Strange Attractors: Creating Patterns in Chaos

by

Julien C. Sprott

Converted to PDF by Robert Coldwell
8/1/2000
coldwell@earthlink.net

Contents

Why This Book Is for You

Art and science sometimes appear in juxtaposition, one aesthetic, the other analytical. This book bridges the two cultures. I have written it for the artist who is willing to devote a modicum of effort to understanding the mathematical world of the scientist and for the scientist who often overlooks the beauty that lurks just beneath even the simplest equations.

If you are neither artist nor scientist, but own a personal computer for which you would like to find an exciting new use, this book is also for you. Fractals generated by computer represent a new art form that anyone can appreciate and appropriate. You don't have to know mathematics beyond elementary algebra, and you don't have to be an expert programmer. This book explains a simple, new technique for generating a class of fractals called strange attractors. Unlike other books about fractals that teach you to reproduce well-known patterns, this one will let you produce your own unlimited variety of displays and musical sounds with a single program. Almost none of the patterns you produce will ever have been seen before.

To get the most out of this book, you will need a personal computer, though it need not be a fancy one. It should have a monitor capable of displaying graphics, preferably in color. Some knowledge of BASIC is useful, although you can just type in the listings even if you don't understand them completely. For those of you who are C programmers, I have provided an appendix with an equivalent version in C. You may find the exercises in this book an enjoyable way to hone your programming skills. As you progress through the book, you will gradually develop a very sophisticated computer program. Each step is relatively simple and brings exciting new things to see and explore. Alternately, you can use the accompanying disk immediately to begin making your own collection of strange attractors.

Strange Attractors

How to find them, those regions
Of space where the equation traces
Over and over a kind of path,
Like the moth that batters its way
Back toward the light
Or, hearing the high cry of the bat,
Folds its wings in a rolling dive?

And ourselves, fluttering toward and away
In a pattern that, given enough
Dimensions and point-of-view,
Anyone living there could plainly see—
Dance and story, advance, retreat,
A human chaos that some slight
Early difference altered irretrievably?

For one, the sound of her mother
Crying. For this other,
The hands that soothed
When he was sick. For a third,
The silence that collects
Around certain facts. And this one,
Sent to bed, longing for a nightlight.

Though we think this time to escape,
Holding a head up, nothing wrong,
Finding a way to beat the system,
Talking about anything else—
Travel, the weather, time
At the flight simulator—for some
The journey circles back

To those strange, unpredictable attractors,
Secrets we can neither speak nor leave.

—Robin S. Chapman

Chapter 1

Order and Chaos

This chapter lays the groundwork for everything that follows in the book. Nearly all the essential ideas, mathematical techniques, and programming tools you need are developed here. Once you've mastered the material in this chapter, the rest of the book is smooth sailing.

1.1 Predictability and Uncertainty

The essence of science is predictability. Halley's comet will return to the vicinity of Earth in the year 2061. Not only can astronomers predict the very minute when the next solar eclipse will occur but also the best vantage point on Earth from which to view it. Scientific theories stand or fall according to whether their predictions agree with detailed, quantitative observation. Such successes are possible because most of the basic laws of nature are *deterministic*, which means they allow us to determine exactly what will happen next from a knowledge of present conditions.

However, if nature is deterministic, there is no room for free will. Human behavior would be predetermined by the arrangements of the molecules that make up our brains. Every cloud that forms or flower that grows would be a direct and inevitable result of processes set into motion eons ago and over which there is no possibility for exercising control. Perfect predictability is dull and uninteresting. Such is the philosophical dilemma that often separates the arts from the sciences.

One possible resolution was advanced in the early decades of the 20th century when it was discovered that the quantum mechanical laws that govern the behavior of atoms and their constituents are apparently *probabilistic*, which means they allow us to predict only the probability that something will happen. Quantum mechanics has been extremely successful in explaining the submicroscopic world, but it was never fully embraced by some scientists, including Albert Einstein, who until his dying day insisted that he did not believe that God plays dice with the Universe.

Since the 1970s science has been undergoing an intellectual revolution that may be as significant as the development of quantum mechanics. It is now widely understood that deterministic is not the same as *predictable*. An example is the weather. The weather is governed by the atmosphere, and the atmosphere obeys

deterministic physical laws. However, long-term weather predictions have improved very little as a result of careful, detailed observations and the unleashing of vast computer resources.

The reason for this unpredictability is that the weather exhibits extreme sensitivity to initial conditions. A tiny change in today's weather (the initial conditions) causes a larger change in tomorrow's weather and an even larger change in the next day's weather. This sensitivity to initial conditions has been dubbed the *butterfly effect*, because it is hypothetically possible for a butterfly flapping its wings in Brazil to set off tornadoes in Texas. Since we can never know the initial conditions with perfect precision, long-term prediction is impossible, even when the physical laws are deterministic and exactly known. It has been shown that the *predictability horizon* in weather forecasting cannot be more than two or three weeks.

Unpredictable behavior of deterministic systems has been called *chaos*, and it has captured the imagination of the scientist and nonscientist alike. The word "chaos" was introduced by Tien-Yien Li and James A. Yorke in a 1975 paper entitled "Period Three Implies Chaos." The term "strange attractors," from which this book takes its title, first appeared in print in a 1971 paper entitled "On the Nature of Turbulence," by David Ruelle and Floris Takens. Some people prefer the term "chaotic attractor," because what seemed strange when first discovered in 1963 is now largely understood.

It's not hard to imagine that if a system is complicated (with many springs and wheels and so forth) and hence governed by complicated mathematical equations, then its behavior might be complicated and unpredictable. What has come as a surprise to most scientists is that even very simple systems, described by simple equations, can have chaotic solutions. However, *everything* is not chaotic. After all, we can make accurate predictions of eclipses and many other things. An even more curious fact is that the same system can behave either predictably or chaotically, depending on small changes in a single term of the equations that describe the system. For this reason, chaos theory holds promise for explaining many natural processes. A stream of water, for example, exhibits smooth (*laminar*) flow when moving slowly and irregular (*turbulent*) flow when moving more rapidly. The transition between the two can be very abrupt. If two sticks are dropped side-by-side into a stream with laminar flow, they stay close together, but if they are dropped into a turbulent stream, they quickly separate.

Chaotic processes are not random; they follow rules, but even simple rules can produce extreme complexity. This blend of simplicity and unpredictability also occurs in music and art. A piece of music that consists of random notes or of an endless repetition of the same sequence of notes would be either disastrously

discordant or unbearably boring. Likewise, a work of art produced by throwing paint at a canvas from a distance or by endlessly replicating a pattern, as in wallpaper, is unlikely to have aesthetic appeal. Nature is full of visual objects, such as clouds and trees and mountains, as well as sounds, like the cacophony of excited birds, that have both structure and variety. The mathematics of chaos provides the tools for creating and describing such objects and sounds.

Chaos theory reconciles our intuitive sense of free will with the deterministic laws of nature. However, it has an even deeper philosophical ramification. Not only do we have freedom to control our actions, but also the sensitivity to initial conditions implies that even our smallest act can drastically alter the course of history, for better or for worse. Like the butterfly flapping its wings, the results of our behavior are amplified with each day that passes, eventually producing a completely different world than would have existed in our absence!

1.2 Bucks and Bugs

Enough philosophizing—it's time to look at a specific example. This example requires some mathematics, but the equations are not difficult. The ideas and terminology are important for understanding what is to follow.

Suppose you have some money in a bank account that provides interest, compounded yearly, and that you don't make any deposits or withdrawals. Let's let *X* represent the amount of money in your account. When the time comes for the bank to credit your interest, its computer does so by multiplying *X* by some number. With an interest rate of 10%, the number is 1.1, and your new balance is 1.1 *X*. If your balance in the nth year is X_n (where n is 1 after the first year, 2 after the second, and so forth), your balance in the year n +1 is

$$X_{n+1} = R\,X_n \qquad \text{(Equation 1A)}$$

where *R* is equal to 1.0 plus your interest rate. (*R* is 1.1 in this example.)

You probably know that such compounding leads to exponential growth. In terms of the initial amount X_0, the amount in your account after n years is

$$X_n = X_0 R^n \qquad \text{(Equation 1B)}$$

After 50 years at 10% yearly interest, you will have $117.39 for every dollar you initially invested. The bank can afford to do this only because of inflation and

because money is loaned at an even higher interest rate.

Equation 1A is applicable to more than compound interest. It's how many of us have our salaries determined. It also describes population growth. Imagine some species of bug that lives for a season, lays its eggs, and then dies (thus avoiding the confusion of overlapping generations). The next year the eggs hatch, and the number of bugs is some constant *R* times the number in the previous year. If *R* is less than 1, the bugs die out over a number of years; and if *R* is greater than 1, their number grows exponentially.

You also know that exponential growth cannot go on forever, whether it be bucks in the bank, bugs in the back yard or people on the planet. Eventually something happens, such as the depletion of resources, to slow down or even reverse growth. Mass starvation, disease, crime, and war are some of the mechanisms that limit unbridled human population growth. Thus we need to modify Equation 1A in some way if it is to model growth patterns in nature more closely.

Perhaps the simplest modification is to multiply the right-hand side of Equation 1A by a term such as (1 - *X*), whose value approaches 1 as *X* gets smaller (much less than 1) but is less than 1 as *X* increases. Since the population dies abruptly as *X* approaches 1, we must think of *X* = 1 as representing some large number of dollars or bugs (say a million or a billion); otherwise we would never get very far! So our modified equation, called the *logistic equation*, is

$$X_{n+1} = R\, X_n\, (1 - X_n) \qquad \text{(Equation 1C)}$$

Now you're going to get your first homework assignment. Take your pocket calculator and start with a small value of *X*, say 0.1. To reduce the amount of work you have to do, use a fairly large value of *R*, say 2, corresponding to a doubling every year. Run *X* through Equation 1C a few times and see what happens. This process is called *iteration*, and the successive values are called *iterates*. If you did it right, you should see that *X* grows rapidly for the first couple of steps, and then it levels off at a value of 0.5. The first few values should be approximately 0.1, 0.18, 0.2952, 0.4161, 0.4859, 0.4996, and 0.5. Compare your results with the unbounded growth of Equation 1A.

You might have predicted the above result, if you had thought to set X_{n+1} equal to X_n in Equation 1C and solved for X_n. This value is called a *fixed-point solution* of the equation, because if *X* ever has that value, it remains fixed there forever. Such a fixed-point solution is sometimes called a *point attractor*, because every initial value of *X* between 0 and 1 is attracted to the fixed point upon repeated iteration of Equation 1C. Try initial values of *X* = 0.2 and *X* = 0.8. A fixed point is also

called a *critical point*, a *singular point*, or a *singularity*.

If you're curious, you might wonder what happens if you start with a value of *X* less than 0, such as -0.1, or greater than 1, such as 1.1. You should verify that the iterates are negative and that they get larger and larger, eventually approaching minus infinity. We say that the solution is *unbounded* and that it attracts to infinity. Thus the values of *X* = 0 and *X* = 1 are like a watershed. Between these values the solution is *bounded*, and outside these values it is unbounded.

The region between *X* = 0 and *X* = 1 is called a *basin of attraction* because it resembles a bathroom basin in which drops of water find their way to the drain from wherever they start. *X* = 0 is also a fixed point, but it is *unstable* because values either slightly above or slightly below zero move away from zero. Such an unstable fixed point is sometimes called a *repellor*. Chaos can result when two or more repellors are present; the iterates then bounce back and forth like a baseball runner caught in a squeeze play.

Equations that exhibit chaos have solutions that are unstable but bounded; the solution never settles down to a fixed value or even to a repeating pattern, but neither does it move off to infinity. Sometimes we say that such equations are *linearly unstable* but *nonlinearly stable*. Small perturbations to the system grow, but the growth ceases when the nonlinear terms become important, as eventually they must. Another way to say it is that the fixed points are *locally unstable*, but the system is *globally stable*. In this case initial conditions are drawn to a special type of attractor called a *strange attractor*, which is not a point or even a finite set of points but rather a complicated geometrical object whose properties constitute the subject of this book.

See what happens if you substitute *X* = 0 or *X* = 1 into the logistic equation. As a check on your calculations, or in case you didn't do your homework, Table 1-1 shows the successive iterates of *X* for each of the cases we have discussed.

Table 1-1. Iterates of the logistic equation for various initial values of *X* with *R*=2

n = 0	n = 1	n = 2	n = 3	n = 4	n = 5	n = 6
0.1	0.18	0.2952	0.4161	0.4859	0.4996	0.5
0.2	0.32	0.4352	0.4916	0.4999	0.5	0.5
0.8	0.32	0.4352	0.4916	0.4999	0.5	0.5
-0.1	-0.22	-0.5368	-1.6499	-8.7442	-170.41	-58421
1.1	-0.22	-0.5368	-1.6499	-8.7442	-170.41	-58421
0.0	0.0	0.0	0.0	0.0	0.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0

An equation, such as the logistic equation, that predicts the next value of a quantity from the previous value is called an *iterated map* because it is like a road map in which each point on the earth is mapped to a corresponding point on a piece of paper. The logistic equation is a one-dimensional map because the various X values can be thought of as lying along a straight line that stretches from minus infinity to plus infinity. Each iteration of the map moves every point along the line to a new position on the line. For the example above with $R = 2$, all the points between $X = 0$ and $X = 1$ walk toward $X = 0.5$, where they stop and remain. Other points run faster and faster toward the end of the line that stretches to minus infinity.

The logistic equation is an example of a *quadratic iterated map*, so called because if you multiply out the right-hand side of Equation 1C, it has not only a linear term RX_n but also a quadratic (squared) term $-RX_n^2$. Quadratic maps are *noninvertable* because you can find X_{n+1} from X_n, but can't go backward because there are two values of X_n that produce the same X_{n+1}, and there is no way of knowing from which it came. For example, Table 1-1 shows that $X_0 = 0.2$ and $X_0 = 0.8$ both produce $X_1 = 0.32$. These are the two roots of the quadratic equation that you get if you try to solve for X_n in Equation 1C in terms of X_{n+1}.

The graph of X_{n+1} versus X_n is a curve called a *parabola*. Because a parabola is not a straight line, the map is said to be *nonlinear*. Chaos and strange attractors require nonlinearity. The interesting and surprising behavior of nonlinear iterated maps is the basis for much of this book.

The first surprising result occurs if you iterate Equation 1C with $R = 3.2$ and an initial value of X in the range of 0 to 1. After a few iterations the solution will alternate between two values of approximately 0.5130 and 0.7995. This is called a *period-2 limit cycle*. Like the fixed point, the limit cycle is another type of simple attractor. It is sometimes called a *periodic* or *cyclic attractor*.

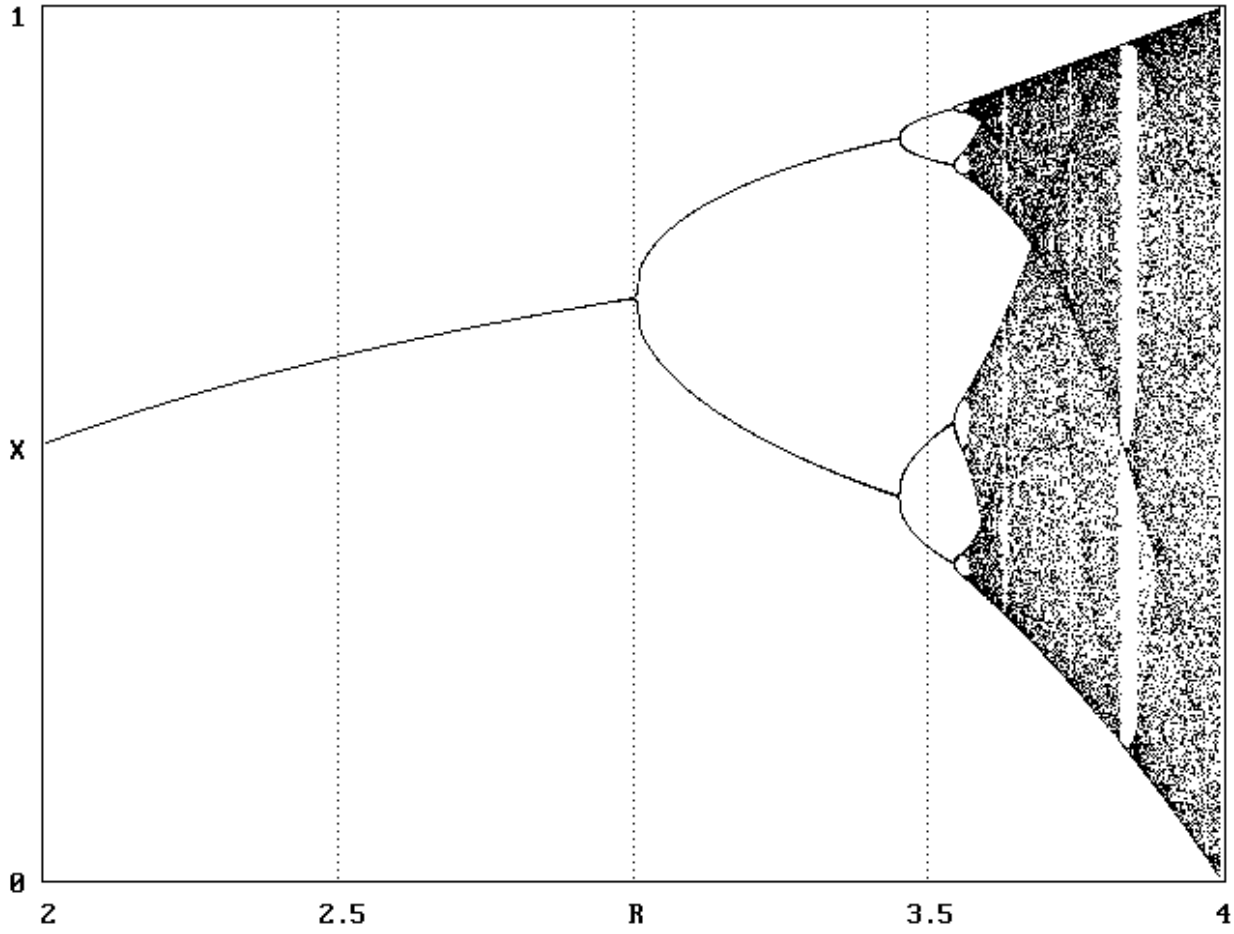
It's not hard to see how cyclic behavior might arise in nature. If the population of beetles grows too large, they deplete the plants on whom they depend for food. With too few plants, the beetles die out, allowing the number of plants to recover, leading to the next cycle of beetle growth, and so forth.

Increase R a bit more to 3.5, and repeat the calculation. The result is a *period-4 limit cycle* with four values of approximately 0.5009, 0.8750, 0.3828, and 0.8269. If you keep increasing R by ever smaller amounts, the period of the limit cycle doubles repeatedly, finally reaching chaotic behavior (an infinite period) at about $R = 3.5699456$. This value is sometimes called the *Feigenbaum point*, after Mitchell J. Feigenbaum, a contemporary mathamatician who discovered many of the interesting properties of one-dimensional maps.

When chaos occurs, the successive iterates fluctuate in an apparently random and irreproducible manner. The chaotic behavior persists up to $R = 4$ except for an infinite number of small periodic windows. For R greater than 4, the solution is unbounded, and the iterates attract rapidly to minus infinity.

The behavior described above can be summarized in a *bifurcation diagram*, as shown in Figure 1-1, in which the limiting iterated values of the logistic equation, after discarding the first few hundred iterates, are plotted for a range of R from 2 to 4. This plot is called the *Feigenbaum diagram*, and it resembles a tree on its side. ("Feigenbaum," appropriately but coincidentally, is German for "fig tree.") You see the fixed-point solution for R less than 3, the period-doubling route to chaos, and the periodic windows at large R. The chaotic regions toward the right side of the figure are characterized by values of X that span a wide range and eventually fill the region densely with points.

Figure 1-1. Bifurcation diagram for the logistic equation, $X_{n+1} = RX_n (1 - X_n)$



Each period doubling is called a *bifurcation* because a single solution splits into a pair of solutions. These splittings are called *pitchfork bifurcations* for obvious reasons. Note the period-3 window at about $R = 3.84$. The period-3 region begins abruptly when R is increased slightly from within the chaotic region to its left in what is called a *tangent* or *saddle-node bifurcation*. Careful inspection of the period-3 window shows that it also undergoes a period-doubling sequence at about $R = 3.85$. Solutions with every period can be found somewhere between $R = 3$ and $R = 4$.

Successive period doublings occur with ever-increasing rapidity as one moves from left to right in Figure 1-1. The ratio of the width of each region to the width of the previous region approaches a constant equal to 4.669201660910..., called the *Feigenbaum number*. Even more remarkable is that this number arises in many different chaotic systems in nature as well as in the solutions of equations. The universality of the Feigenbaum number in chaos is reminiscent of the ubiquity of the number π in Euclidean geometry.

With $R = 4$ the solutions occupy the entire interval from $X = 0$ to $X = 1$. Eventually X takes on a value arbitrarily close to any point in that interval (a characteristic called *topological transitivity*). Curiously, however, infinitely many initial values of X don't lead to a chaotic solution even for $R = 4$. For example $X_0 = 0.5$ and $X_0 = 0.75$ lead to unstable fixed points, while $X_0 = 0.345491...$ and $X_0 = 0.904508...$ produce an unstable period-2 limit cycle. By unstable we mean that if the initial values are wrong by even the slightest amount, successive iterates will wander ever farther away.

Even though there are infinitely many nonchaotic initial values between zero and one, the chance that you will find one by randomly guessing is negligible. For every such value, there are infinitely many others that produce chaos. Such a seemingly paradoxical entity is an example of a *Cantor set*, named after the 19th-century Russian-born German mathematician Georg Cantor who is often credited with developing a mathematically rigorous concept of infinity.

A Cantor set contains infinitely many members (in fact, uncountably infinitely many), but its members represent a zero fraction of the total! For example, infinitely many points are required to cover completely the circumference of a circle, but this number of points doesn't even begin to cover its interior. Such a collection (or set) of points, although infinite in number, is said to comprise a *set of measure zero*, because the points fill a negligible portion of the plane. An attractor is a set of measure zero, but its basin of attraction has a nonzero measure.

Few people would have guessed that such complexity could arise from such underlying simplicity. Furthermore, the logistic equation is only the simplest of an endless variety of equations that can exhibit chaos. It is this dichotomy of simplicity

and complexity that makes chaos beautiful to the mathematician and artist alike. In the bifurcation diagram of the logistic equation, we have something with aesthetic appeal, and it came from a simple quadratic equation!

1.3 The Butterfly Effect

If our goal is to seek chaotic behavior in the solution of equations, we need a simple way to test for chaos. For this purpose we use the fact that chaotic processes exhibit extreme sensitivity to initial conditions, in contrast to regular processes in which different starting points usually converge to the same sequence of points on a simple attractor.

Suppose we iterate the logistic equation with two initial values of X that differ by only a tiny amount. Think of these values as representing two states of the atmosphere that differ only by the flapping of the wings of a butterfly. If successive iterates are attracted to a fixed point as they are for $R = 2$, the difference between the two solutions must get smaller and smaller as the fixed point is approached. A similar thing happens for a limit cycle. The difference between the two solutions will on average decrease exponentially.

If the solution is chaotic, as is the logistic equation for $R = 4$, the successive iterates for the two cases initially on average get farther apart; the difference usually increases exponentially. If the difference doubles on average with every iteration, we say the Lyapunov exponent is 1. If it is reduced by half, we say the *Lyapunov exponent* is -1. The name comes from the late-19th-century Russian mathematician Aleksandr M. Lyapunov (sometimes transliterated Liapunov or Ljapunov).

You can think of the Lyapunov exponent as the power of 2 by which the difference between two nearly equal X values changes on average for each iteration. Thus the difference between the values changes by an average of 2^L for each iteration. If L is negative, the solutions approach one another; if L is positive, we have sensitivity to initial conditions and hence chaos.

One way to detect chaos is to iterate the equation with two nearly equal initial values and see if, after many iterations, the values are closer together or farther apart. Another way is to make use of a principle of calculus that says that the difference in the solutions after one iteration divided by the difference before the iteration, provided the difference is small, is equal to the derivative of the equation for the map, which for the logistic equation is

$$\Delta X_{n+1} / \Delta X_n = R(1 - 2X_n) \qquad \text{(Equation 1D)}$$

where ΔX is the difference between the two values of X. In Equation 1D, ΔX_n is the difference in the X values after n iterations, and ΔX_{n+1} is the difference after $n+1$ iterations.

Since ΔX increases by the factor on the right of Equation 1D for each iteration, the proper way to calculate the average is to start with a value of 1 and multiply it repeatedly by the right-hand side of Equation 1D at each iteration, then divide the result by the number of iterations, and finally take the logarithm to the base 2 of the absolute value of the result to get the Lyapunov exponent. If you prefer an equation, the preceding description is equivalent to

$$L = \sum \log_2 | R (1 - 2X_n) | / N \qquad \text{(Equation 1E)}$$

where the vertical bars mean that you are to disregard the sign of the quantity inside, and $\sum$ means to sum the quantity to its right from a value of $n = 1$ to a value of $n = N$, where N is some large number. The larger the value of N, the more accurate the estimate of L.

Suppose you knew the value of X to within 0.01 for an iterated map with $L = 1$. After one iteration the uncertainty would be about 0.02, and after two iterations the uncertainty would be about 0.04, and so forth. After about seven iterations, the error would exceed 1, and your prediction would be totally worthless. If the X values are expressed as binary numbers, each iteration would result in throwing away the rightmost (least significant) binary digit (*bit*). Thus the units of L are bits per iteration. Sometimes L is expressed in terms of the natural logarithm (base e) rather than $\log_2$. The Lyapunov exponent is the rate at which information is lost when a map is iterated.

It is as if a succession of cartographers each copied maps from one another, but every time one was copied it was only half as accurate as the previous one. If the original map were accurate to 1%, the next copy would be accurate to 2%, and the seventh generation copy would bear no relation to the original. If the Lyapunov exponent were -1, one bit of information would be *gained* at each iteration. Even a completely unknown initial condition would eventually be perfectly accurate as it approached the known fixed point or limit cycle. Unfortunately, negative Lyapunov exponents are not the rule in cartography; otherwise all our maps would be self-correcting!

Figure 1-2. Lyapunov exponent for the logistic equation

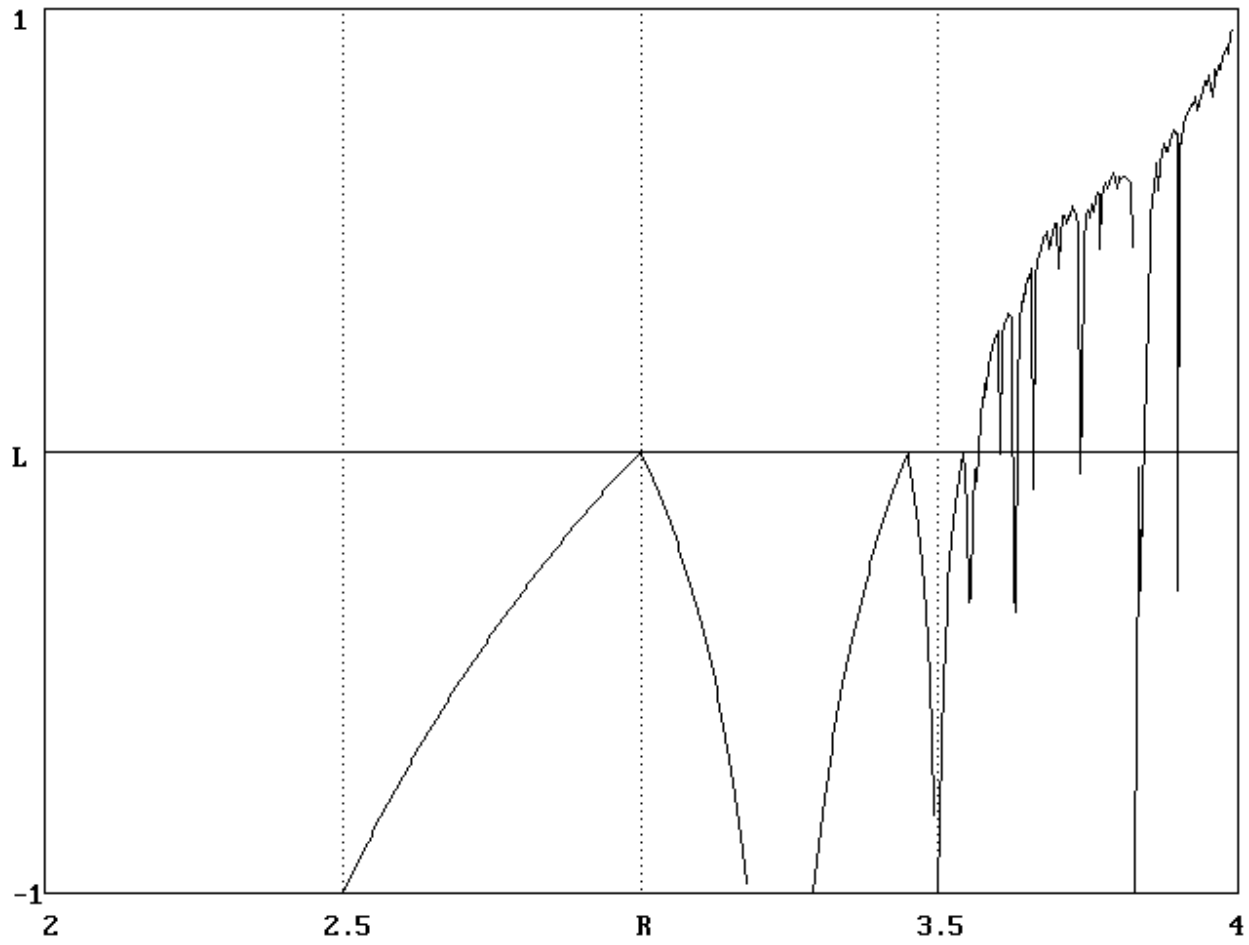


Figure 1-2 shows the Lyapunov exponent for the logistic equation using values of *R* from 2 to 4. The Lyapunov exponent is 1.0 at *R* = 4 because that value causes the interval of *X* from 0 to 1 to be mapped backed onto itself with a single fold at *X* = 0.5. Thus information is lost at a rate of 1 bit per iteration, because each iterate has two possible predecessors. You can also see some of the periodic windows where *L* dips below zero toward the right edge of the plot. Also note that *L* is zero wherever a bifurcation occurs, for example at *R*=2. At these points the solution is fraught with indecision over which branch to take, and the initial uncertainty persists forever, neither increasing nor decreasing.

1.4 The Computer Artist

By now you have probably surmised that the operations we have described are best carried out by a computer. The equations are simple, but they must be applied repeatedly. This is precisely the kind of task at which computers excel.

There are dozens of computer types and programming languages to choose from. Currently the most popular computers are those based on the IBM PC running the MS-DOS or IBM-DOS operating system (hereafter simply called DOS). The most widely available programming language is BASIC (Beginner's All-purpose Symbolic Instruction Code), which usually comes bundled with the operating system software included with the computer. A version of BASIC called QBASIC has been included with DOS since version 5.0. BASIC may not be the most advanced computer language, but it is one of the easiest to learn and to use, its commands are close to ordinary English, and it is more than adequate for our purposes. Furthermore, modern versions of BASIC compare favorably with the best of the other languages.

The American National Standards Institute (ANSI) has established a standard for the BASIC language, but it is somewhat limited, and most versions of BASIC have many additions and embellishments. We will intentionally use a primitive dialect to ensure compatibility with most modern implementations and to simplify the translation into incompatible versions. In particular, the programs in this book should run without modification under Microsoft BASICA, GW-BASIC, QBASIC, QuickBASIC, VisualBASIC for MS-DOS; Borland International Turbo BASIC (no longer available); and Spectra Publishing PowerBASIC on IBM PCs or compatibles. You will be happiest using a modern compiled BASIC such as VisualBASIC or PowerBASIC on a fast computer with a math coprocessor.

Appendix C includes information on translating the computer programs into other, partially incompatible dialects of BASIC, as well as source code for use with VisualBASIC for Windows and Microsoft QuickBASIC for the Macintosh. Appendix D contains a translation into Microsoft QuickC. The BASIC programs use line numbers, which have been obsolete since the mid-1980s, but they are harmless, and they provide a convenient way to reference lines of the program and to indicate where in the program a change is to be made.

If you follow sequentially through this book, you will need to add and change a only few lines of the program as you meet each new idea. Your program will gradually grow more versatile as you work through the book. In the end you will have a powerful program that can reproduce all the examples in this book as well as an endless variety of new ones. Hence you should avoid the temptation to

eliminate or to change the line numbers, at least until you have a fully functional program. You may prefer to jump to Appendix B where you will find the complete final program, which is also provided on the accompanying disk along with source listings in BASIC, Microsoft QuickC, Borland Turbo C++ and a ready-to-run executable version of the program.

If you are an experienced programmer, you might ridicule some of the quaint program listings. Many powerful programming structures such as block IF statements, DO LOOPs, and callable subroutines with local variables that produce beautifully structured programs are now standard, but they have been avoided to allow backwards compatibility with more primitive versions of BASIC. They also often impose a small speed penalty. The dreaded GOTO statement has been used primarily to bypass blocks of code in deference to BASIC versions that don't support block IF statements. Lines of the program that are bypassed by a GOTO are usually indented. Blocks of the program contained within FOR...NEXT loops have also been indented. In the interest of structure and simplicity, the programs have been written using numerous small modular subroutines, each with a single entry point beginning with a comment line, and a single exit point containing a RETURN statement, albeit with global variables. The individual subroutines are separated with blank lines. It should be relatively easy for an experienced programmer to rewrite the program in a more modern format.

The program listing **PROG01** iterates the logistic equation for $R = 4$ with an initial value of $X = 0.05$ and makes a graph of each iterate versus its predecessor. The program looks more complicated than it actually is because the various operations have been relegated to subroutines to provide a template for the more versatile cases to follow.

PROG01. Program for iterating and graphing the logistic equation

```
1000 REM LOGISTIC EQUATION

1010 DEFDBL A-Z                 'Use double precision

1030 SM% = 12                   'Assume VGA graphics

1190 GOSUB 1300                 'Initialize

1200 GOSUB 1500                 'Set parameters

1210 GOSUB 1700                 'Iterate equations
```

```
1220 GOSUB 2100                 'Display results
1230 GOSUB 2400                 'Test results
1240 ON T% GOTO 1190, 1200, 1210
1250 CLS
1260 END

1300 REM Initialize
1320 SCREEN SM%                 'Set graphics mode
1350 WINDOW (-.1, -.1)-(1.1, 1.1)
1360 CLS
1420 RETURN

1500 REM Set parameters
1510 X = .05                    'Initial condition
1560 R = 4                      'Growth rate
1570 T% = 3
1590 LINE (-.1, -.1)-(1.1, 1.1), , B
1630 RETURN

1700 REM Iterate equations
1720 XNEW = R * X * (1 - X)
2030 RETURN
```

```
2100 REM Display results

2300 PSET (X, XNEW)             'Plot point on screen

2320 RETURN



2400 REM Test results

2490 IF LEN(INKEY$) THEN T% = 0 'Respond to user key stroke

2510 X = XNEW                   'Update value of X

2550 RETURN
```

If, when you first run the program, your computer reports an error, it is probably in one of the following lines:

Line 1010: Be sure your version of BASIC supports double-precision (four-byte) floating-point variables. If it doesn't, you may omit this line, but then you probably will have to change the 4 in line 1560 to 3.99999 to avoid overflow resulting from round-off errors. With modern versions of BASIC and a computer with a math coprocessor, there is no penalty, and considerable advantage, in using double precision. Because of the finite precision of computer arithmetic, all cases will eventually repeat, but with double precision the average number of iterations required before this happens is acceptably large.

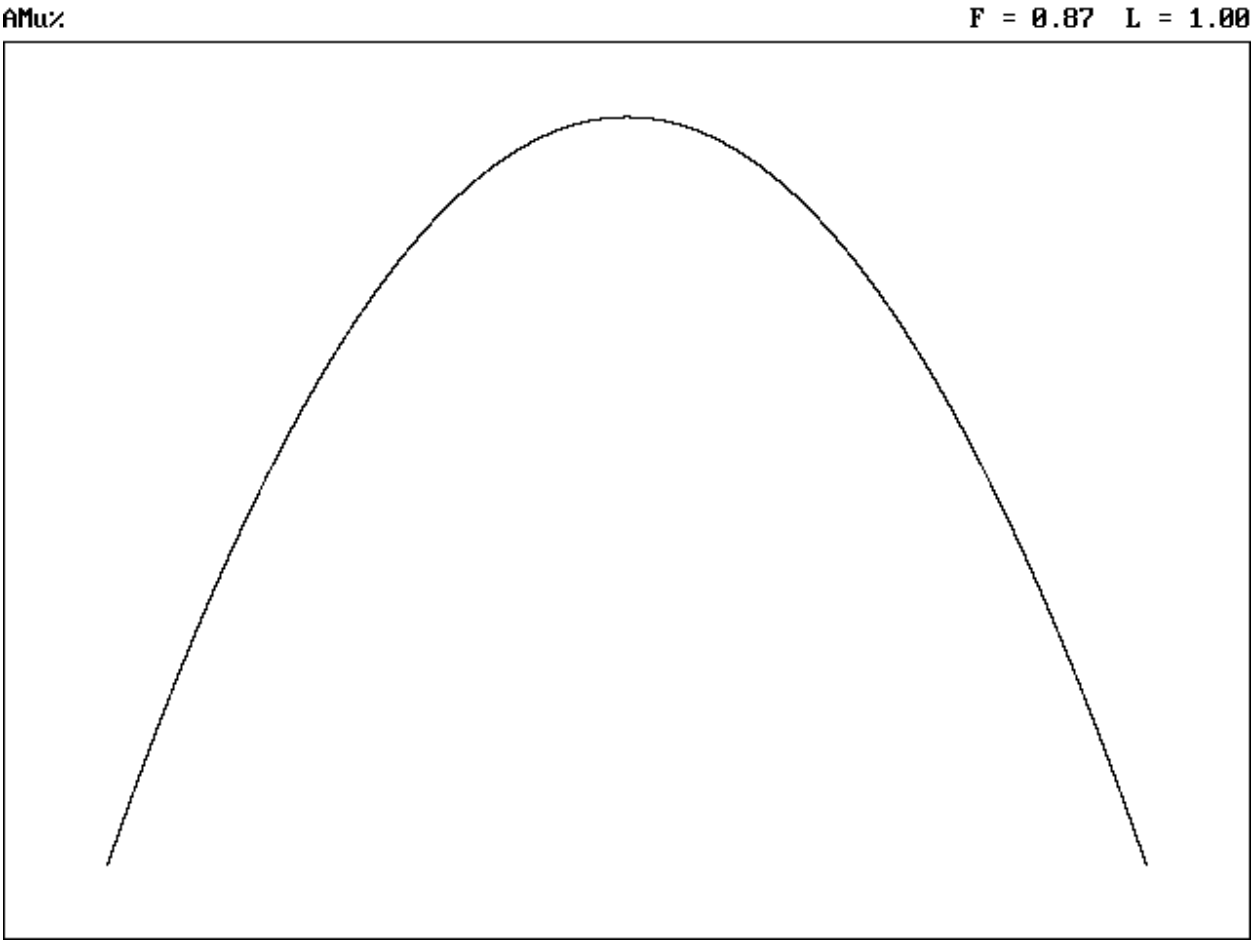
Line 1320: Either your version of BASIC doesn't require this command or your computer or compiler doesn't support VGA graphics. Try reducing the 12 in line 1030 to a lower number until you find one that works. If none works, try eliminating line 1320 altogether.

Line 1350: The WINDOW command defines the coordinates of the lower-left and upper-right corners of the graphics window for subsequent PSET and LINE commands. If your version of BASIC doesn't support this command, you must delete this line and convert all the parameters in the PSET and LINE commands to address screen pixels. In this case try replacing line 2300 with PSET (200 * X, 200 - 200 * XNEW). One advantage of using the WINDOW command is that when a version of BASIC comes along that supports higher screen resolutions, the program can be easily recompiled to take advantage of it.

Other errors: Look carefully for typographical errors, or consult your BASIC manual to determine compatibility.

The correct program should produce a plot of the logistic parabola, as shown in Figure 1-3. Try different initial values of *X* (line 1510) and different values of *R* (line 1560) to confirm the behavior predicted for the logistic equation.

Figure 1-3. The logistic parabola from PROG01



The logistic parabola comes from a chaotic solution, but it doesn't look very complicated, and it would hardly qualify as art. With one small change we can make things more interesting and, at the same time, illustrate sensitivity to initial conditions. Instead of plotting each iterate versus its immediate predecessor, we could plot it versus its second or third or fourth predecessor. Let's save the last 500 iterates and provide the option to plot *X* versus any one of them.

The changes that you need to make in the program **PROG01** to accomplish this are shown in the listing **PROG02**. You can either go through the program and change or add lines as necessary or type the listing and save it in ASCII format and then use the MERGE command supported by many (mostly old) versions of BASIC to update the previous version of the program.

PROG02. Changes required in PROG01 to plot the fifth previous iterate

```
1000 REM LOGISTIC EQUATION (5th Previous Iterate)

1020 DIM XS(499)

1040 PREV% = 5                  'Plot versus fifth previous iterate



1580 P% = 0



2210 XS(P%) = X

2220 P% = (P% + 1) MOD 500

2230 I% = (P% + 500 - PREV%) MOD 500

2300 PSET (XS(I%), XNEW)               'Plot point on screen
```

If you set *PREV%* = 1 in line 1040, the result is the same as for **PROG01**. However, if you set *PREV%* equal to 2, you see the logistic parabola change into a curve with two humps. Each time you increase *PREV%* by 1, you double the number of humps in the curve. Thus *PREV%* = 5 results in 16 oscillations, as shown in Figure 1-4.

Figure 1-4. The logistic parabola after five iterations from PROG02

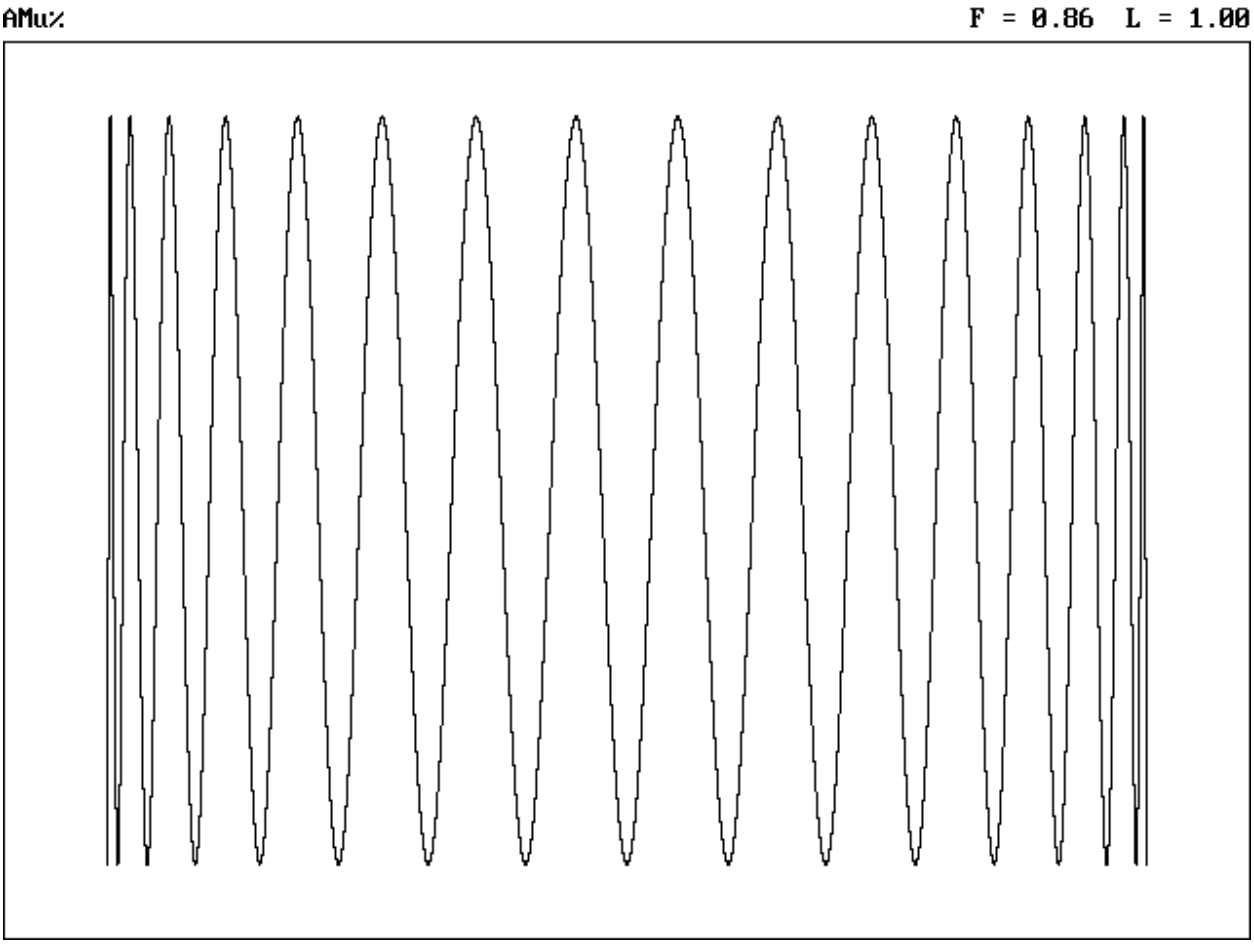


Figure 1-4 provides a good graphical illustration of the sensitivity to initial conditions. The horizontal axis represents all possible initial conditions from zero to one. The vertical axis shows the value from zero to one corresponding to each initial condition after five iterations. It's not hard to see that two nearby points on the horizontal axis usually translate into two very different values along the vertical axis after five iterations. Try using *PREV%* = 10, and convince yourself that information about the initial condition is almost completely lost after ten iterations.

This exercise provides a good insight into the way a strange attractor is formed geometrically. The logistic parabola, which began as a line (a one-dimensional object), is stretched and folded with each iteration, eventually filling the entire plane (a two-dimensional object) after many iterations. Perhaps it reminds you of those taffy machines that repeatedly stretch and fold the taffy, causing two nearby specks in the taffy after a while to be nowhere near one

another. On average the distance between the specks initially increases at an exponential rate.

You should be able to think of many other examples of sensitivity to initial conditions. When you stir your coffee to mix in the cream, you're relying on a chaotic process. Two sticks dropped into the water close together just above a waterfall eventually end up far apart. Try laying two identical garden hoses side by side, and turn on the water in each one at the same time without holding the ends. Chaotic processes are all around us. Their mathematical solutions usually produce chaotic strange attractors, whose diversity and beauty we are about to explore.

Chapter 2

Wiggly Lines

In this chapter we will teach the computer to search for chaotic solutions of simple equations with a single variable. The solutions are segments of lines, but the lines can wiggle in an incredibly complicated manner.

2.1 More Knobs to Twiddle

The logistic equation (Equation 1C) is an example of a *dynamical system*. Such systems are described by deterministic initial-value equations. This particular system has a single parameter *R* whose value determines the solution's behavior for all initial values of *X* within the basin of attraction. This parameter is like a knob on a radio or on a stove that you can turn up or down to control the sound emitted by the radio or the convection in a pot of boiling soup.

You can do a simple experiment to observe the period-doubling route to chaos. Go into your bathroom or kitchen and turn on the tap, only slightly, to produce a regular periodic pattern of drips. Now slowly open the tap until the pattern becomes chaotic. Just before the onset of chaos, if you are sufficiently careful and patient, you should observe one or more period doublings where the sound changes to something like "drip drip—drip drip—drip drip." The knob that controls the flow rate corresponds to the parameter *R* in the logistic equation. The dripping faucet has been extensively studied by Robert Shaw and discussed at length in his book *The Dripping Faucet as a Model Chaotic System*.

Usually a dynamical system has more than one knob. Your kitchen faucet probably has independent control of the flow rate and the temperature of the water. With more knobs, you might expect to increase the variety of ways the system can behave. Such knobs are called *control parameters*.

The formula for the most general one-dimensional quadratic iterated map is

$$X_{n+1} = a_1 + a_2X_n + a_3Xn_2 \qquad \text{(Equation 2A)}$$

where a_1, a_2, and a_3 are three control parameters. By exploring all combinations of their values, we expect eventually to observe every possible peculiar solution that the equation can have.

You might think that the initial condition X_0 is a fourth knob, but if the system is chaotic, the solution is generally a strange attractor, and all initial conditions within the basin of attraction look the same after many iterations. Of course there is no guarantee that a particular choice of X_0 lies within the basin, but values of X_0 close to zero are within the basin about half the time, and there are so many chaotic solutions over the range of the other three parameters that we can well afford to discard half of them.

The search for strange attractors proceeds as follows. Choose values for a_1, a_2, and a_3 arbitrarily. Start with a value of X_0 near zero. Iterate Equation 2A repeatedly until the solution either exceeds some large number, in which case it is presumably unbounded, or until the Lyapunov exponent becomes small or negative, in which case the solution is probably a fixed point or limit cycle. In either event, choose a different combination of a_1, a_2, and a_3, and start over. If, after a few thousand iterations, the solution is bounded (X is not enormous) and the Lyapunov exponent is positive, then it is likely that you have found a strange attractor.

2.2 Randomness and Pseudorandomness

To choose values of a_1, a_2, and a_3, we can use the random-number generator provided with most computer languages. The random numbers thus produced are usually uniformly distributed between zero and one. You may wonder how a computer, the epitome of determinism, could ever produce a random number. This question deserves a digression because the answer provides yet another example of the very issues we have been discussing.

One way to produce a random number is to start with a value of X (the *seed*) between zero and one and iterate the logistic equation with $R = 4$ a few dozen times. The result is a new number in the range of zero to one that is related to the seed in a complicated and sensitive way. This number is then used as the seed for the next random number, which is produced in the same way. A given seed will produce the same sequence of random numbers, but the sequence may not be the same on different computers or with different languages or even with different versions of the same language because of the way the numbers are rounded.

However, this method of producing random numbers is not optimal. First, the numbers are not uniformly distributed over the range. They tend to cluster near zero and one as the darkness of the right-hand side of Figure 1-1 suggests. Also, multiplying a non-integer number by itself many times is a relatively slow process on a computer.

Instead, computers usually get their random numbers using the *linear congruential method*:

$$X_{n+1} = (aX_n + b) \bmod c \qquad \text{(Equation 2B)}$$

In the mod (*modulus*) operation, the quantity to the left of the mod ($aX_n + b$) is divided by the quantity to its right (c), and the remainder is kept rather than the quotient. All the quantities in Equation 2B are integers. The constants a, b, and c are carefully chosen to maximize the number of steps required for the sequence to repeat, which in any case can never exceed c. The numbers are uniformly distributed from zero to c - 1, but they can be transformed to the range zero to one by simply dividing X_{n+1} by c. The numbers appear to be random, but since they are produced using a deterministic procedure, they are often called *pseudorandom*. Equation 2B is another example of a one-dimensional chaotic map, which is related to the *shift map*.

Truly random numbers should satisfy infinitely many conditions. Not only must the numbers be uniform over the interval, but there should be no detectable relation between the numbers and any of their predecessors. In particular, the sequence should repeat only after a very large number of steps. Most random-number generators are deficient in certain ways. For example, the random numbers produced by Microsoft QBASIC 1.0, QuickBASIC 4.5, and VisualBASIC for DOS 1.0 repeat after 16,777,216 steps, and this number is too small for some of our purposes.

The situation can be greatly improved by shuffling the numbers. Suppose we maintain a table of a hundred or so random numbers. When we want one, we randomly take an entry from the table and replace it with a new random number. With this simple modification, the pseudorandom numbers generated by the computer are sufficiently random for our purpose.

You should always remember that the sequence of random numbers generated by a digital computer will eventually repeat. You must take care to ensure that over the duration of a calculation, such a repetition does not occur. You must also reseed the random-number generator using a truly random seed, such as one based on the time of day the program is started, if you are to avoid repeating the same sequence each time you run the program.

2.3 What's in a Name?

When we begin to choose random values for the coefficients a_1, a_2, and a_3, we are immediately confronted with two issues. The first is the range of values that the coefficients may have, and the second is the amount by which two values of a coefficient must differ to produce attractors that are visibly different.

We can address the first issue by referring to the logistic equation (Equation 1C). When the value of *R* is too small (less than about 3.5), there are no chaotic solutions, and when the value of *R* is too large (greater than 4), all the solutions are unbounded. A similar situation occurs for the more general one-dimensional quadratic map in Equation 2A. Thus we want to limit the coefficients to values whose magnitudes (positive or negative) are of order unity. That is, 0.1 is probably too small a value and 10 is probably unnecessarily large. This assumption can be verified by numerical experiment.

The second issue requires a subjective judgment of how dissimilar two attractors must look before we consider them to be different. In practice, a change in one of the coefficients by an amount of order 0.1 generally produces an object that is noticeably different. If we let each coefficient take on values ranging from -1.2 to 1.2 in steps of 0.1, we will have 25 possible values. We can associate each with a letter of the alphabet, A through Y, and have a convenient way to catalog and replicate the attractors. Limiting the coefficients to 25 values may seem excessively restrictive, but since there are three coefficients for one-dimensional quadratic maps, there are 25^3 or 15,625 different combinations.

The coefficients that correspond to the logistic equation with $R = 4$ are $a_1 = 0$, $a_2 = 4$, and $a_3 = -4$, and they fall outside the range of -1.2 to 1.2. Thus for some purposes, it is convenient to take a larger range. A convenient way to extend the range is to use the ASCII (American Standard Code for Information Interchange) character set summarized in Table 2-1.

Table 2-1. ASCII character set and associated coefficient values

Char	Dec	Coeff	Char	Dec	Coeff	Char	Dec	Coeff
	32	-4.5	#	64	-1.3	`	96	1.9
!	33	-4.4	A	65	-1.2	a	97	2.0
"	34	-4.3	B	66	-1.1	b	98	2.1

Char	Dec	Coeff	Char	Dec	Coeff	Char	Dec	Coeff
#	35	-4.2	C	67	-1.0	c	99	2.2
$	36	-4.1	D	68	-0.9	d	100	2.3
%	37	-4.0	E	69	-0.8	e	101	2.4
&	38	-3.9	F	70	-0.7	f	102	2.5
‘	39	-3.8	G	71	-0.6	g	103	2.6
(	40	-3.7	H	72	-0.5	h	104	2.7
)	41	-3.6	I	73	-0.4	i	105	2.8
*	42	-3.5	J	74	-0.3	j	106	2.9
+	43	-3.4	K	75	-0.2	k	107	3.0
,	44	-3.3	L	76	-0.1	l	108	3.1
-	45	-3.2	M	77	0.0	m	109	3.2
.	46	-3.1	N	78	0.1	n	110	3.3
/	47	-3.0	O	79	0.2	o	111	3.4
0	48	-2.9	P	80	0.3	p	112	3.5
1	49	-2.8	Q	81	0.4	q	113	3.6
2	50	-2.7	R	82	0.5	r	114	3.7
3	51	-2.6	S	83	0.6	s	115	3.8
4	52	-2.5	T	84	0.7	t	116	3.9
5	53	-2.4	U	85	0.8	u	117	4.0
6	54	-2.3	V	86	0.9	v	118	4.1
7	55	-2.2	W	87	1.0	w	119	4.2
8	56	-2.1	X	88	1.1	x	120	4.3

Char	Dec	Coeff	Char	Dec	Coeff	Char	Dec	Coeff
9	57	-2.0	Y	89	1.2	y	121	4.4
:	58	-1.9	Z	90	1.3	z	122	4.5
;	59	-1.8	[	91	1.4	{	123	4.6
<	60	-1.7	\	92	1.5	|	124	4.7
=	61	-1.6	]	93	1.6	}	125	4.8
>	62	-1.5	^	94	1.7	~	126	4.9
?	63	-1.4	_	95	1.8	_	127	5.0

ASCII codes from 0 to 31 are reserved for control codes—things like backspace, carriage return, and line feed. Codes from 128 to 255 can also be used, but there is no universal character set associated with them. By making use of all the ASCII characters from 0 to 255, we can accommodate coefficients in the range of -7.7 to 17.8. The characters listed in the table will suffice for most of our needs, however.

With such a coding scheme, we can represent each attractor by a sequence of characters, with each character corresponding to one of the coefficients. The sequence can be thought of as the name of the attractor. We preface the name with a character that indicates the type of equation. Let's use the letter A to represent one-dimensional quadratic maps. Thus the logistic equation coded in this way is AMu%. Note that the letters in the name are case sensitive (u and U are different), so you should be careful when typing them. Such names may look strange, which is perhaps appropriate for strange attractors, and you shouldn't try to pronounce them! However, they do provide a convenient and compact method for saving everything you need to reproduce an attractor.

2.4 The Computer Search

Before embarking on a search for strange attractors, we need to generalize the formula given in Equation 1E for the Lyapunov exponent of the logistic equation. The generalization is easily obtained using differential calculus, and the result is

$$L = \sum \log_2 | a_2 + 2a_3X_n | \ / \ N \qquad \text{(Equation 2C)}$$

The program changes that are required to perform a search for strange attractors in one-dimensional quadratic iterated maps are given in the listing **PROG03**.

PROG03. Changes required in PROG02 to search for strange attractors in one-dimensional quadratic maps

```
1000 REM ONE-D MAP SEARCH

1020 DIM XS(499), A(504), V(99)

1050 NMAX = 11000               'Maximum number of iterations

1160 RANDOMIZE TIMER            'Reseed random-number generator


1360 CLS : LOCATE 13, 34: PRINT "Searching..."


1560 GOSUB 2600                 'Get coefficients

1580 P% = 0: LSUM = 0: N = 0: NL = 0

1590 XMIN = 1000000!: XMAX = -XMIN


1720 XNEW = A(1) + (A(2) + A(3) * X) * X

2020 N = N + 1


2110 IF N < 100 OR N > 1000 THEN GOTO 2200

2120    IF X < XMIN THEN XMIN = X

2130    IF X > XMAX THEN XMAX = X
```

```
2140    YMIN = XMIN: YMAX = XMAX

2200 IF N = 1000 THEN GOSUB 3100        'Resize the screen

2250 IF N < 1000 OR XS(I%) <= XL OR XS(I%) >= XH OR XNEW <= XL OR XNEW >= XH THEN
GOTO 2320



2410 IF ABS(XNEW) > 1000000! THEN T% = 2    'Unbounded

2430 GOSUB 2900                 'Calculate Lyapunov exponent

2460 IF N >= NMAX THEN T% = 2   'Strange attractor found

2470 IF ABS(XNEW - X) < .000001 THEN T% = 2 'Fixed point

2480 IF N > 100 AND L < .005 THEN T% = 2    'Limit cycle



2600 REM Get coefficients

2660 CODE$ = "A"

2680 M% = 3

2690    FOR I% = 1 TO M%        'Construct CODE$

2700        GOSUB 2800          'Shuffle random numbers

2710        CODE$ = CODE$ + CHR$(65 + INT(25 * RAN))

2720    NEXT I%

2730 FOR I% = 1 TO M%           'Convert CODE$ to coefficient values

2740    A(I%) = (ASC(MID$(CODE$, I% + 1, 1)) - 77) / 10

2750 NEXT I%

2760 RETURN



2800 REM Shuffle random numbers
```

```
2810 IF V(0) = 0 THEN FOR J% = 0 TO 99: V(J%) = RND: NEXT J%
2820 J% = INT(100 * RAN)
2830 RAN = V(J%)
2840 V(J%) = RND
2850 RETURN

2900 REM Calculate Lyapunov exponent
2910 DF = ABS(A(2) + 2 * A(3) * X)
3030 IF DF > 0 THEN LSUM = LSUM + LOG(DF): NL = NL + 1
3040 L = .721347 * LSUM / NL
3070 RETURN

3100 REM Resize the screen
3120 IF XMAX - XMIN < .000001 THEN XMIN = XMIN - .0000005: XMAX = XMAX + .0000005
3130 IF YMAX - YMIN < .000001 THEN YMIN = YMIN - .0000005: YMAX = YMAX + .0000005
3160 MX = .1 * (XMAX - XMIN): MY = .1 * (YMAX - YMIN)
3170 XL = XMIN - MX: XH = XMAX + MX: YL = YMIN - MY: YH = YMAX + MY
3180 WINDOW (XL, YL)-(XH, YH): CLS
3310 LINE (XL, YL)-(XH, YH), , B
3460 RETURN
```

Here are six points to note about **PROG03**:

1. The maximum number of iterations (*NMAX* in line 1050) has been set

arbitrarily to 11,000. This is the number of iterations after which a strange attractor is assumed to have been found if the magnitude of *X* never exceeded one million and the Lyapunov exponent is positive (actually greater than 0.005). You can decrease *NMAX* to speed the rate at which attractors are found, or you can increase *NMAX* if you have a very fast computer or want to give the displays more time to develop. The number of iterations is a parameter that you can adjust for the most visually appealing result. Most of the figures in this book were made with *NMAX* set at between about 500,000 and 10 million, and they required between about a minute and an hour to produce.

2. The seed for the random-number generator is taken in line 1160 as the number of seconds lapsed since midnight (TIMER). This choice ensures that a new sequence of random numbers is produced each time the program is run, except in the unlikely event that it is run at exactly the same time each day.

3. After 1000 iterations (line 2200), the screen is resized and erased by the subroutine in lines 3100 through 3460 using the minimum and maximum values of *X* between the 100th and 1000th iteration, allowing a 10% border around the attractor.

4. To save time, the difference between each value of *X* and its predecessor is evaluated in line 2470, and if the difference is less than one millionth, the solution is assumed to be a fixed point even if the Lyapunov exponent is still positive.

5. The Lyapunov exponent is not used as a criterion until after 100 iterations (line 2480) to ensure that its value is reasonably accurate.

6. The coefficients of the equation are chosen in line 2710 using random numbers that have been shuffled by the subroutine in lines 2800 through 2850 to minimize the chance of repeating the same search sequence.

The criterion for detecting a strange attractor is somewhat subjective. There will always be borderline cases for which no amount of computing will suffice to distinguish between a strange attractor and a periodic solution with a very long period. However, our interest here is in finding visually interesting attractors quickly, and so we can afford to make occasional mistakes. Such mistakes account for only a small fraction of cases.

Of the 15,625 combinations of coefficients, exactly 364 (2.3%) are chaotic by these criteria. Some of the more visually interesting ones are shown in Figures 2-1 through 2-4, in which the values are plotted versus their fifth previous iterate. For each case, the code and the Lyapunov exponent are shown at the top of the graph.

Figure 2-1. One-dimensional quadratic map

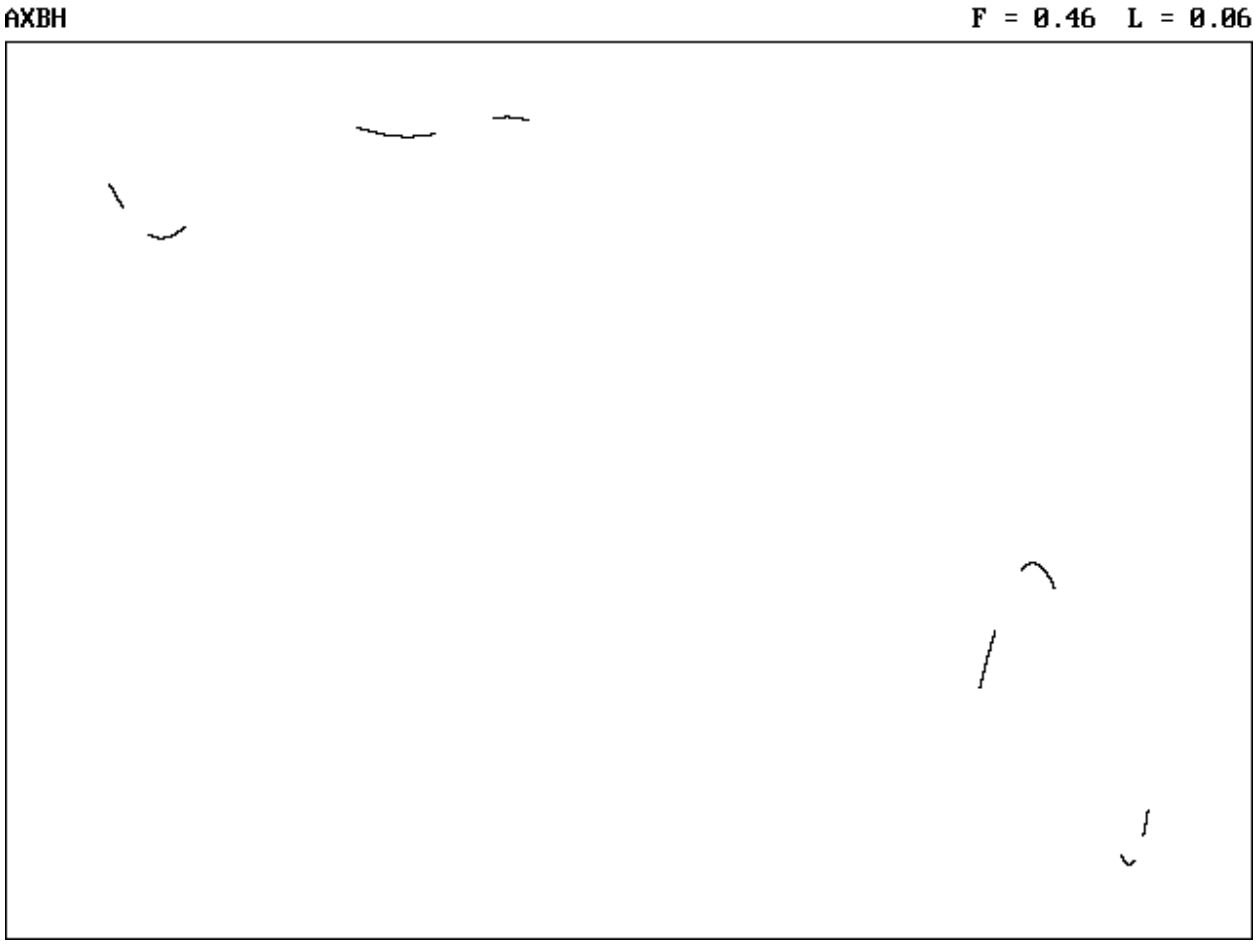


Figure 2-2. One-dimensional quadratic map

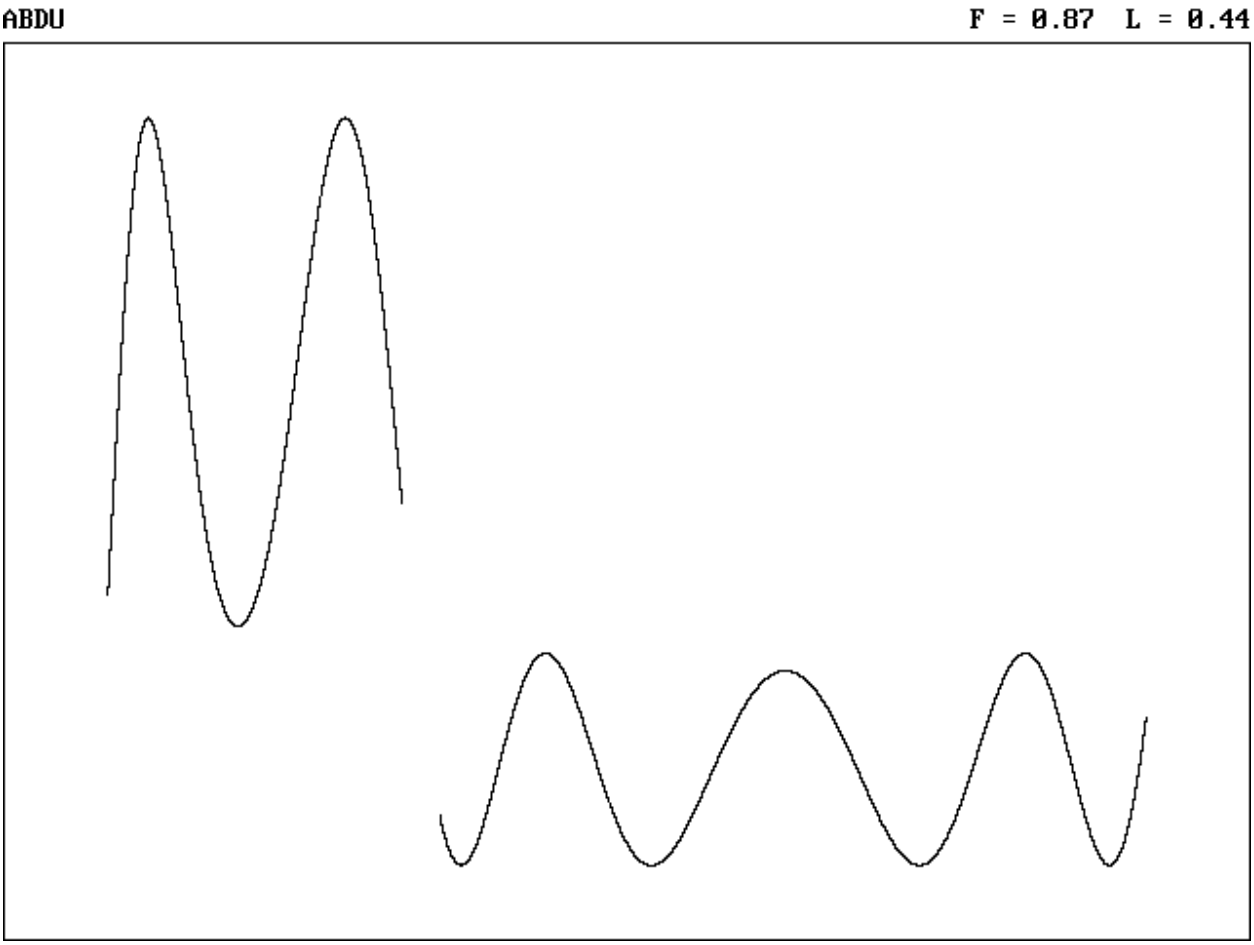


Figure 2-3. One-dimensional quadratic map

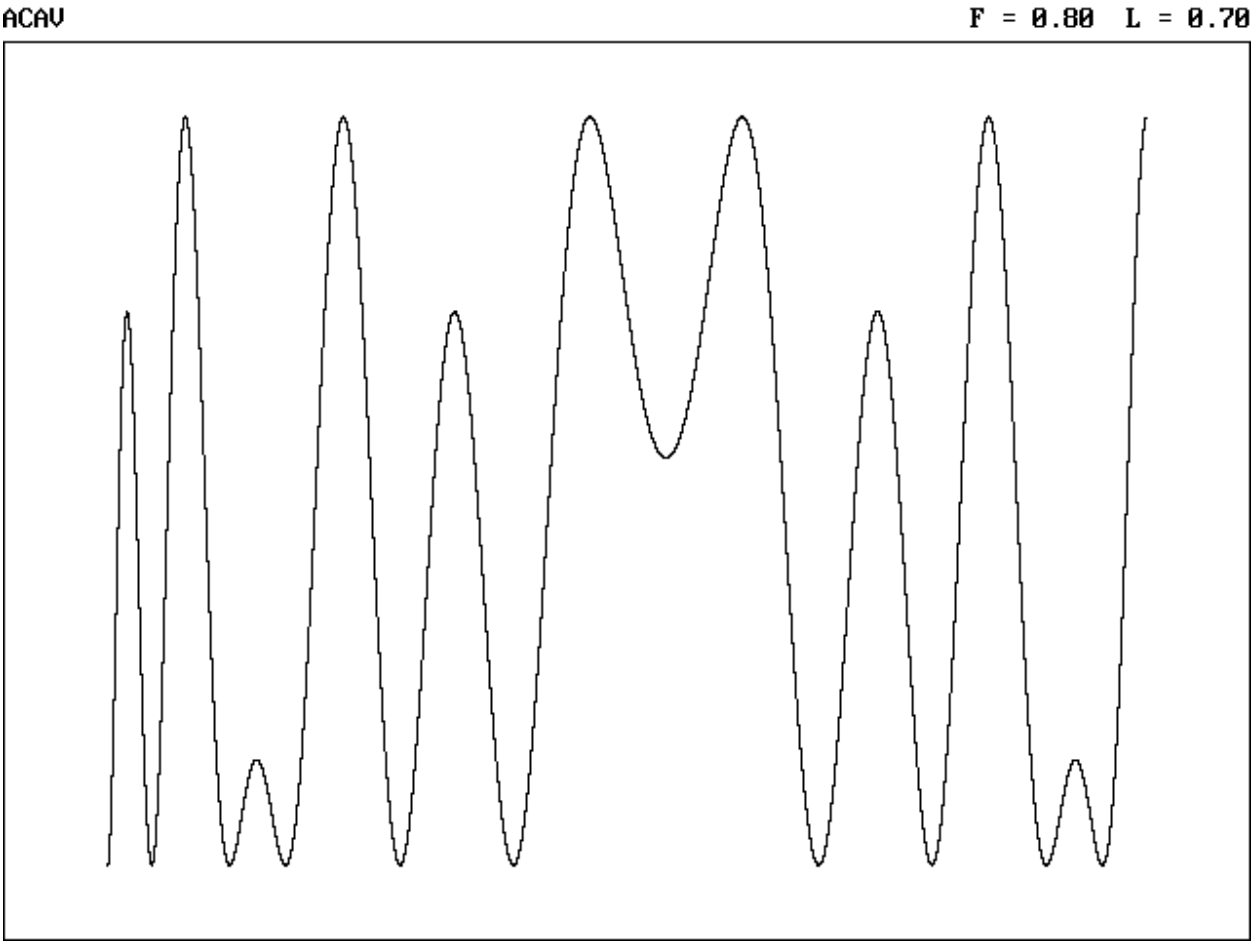
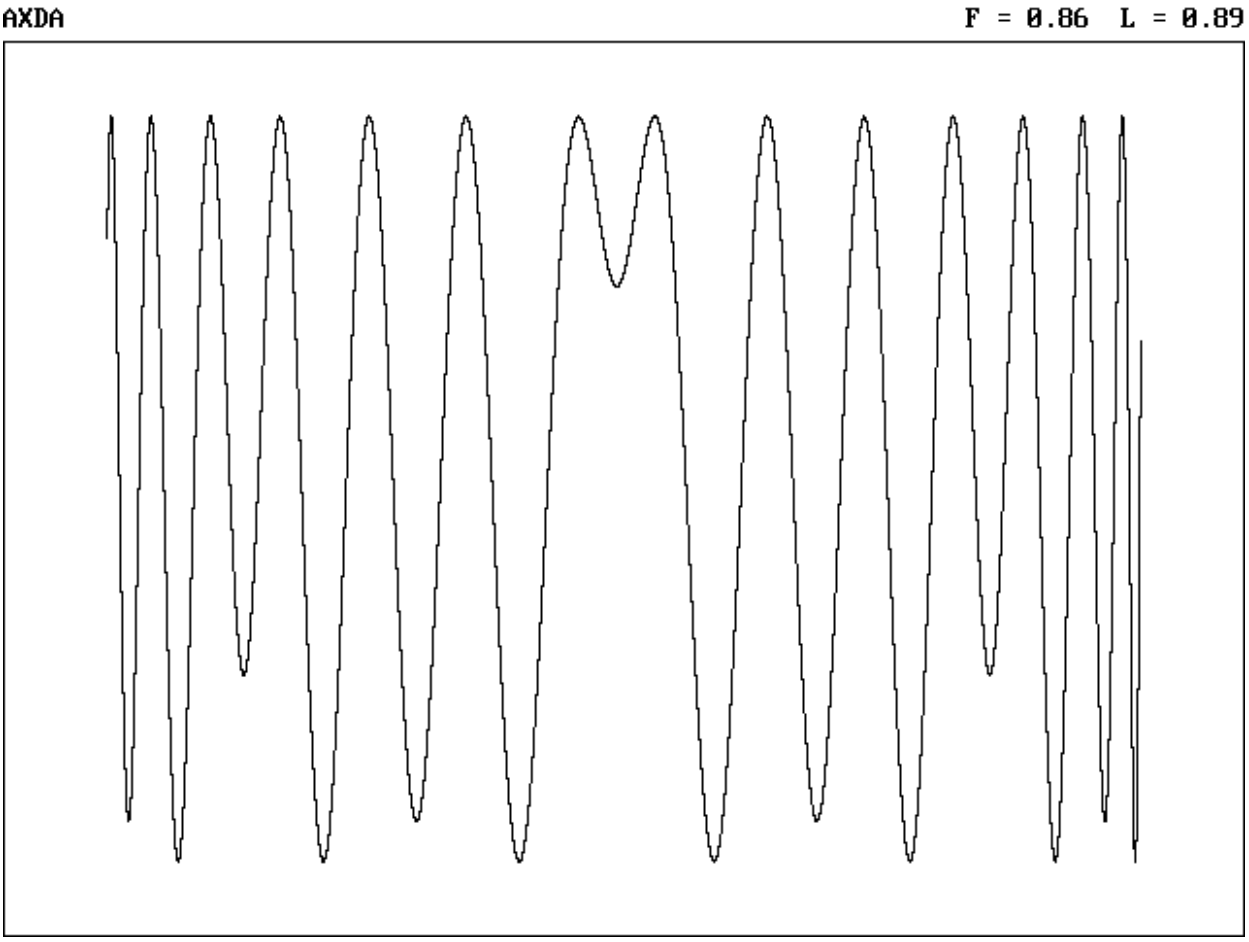


Figure 2-4. One-dimensional quadratic map



The search for strange attractors is potentially time-consuming if you have an old computer without a math coprocessor or if you are using a BASIC interpreter rather than a compiler. Even if the search is reasonably fast on your computer, be forewarned that it will slow down considerably as you advance to the more complicated equations later in the book. Perhaps this is a good time to summarize some of your options for making the program run faster.

When comparing calculation speeds of various computers and compilers, you must do the comparison with the actual program or a benchmark that accurately reflects its mix of instructions, graphics, and disk access. With computer speeds doubling approximately every two years, speed will eventually cease to be a consideration for the calculations described in this book. Meanwhile, you need

to consider the alternatives.

Table 2-2 lists the average number of strange attractors found by **PROG03** per hour using various versions of BASIC on a 33-MHz 80486DX-based computer with and without a math coprocessor. The exact numbers are less important than the relative values. They provide a good indication of how the various versions of BASIC compare on calculations of the type that are used throughout this book.

Table 2-2. Strange attractors found per hour by PROG03 with various versions of BASIC

Publisher	Program	Ver	Type	Attractors/hour	
				No copro	Coproc
Microsoft	GW-BASIC	3.2	Interpreter	92	92
Microsoft	QBASIC	1.0	Interpreter	73	73
Microsoft	QuickBASIC	4.5	Interpreter	78	396
Microsoft	QuickBASIC	4.5	Compiler	98	390
Microsoft	VB for DOS	1.0	Interpreter	72	393
Microsoft	VB for DOS	1.0	Comp (alternate)	315	316
Microsoft	VB for DOS	1.0	Comp (emulate)	139	418
Borland	Turbo BASIC	1.1	Compiler	96	400
Spectra	PowerBASIC	3.0	Comp (procedure)	246	1419
Spectra	PowerBASIC	3.0	Comp (emulate)	123	1683

QuickBASIC and VisualBASIC for MS-DOS can be run from the editor environment, where they function much like an interpreter, or they can be used to compile a stand-alone executable program. VisualBASIC can be compiled with either of two floating point math packages; the *alternate* package is faster for machines without a coprocessor, and the *emulate* package is faster for machines with a coprocessor. Turbo BASIC is now obsolete and has been replaced by PowerBASIC.

PowerBASIC, like VisualBASIC, can be compiled with either of two floating point math packages; the *procedure* package is similar to the VisualBASIC alternate package. A third math package, *NPX (87)* is the same as emulate, except it cannot work on a machine without a math coprocessor. The tests were done with all error trapping turned off, which is inadvisable until you have a thoroughly debugged program.

If you launch the program from Microsoft Windows, you might find the computation speeds considerably different from those in Table 2.2. In one test, the PowerBASIC speeds were cut in half, and the QuickBASIC speeds were increased slightly from the values obtained when the program was run directly from DOS. You should do your own speed tests to see what configuration provides the optimum performance on your computer and operating system.

The executable program on the disk that accompanies this book was compiled with PowerBASIC using the procedure package. If you have PowerBASIC and a math coprocessor, you can recompile the program using the emulate or NPX (87) package to achieve a slight improvement in speed.

2.5 Wiggles on Wiggles

The preceding figures consist of segments of wiggly lines, so they are not very artistic. To make things more interesting, we can consider one-dimensional maps of higher order. By this we mean that we will not stop with quadratic (X_2) maps, but we will consider equations containing *cubic* (X_3), *quartic* (X_4), *quintic* (X_5), and even higher terms.

In one sense, considering higher-order terms is equivalent to plotting each iterate versus an iterate earlier than the immediately previous one. For example, two successive iterations of the second-order Equation 2A yields

$$X_{n+2} = a_1(1+a_2+a_1a_3) + (a_3a_2+2a_1a_3)X_n$$
$$+ a_3(a_2+2a_1a_3+a_2^2)X_n^2 + 2a_2a_3^2X_n^3 + a_3^3X_n^4 \qquad \text{(Equation 2D)}$$

which is a fourth-order *polynomial*. However, there are only three parameters—a_1, a_2, and a_3—from which the five coefficients are uniquely determined.

A simpler and more general procedure is to allow each term in the polynomial to have its own coefficient, which for fifth order gives

$$X_{n+1} = a_1 + a_2X_n + a_3X_n^2 + a_4X_n^3 + a_5X_n^4 + a_6X_n^5 \qquad \text{(Equation 2E)}$$

With six coefficients, each with 25 possible values, there are 25^6 or about 244 million different combinations. Even if only a small percentage of them is chaotic, we would have to look at one every second for about a year before we would see them all.

The generalization of the expression for the Lyapunov exponent for a fifth-order map is given by

$$L = \sum \log_2 | a_2 + 2a_3X_n + 3a_4X_n^2 + 4a_5X_n^3 + 5a_6X_n^4 | \ / \ N \qquad \text{(Equation 2C)}$$

With these equations in hand, we can easily modify the program in **PROG04** to search for one-dimensional attractors of up to fifth order. In our coding scheme, a first letter of B represents third order, C represents fourth order, and D represents fifth order. The program is written so that even higher orders can be produced by changing the quantity *OMAX%* in line 1060.

PROG04. Changes required in PROG03 to search for strange attractors in one-dimensional maps of order up to *OMAX%*

```
1000 REM ONE-D MAP SEARCH (Polynomials up to 5th Order)

1060 OMAX% = 5                  'Maximum order of polynomial



1720 XNEW = A(O% + 1)

1730 FOR I% = O% TO 1 STEP -1

1830    XNEW = A(I%) + XNEW * X

1930 NEXT I%



2650 O% = 2 + INT((OMAX% - 1) * RND)

2660 CODE$ = CHR$(63 + O%)

2680 M% = O% + 1
```

```
2910 DF = 0

2930 FOR I% = O% TO 1 STEP -1

2940    DF = I% * A(I% + 1) + DF * X

2970 NEXT I%

3000 DF = ABS(DF)
```

PROG04 produces an interesting array of shapes, samples of which are shown in Figures 2-5 through 2-10. The objects are still segments of lines, but the wiggles themselves have wiggles, and the underlying determinism is less obvious than before.

Figure 2-5. One-dimensional cubic map

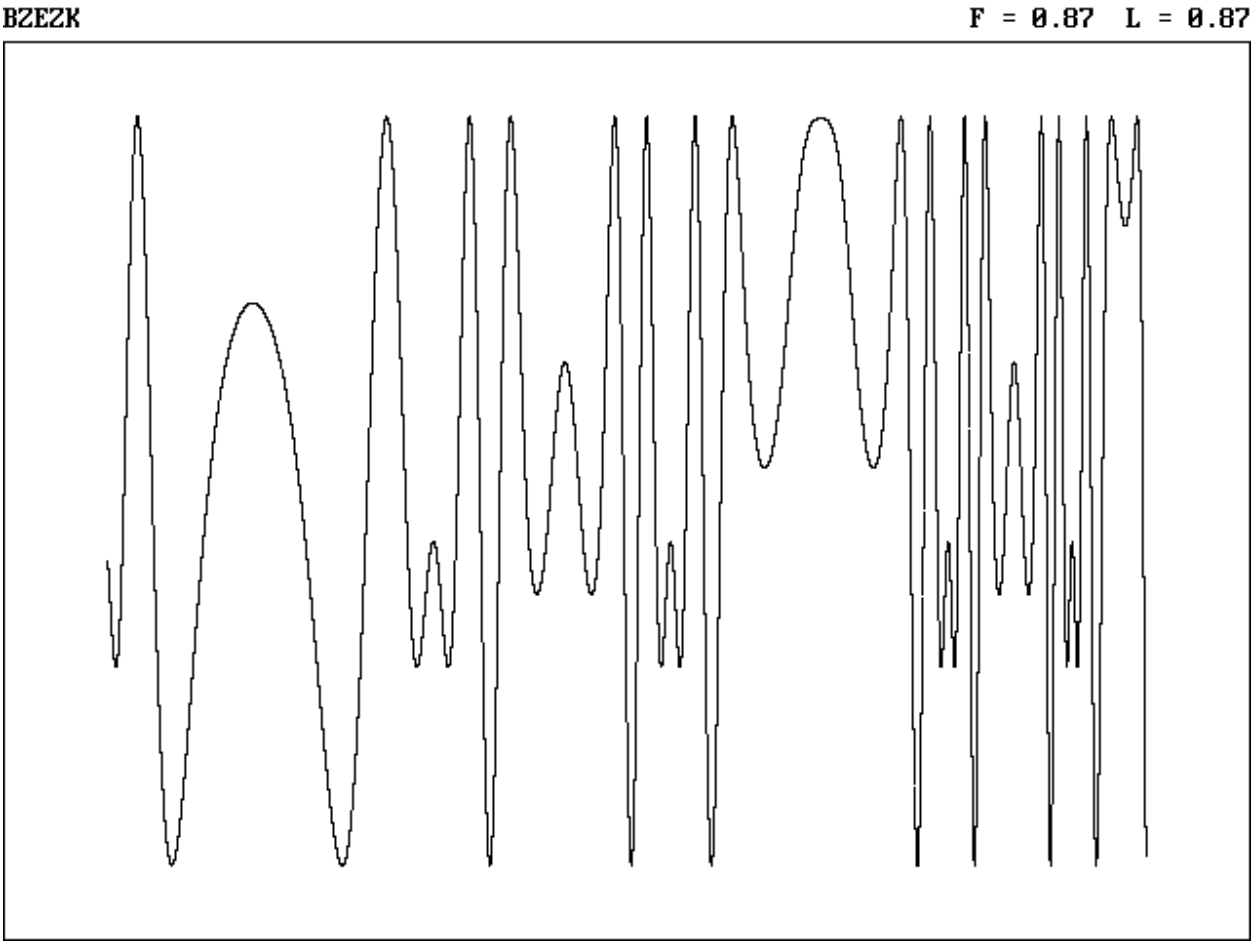


Figure 2-6. One-dimensional quartic map

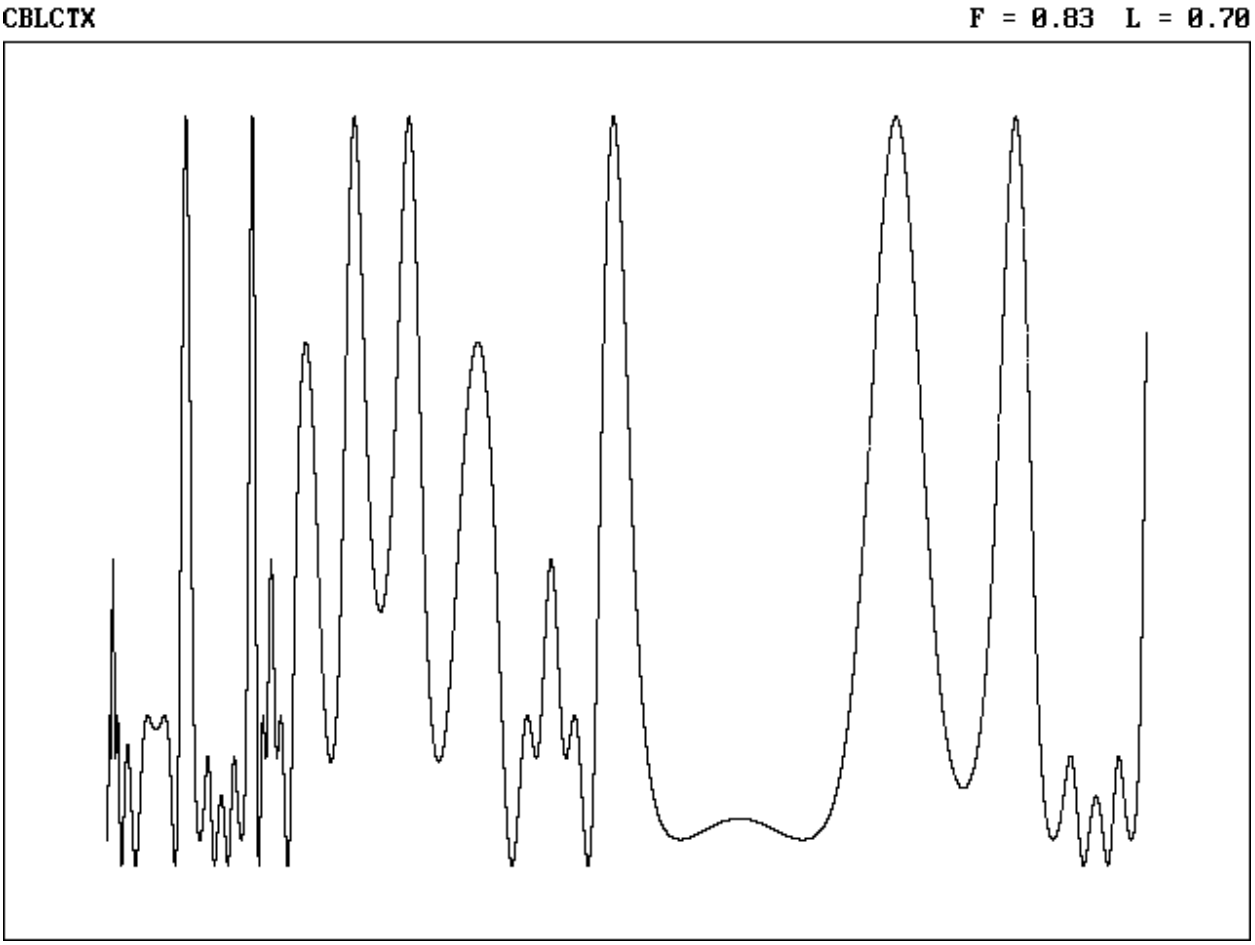


Figure 2-7. One-dimensional quartic map

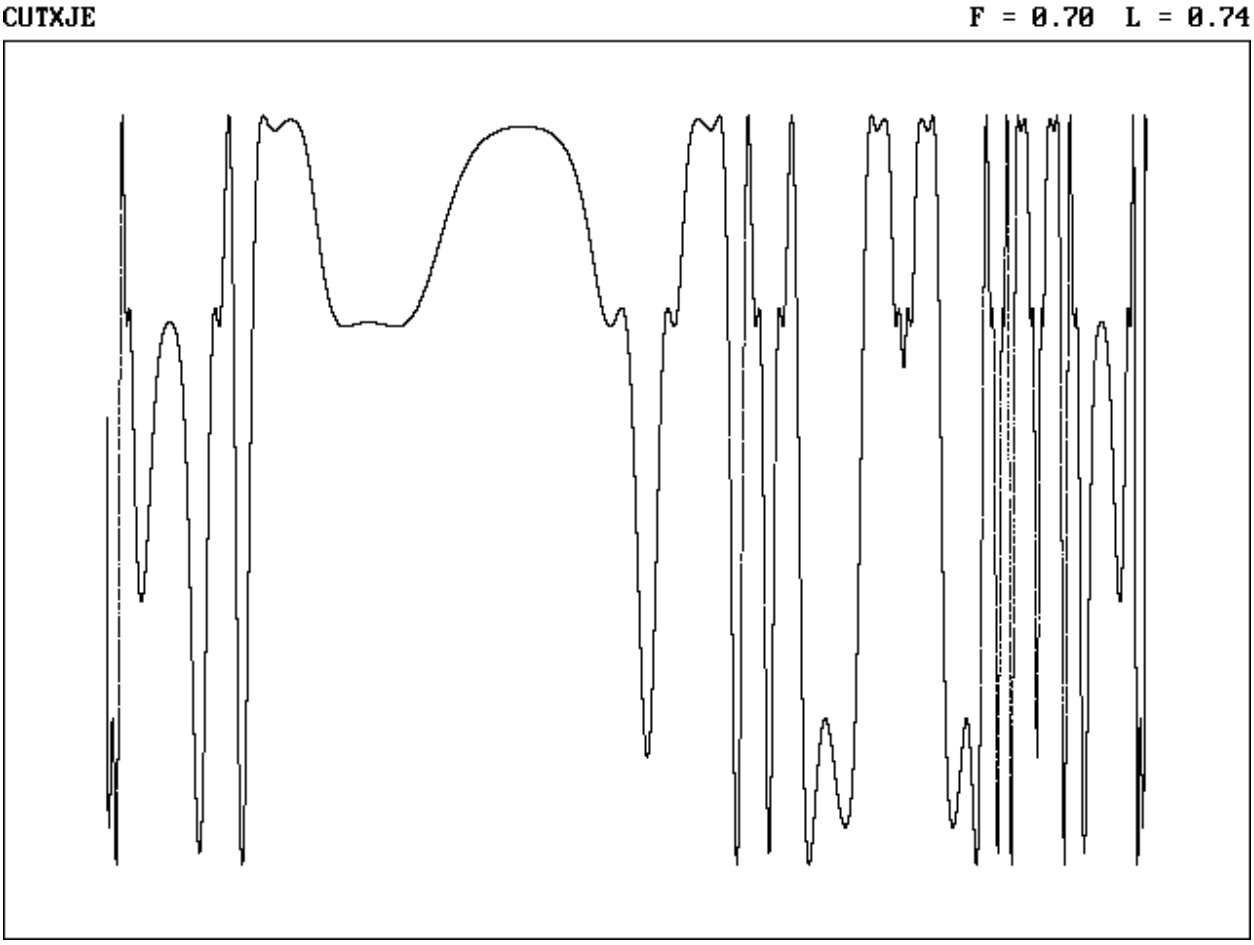


Figure 2-8. One-dimensional quintic map

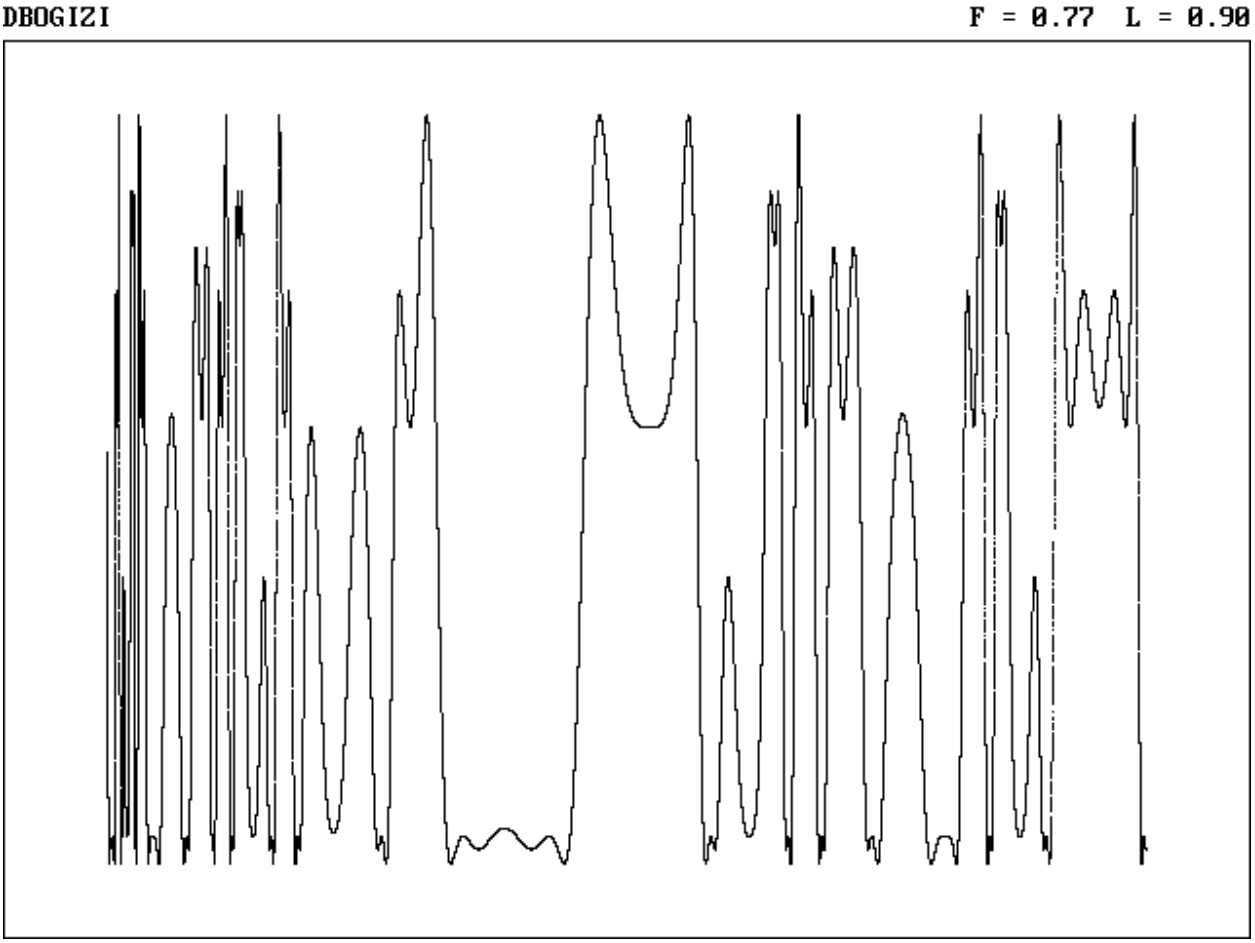


Figure 2-9. One-dimensional quintic map

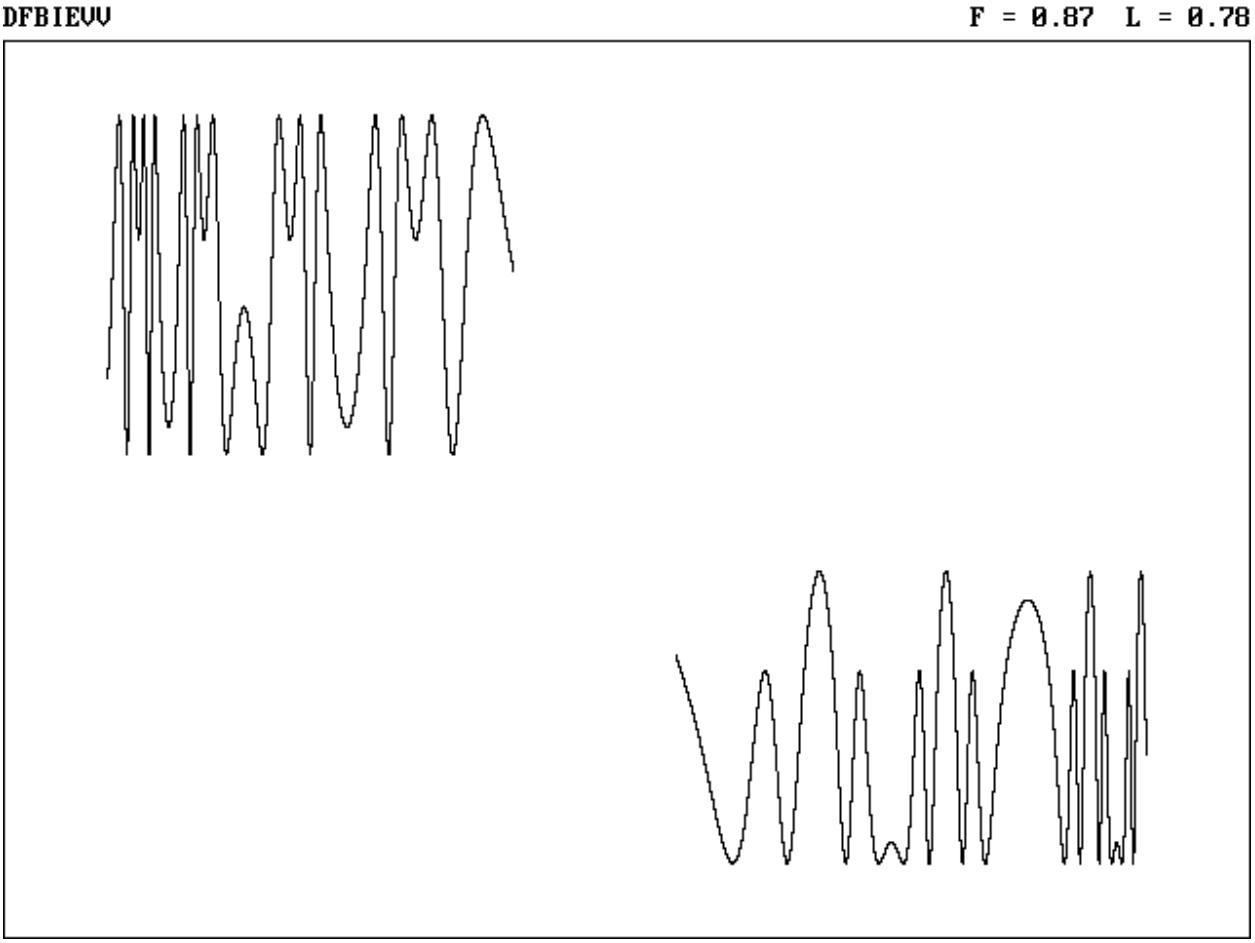
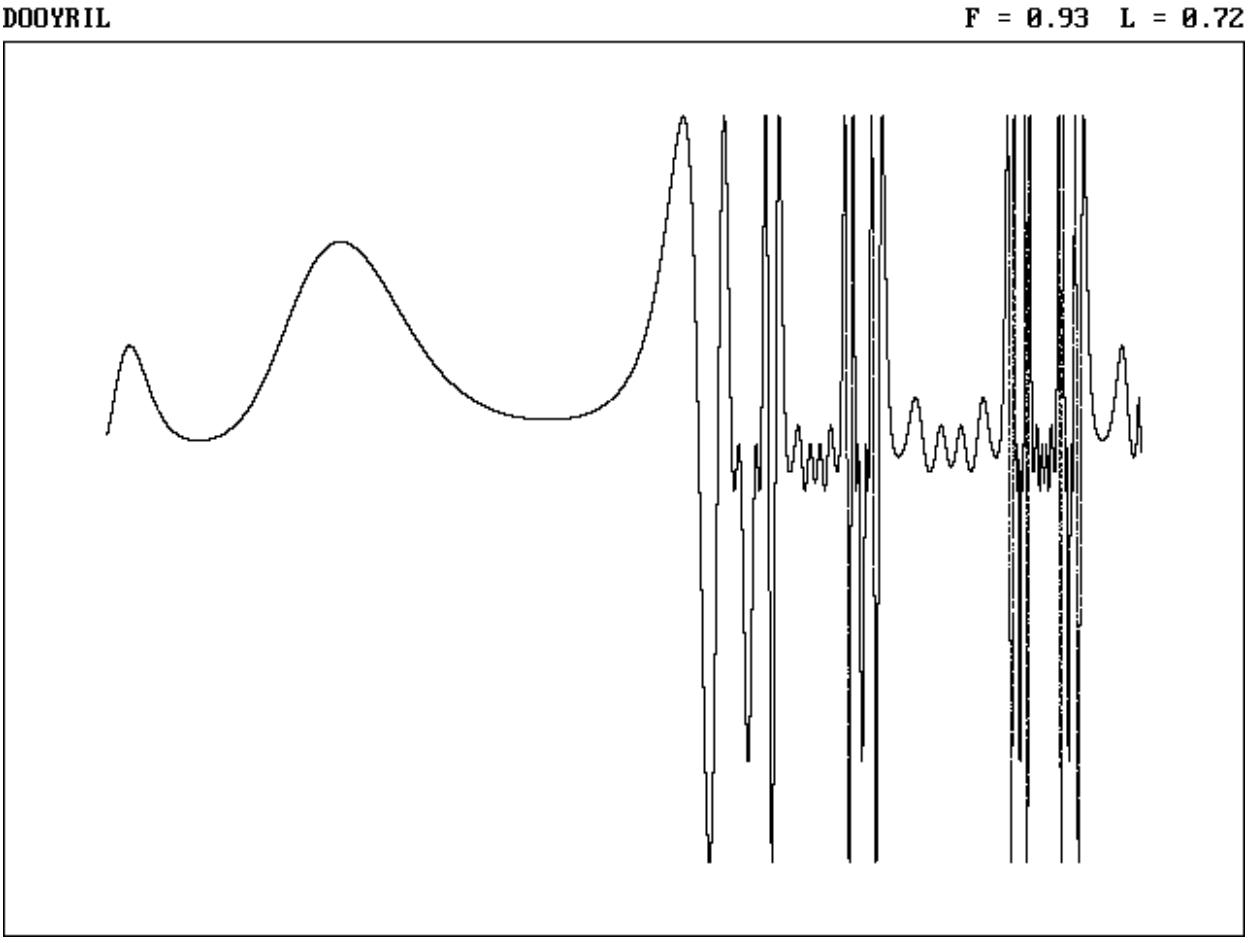


Figure 2-10. One-dimensional quintic map



2.6 Making Music

If the preceding figures don't qualify as art, perhaps they qualify as music. Since the quantity *X* behaves in a deterministic yet unpredictable way, it may be that a sequence of musical notes determined by *X* will mimic the order and unpredictability that characterize music. It's easy to test.

Suppose we allow the notes to span three octaves from A-220 to A-1760. The letter refers to the musical note, and the numbers refer to the frequency in cycles per second (called *Hertz*). We'll allow the notes to take one of twelve distinct values corresponding to the even-tempered scale, and for simplicity we'll assume all the notes to be of the same duration. Thus the range of possible values of *X* is divided into 36 intervals, and each successive iterate of *X* is converted into the corresponding musical note. **PROG05** shows the changes necessary to accomplish this.

PROG05. Changes required in PROG04 to produce chaotic music

```
1000 REM ONE-D MAP SEARCH (With Sound)

1100 SND% = 1                    'Turn sound on


2310    IF SND% = 1 THEN GOSUB 3500     'Produce sound


2490 Q$ = INKEY$: IF LEN(Q$) THEN GOSUB 3600        'Respond to user command


3500 REM Produce sound

3510 FREQ% = 220 * 2 ^ (CINT(36 * (XNEW - XL) / (XH - XL)) / 12)

3520 DUR = 1

3540 SOUND FREQ%, DUR: IF PLAY(0) THEN PLAY "MF"

3550 RETURN
```

```
3600 REM Respond to user command

3610 T% = 0

3630 IF ASC(Q$) > 96 THEN Q$ = CHR$(ASC(Q$) - 32)

3770 IF Q$ = "S" THEN SND% = (SND% + 1) MOD 2: T% = 3

3800 RETURN
```

The program allows you to toggle the sound on and off by pressing the **S** key. Pressing any other key exits the program. You might wish to experiment with the duration *DUR* of the *SOUND* statement in line 3520. Increasing its value from 1 (corresponding to approximately 0.055 seconds) makes the sounds more musical, but then the calculation takes longer.

The use of sound to help interpret data generated by a computer is a technique that is relatively unexplored. The method is sometimes called *sonification*. In some cases, patterns and structure in data can be more readily discerned audibly than visually. This technique was used to advantage in interpreting data from the Voyager spacecraft as it detected plasma waves near Jupiter and micrometeorites as it crossed through the rings of Saturn. The repetitive sound of a simple limit cycle contrasts sharply with the nonrepetitive waverings of a chaotic time series.

Chapter 3

Pieces of Planes

Whereas the last chapter discussed one-dimensional maps whose graphs are segments of lines, this chapter deals with two-dimensional maps whose graphs are pieces of planes and which thus produce much more interesting displays. This chapter provides the minimum tools for creating attractors that genuinely qualify as art. Armed with only the information contained here, you have such a great variety of available patterns that you hardly need to proceed beyond this chapter. But if you do stop here, you miss some delightful surprises.

3.1 Quadratic Maps in Two Dimensions

In the discussion so far, the maps have involved a single variable *X* whose value changes with each iteration of the equation. Such maps are said to be one-dimensional because the values of *X* can be thought of as lying along a line, and a line is a one-dimensional object. By plotting each value of *X* versus a previous value of *X*, the line can be made to wiggle with considerable complexity; but it always remains a line, and lines are of limited interest and beauty.

The situation is more interesting when you consider iterated maps that involve two variables, *X* and *Y*. In such a case, each iterate produces a point in a plane, where *X*, by convention, represents the horizontal coordinate of the point, and *Y* represents the vertical coordinate. With successive iteration, the points fill in some portion of the plane. The visually interesting cases, as usual, are the chaotic ones.

Such two-dimensional maps might arise, for example, from an ecological model only slightly more complicated than the logistic equation. A classic example is the *predator-prey problem* in which *X* represents the prey and *Y* the predator. In a simple linear model, the solution is a fixed point (a unique number of both predators and prey) or a limit cycle (both the number of predators and the number of prey oscillate, reaching their maximum values at different times, but eventually repeating). When nonlinear terms are introduced into the model, the population of each species can behave chaotically. You can think of each point that makes up such an attractor as the population of predators and prey in successive years. Since such complexity arises from these very simple models, it's easy to understand why ecologists might have trouble predicting the fate of biological species!

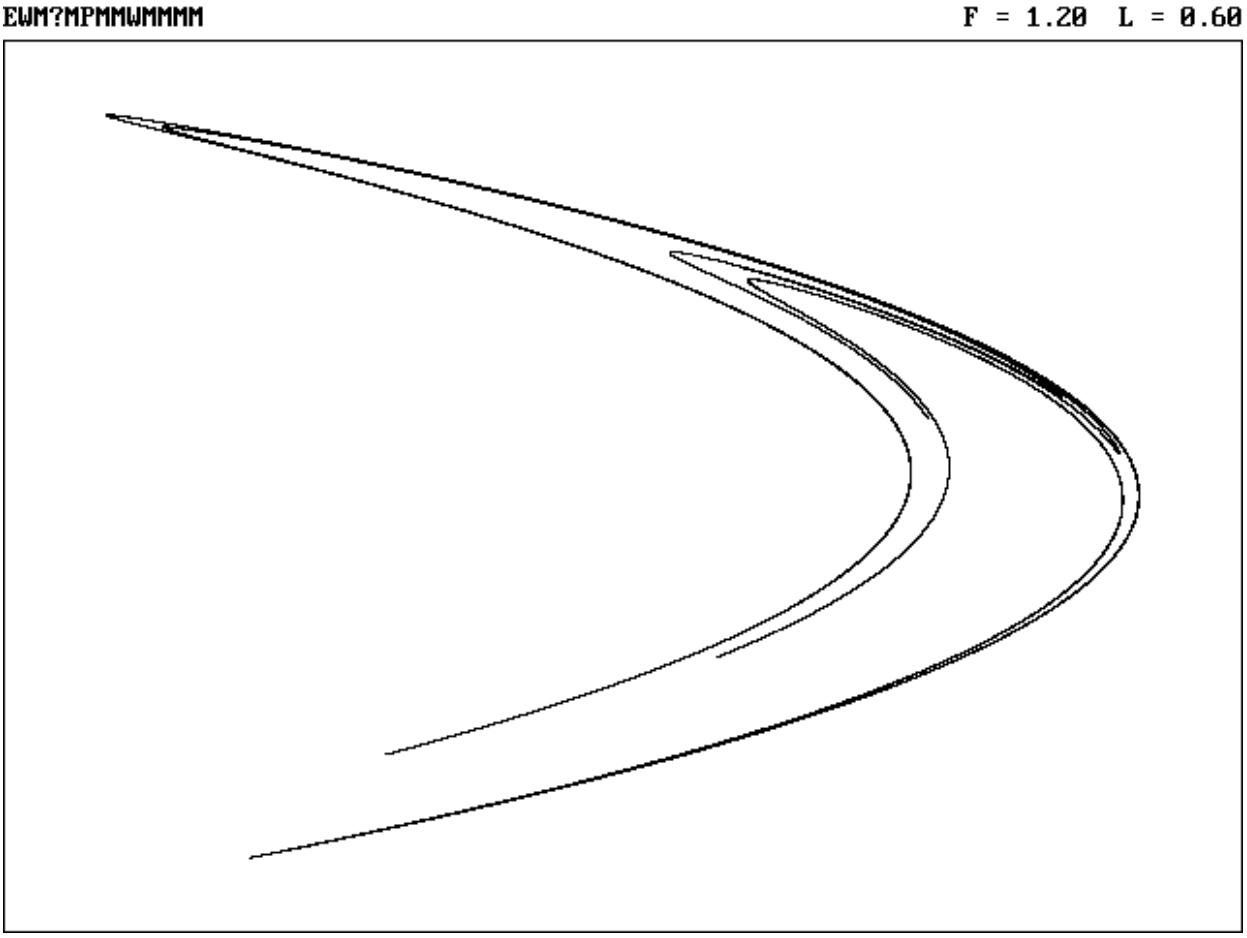
Perhaps the best known chaotic two-dimensional map is the *Hénon* map (proposed by the French astronomer Michel Hénon in 1976), whose equations are

$$X_{n+1} = 1 + aX_n^2 + bY_n$$

$$Y_{n+1} = X_n \qquad \text{(Equation 3A)}$$

The quantities *a* and *b* are the control parameters, analogous to *R* in the logistic equation. Hénon used the values a = -1.4 and b = 0.3. The necessary nonlinearity is provided by the X^2 term in the first equation. The Hénon map is special because the net contraction of a set of initial points covering an area of the *XY* plane is constant with each iteration. The area occupied by the points is 30% of the area at the previous iteration (from the bY_n term). Other values of b can be used, but not all values produce chaotic solutions. Unlike the logistic map, the Hénon map is invertable; there is a unique value for X_n and Y_n corresponding to each X_{n+1} and Y_{n+1}. You may have seen an alternate form of the Hénon equations in which the factor *b* appears instead in the second equation and the sign preceding the X^2 term is negative. The result of repeated iteration of Equation 3A is shown in Figure 3-1.

Figure 3-1. The Hénon map



The resulting graph is more than a line but less than a surface. What resembles a single line is a pair of lines, each of which is, in turn, another pair of lines, and so forth to however close you look or whatever magnification you choose. This *self-similarity* is a common characteristic of a class of objects that are called *fractals*.

Fractals are to chaos what geometry is to algebra—the visual expression of the mathematical idea. Approaching an understanding of chaos through such visual means is appealing to those with an aversion to conventional mathematics. The *Euclidean geometry* we learned in high school originated with the ancient Greeks and was developed more fully by the French mathematician Descartes and others in the 1600s. It deals with simple shapes such as lines, circles, and spheres. Euclidean geometry is now being augmented by *fractal geometry*, whose father and champion is the contemporary mathematician, Benoit Mandelbrot. Fractals appeared in art, such as in the drawings of the Dutch artist Maurits C. Escher, before

they were widely appreciated by mathematicians and scientists.

Some fractals are exactly self-similar, which means that they look the same no matter how much you magnify them. Others, such as most of the ones in this book, only have regions that are self-similar. There is no part of the Hénon map where you can zoom in and find a miniature replica of the entire map. Other fractals are only *statistically self-similar*, which means that a magnified portion of the object has the same amount of detail as the whole, but it is not an exact replica of it. Nearly all strange attractors are fractals, but not all fractals arise from strange attractors.

The Hénon map produces an object with a *fractal dimension* that is a fraction intermediate between one and two. The fractal dimension is a useful quantity for characterizing strange attractors. Isolated points have dimension zero, line segments have dimension one, surfaces have dimension two, and solids have dimension three. Strange attractors generally have noninteger dimensions.

Some authors make a distinction between strange attractors, which have non-interger dimension, and chaotic attractors, which exhibit sensitivity to initial conditions.

Since the Hénon map has X^2 as its highest-order term, it is a *quadratic map*. The most general two-dimensional iterated quadratic map is

$$X_{n+1} = a_1 + a_2X_n + a_3X_n^2 + a_4X_nY_n + a_5Y_n + a_6Y_n^2$$

$$Y_{n+1} = a_7 + a_8X_n + a_9X_n^2 + a_{10}X_nY_n + a_{11}Y_n + a_{12}Y_n^2 \qquad \text{(Equation 3B)}$$

The two equations in Equation 3B have 12 coefficients. For the Hénon map, $a_1 = 1$, $a_3 = -1.4$, $a_5 = 0.3$, $a_8 = 1$, and the other coefficients are zero. If we use the initial letter E to represent two-dimensional quadratic maps, the code for the Hénon map according to Table 2-1 is EWM?MPM^{2}WM^4, where we have introduced the shorthand M^2 for MM and M^4 for MMMM.

Values of *a* in the range of -1.2 to 1.2 are sufficient to produce an enormous variety of strange attractors. With increments of 0.1, there are 25^{12} or about 6×10^{16} different cases, of which approximately 1.6% or about 10^{15} are chaotic. Viewing them all at a rate of one per second would require over 30 million years! Stated differently, if each one were printed on an 81/2-by-11-inch sheet of paper, the

collection would cover nearly the entire land mass of Earth.

Note that not all the cases are strictly distinct. For example, if you replace *X* with *Y* and *Y* with *-X* in Equation 3B, you produce an attractor rotated 90 degrees counterclockwise from the original. When you do this, be sure to change X_{n+1} and Y_{n+1} as well as X_n and Y_n. Thus the code EM^{4}CMWJM^3? produces a rotated version of the Hénon map. In the same fashion, you can rotate an attractor through 180 degrees by replacing *X* with *-X* and *Y* with *-Y* and through 270 degrees by replacing *X* with *-Y* and *Y* with *X*. Perhaps it's easier just to rotate your computer monitor!

Besides rotations, there are cases that correspond to reflections. When viewed in a mirror, the attractors have left and right reversed, but up and down remain the same. A transformation in which *X* is replaced with *-X* accomplishes this. Thus the code for a reflected Hénon map is ECM[MJM^{2}CM^4. In addition, the reflections can be rotated. Thus there are at least eight so-called *degenerate states* for each attractor, corresponding to rotations and reflections. Such symmetries and degeneracies play an important role in science; they often reduce the amount of work we have to do and provide relations between phenomena that initially appear different.

Additional degenerate cases correspond to scale changes. For example, if you replace *X* by *mX* and *Y* by *nY* with $m = n$, the attractor remains the same except it is reduced in size by a factor of *m*. Some of the coefficients are likely to be outside the allowed range, however. The Hénon map with $m = n = 2$ can be generated with the code ERM1MPM^{2}WM^4. With m not equal to *n*, the horizontal and vertical dimensions are scaled differently, but since the computer rescales the attractor to fit the screen, the visual result is the same.

These degeneracies show that there are many ways to code a particular attractor. Although this is true, there are so many different possible combinations of coefficients that it is very unlikely that two degenerate cases will be found spontaneously. Thus the examples displayed in this chapter represent but a tiny fraction of the possibilities, and you will be generating many other cases, almost none of which have been seen before.

3.2 The Butterfly Effect Revisited

Two-dimensional chaotic iterated maps also exhibit sensitivity to initial conditions, but the situation is more complicated than for one-dimensional maps. Imagine a collection of initial conditions filling a small circular region of the *XY* plane.

After one iteration, the points have moved to a new position in the plane, but they now occupy an elongated region called an *ellipse*. The circle has contracted in one direction and expanded in the other. With each iteration, the ellipse gets longer and narrower, eventually stretching out into a long filament. The orientation of the filament also changes with each iteration, and it wraps up like a ball of taffy.

Thus two-dimensional chaotic maps have not a single Lyapunov exponent but two—a positive one corresponding to the direction of expansion, and a negative one corresponding to the direction of contraction. The signature of chaos is that at least one of the Lyapunov exponents is positive. Furthermore, the magnitude of the negative exponent has to be greater than the positive one so that initial conditions scattered throughout the basin of attraction contract onto an attractor that occupies a negligible portion of the plane. The area of the ellipse continually decreases even as it stretches to an infinite length.

There is a proper way to calculate both of the Lyapunov exponents. For the mathematically inclined, the procedure involves summing the logarithms of the eigenvalues of the Jacobian matrix of the linearized transformation with occasional Gram-Schmidt reorthonormalization. This method is slightly complicated, so we will instead devise a simpler procedure sufficient for determining the largest Lyapunov exponent, which is all we need in order to test for chaos.

Suppose we take two arbitrary but nearby initial conditions. The first few iterations of the map may cause the points to get closer together or farther apart, depending on the initial orientation of the two points. Eventually, the points will come arbitrarily close in the direction of the contraction, but they will continue to separate in the direction of the expansion. Thus if we wait long enough, the rate of separation will be governed only by the largest Lyapunov exponent. Fortunately, this usually takes just a few iterations.

However, because the separation grows exponentially for a chaotic system, the points quickly become too far apart for an accurate estimate of the exponent. This problem can be remedied if, after each iteration, the points are moved back to their original separation along the direction of the new separation. The Lyapunov exponent is then determined by the average of the distance they must be moved for each iteration to maintain a constant small separation. If the two solutions are separated by a distance d_n after the nth iteration, and the separation after the next iteration is d_{n+1}, the Lyapunov exponent is determined from

$$L = \sum \log_2 (d_{n+1} / d_n) / N \qquad \text{(Equation 3C)}$$

where the sum is taken over all iterations from $n = 0$ to $n = N-1$. After each iteration,

the value of one of the iterates is changed to make $d_{n+1} = d_n$. For the cases here, dn equals 10^{-6}. This procedure also allows us to deal with maps of three and even higher dimensions in which there are more than two Lyapunov exponents.

3.3 Searching the Plane

We now have all the tools in hand to conduct a computer search for attractors in two dimensions. The procedure is the same as for one-dimensional maps, except the Lyapunov exponent calculation is done differently and the *X* and *Y* variables are plotted as a point in the plane after each iteration. **PROG06** shows the changes needed to accomplish such a search.

PROG06. Changes required in PROG05 to search for two-dimensional quadratic strange attractors

```
1000 REM TWO-D MAP SEARCH

1060 OMAX% = 2                  'Maximum order of polynomial

1070 D% = 2                     'Dimension of system

1100 SND% = 0                   'Turn sound off


1520 Y = .05

1550 XE = X + .000001: YE = Y

1590 XMIN = 1000000!: XMAX = -XMIN: YMIN = XMIN: YMAX = XMAX


1720 XNEW = A(1) + X * (A(2) + A(3) * X + A(4) * Y)

1730 XNEW = XNEW + Y * (A(5) + A(6) * Y)

1830 YNEW = A(7) + X * (A(8) + A(9) * X + A(10) * Y)

1930 YNEW = YNEW + Y * (A(11) + A(12) * Y)
```

```
2140    IF Y < YMIN THEN YMIN = Y
2150    IF Y > YMAX THEN YMAX = Y
2240 IF D% = 1 THEN XP = XS(I%): YP = XNEW ELSE XP = X: YP = Y
2250 IF N < 1000 OR XP <= XL OR XP >= XH OR YP <= YL OR YP >= YH THEN GOTO 2320
2300    PSET (XP, YP)                   'Plot point on screen

2410 IF ABS(XNEW) + ABS(YNEW) > 1000000! THEN T% = 2    'Unbounded
2470 IF ABS(XNEW - X) + ABS(YNEW - Y) < .000001 THEN T% = 2
2520 Y = YNEW

2660 CODE$ = CHR$(59 + 4 * D% + O%)
2680 M% = 1: FOR I% = 1 TO D%: M% = M% * (O% + I%): NEXT I%

2910 XSAVE = XNEW: YSAVE = YNEW: X = XE: Y = YE: N = N - 1
2930 GOSUB 1700                 'Reiterate equations
2940 DLX = XNEW - XSAVE: DLY = YNEW - YSAVE
2960 DL2 = DLX * DLX + DLY * DLY
2970 IF CSNG(DL2) <= 0 THEN GOTO 3070       'Don't divide by zero
2980    DF = 1000000000000# * DL2
2990    RS = 1 / SQR(DF)
3000    XE = XSAVE + RS * (XNEW - XSAVE): YE = YSAVE + RS * (YNEW -     YSAVE)
3020    XNEW = XSAVE: YNEW = YSAVE
```

```
3030    IF DF > 0 THEN LSUM = LSUM + LOG(DF): NL = NL + 1

3040    L = .721347 * LSUM / NL


3110 IF D% = 1 THEN YMIN = XMIN: YMAX = XMAX
```

This program produces an incredible variety of interesting patterns, a small selection of which is shown in Figures 3-2 through 3-17. Admire the beauty and variety of these figures, and then make some of you own by running the program. If your computer has a printer, use the **Print Screen** key to print any that you find especially appealing.

Figure 3-2. Two-dimensional quadratic map

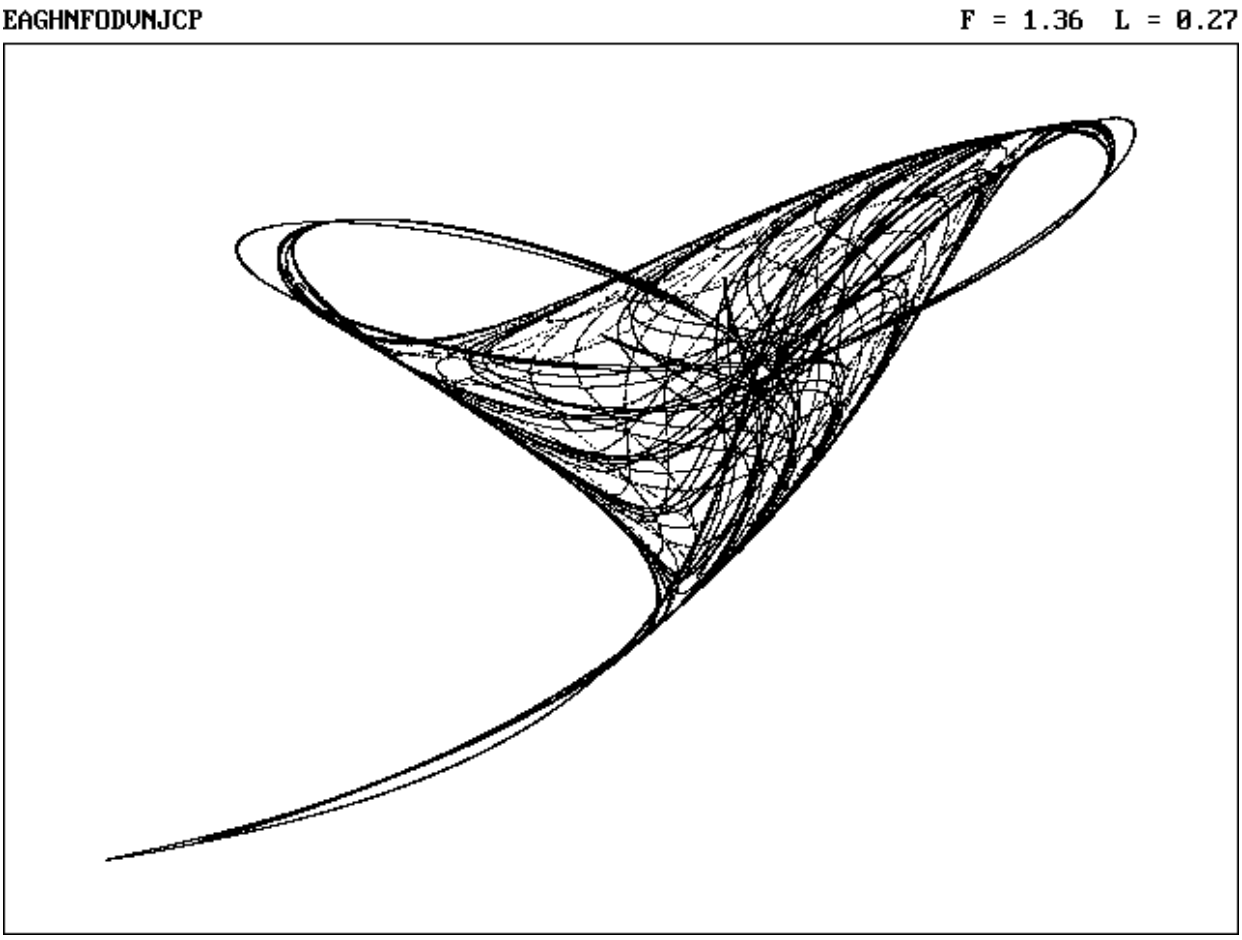


Figure 3-3. Two-dimensional quadratic map

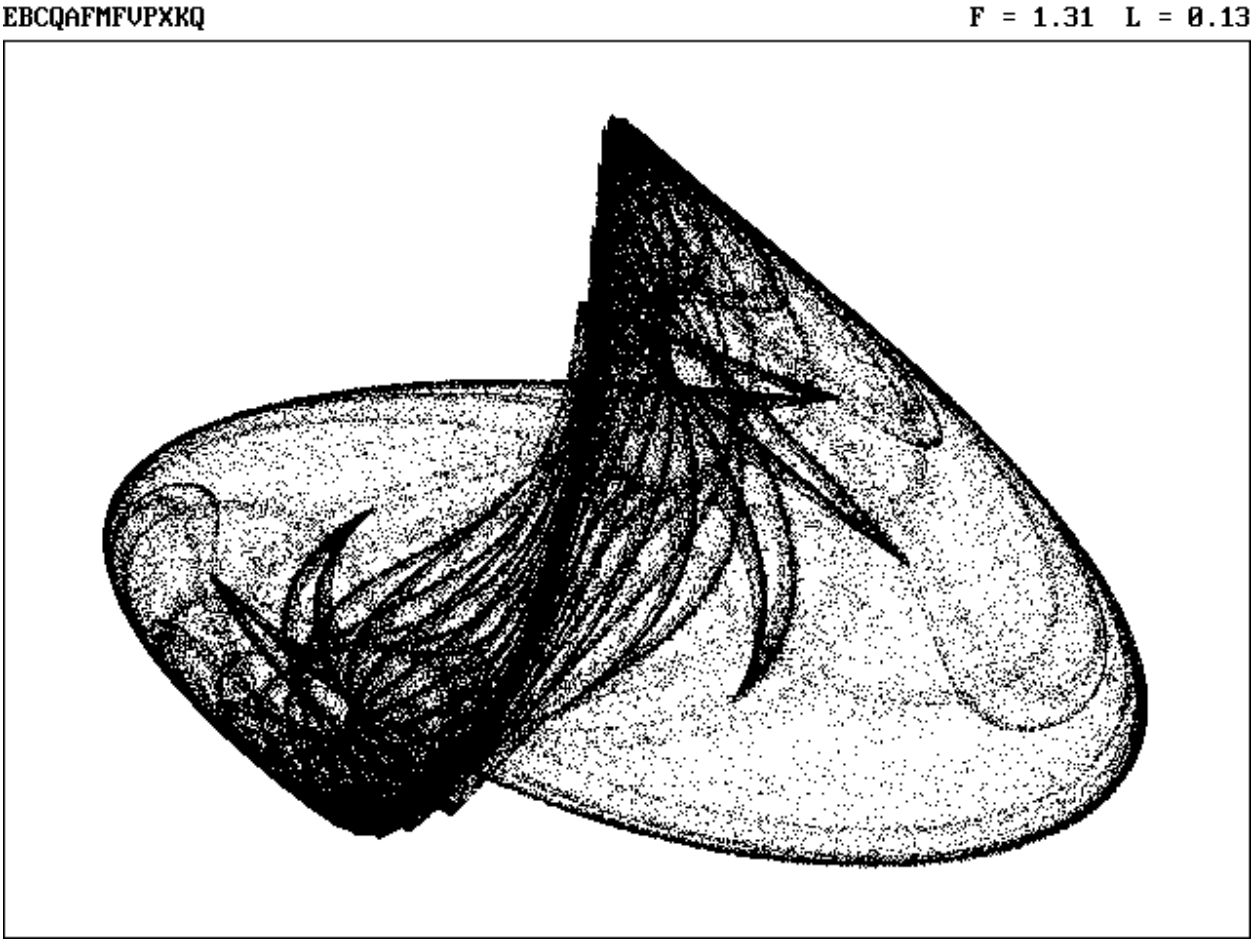


Figure 3-4. Two-dimensional quadratic map

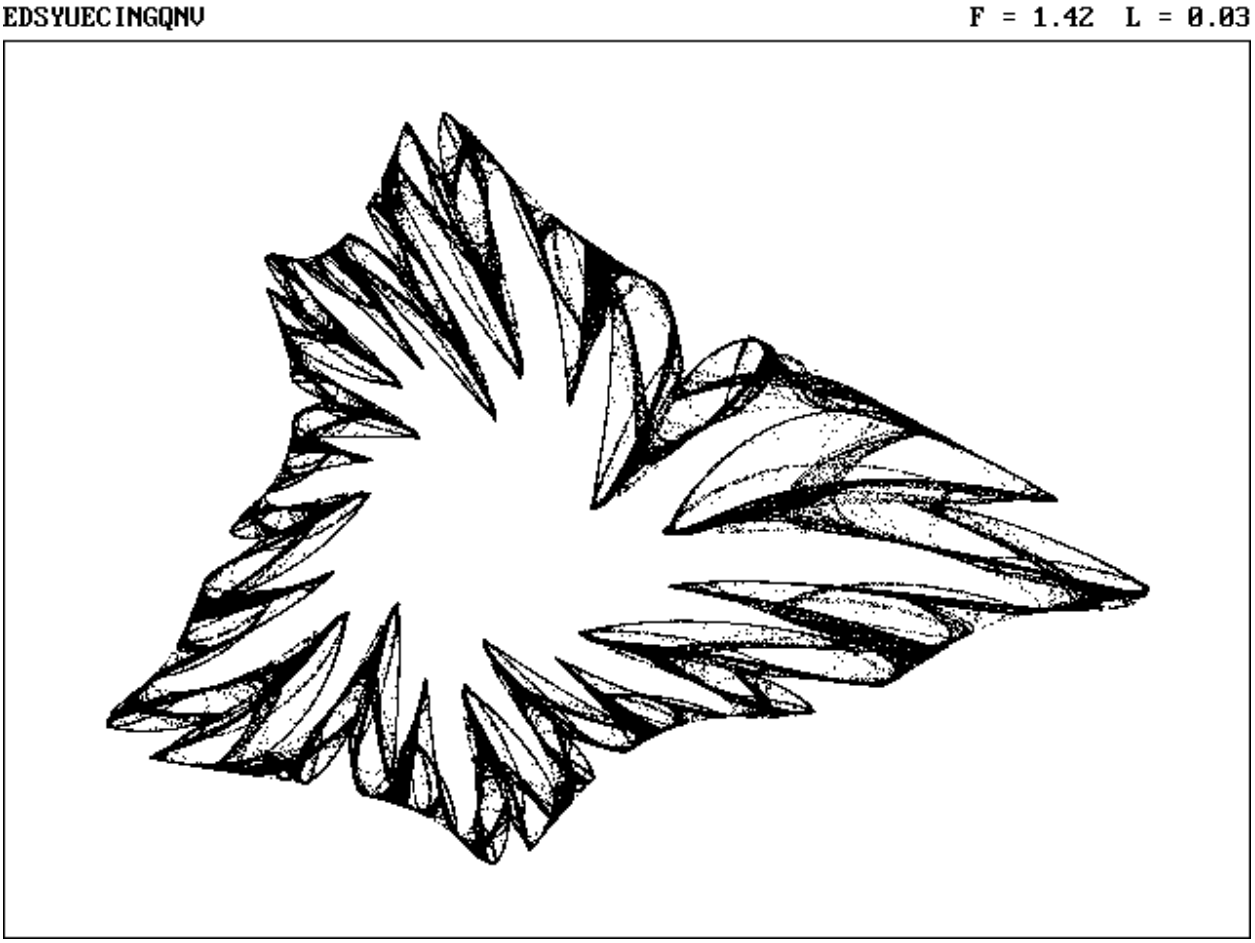


Figure 3-5. Two-dimensional quadratic map

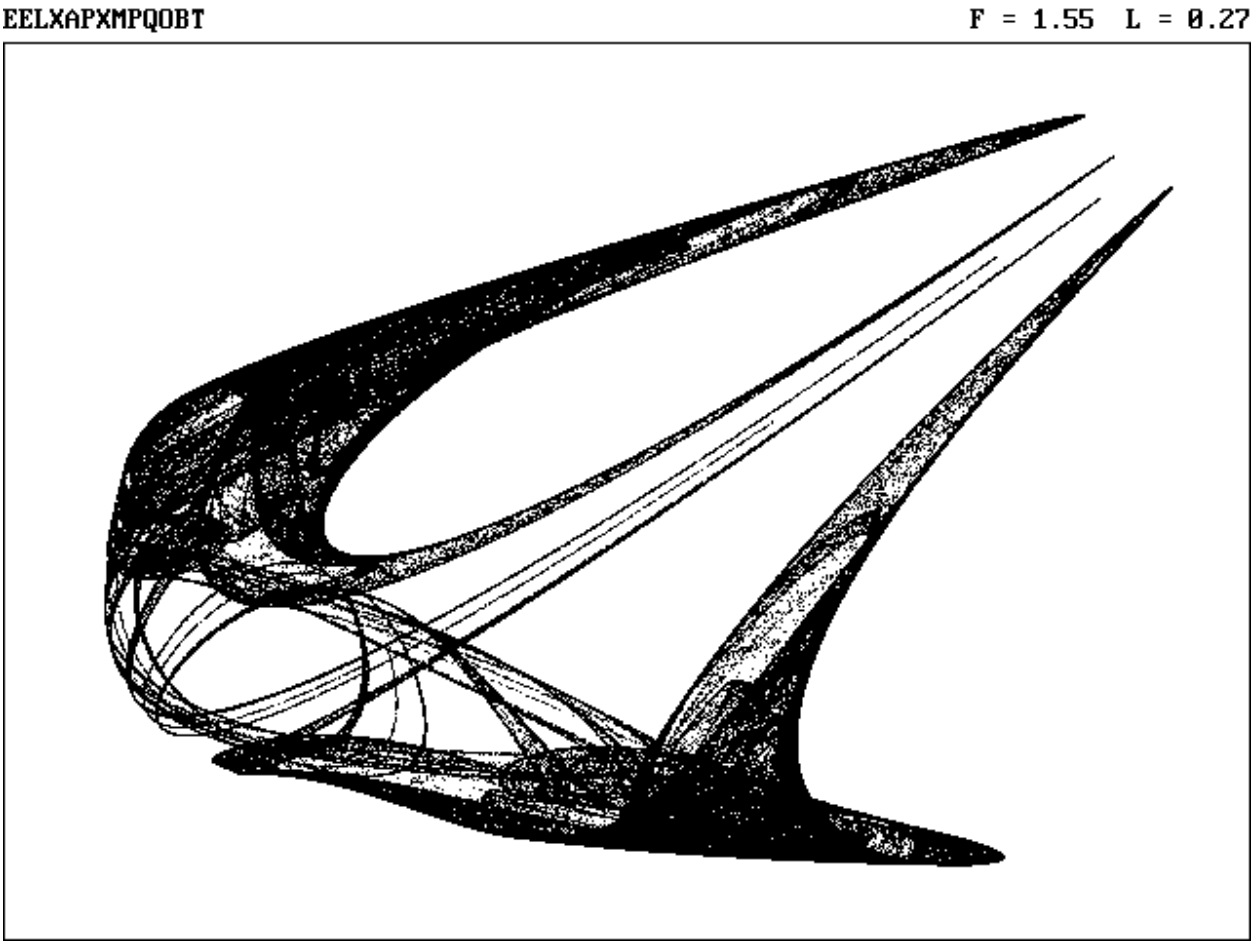


Figure 3-6. Two-dimensional quadratic map

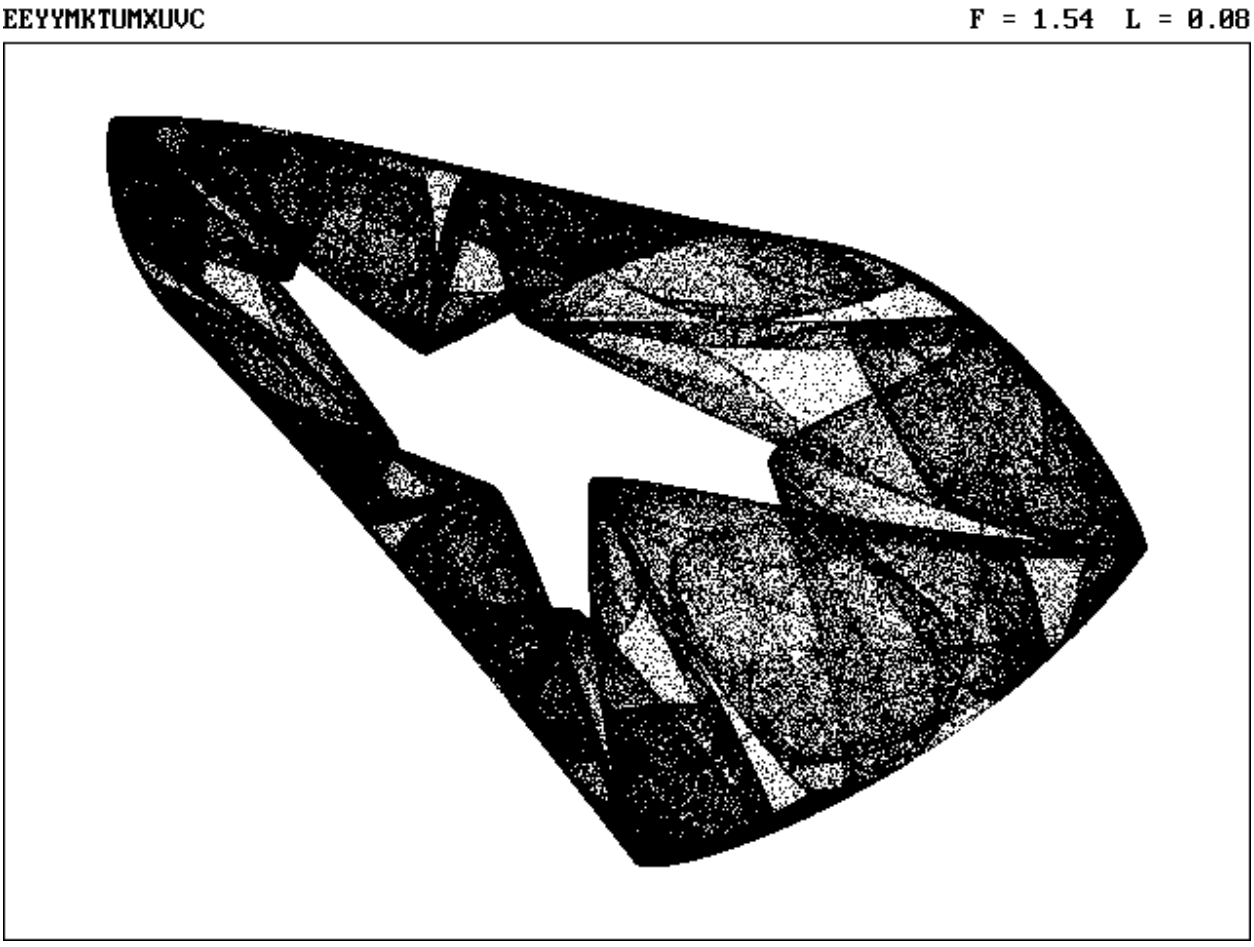


Figure 3-7. Two-dimensional quadratic map

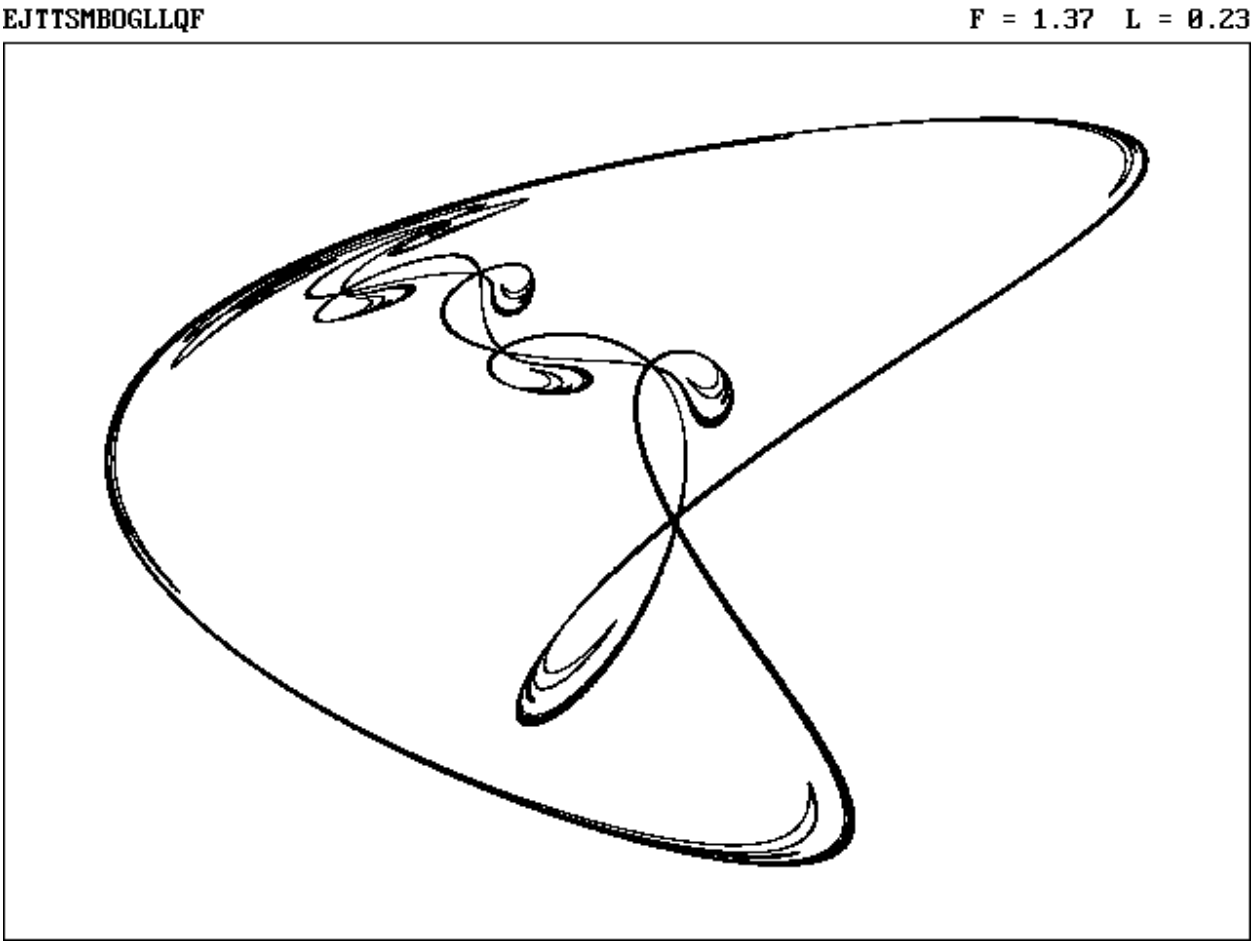


Figure 3-8. Two-dimensional quadratic map

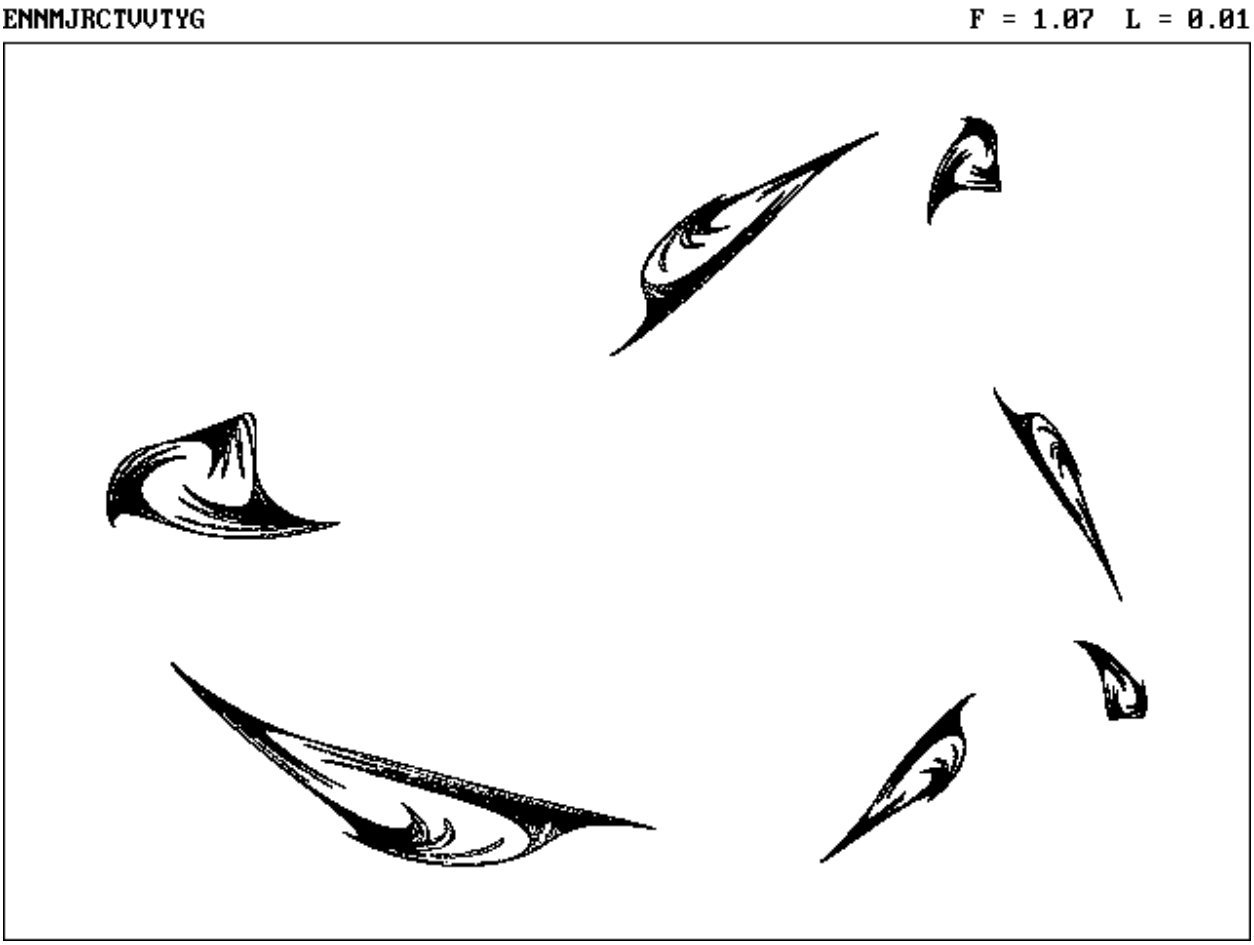


Figure 3-9. Two-dimensional quadratic map

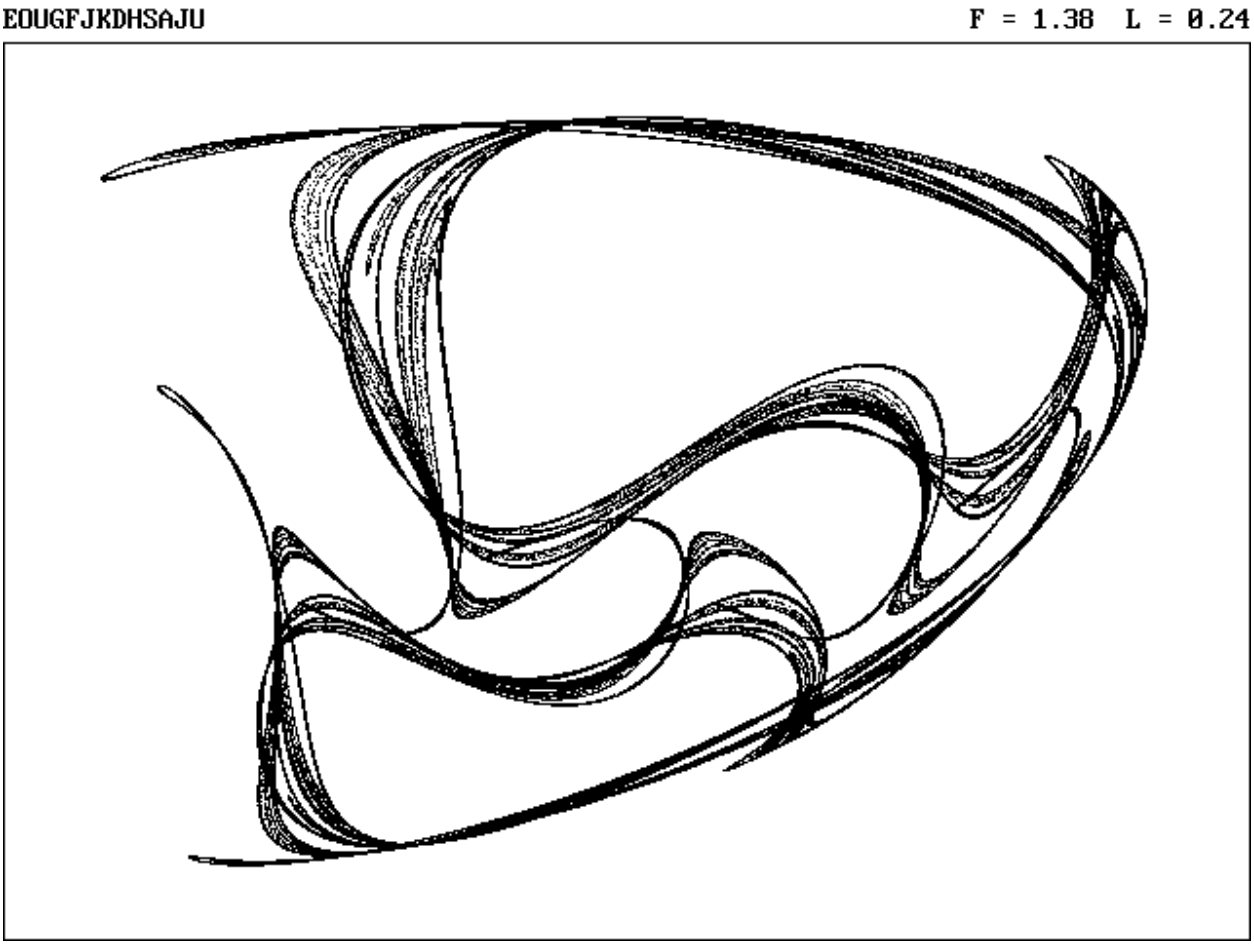


Figure 3-10. Two-dimensional quadratic map

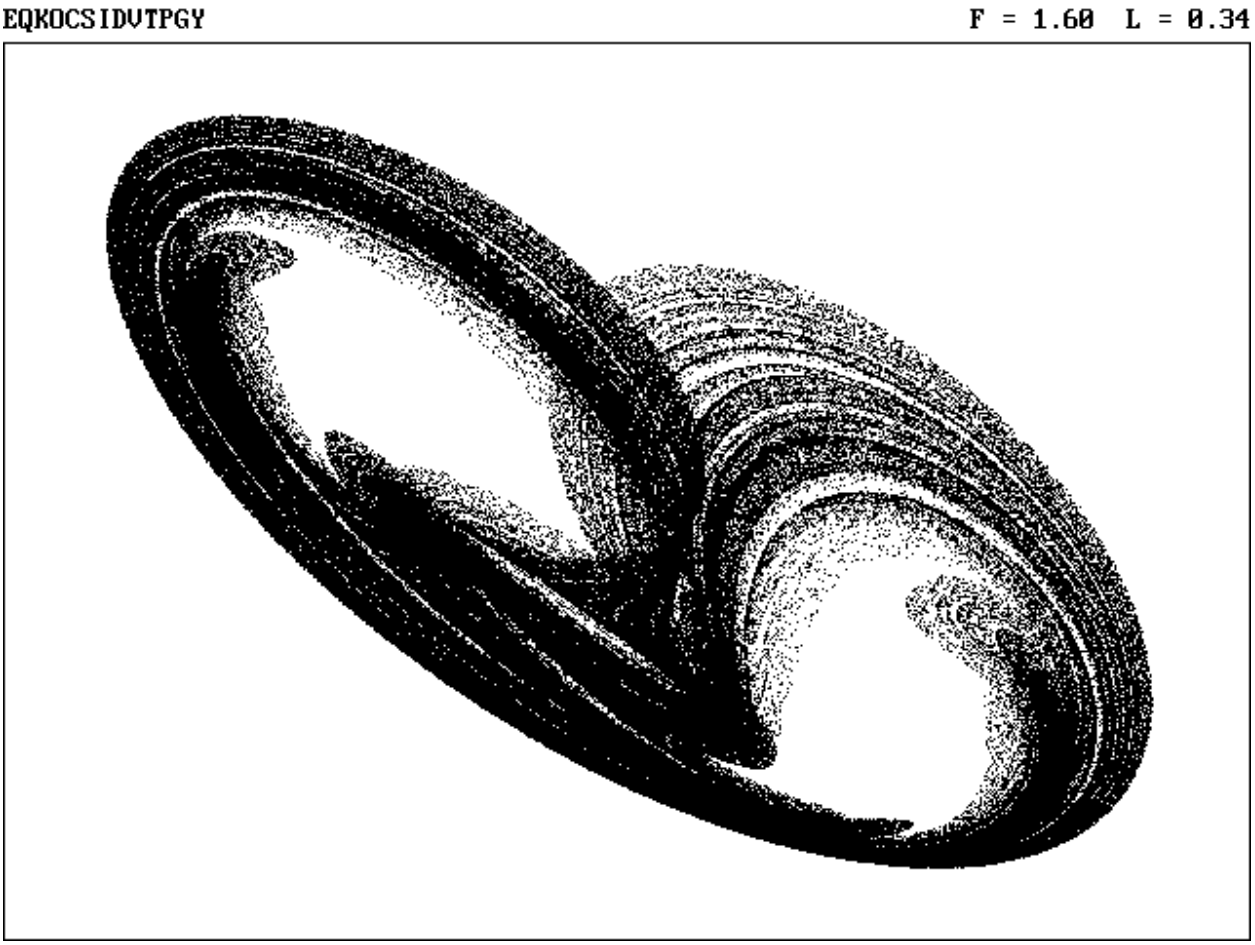


Figure 3-11. Two-dimensional quadratic map

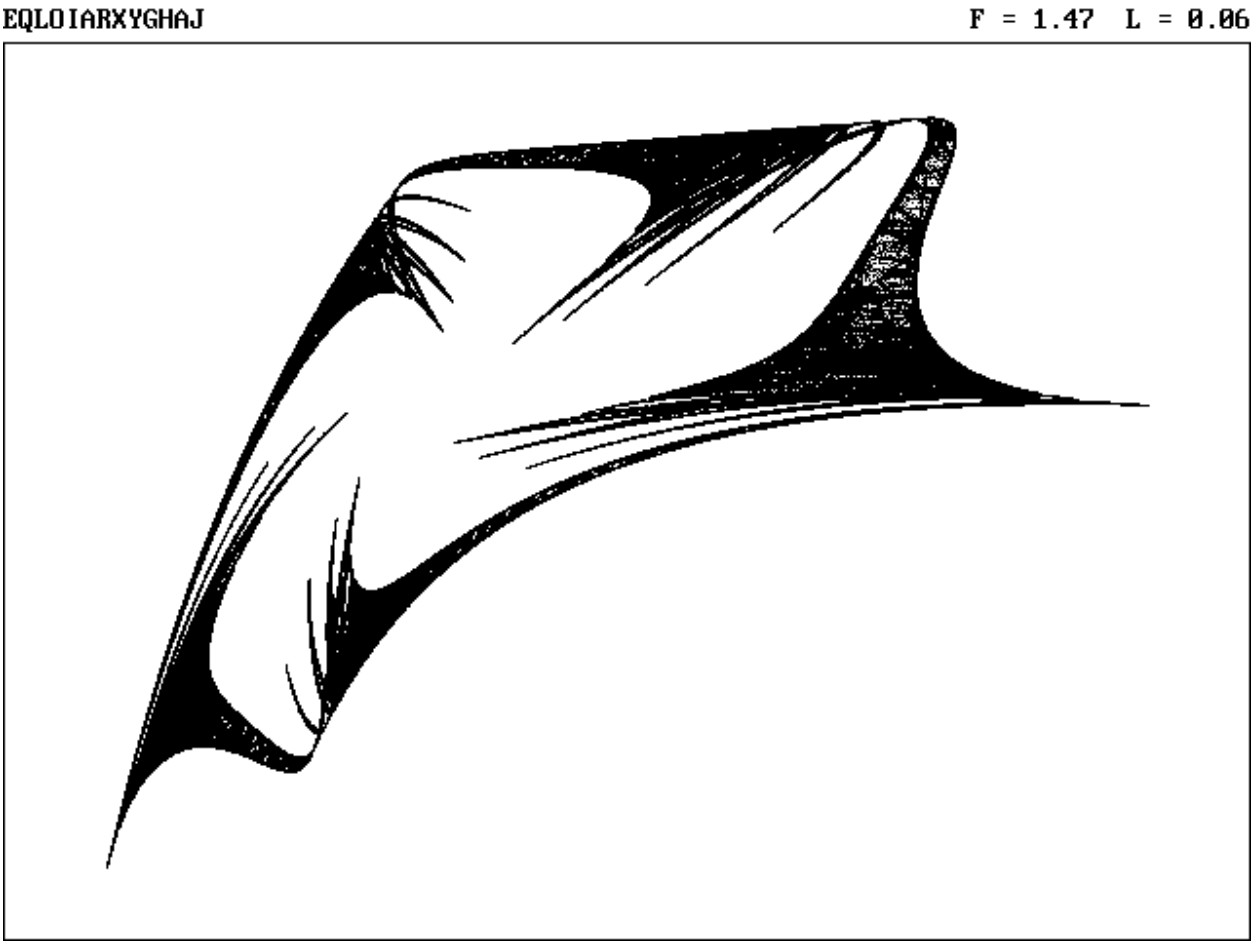


Figure 3-12. Two-dimensional quadratic map

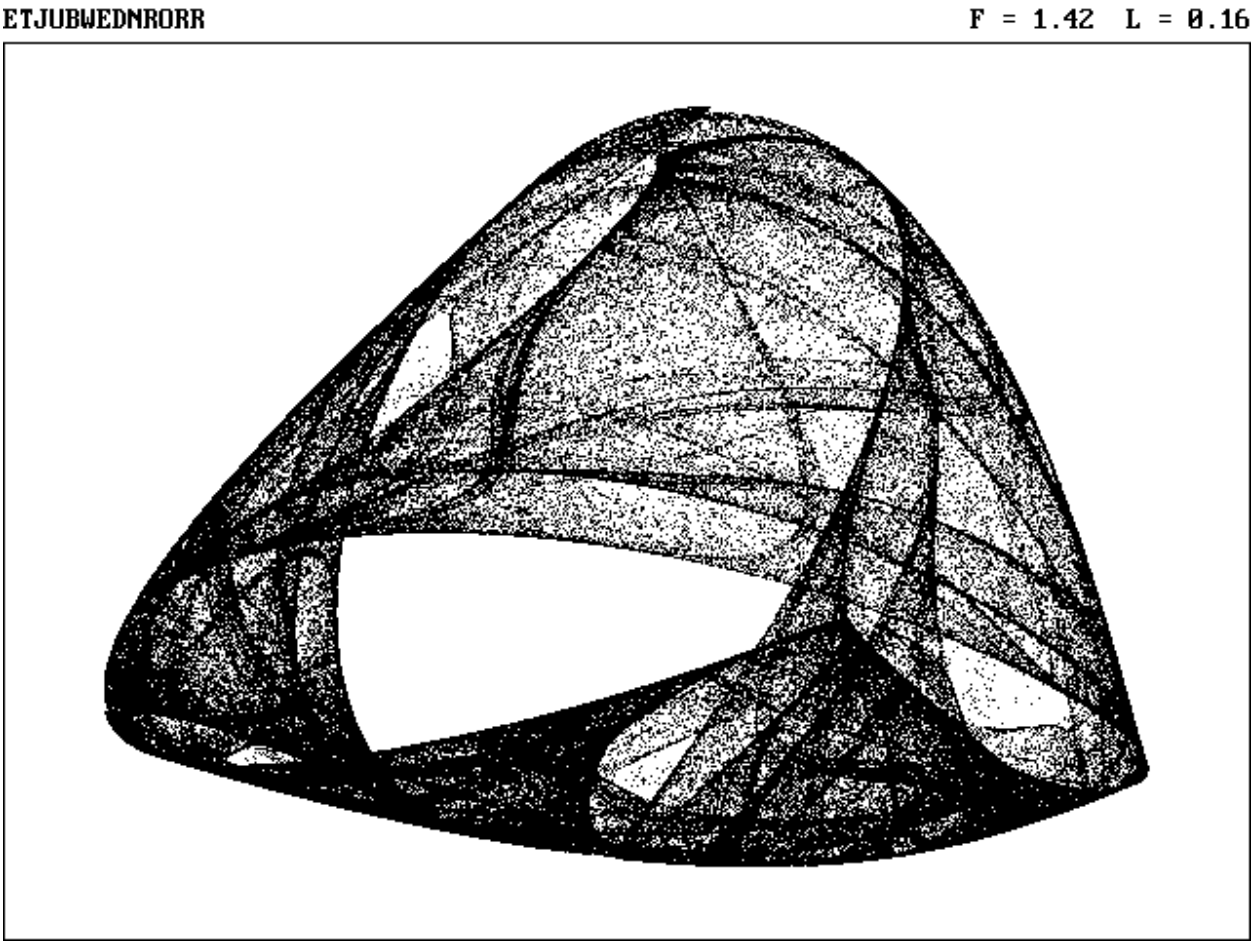


Figure 3-13. Two-dimensional quadratic map

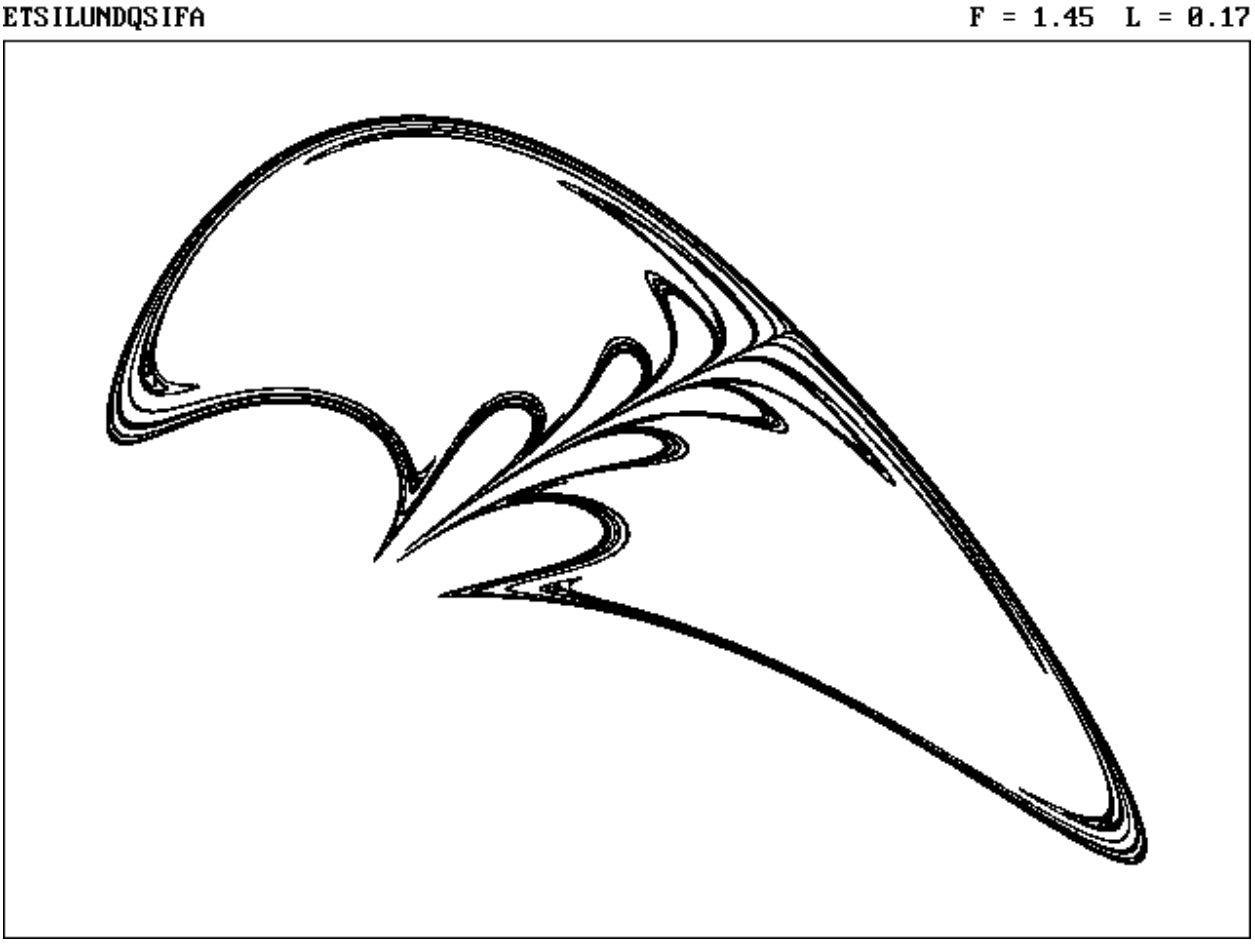


Figure 3-14. Two-dimensional quadratic map

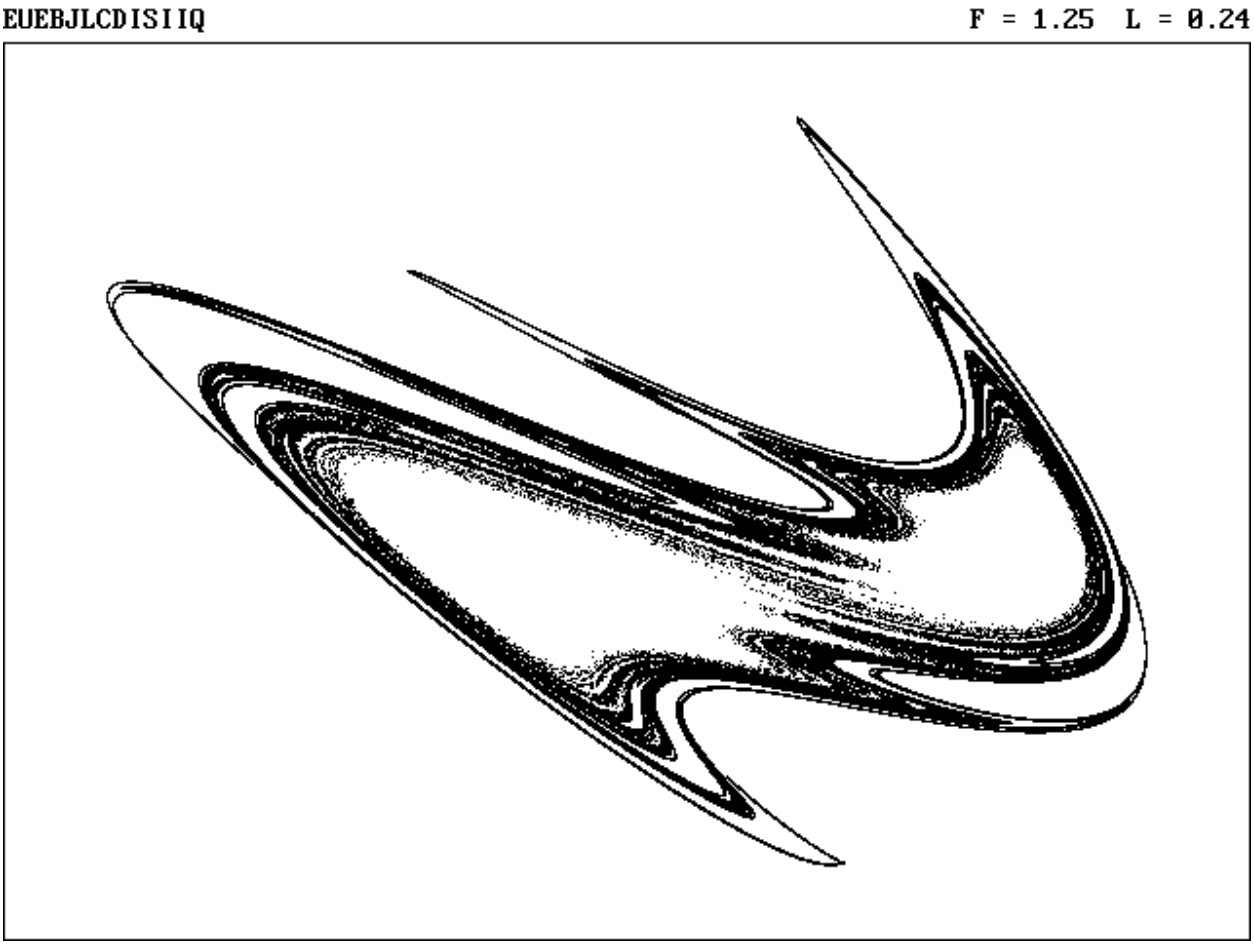


Figure 3-15. Two-dimensional quadratic map

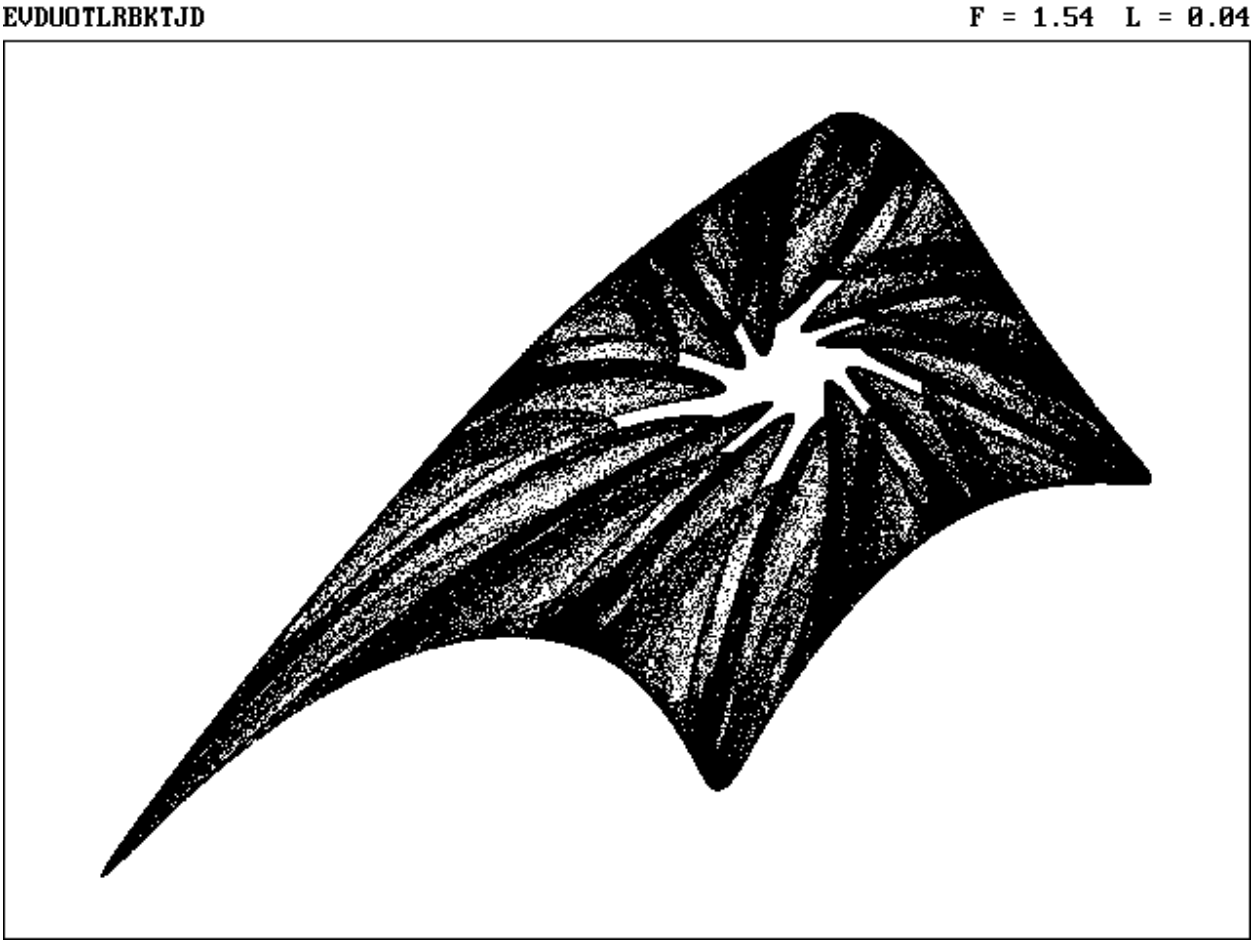


Figure 3-16. Two-dimensional quadratic map

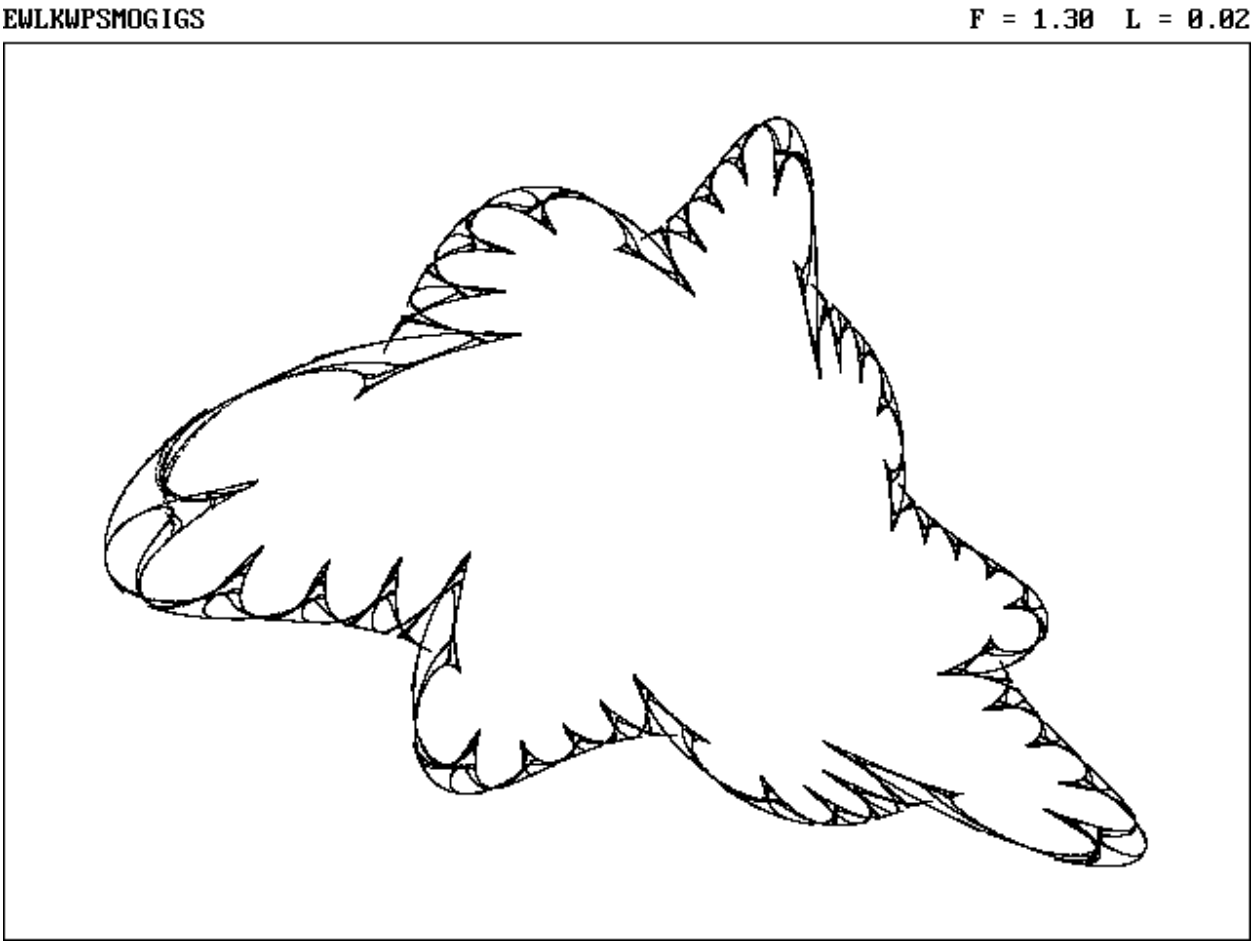
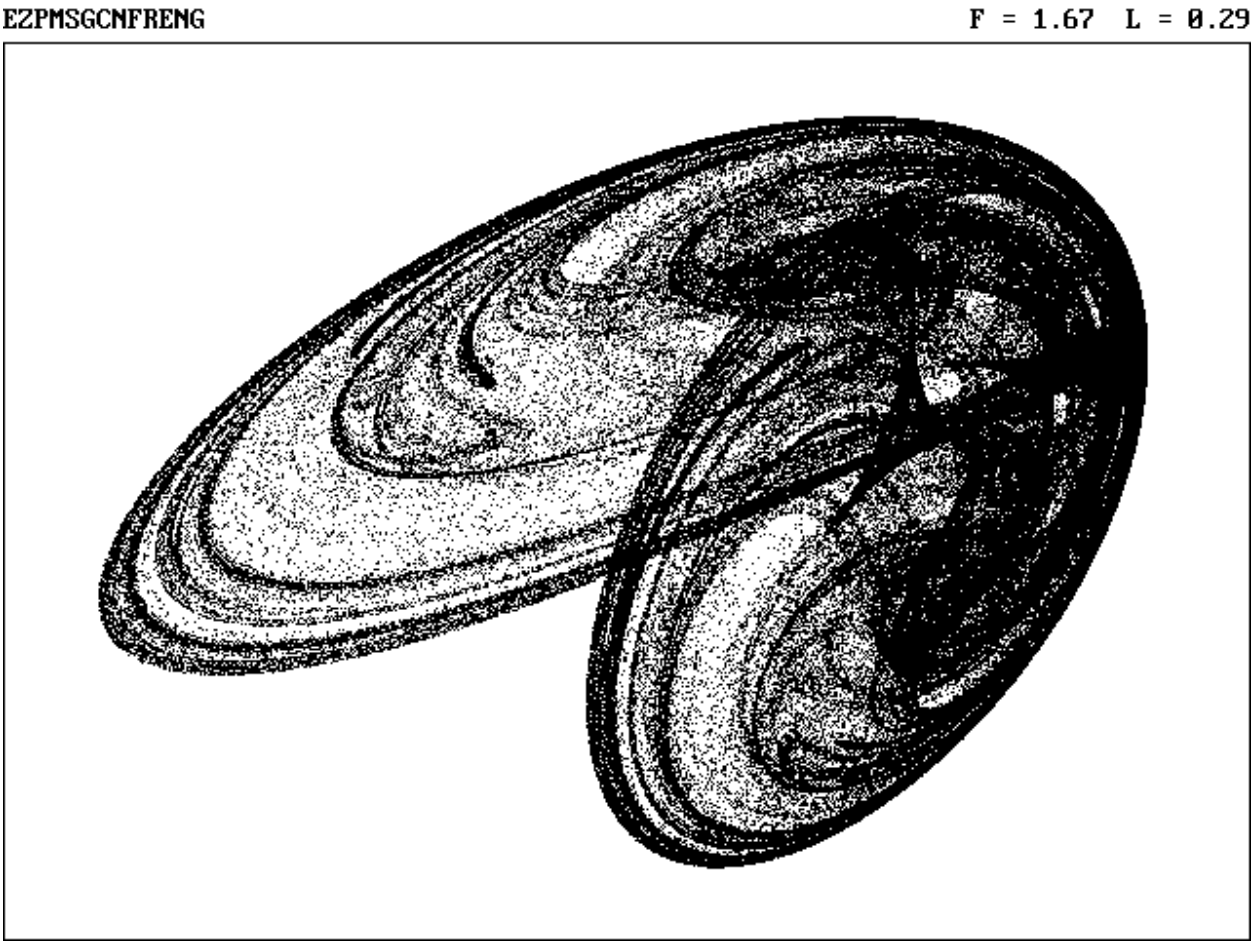


Figure 3-17. Two-dimensional quadratic map



If you are an experienced programmer, you might consider writing a screen-saver program based on **PROG06**. Such a *terminate-and-stay-resident* (TSR) program is run once when the computer is turned on and leaves a portion of itself in memory, constantly monitoring keyboard and mouse activity. When there is no user activity for, say, five minutes, it blanks the screen and begins displaying a succession of unique strange attractors to prevent screen burn-in. The original screen is restored whenever a key is pressed or the mouse is moved. PowerBASIC version 3.0 allows you to do this easily by inserting the program between POPUP statements.

3.4 The Fractal Dimension

The previous figures differ considerably in how densely they fill the plane. Some are very thin, others are thick. A good contrast is provided by Figures 3-16 and 3-17. Figure 3-16 resembles a piece of string that has been laid down in a complicated shape on the page, whereas Figure 3-17 looks like a twisted piece of paper with many holes in it. Thus the object in Figure 3-16 has a fractal dimension close to 1, and the object in Figure 3-17 has a fractal dimension closer to 2.

It is possible to be more explicit and to calculate the fractal dimension exactly. Consider two simple cases, one in which successive iterates lie uniformly along a straight line that goes diagonally across the page, and the other in which successive iterates gradually fill the entire plane, as if they were grains of pepper sprinkled on the paper from a great height. The first case has dimension 1, and the second has dimension 2. How would we calculate the dimension, given the *X* and *Y* coordinates of an arbitrary collection of such points?

One method is to draw a small circle somewhere on the plane that surrounds at least one of the points. We then draw a second circle with the same center but with twice the radius. Now we count the number of points inside each circle. Let's say the smaller circle encloses N_1 points and the larger circle encloses N_2 points. Obviously N_2 is greater than or equal to N_1 because all the points inside the inner circle are also inside the outer circle.

If the points are widely separated, then N_2 equals N_1. If the points are part of a straight line, the larger circle on average encloses twice as many points as the smaller circle, but if the points are part of a plane, the larger circle on average encloses four times as many points as the smaller circle, because the area of a circle is proportional to the square of its radius. Thus for these simple cases the dimension is given by

$$F = \log_2 (N_2 / N_1) \qquad \text{(Equation 3D)}$$

It is hardly surprising that if you do this operation for the cases shown in the figures, the quantity *F* is neither 0 nor 1 nor 2 but rather a fraction.

With real data, a number of practical considerations determine the accuracy of the result and the amount of computation required to obtain it:

1. Is a circle the best shape, or would a square, rectangle, triangle, or some other shape be better?

2. How large should the circle be?

3. Is doubling the size of the circle optimal, or would some other factor be better?

4. Where should the circles be placed, and how many circles are required to obtain a representative average?

5. How many points are needed to produce a reliable fractal dimension?

Let's address each of these questions in turn.

There is nothing special about circles. A rectangle, triangle, or any other two-dimensional figure would suffice, because the area scales as the square of the linear dimension in each case. However, a circle is convenient because it is easy to tell whether a given point is in its interior by comparing the radius of the circle with the distance of the point from its center.

The optimum size of the circle represents a compromise. Ideally, the circles should be invisibly small, because the dimension is defined in the limit of infinite resolution. However, if the circles are too small, they contain too few points to produce a statistically meaningful result, unless an unreasonably large number of iterations is performed. We somewhat arbitrarily use circles with a radius equal to about 2% of the diagonal of the smallest rectangle that contains the attractor.

Similarly, doubling the radius of the circle is arbitrary. Small values degrade the statistics, and large values miss too much of the fine-scale structure. We will use a value of ten, with the smaller circle about 0.6% the size of the attractor and the larger circle about 6% the size of the attractor. Thus in Equation 3D we will use logarithms of base 10 ($\log_{10}$) instead of base 2 ($\log_2$).

Ideally, the circles should be placed uniformly or randomly over the plane. However, if we were to do this, most of the circles would be empty, and a very long calculation would be required to obtain an accurate estimate of the dimension. Instead, we center the circles on the data points themselves. In this way the circles tend to enclose many points. However, it represents a different type of average because it weighs more heavily the portions of the attractor where the points are most dense. Technically, what we are calculating is called the *correlation dimension*, because it involves the number of other points that are correlated with each point in the data set. The correlation dimension is never greater than the fractal dimension, but it tends not to be much smaller either.

The correlation dimension is only one of many ways to define the dimension of an attractor. The various methods differ in how the regions of the attractor are weighed in the average. It is probably the easiest method to implement, and it gives more reliable results than the fractal dimension when the dimension of the attractor is greater than about two. The fractal dimension is also called the *capacity dimension*, and it is closely related to the *Hausdorff-Besicovitch dimension*. Furthermore, the correlation dimension is probably a more meaningful measure of the strangeness of the attractor, because it includes information about its formation as well as its final appearance.

The number of data points required to provide an accurate estimate of the dimension is a question still being debated in the scientific literature. Therefore, we will use a heuristic approach and continually update the dimension estimate with each iteration, giving you an opportunity to decide when it seems to have converged to a unique value. To do this, we must modify the procedure slightly. Rather than count the number of data points within a circle, which would require that the calculation run to conclusion with the coordinates of all the points saved, we use the equivalent method of determining the probability that two randomly chosen points are within a certain distance of one another. To do this, the distance of each new iterate from one of its randomly chosen predecessors is calculated. Now you see why we bothered to save the last 500 iterates! We exclude the most recent 20 points, because the iterates are likely to be abnormally highly correlated with their recent predecessors. Thus, with each iteration, we have only one additional calculation to do in which we compare the distance of the iterate to one of its randomly chosen predecessors and increment N_1 and N_2, as appropriate. **PROG07** shows the changes needed to calculate and display the fractal dimension.

PROG07. Changes required in PROG06 to calculate and display the fractal dimension

```
1000 REM TWO-D MAP SEARCH (With Fractal Dimension)
1020 DIM XS(499), YS(499), A(504), V(99)

1580 P% = 0: LSUM = 0: N = 0: NL = 0: N1 = 0: N2 = 0
1620 TWOD% = 2 ^ D%

2210 XS(P%) = X: YS(P%) = Y

2440 GOSUB 3900                 'Calculate fractal dimension

3030    LSUM = LSUM + LOG(DF): NL = NL + 1
3060    IF N > 1000 AND N MOD 10 = 0 THEN LOCATE 1, 76: PRINT USING "##.##"; L;

3170 XL = XMIN - MX: XH = XMAX + MX: YL = YMIN - MY: YH = YMAX + 1.5 * MY
3190 YH = YH - .5 * MY
3400 LOCATE 1, 1: PRINT CODE$
3420 LOCATE 1, 63: PRINT "F =": LOCATE 1, 73: PRINT "L ="

3900 REM Calculate fractal dimension
3910 IF N < 1000 THEN GOTO 4010       'Wait for transient to settle
3920 IF N = 1000 THEN D2MAX = (XMAX - XMIN) ^ 2 + (YMAX - YMIN) ^ 2
3930 J% = (P% + 1 + INT(480 * RND)) MOD 500
3940 DX = XNEW - XS(J%): DY = YNEW - YS(J%)
```

```
3950 D2 = DX * DX + DY * DY

3960 IF D2 < .001 * TWOD% * D2MAX THEN N2 = N2 + 1

3970 IF D2 > .00001 * TWOD% * D2MAX THEN GOTO 4010

3980    N1 = N1 + 1

3990    F = .434294 * LOG(N2 / (N1 - .5))

4000    LOCATE 1, 66: PRINT USING "##.##"; F;

4010 RETURN
```

At this point you might want to examine the fractal dimension of the various figures in this book as well as the dimension of those you create with **PROG07**. One thing you will notice is that the dimension of objects that resemble lines is often less than 1.0. One reason is that the points that make up the line are seldom uniformly distributed along its length. Remember that the correlation dimension is usually smaller than the fractal dimension. They are equal if the points are uniformly distributed over the attractor. The correlation dimension of a line consisting of a uniform distribution of points along its length would be exactly 1.0.

Also note that the dimension of most attractors varies considerably from one part of the attractor to another. Figure 3-11 is a good example of one in which parts of the attractor resemble thin lines and other parts resemble filled-in planes. It obviously is simplistic to characterize such an object by a single average dimension.

The dimension also depends on scale. It is properly defined in the limit where one zooms in very tightly on the attractor to observe its finest detail. However, a calculation in this limit would take forever because an infinite number of iterations would be required to collect enough points to reveal the detail. Figure 3-13 is an example of an attractor that is nearly one-dimensional on a large scale but closer to two-dimensional on a fine scale. Our calculation provides what might be called a *visual dimension* because it is taken on a scale close to what the eye can visually resolve. In any case, you should not ascribe undue significance to the calculated dimension.

Also note that we are using the word "dimension" to mean several different things. The maps that we are looking at are two-dimensional because they have two variables, *X* and *Y*. However, the attractor has a smaller dimension. We say the

attractor is *embedded* in a two-dimensional space or that the *embedding dimension* is 2. A point or a line can be embedded in a plane, but a ball cannot.

An attractor usually fills a negligible portion of the space in which it is embedded. That's why it's called an attractor! Points initially distributed throughout the embedding space are drawn to the attractor after a number of iterations, and the remaining space is left empty. Thus the area of an attractor embedded in a two-dimensional space is zero, and the volume of an attractor embedded in a three-dimensional space is zero, and so forth.

It is also interesting that the fractal dimension and the Lyapunov exponents are not entirely independent. It has been conjectured that the fractal dimension is related to the two Lyapunov exponents by

$$F = 1 - L_1 / L_2 \qquad \text{(Equation 3E)}$$

where L_1 is the more positive of the two exponents and is the one we denote by *L* in the figures. If both Lyapunov exponents are known, Equation 3E can be used to define a dimension of the attractor, called the *Lyapunov dimension*. The Lyapunov dimension is also called the *Kaplan-Yorke dimension* after the scientists who proposed an extension of Equation 3E to higher dimensions.

This relation is reasonable because, if the two exponents are equal but of opposite signs ($L_2 = - L_1$), the contraction in one direction is just offset by expansion in the other. A set of initial conditions spread out over a two-dimensional region thus maintains its area upon successive iteration. Such a mapping is said to be *area-preserving*, *symplectic*, or *Hamiltonian*, after the 19th-century Irish astronomer, William Rowan Hamilton. On the other hand, if the contraction is very rapid (L_2 is large and negative), the initial conditions quickly collapse to a very elongated ellipse whose dimension is close to 1. Such a contraction is sometimes called *filamentation*.

Armed with information about the fractal dimension, you can program the computer to be even more selective. For example, the visually appealing attractors tend to have fractal dimensions slightly greater than 1, and thus you could reject those with smaller dimensions or those with dimensions close to 2. We return to this intriguing possibility in Chapter 8.

3.5 Higher-Order Disorder

With one-dimensional maps, the attractors became more interesting when we considered terms higher than quadratic. It is straightforward to do the same with two-dimensional maps. For example, the most general equations for two-dimensional cubic maps are

$$X_{n+1} = a_1 + a_2X_n + a_3X_n^2 + a_4X_n^3 + a_5X_n^2Y_n$$
$$+ a_6X_nY_n + a_7X_nY_n^2 + a_8Y_n + a_9Y_n^2 + a_{10}Y_n^3$$
$$Y_{n+1} = a_{11} + a_{12}X_n + a_{13}X_n^2 + a_{14}X_n^3 + a_{15}X_n^2Y_n$$
$$+ a_{16}X_nY_n + a_{17}X_nY_n^2 + a_{18}Y^n + a_{19}Y_n^2 + a_{20}Y_n^3 \qquad \text{(Equation 3F)}$$

Note that there are 20 coefficients, which vastly increases the number of possible cases. The fourth-order case would have 30 coefficients, and the fifth-order case would have 42 coefficients. If you prefer an equation, a two-dimensional map of order *O* has $(O + 1)(O + 2)$ coefficients. We will code the cubic, quartic, and quintic cases with the letters F, G, and H, respectively.

The changes that must be made to the program to generate attractors in two dimensions up to fifth order are given in **PROG08**.

PROG08. Changes required in PROG07 to generate attractors in two dimensions up to fifth order

```
1000 REM TWO-D MAP SEARCH (Polynomials up to 5th Order)

1020 DIM XS(499), YS(499), A(504), V(99), XY(4), XN(4)

1060 OMAX% = 5                  'Maximum order of polynomial



1720 M% = 1: XY(1) = X: XY(2) = Y

1730 FOR I% = 1 TO D%

1740 XN(I%) = A(M%)

1750 M% = M% + 1
```

```
1760 FOR I1% = 1 TO D%
1770 XN(I%) = XN(I%) + A(M%) * XY(I1%)
1780 M% = M% + 1
1790 FOR I2% = I1% TO D%
1800 XN(I%) = XN(I%) + A(M%) * XY(I1%) * XY(I2%)
1810 M% = M% + 1
1820 IF O% = 2 THEN GOTO 1970
1830 FOR I3% = I2% TO D%
1840 XN(I%) = XN(I%) + A(M%) * XY(I1%) * XY(I2%) * XY(I3%)
1850 M% = M% + 1
1860 IF O% = 3 THEN GOTO 1960
1870 FOR I4% = I3% TO D%
1880 XN(I%) = XN(I%) + A(M%) * XY(I1%) * XY(I2%) * XY(I3%) * XY(I4%)
1890 M% = M% + 1
1900 IF O% = 4 THEN GOTO 1950
1910 FOR I5% = I4% TO D%
1920 XN(I%) = XN(I%) + A(M%) * XY(I1%) * XY(I2%) * XY(I3%) * XY(I4%) * XY(I5%)
1930 M% = M% + 1
1940 NEXT I5%
1950 NEXT I4%
1960 NEXT I3%
1970 NEXT I2%
1980 NEXT I1%
```

```
2000 NEXT I%

2010 M% = M% - 1: XNEW = XN(1): YNEW = XN(2)
```

PROG08 could have been written more compactly, but it is done this way to simplify its extension to even higher dimensions. Examples of attractors produced by this program are shown in Figures 3-18 through 3-41.

Figure 3-18. Two-dimensional cubic map

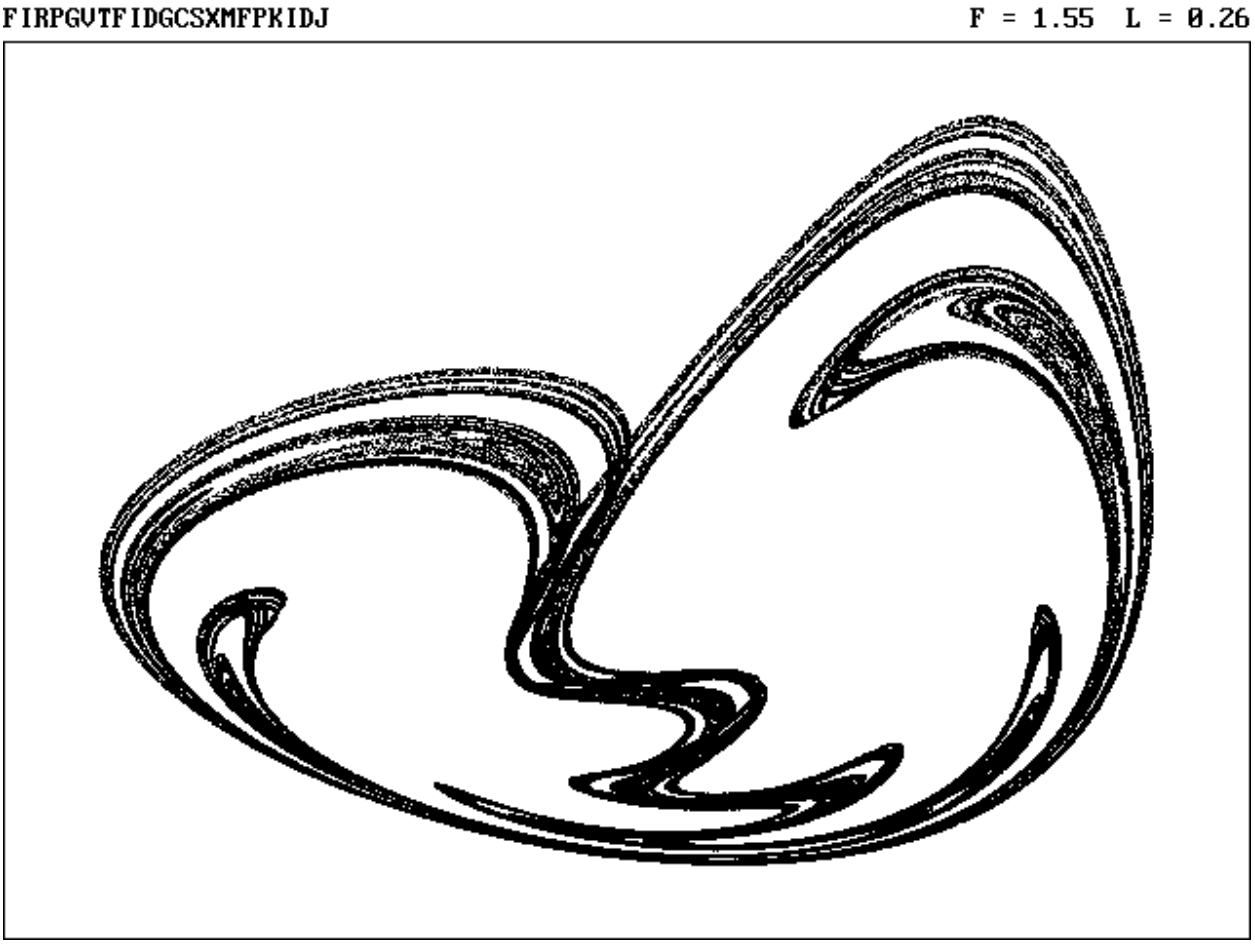


Figure 3-19. Two-dimensional cubic map

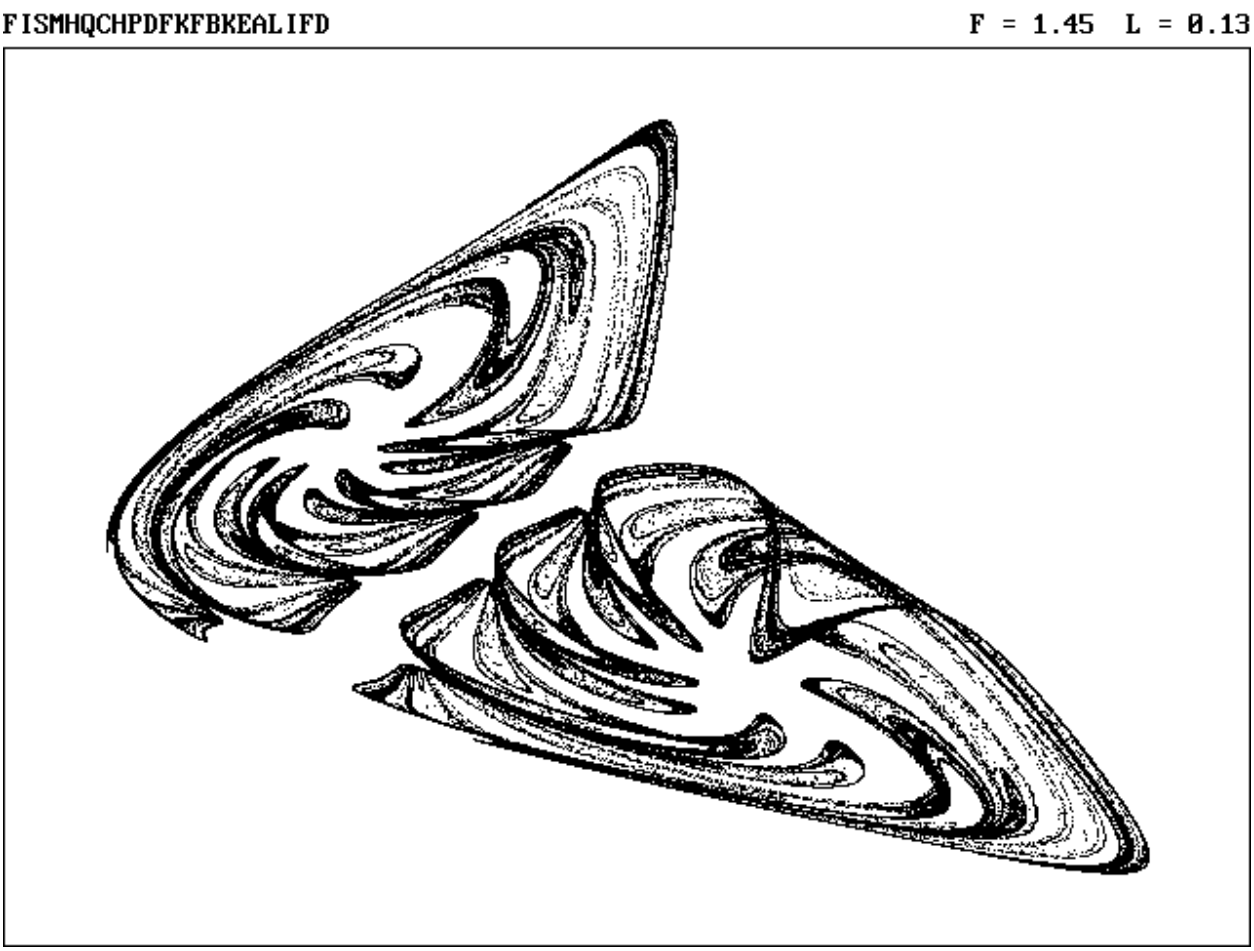


Figure 3-20. Two-dimensional cubic map

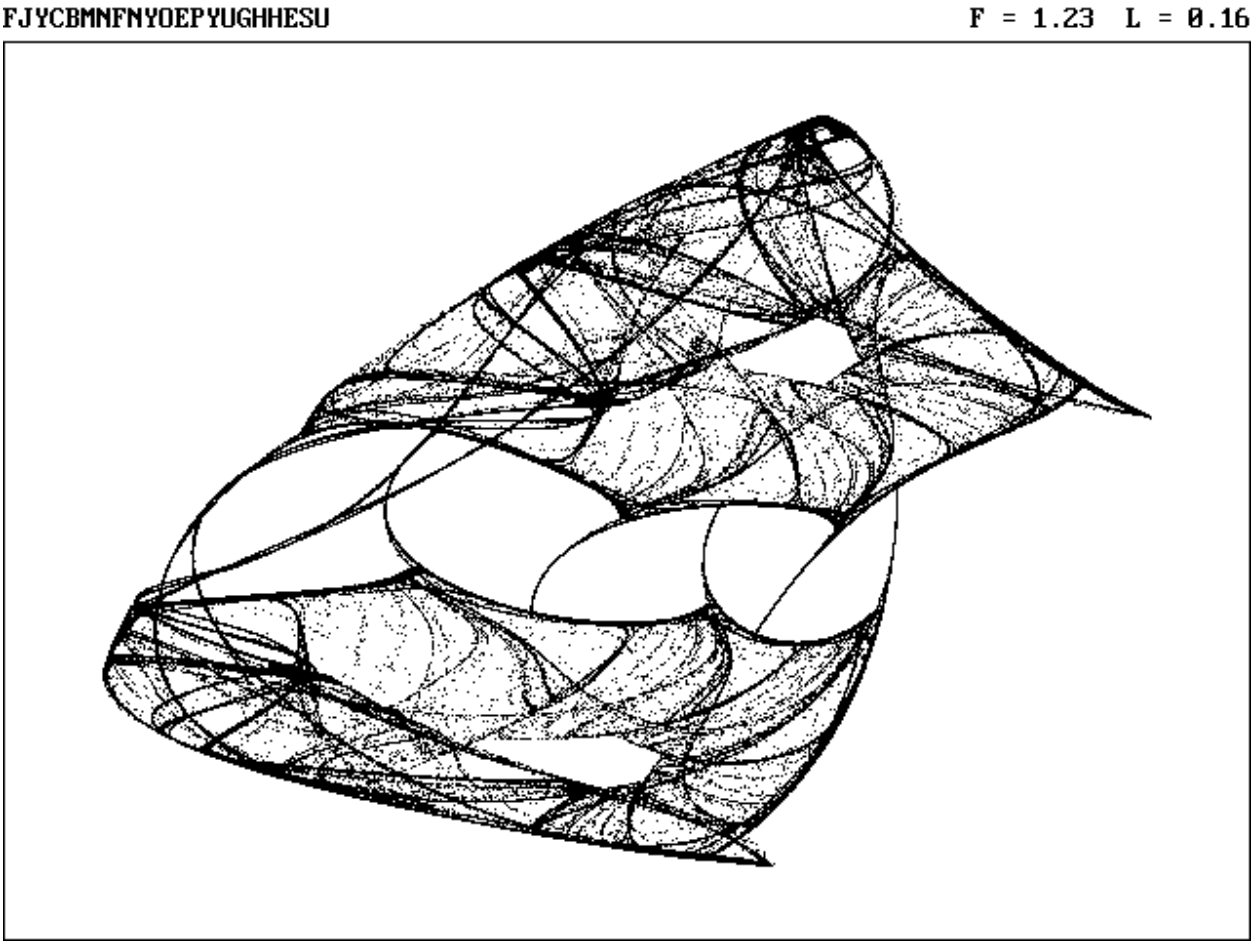


Figure 3-21. Two-dimensional cubic map

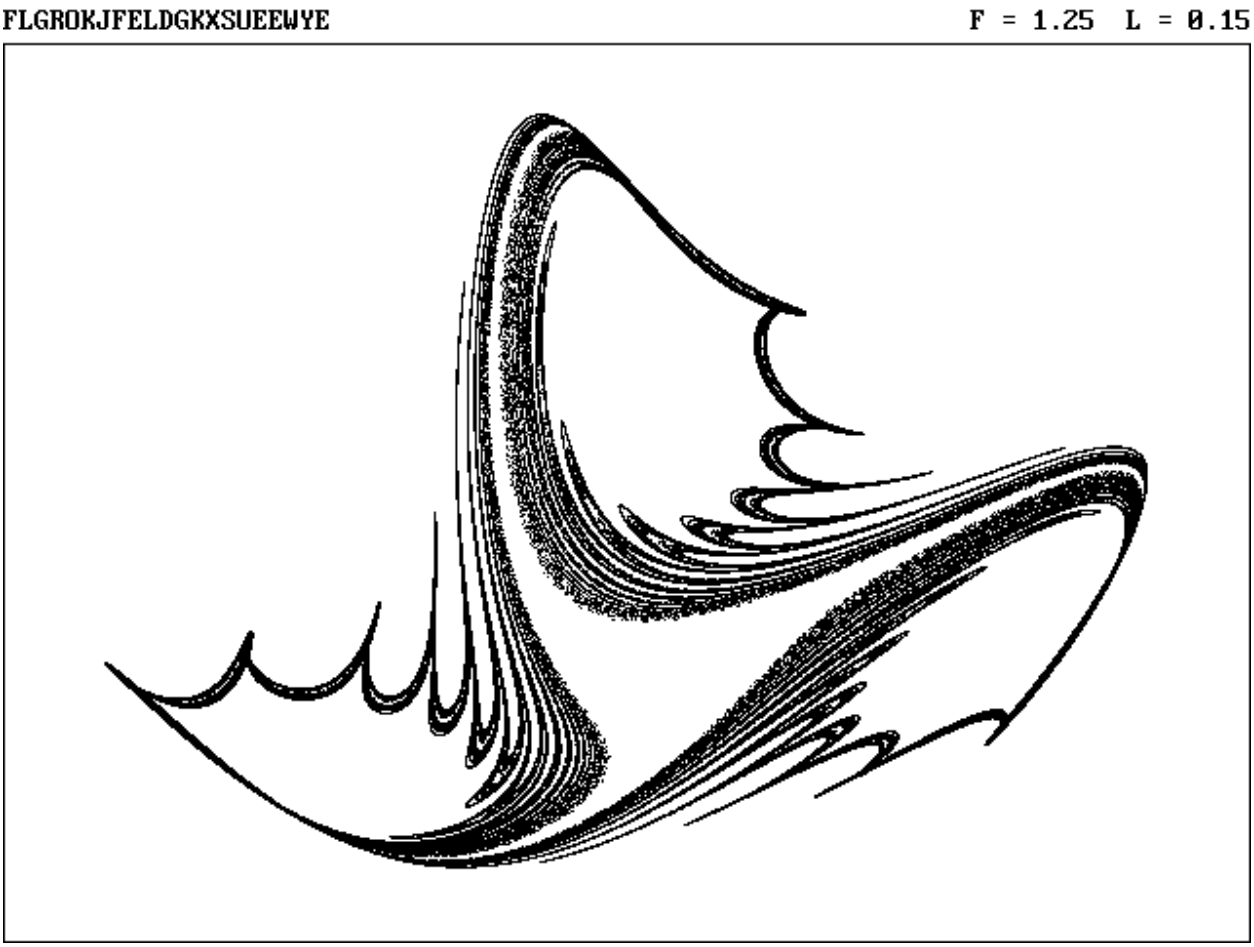


Figure 3-22. Two-dimensional cubic map

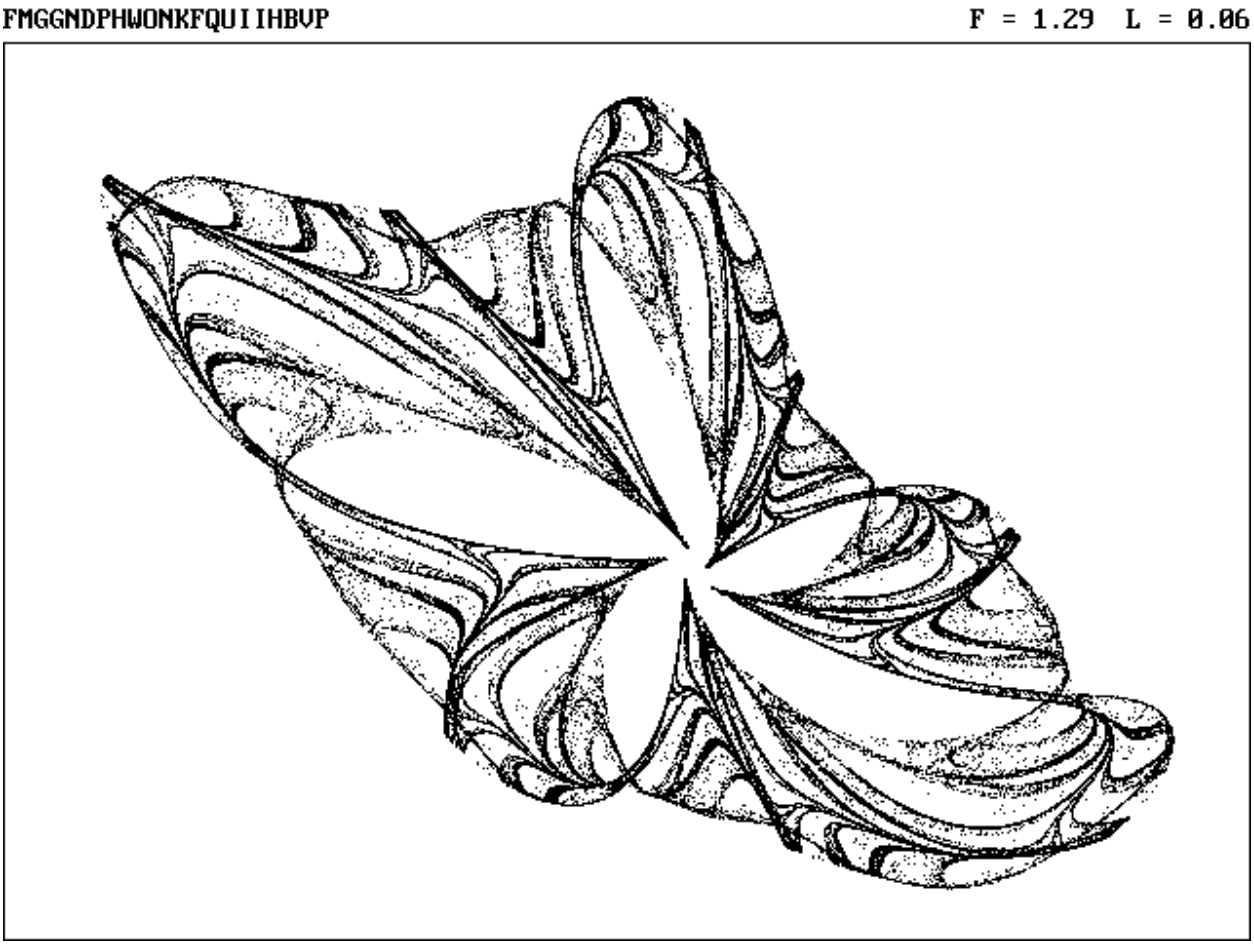


Figure 3-23. Two-dimensional cubic map

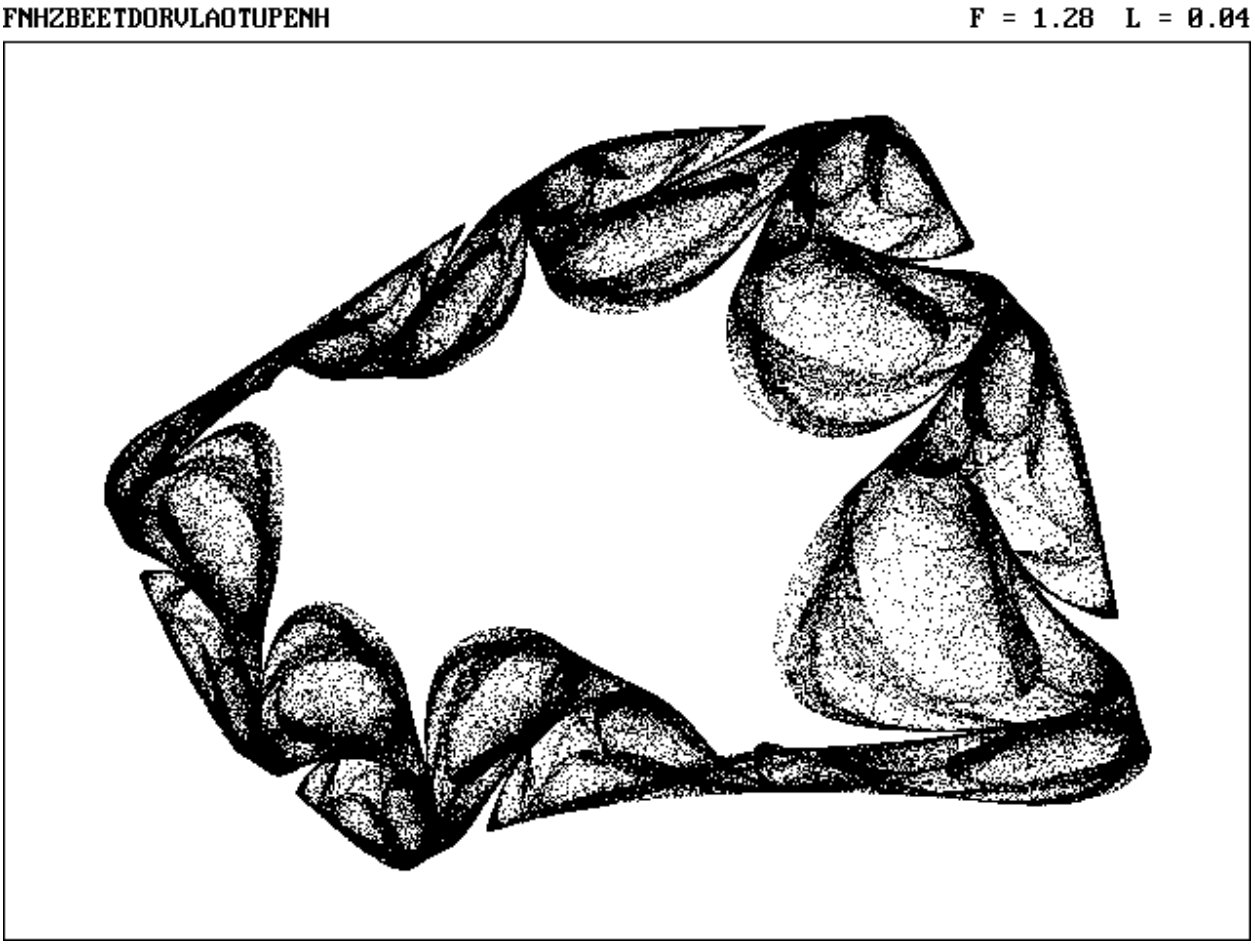


Figure 3-24. Two-dimensional cubic map

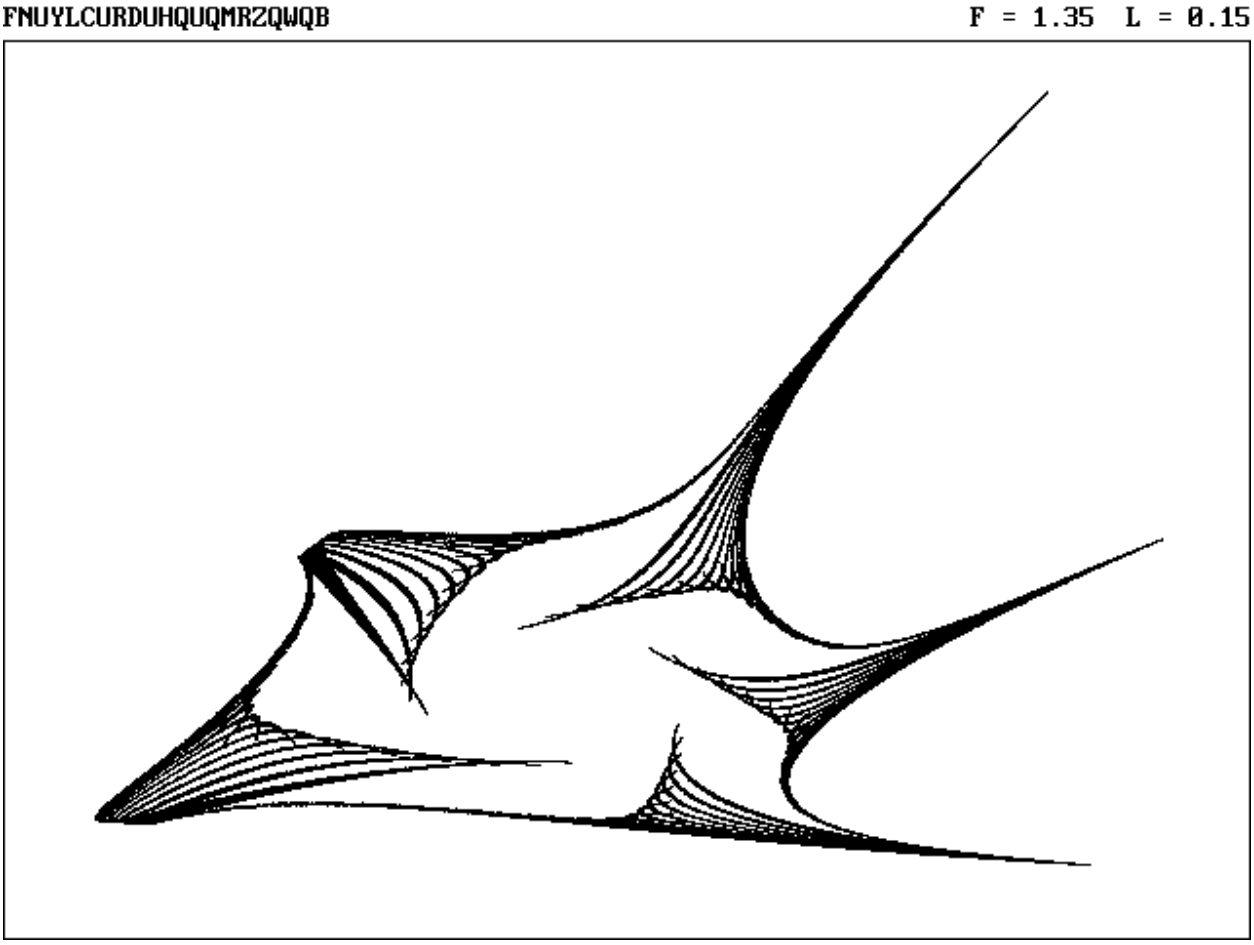


Figure 3-25. Two-dimensional cubic map

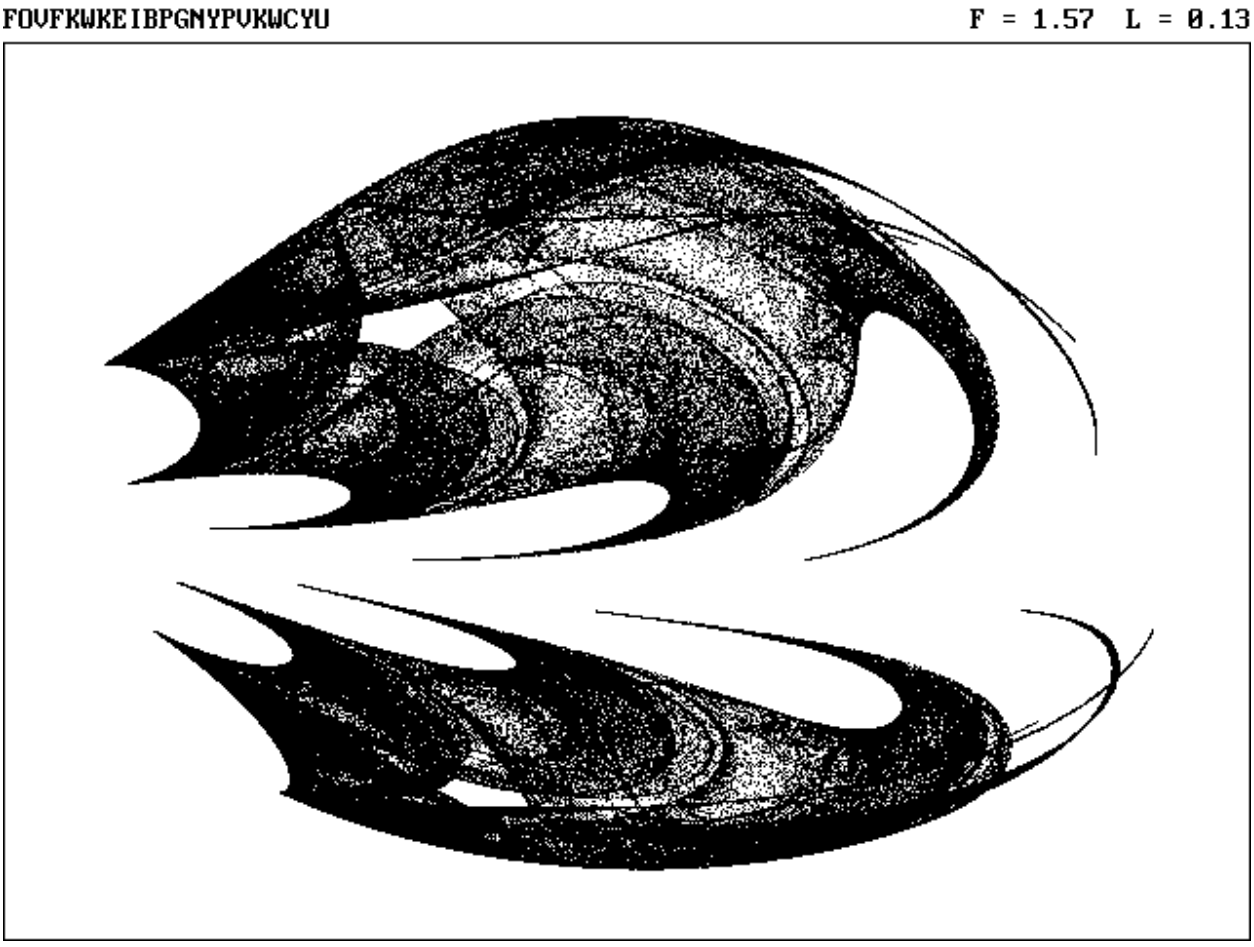


Figure 3-26. Two-dimensional quartic map

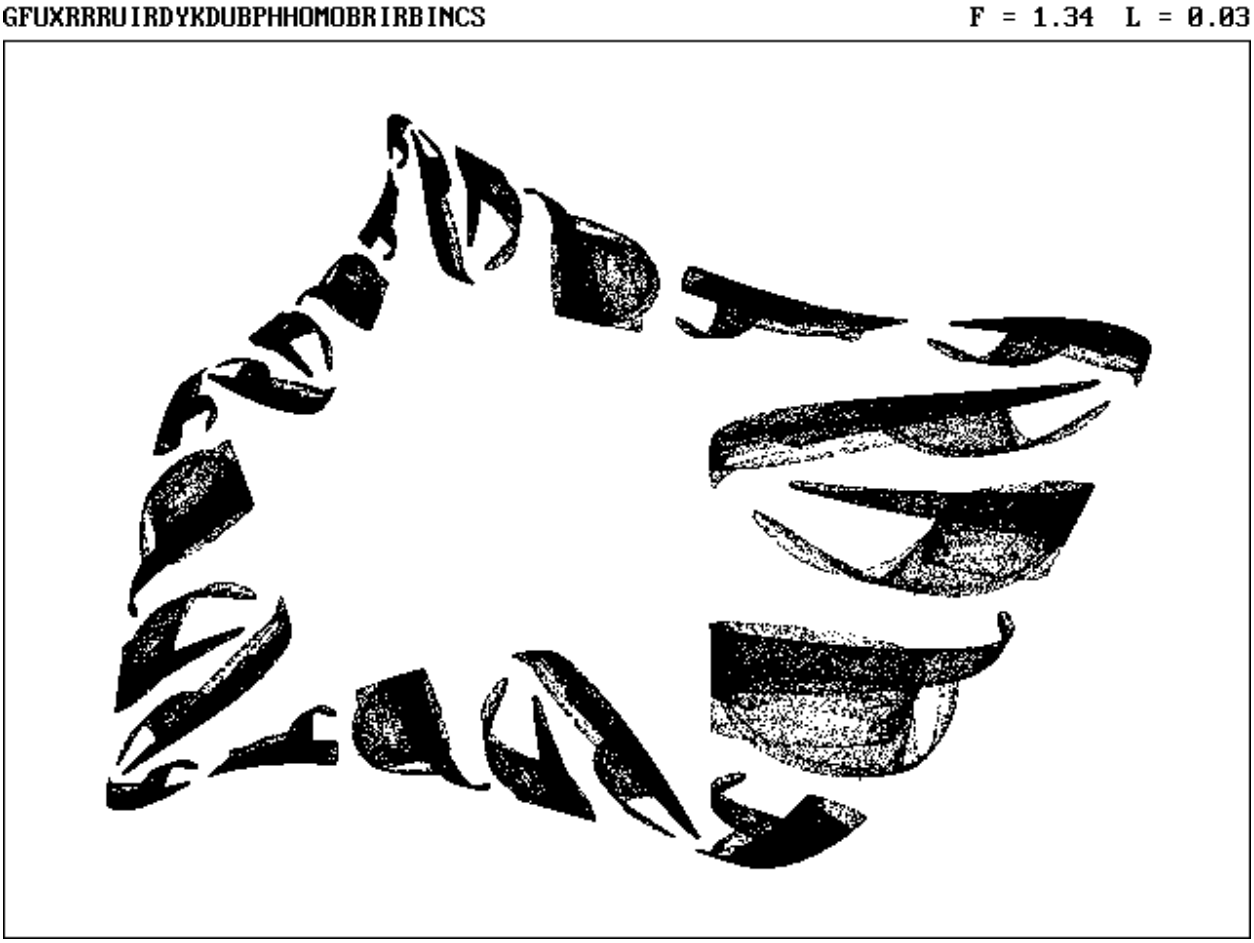


Figure 3-27. Two-dimensional quartic map

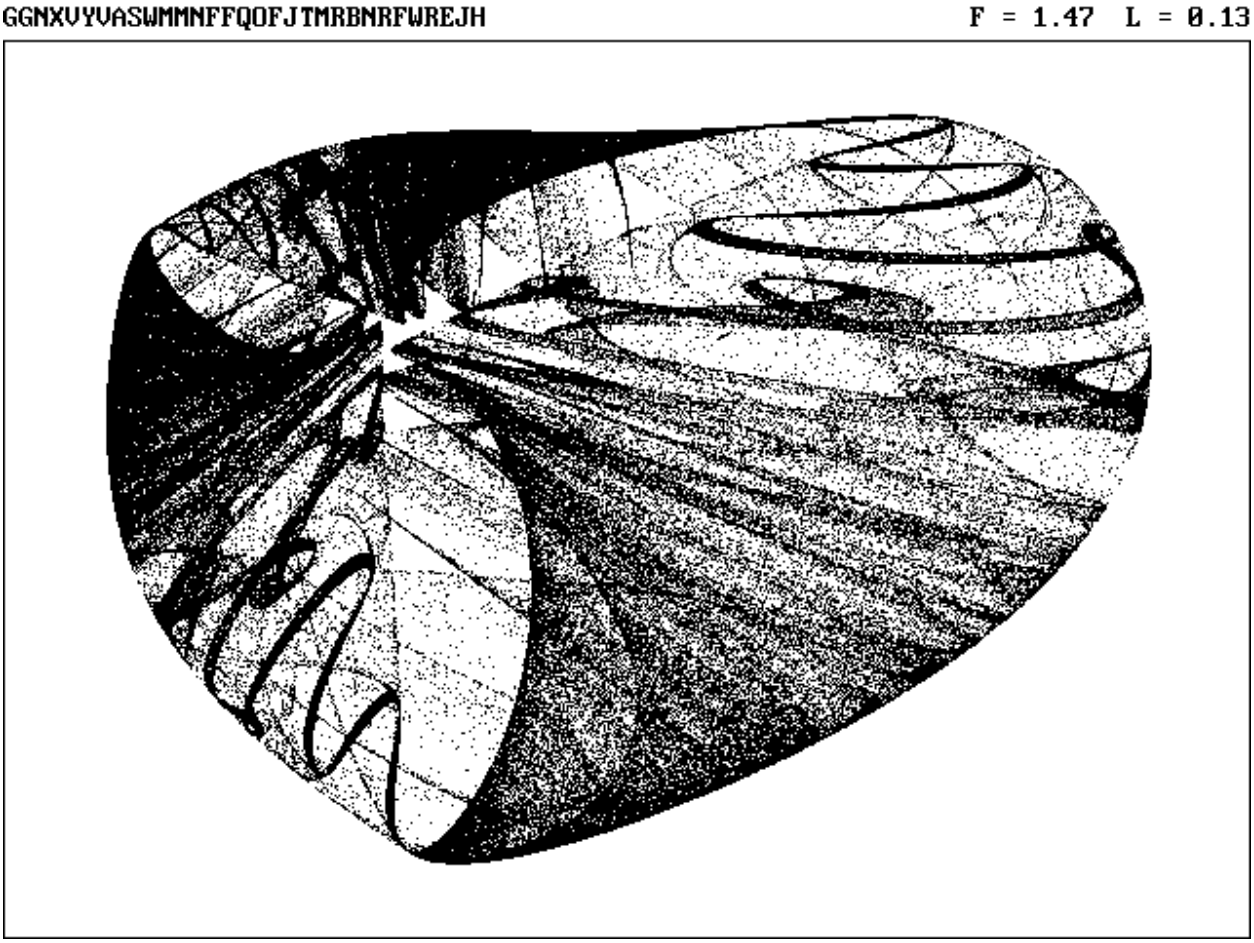


Figure 3-28. Two-dimensional quartic map

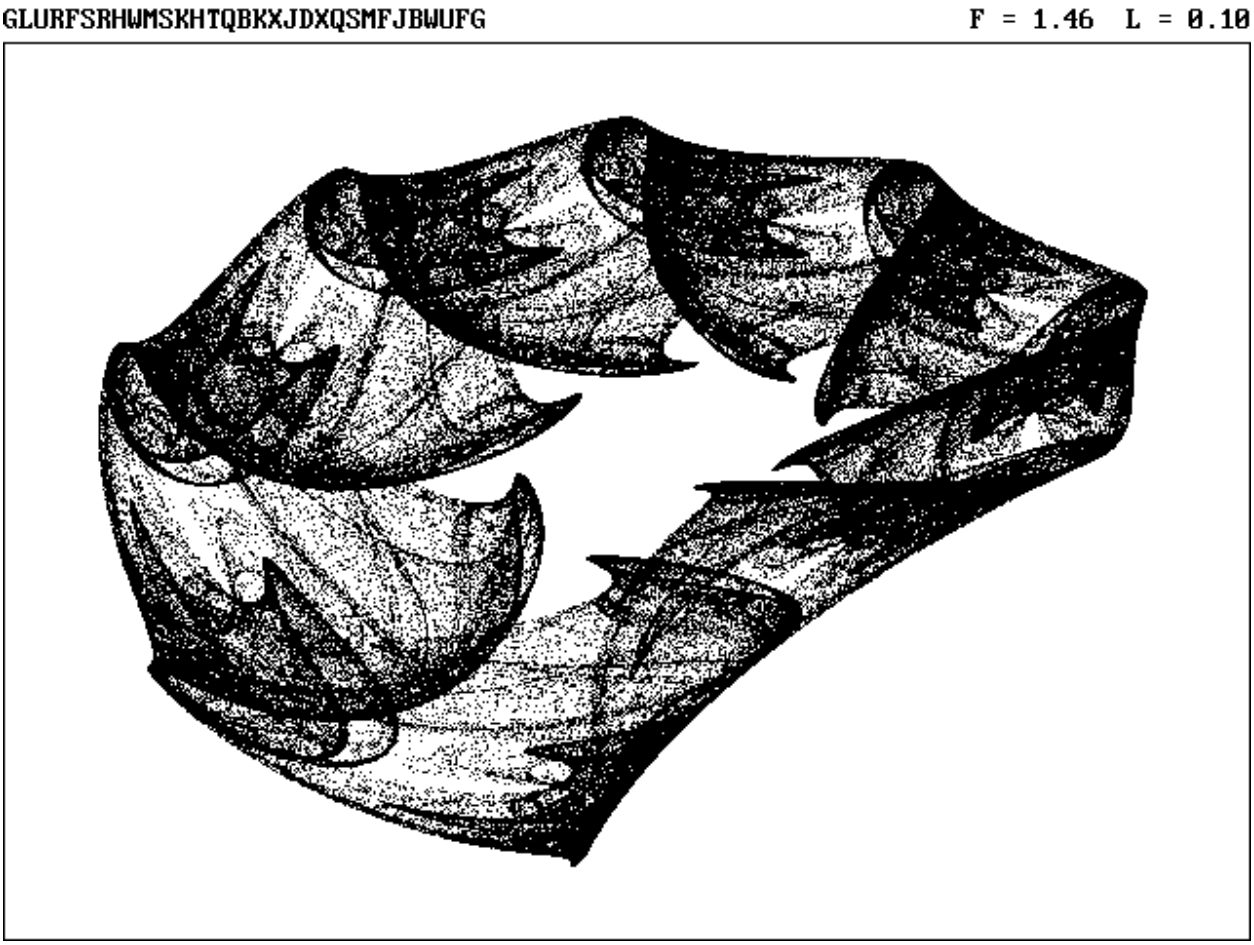


Figure 3-29. Two-dimensional quartic map

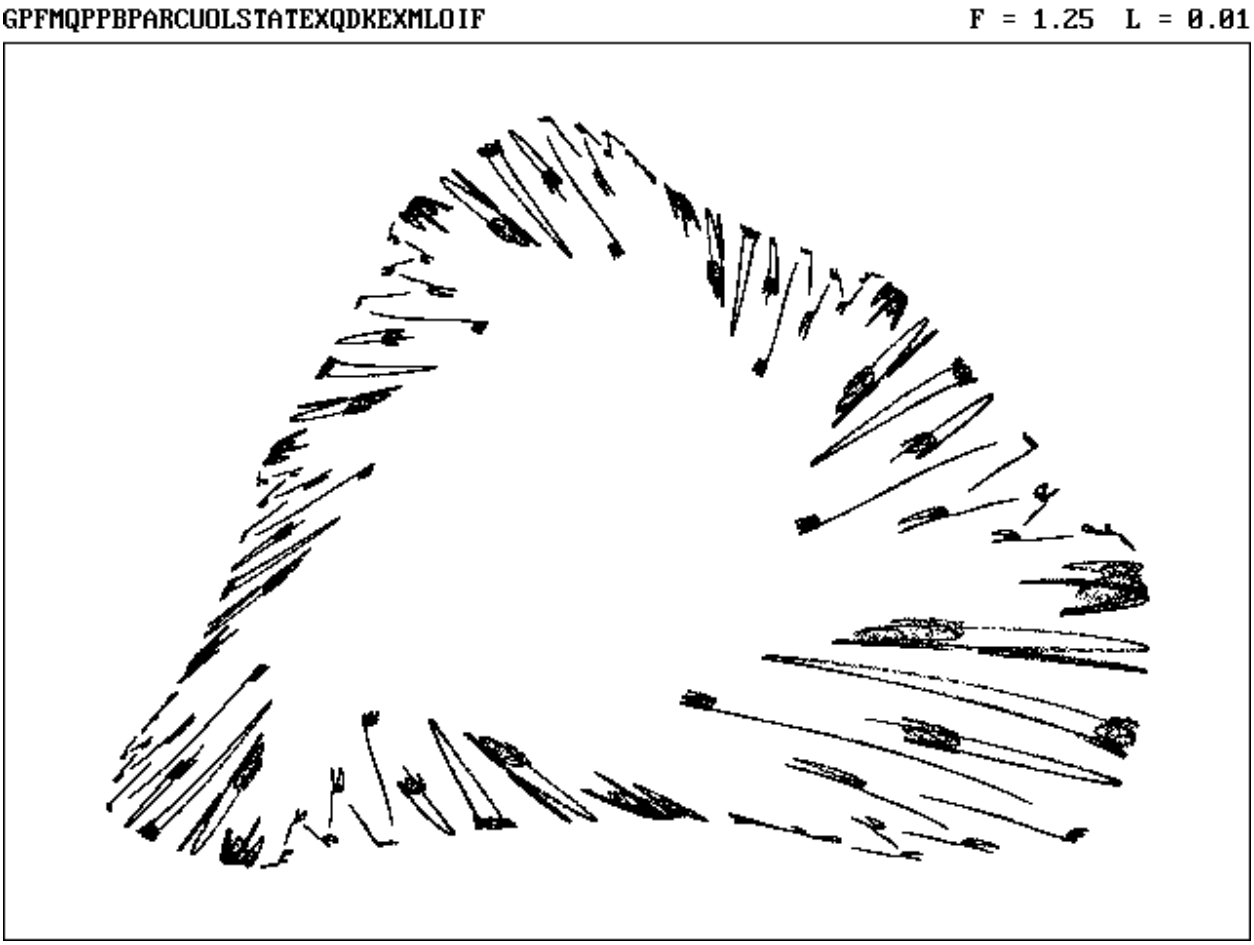


Figure 3-30. Two-dimensional quartic map

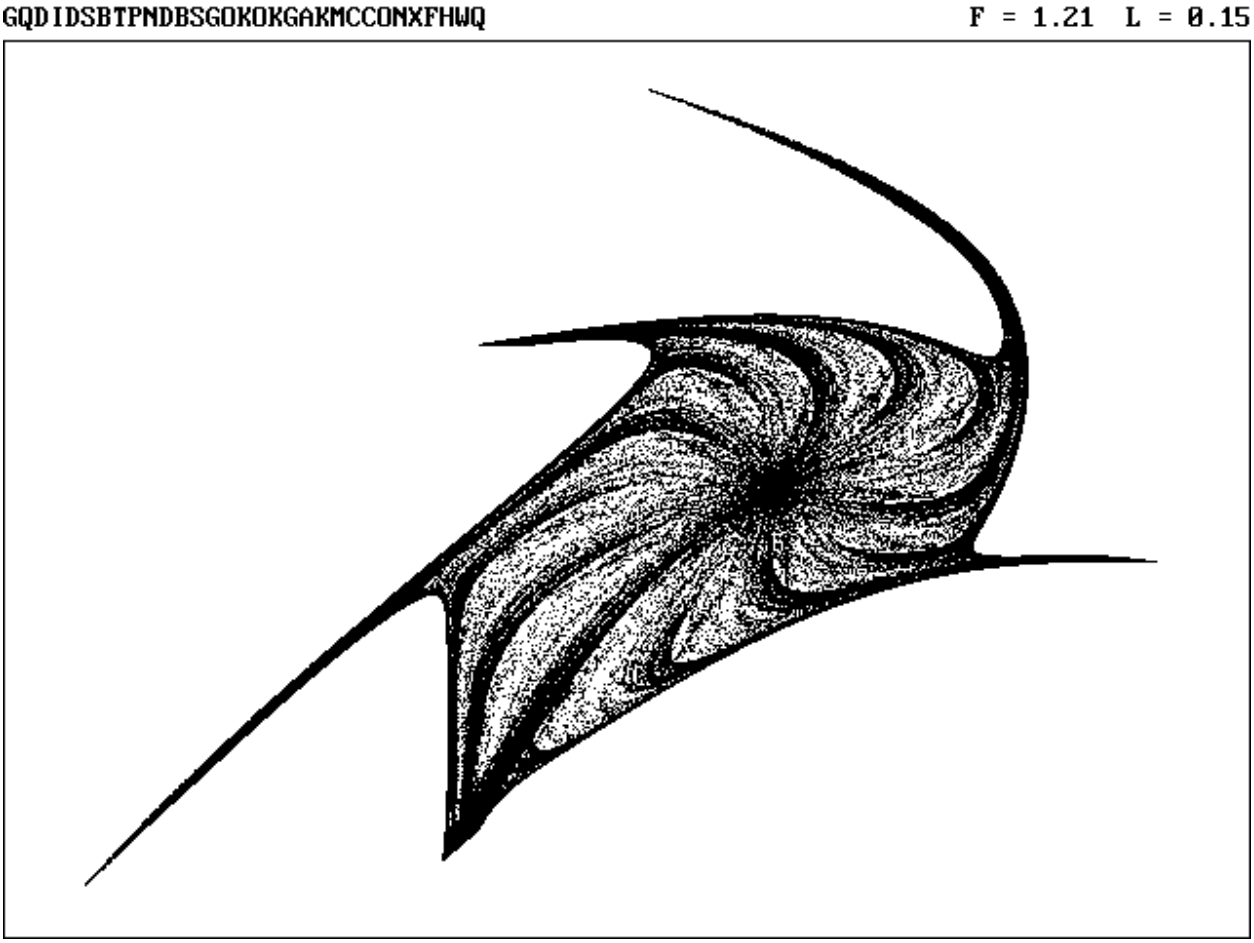


Figure 3-31. Two-dimensional quartic map

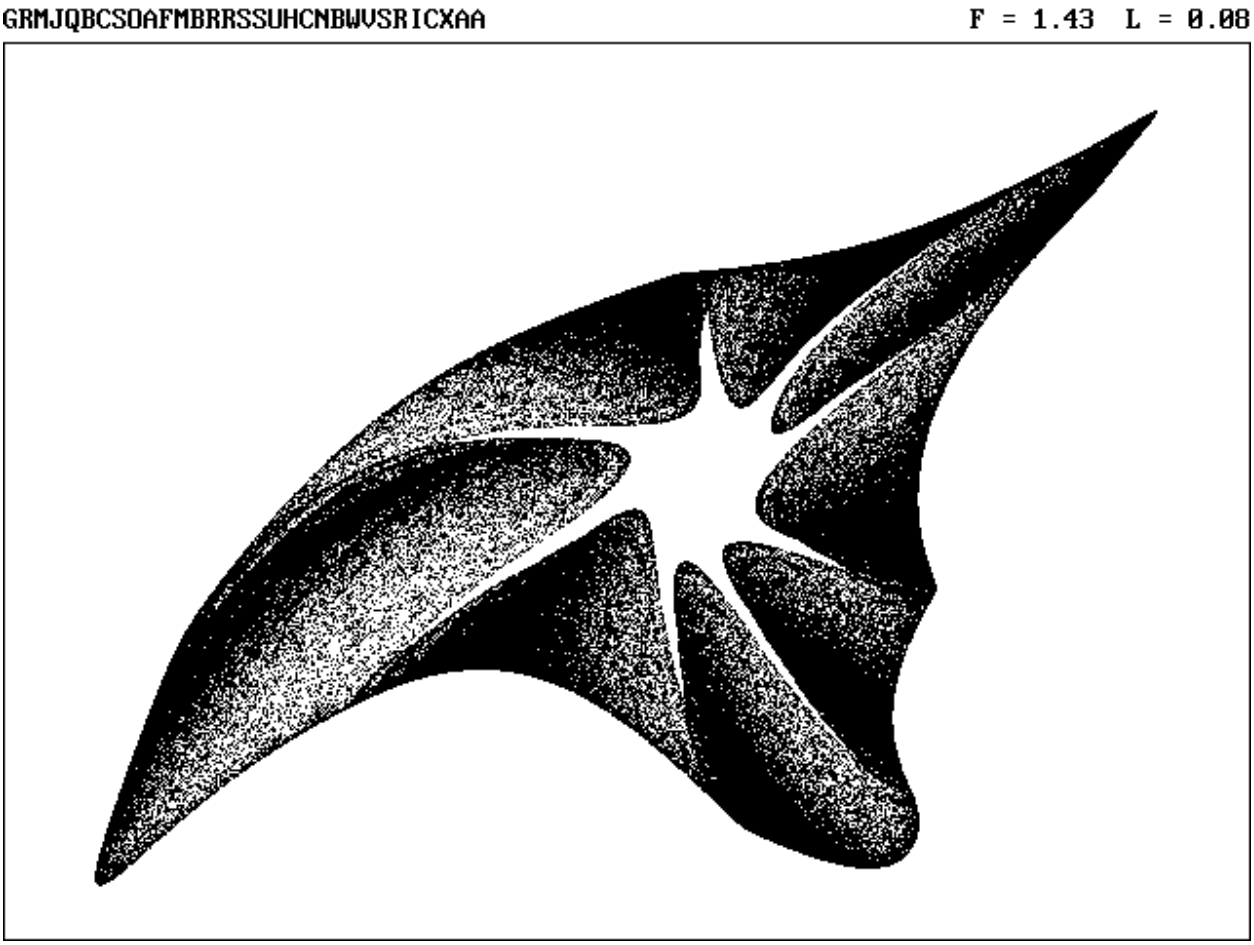


Figure 3-32. Two-dimensional quartic map

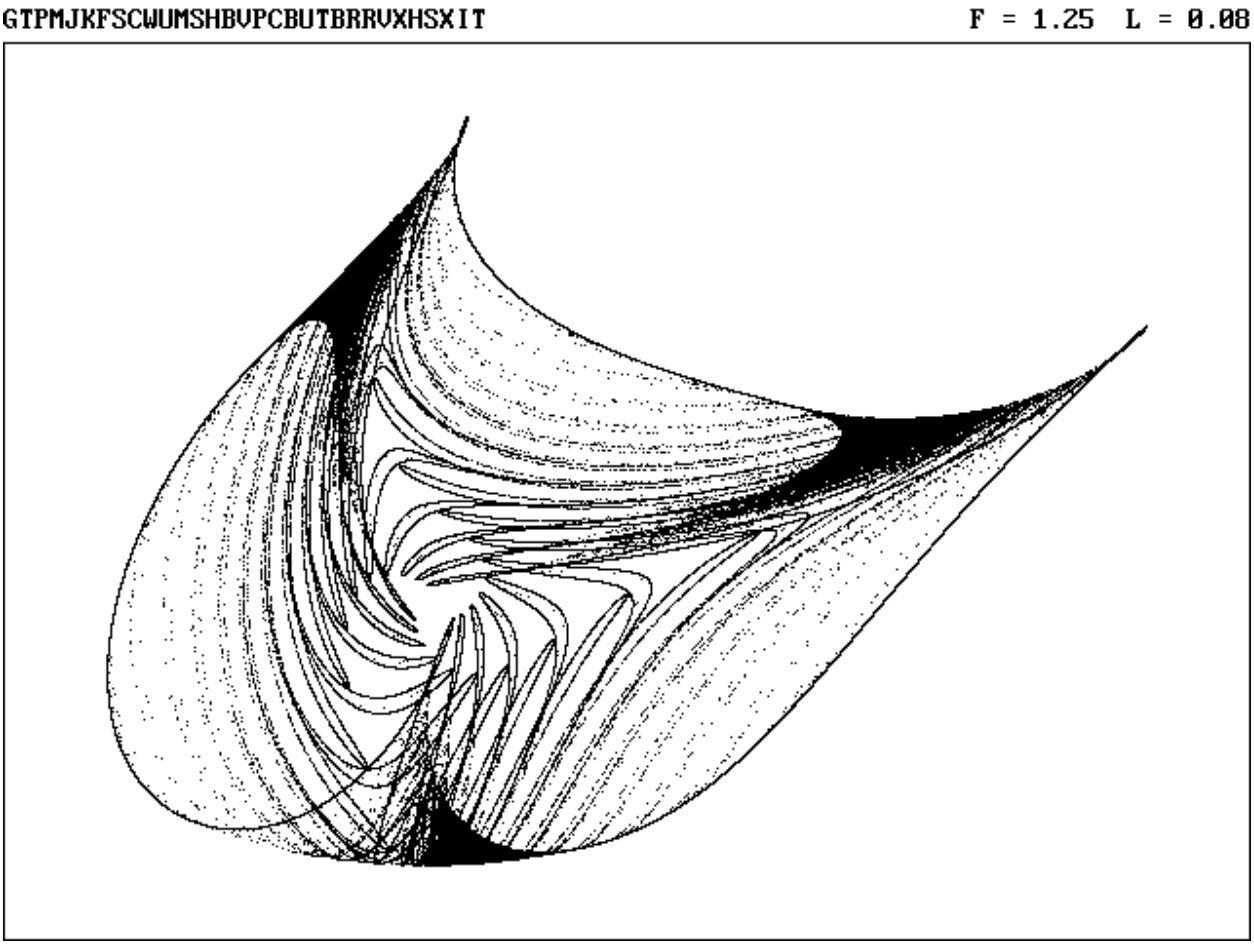


Figure 3-33. Two-dimensional quartic map

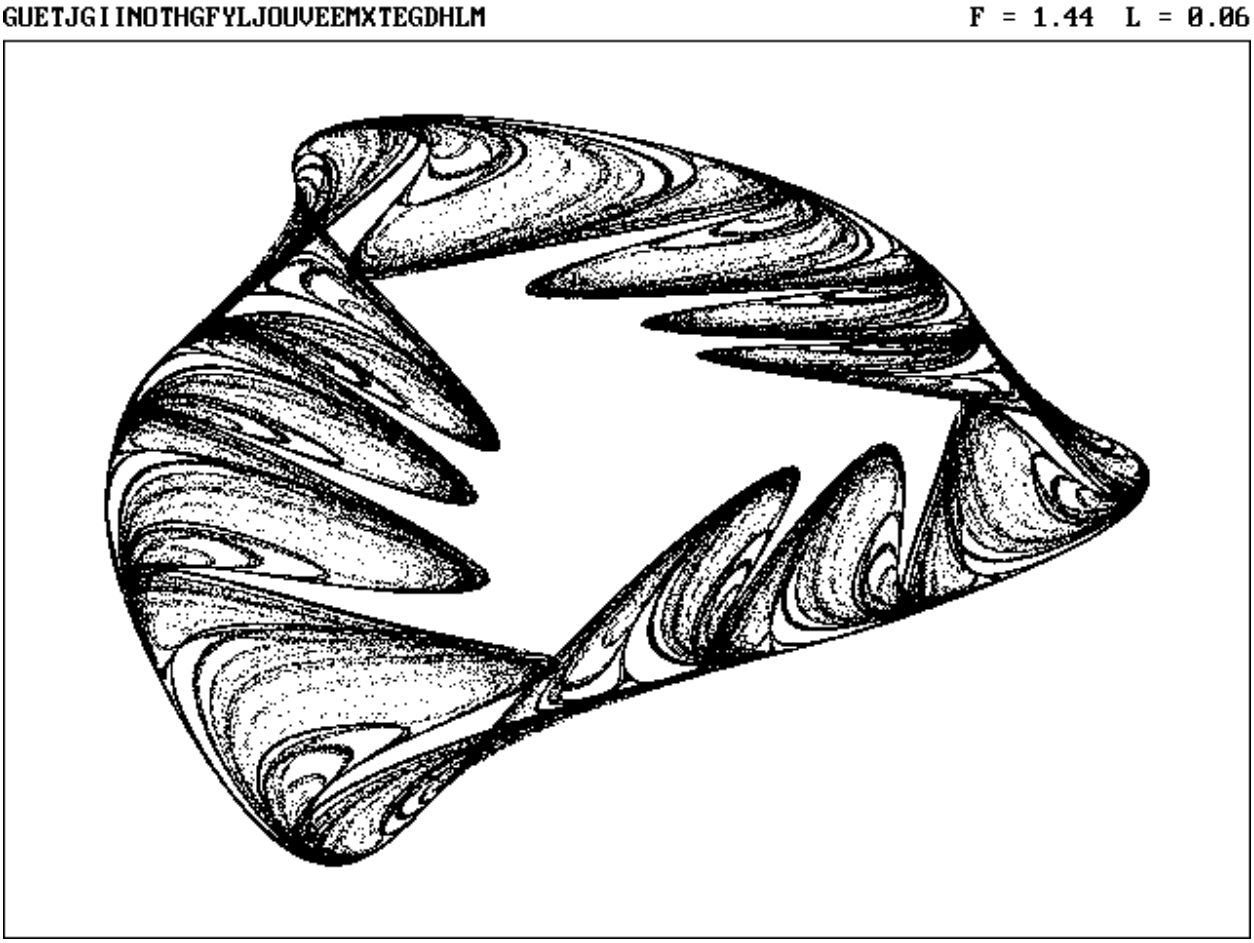


Figure 3-34. Two-dimensional quintic map

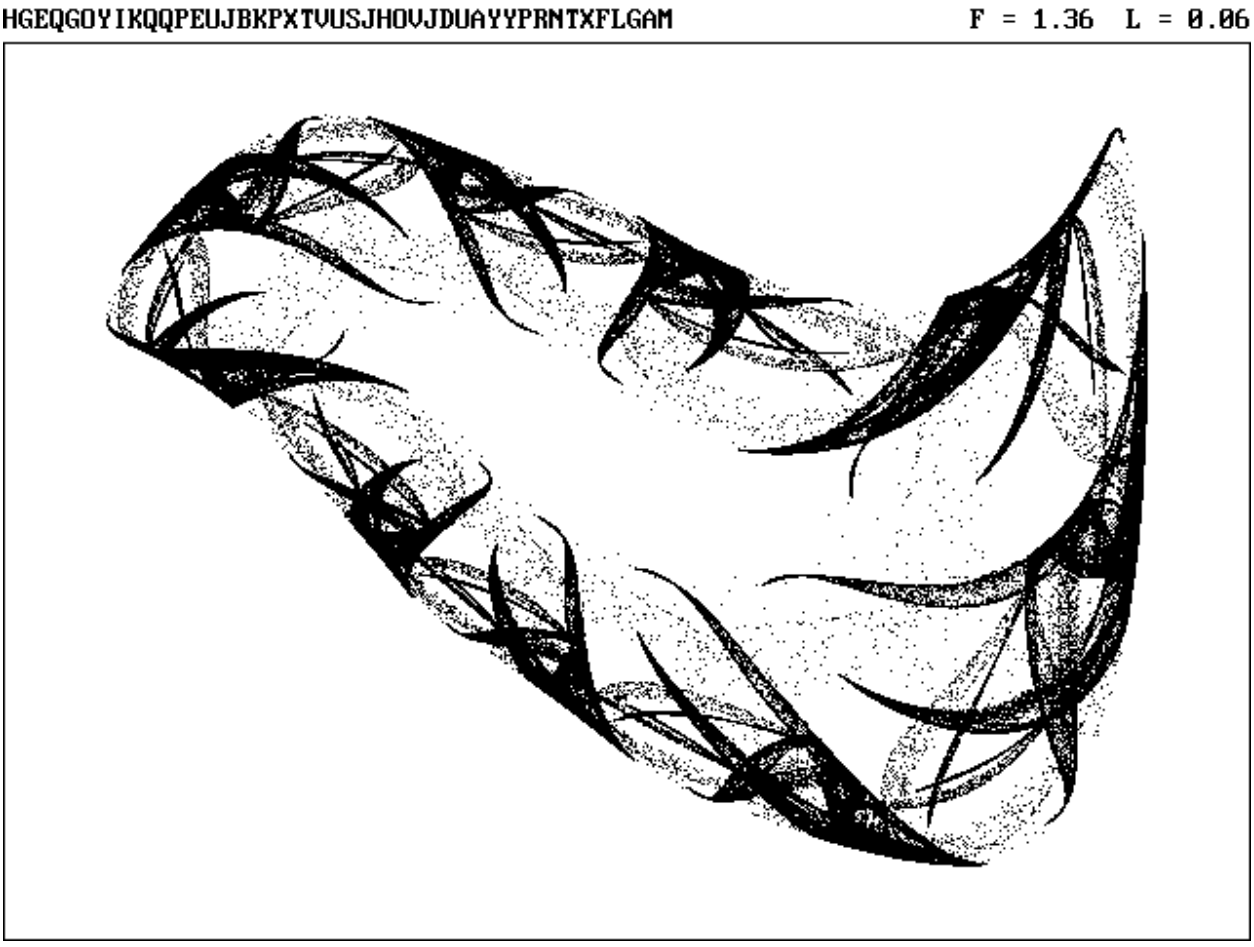


Figure 3-35. Two-dimensional quintic map

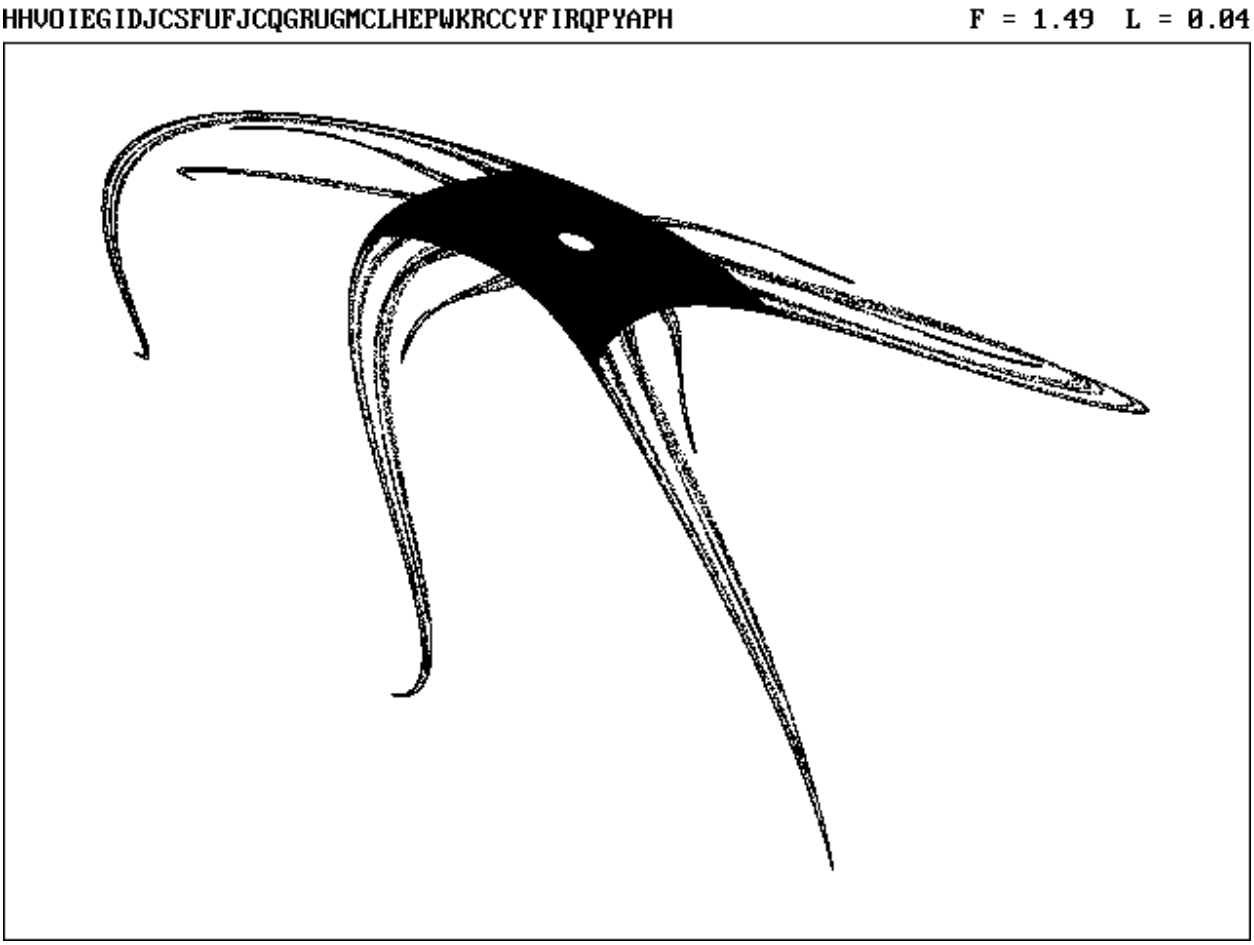


Figure 3-36. Two-dimensional quintic map

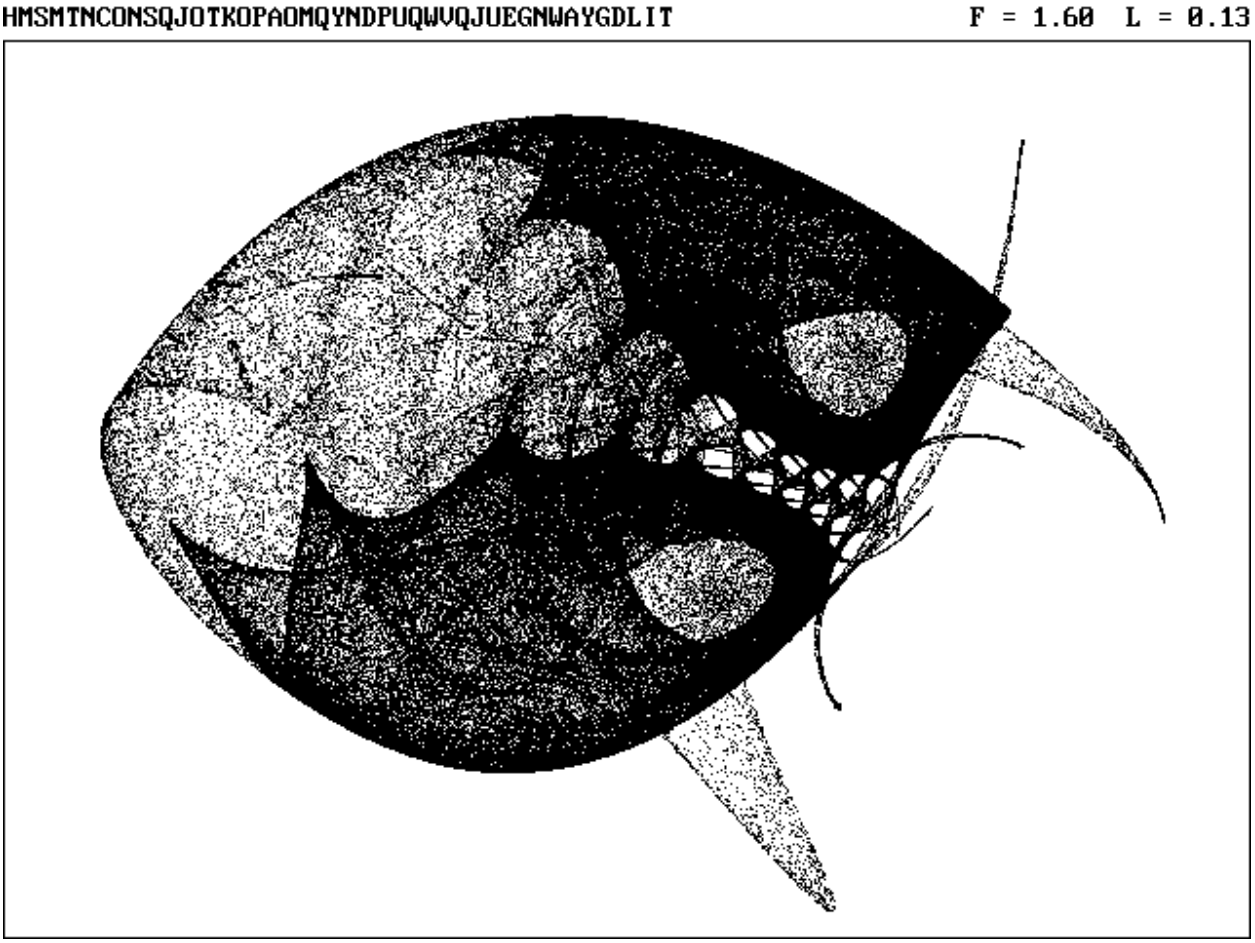


Figure 3-37. Two-dimensional quintic map

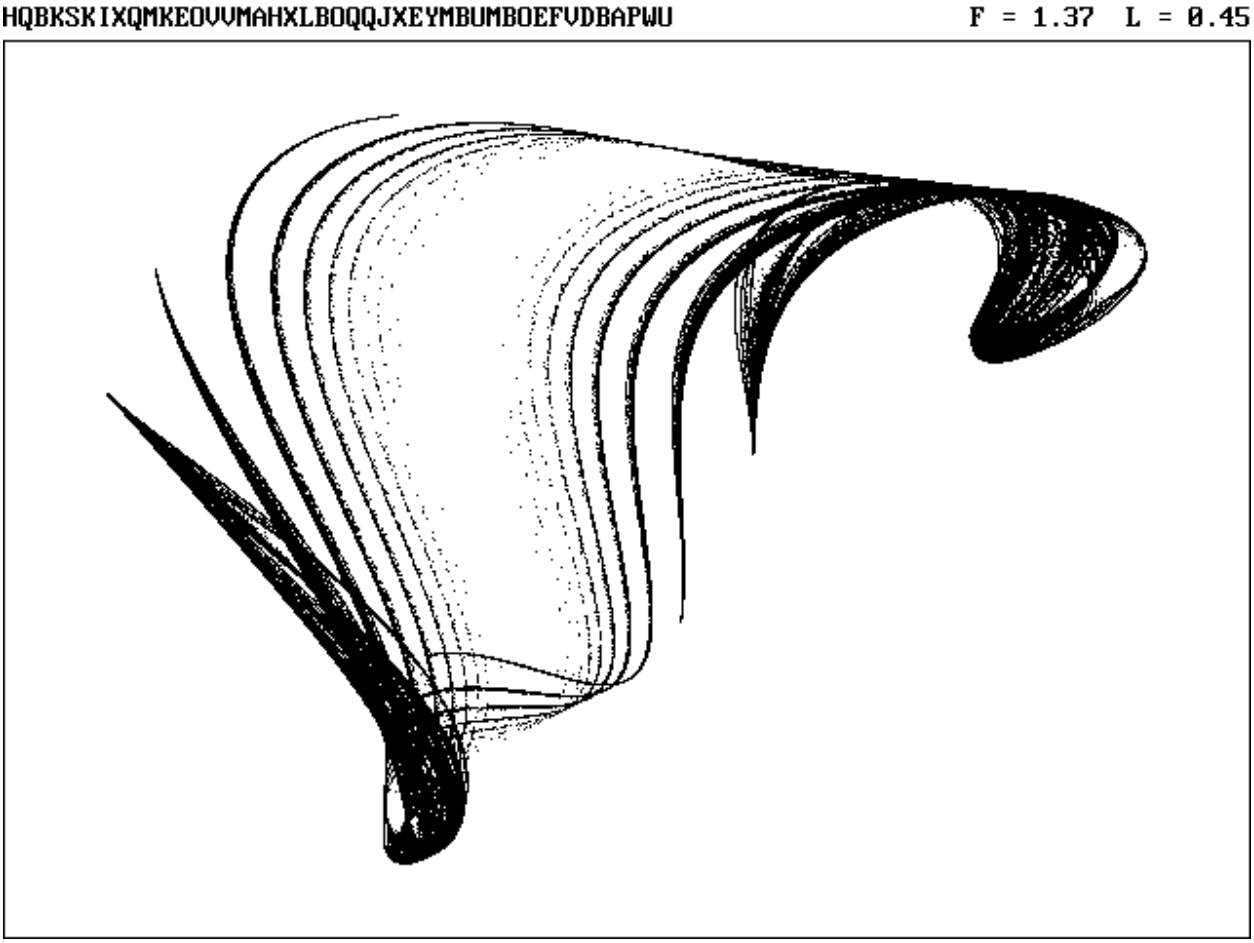


Figure 3-38. Two-dimensional quintic map

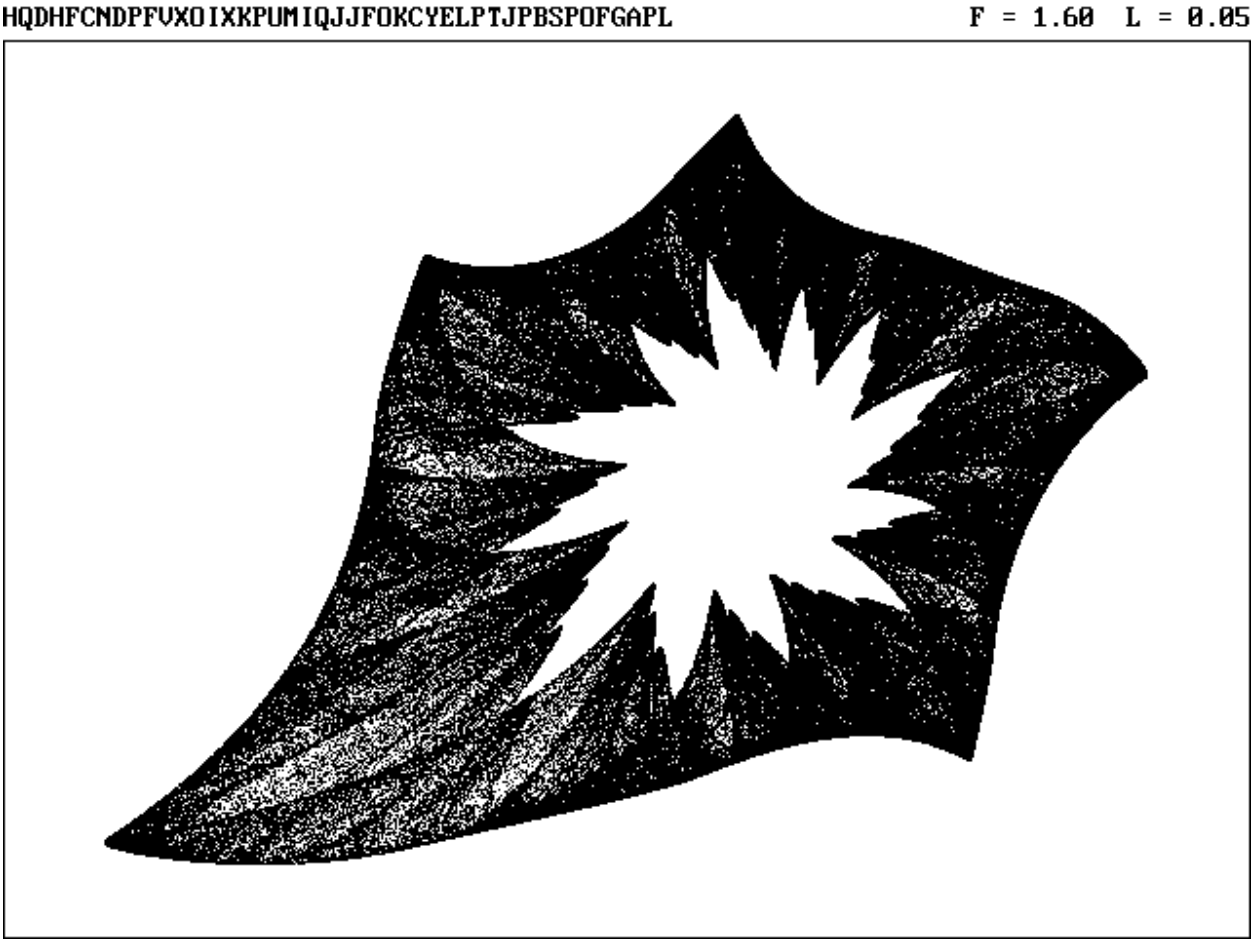


Figure 3-39. Two-dimensional quintic map

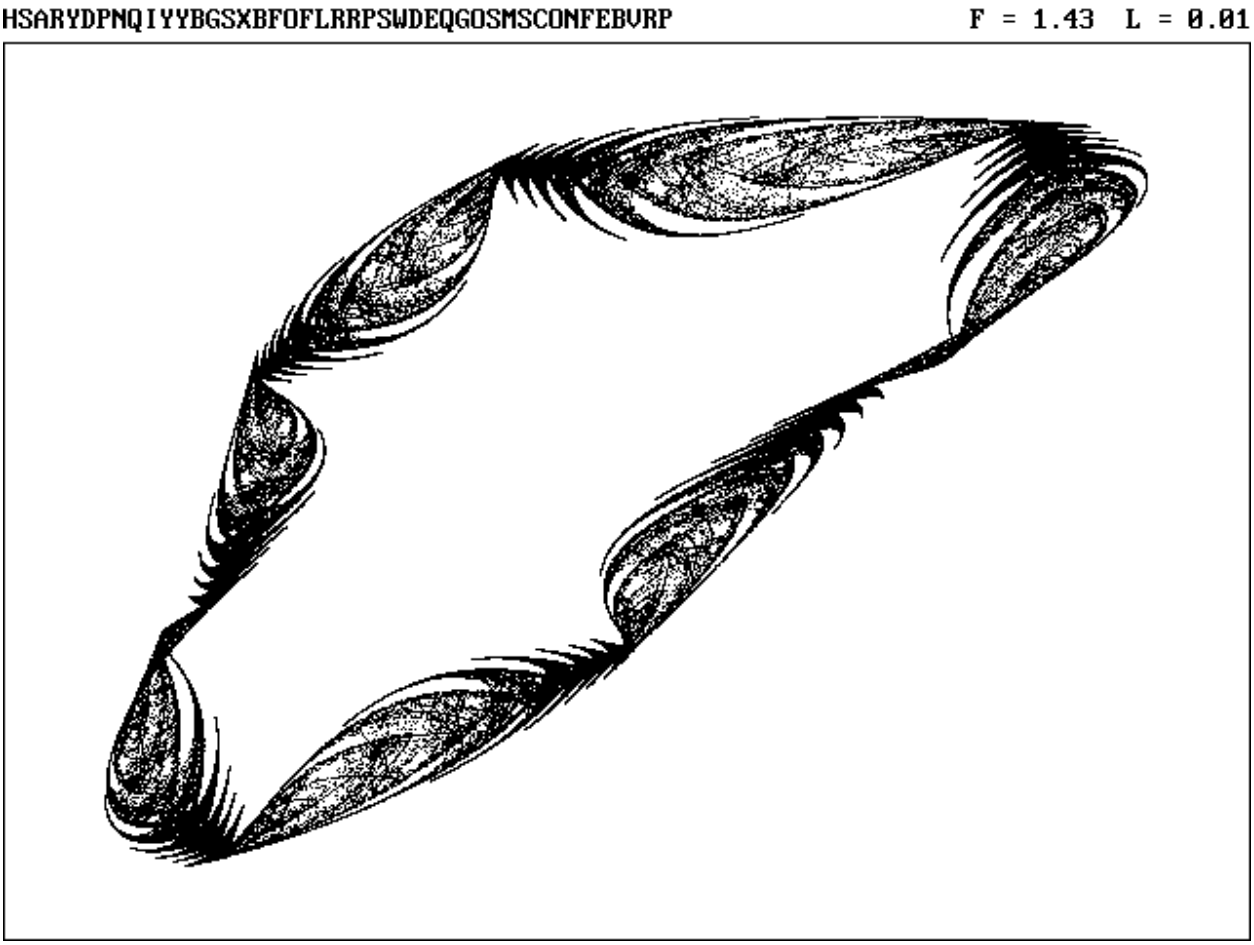


Figure 3-40. Two-dimensional quintic map

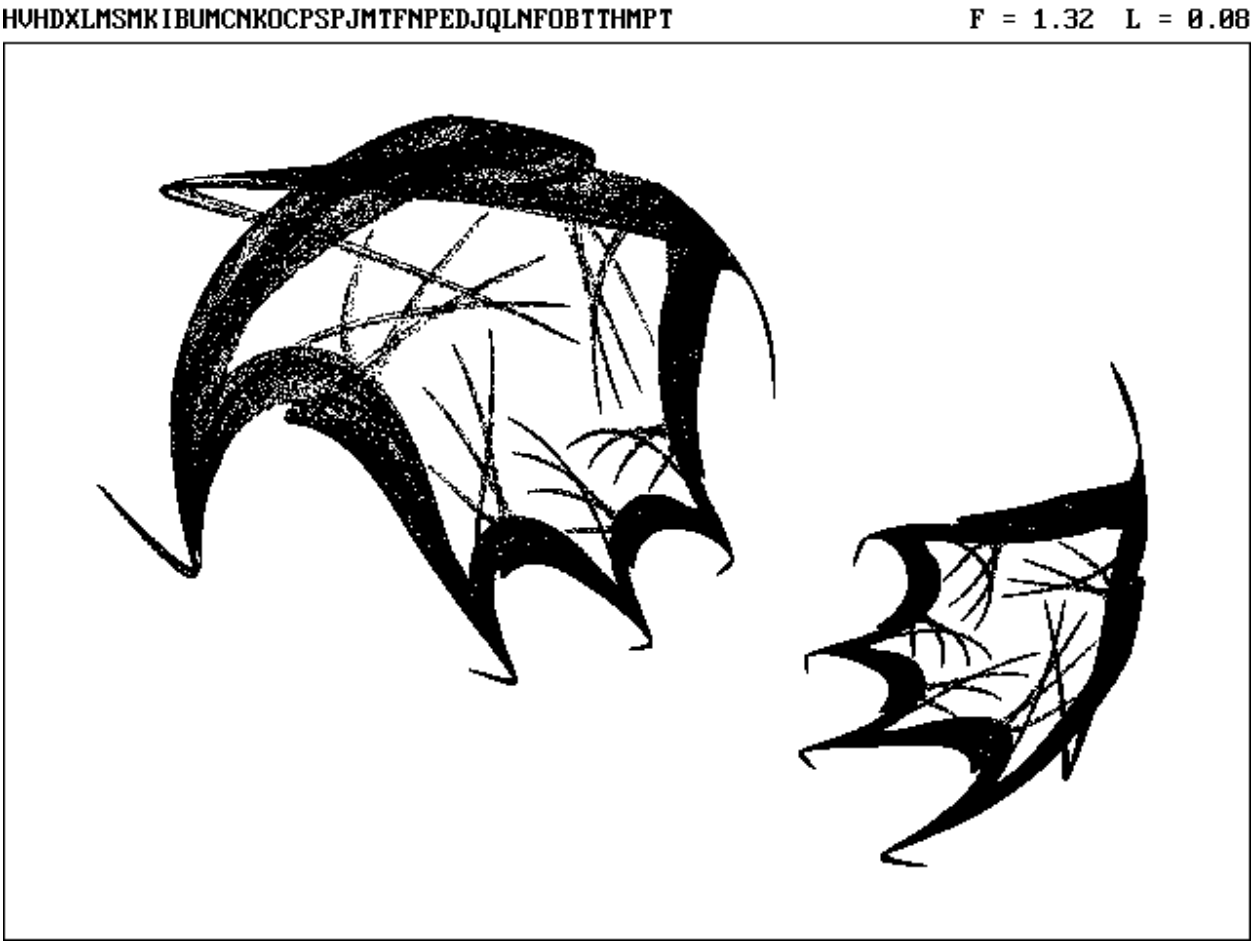
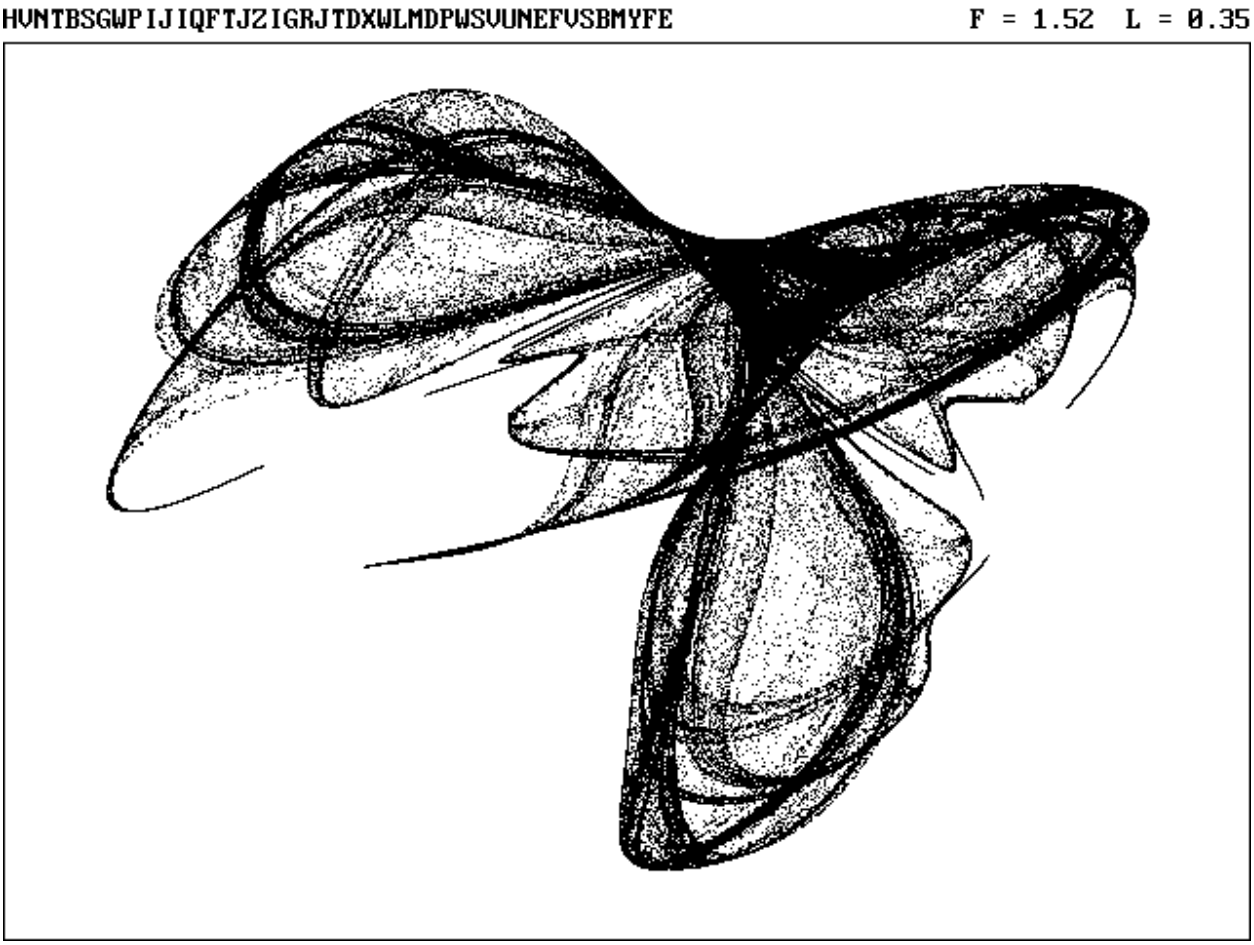


Figure 3-41. Two-dimensional quintic map



Perhaps this is a good point to pause and reiterate in what sense these objects are attractors. If you choose initial values of *X* and *Y* somewhere near the attractor, within its basin of attraction, and substitute these values into the equations that describe the attractor, the new values of *X* and *Y* represent a point in the plane that is closer to the attractor. After a number of iterations, the point works its way to the attractor, and thereafter it moves around on the attractor in some complicated manner, eventually visiting every part of the attractor. The next position can always be simply and accurately predicted from the current position, but the small, inevitable uncertainty in position continually increases so that a long-term prediction is impossible, except to say that the point is somewhere on the attractor. You can think of the attractor as the set of all possible long-term solutions of the equations that produced it.

Besides the error in knowing perfectly the initial conditions, there are also

computer round-off errors at each iteration. Given the extreme sensitivity to small errors, you may wonder whether any computer is capable of calculating correctly such a chaotic process. It is true that if the same chaotic equations are iterated on two computers using different precision or round-off methods, the sequence of iterates is almost certainly completely different after a few dozen iterations. However, the appearance of the attractor is probably the same. In such a case, we say that the solution is *structurally stable* or *robust*. Furthermore, according to the *shadowing lemma*, an appropriate small change in initial conditions produces a chaotic sequence that follows arbitrarily close to the computed one.

Since computers always round the results of calculations to a finite number of digits (or more precisely, *bits*), a limited number of values is allowed. Thus successive iteration of a map always eventually repeats a previously obtained value, whereupon the solution reproduces exactly the same sequence of states as it did before. Strictly speaking, every such solution is periodic, and true chaos cannot be observed with a computer. However, with double-precision floating-point variables, which are normally 64 bits, there are 2^{64} or about 10^{19} possible values. It can be shown that an average periodicity occurs after about the square root of this number of iterations, which is about 3×10^9. Until the number of iterations approaches this value, there is little cause to worry. For maps higher than one dimension, this problem is even less serious because all the variables have to reach a previously existing state at the same time.

It is also interesting to realize that infinitely many periodic solutions are embedded in each attractor. These solutions are called periodic *orbits*. From wherever you start on the attractor, you eventually return to a point arbitrarily close to the starting point. This result is called the *Poincaré recurrence theorem*, after Jules-Henri Poincaré, a French mathematician who a hundred years ago portended the modern era of chaos. Thus by making only a small change in the starting point, it is possible, in principle, to return exactly to the starting point, which implies a periodic orbit with a period equal to the number of iterations required to return. Most of these orbits have very large periods, however.

Every point on the attractor is arbitrarily close to such a periodic orbit, but the chance that a randomly chosen point on the attractor lies on such an orbit is infinitesimal. We say that the periodic orbits are *dense* on the attractor. These orbits, though infinite in number, constitute a Cantor set of measure zero. The periodic orbits are unstable in the sense that if you get just slightly off the orbit, you continue to get farther away with each iteration.

The strange attractors exhibited in this book are examples of *orbital fractals*. They should be distinguished from *escape-time fractals*, which show the basin of

attraction and typically display with color the number of iterations required for points outside the basin to escape beyond some predefined region. The *Mandelbrot* and *Julia* sets are perhaps the best-known escape-time fractals. Escape-time fractals require much longer computing times to develop but provide dazzling displays with exotic fine-scale structures.

3.6 Strange Attractor Planets

The previous figures have obvious beauty, but they generally lack symmetry. Nature mixes symmetry with disorder, and our sense of beauty has developed accordingly. The Earth viewed from outer space is beautiful in part because the irregular features of the clouds and continents are superimposed on a nearly perfect sphere.

There are many ways to do the same with our attractors. Suppose, for example, *X* and *Y* are not the horizontal and vertical positions in a plane but rather the longitude and latitude on the surface of the Earth. The result is an object that might resemble a strange planet with swirling clouds, oceans, canals, craters, and other features.

Note that mapping a plane onto a sphere is a nonlinear transformation. You can't wrap a piece of paper around a globe without a large nonuniform stretching. That's why Greenland looks larger than South America on most flat maps. When a sphere is projected onto a flat computer screen or onto the page of a book, it is stretched so as to magnify the central portion of the attractor and compress the edges.

If θ is the longitude (measured from zero at the right edge) and φ is the latitude (measured from zero at the top), the *X* and *Y* coordinates of the projection of a sphere onto the screen are given by

$$X_p = \cos \_ \sin \_$$

$$Y_p = \cos \_ \qquad \text{(Equation 3G)}$$

We get θ from *X* by a scaling that keeps θ in the range of 0 to π radians (180 degrees), because there is no need to plot points that lie on the back side of the planet. Similarly, we get φ from *Y* by a scaling that keeps φ in the range of 0 (at the North Pole) to π radians (at the South Pole). The program modifications required to accomplish this transformation are given in **PROG09**. This program allows you to

toggle back and forth between the two types of projection by pressing the **P** key.

PROG09. Changes required in PROG08 to project attractor onto a sphere

```
1000 REM TWO-D MAP SEARCH (Projected onto a Sphere)
1110 PJT% = 1                    'Projection is spherical

2260    IF PJT% = 1 THEN GOSUB 4100     'Project onto a sphere

3200 XA = (XL + XH) / 2: YA = (YL + YH) / 2
3310 IF PJT% <> 1 THEN LINE (XL, YL)-(XH, YH), , B
3320 IF PJT% = 1 THEN CIRCLE (XA, YA), .36 * (XH - XL)
3330 TT = 3.1416 / (XMAX - XMIN): PT = 3.1416 / (YMAX - YMIN)

3750 IF Q$ = "P" THEN PJT% = (PJT% + 1) MOD 2: T% = 3: IF N > 999 THEN N = 999

4100 REM Project onto a sphere
4110 TH = TT * (XMAX - XP)
4120 PH = PT * (YMAX - YP)
4130 XP = XA + .36 * (XH - XL) * COS(TH) * SIN(PH)
4140 YP = YA + .5 * (YH - YL) * COS(PH)
4150 RETURN
```

Figures 3-42 through 3-57 show some examples of two-dimensional attractors

projected onto a sphere. Note that the features on the attractors tend to converge at the poles at the tops and bottoms of the figures. This convergence could be suppressed by using an area-preserving transformation that stretches the *Y* values near the poles by the same factor that the *X* values are compressed. The simplest way to produce this effect is to delete line 4140.

Figure 3-42. Two-dimensional quadratic map projected onto a sphere

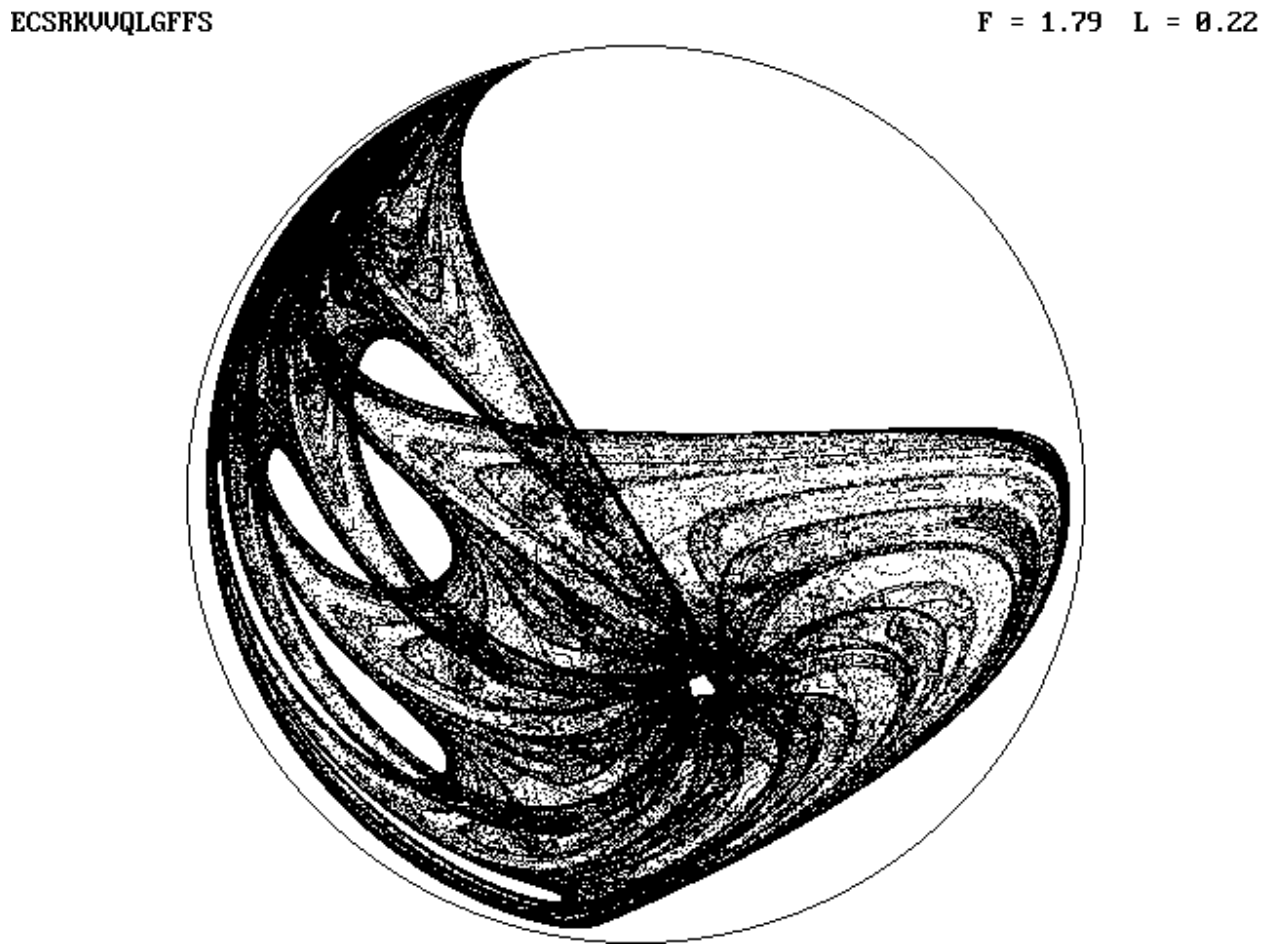


Figure 3-43. Two-dimensional quadratic map projected onto a sphere

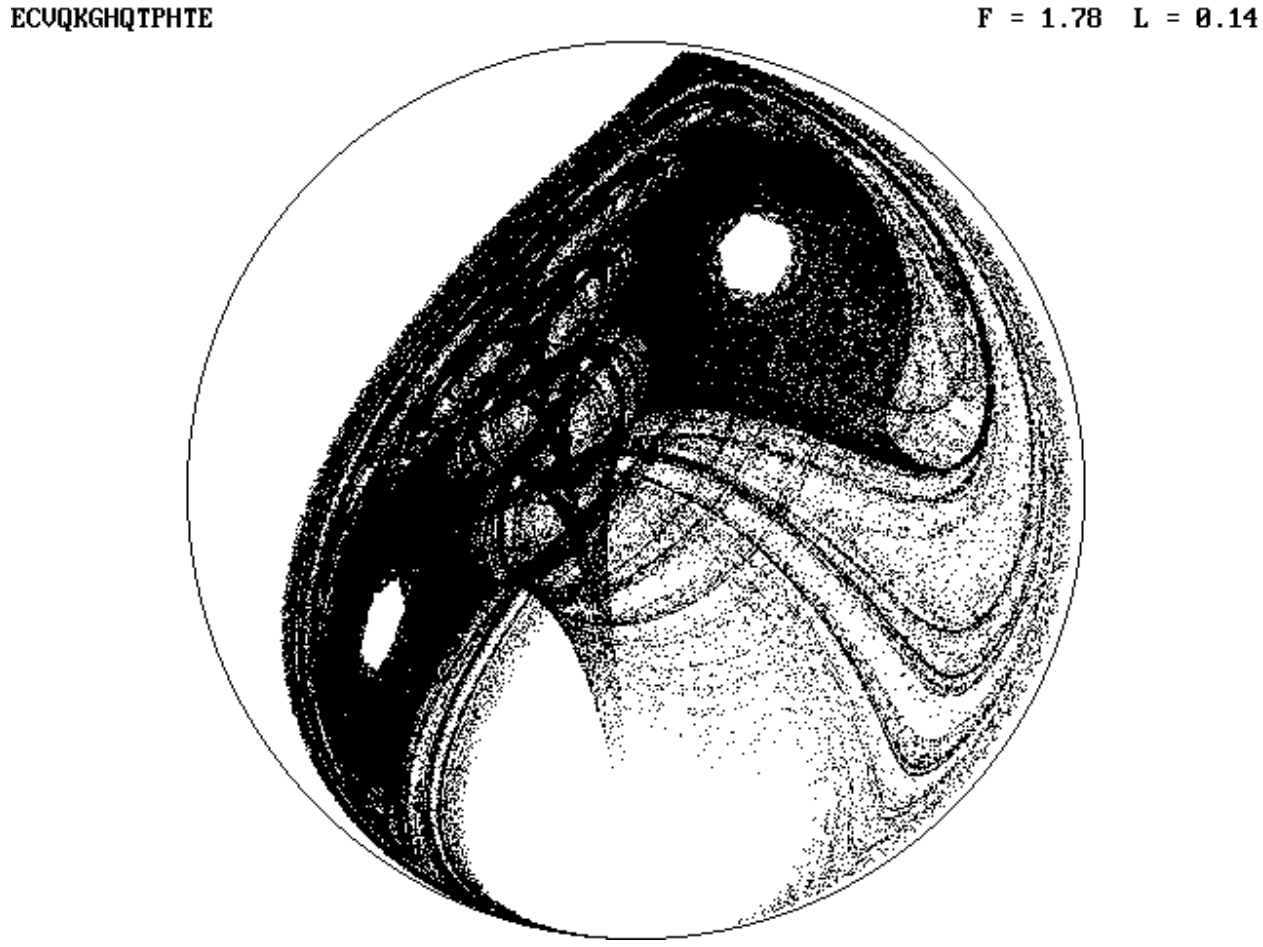


Figure 3-44. Two-dimensional quadratic map projected onto a sphere

Figure 3-45. Two-dimensional quadratic map projected onto a sphere

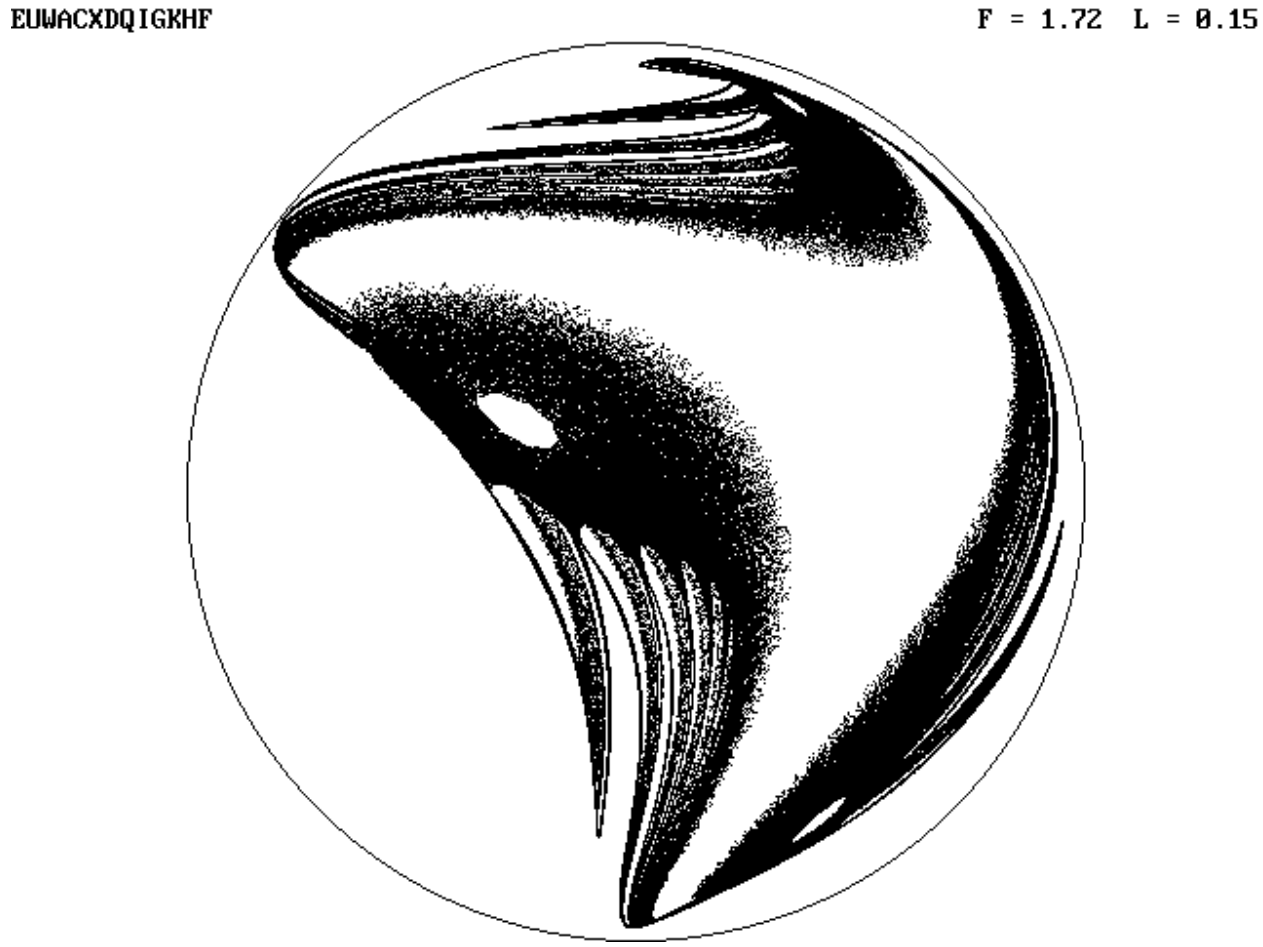


Figure 3-46. Two-dimensional cubic map projected onto a sphere

Figure 3-47. Two-dimensional cubic map projected onto a sphere

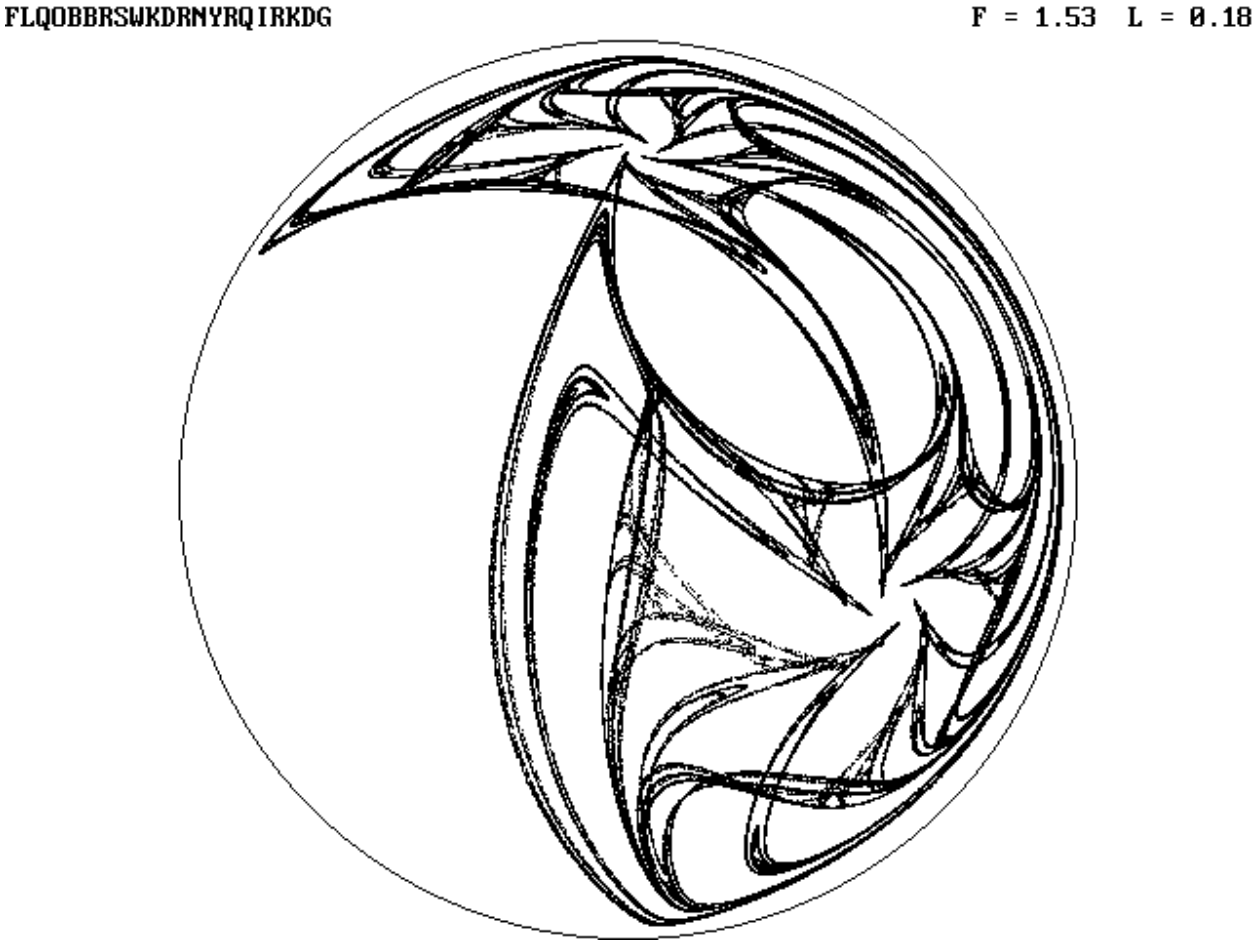


Figure 3-48. Two-dimensional cubic map projected onto a sphere

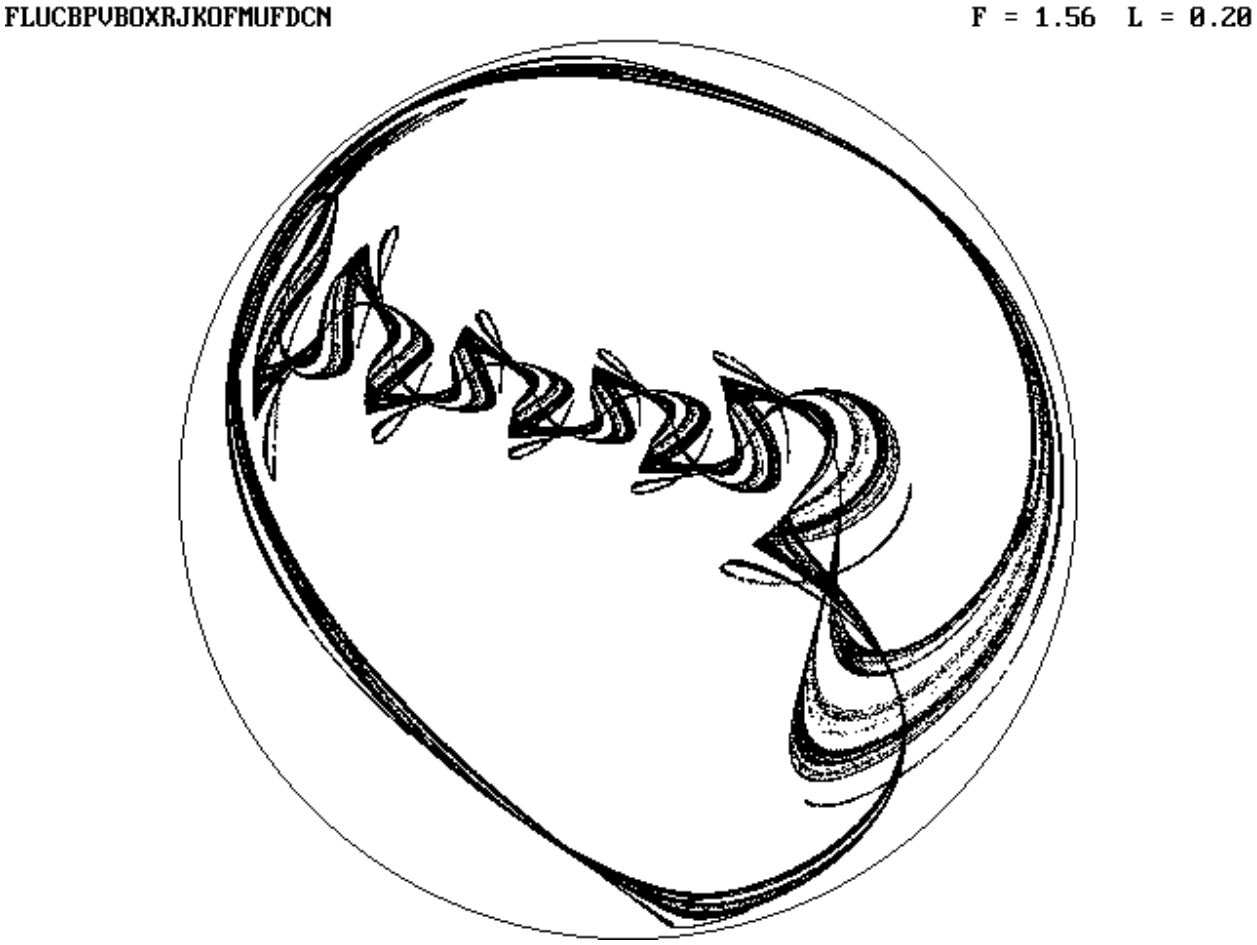


Figure 3-49. Two-dimensional cubic map projected onto a sphere

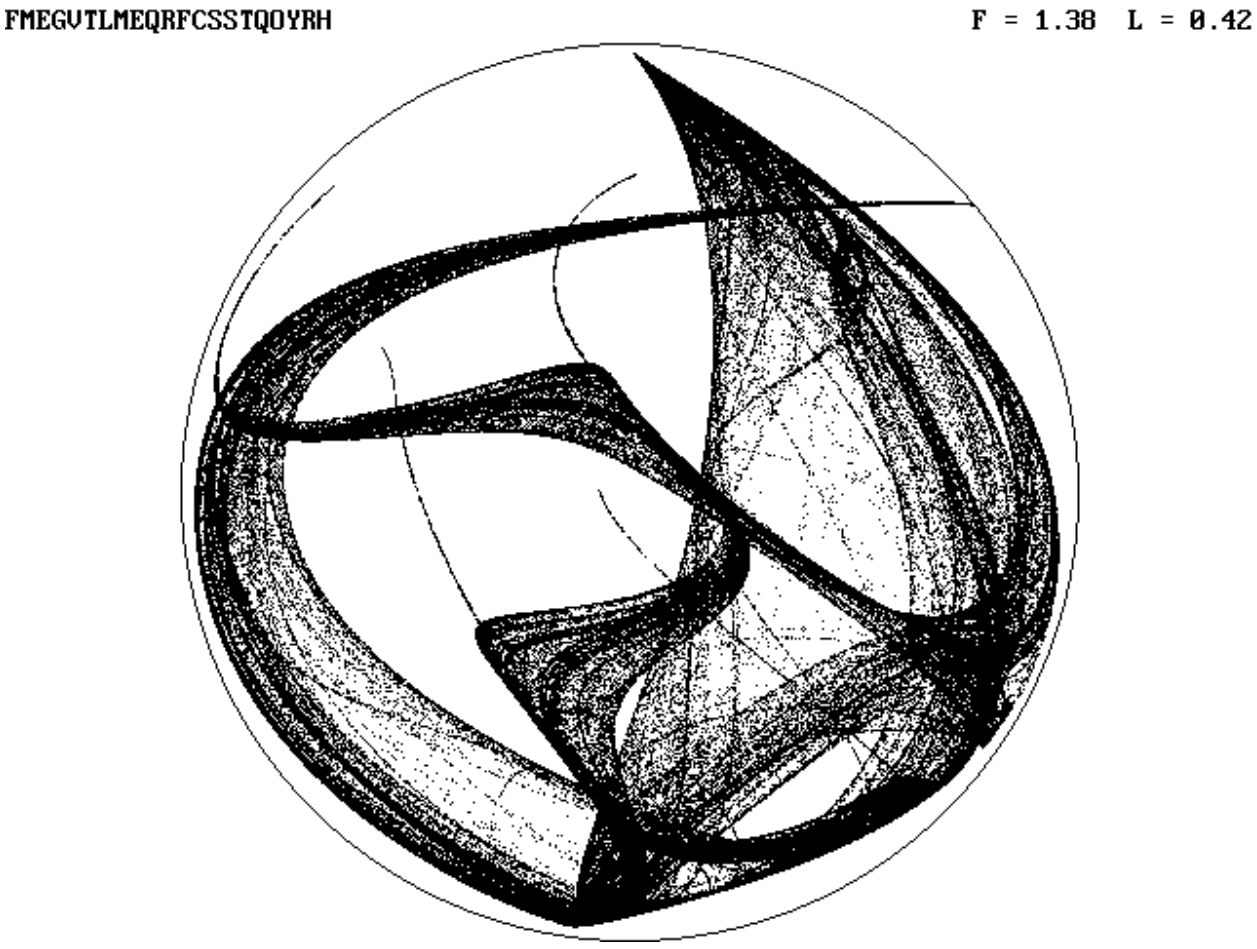


Figure 3-50. Two-dimensional quartic map projected onto a sphere

Figure 3-51. Two-dimensional quartic map projected onto a sphere

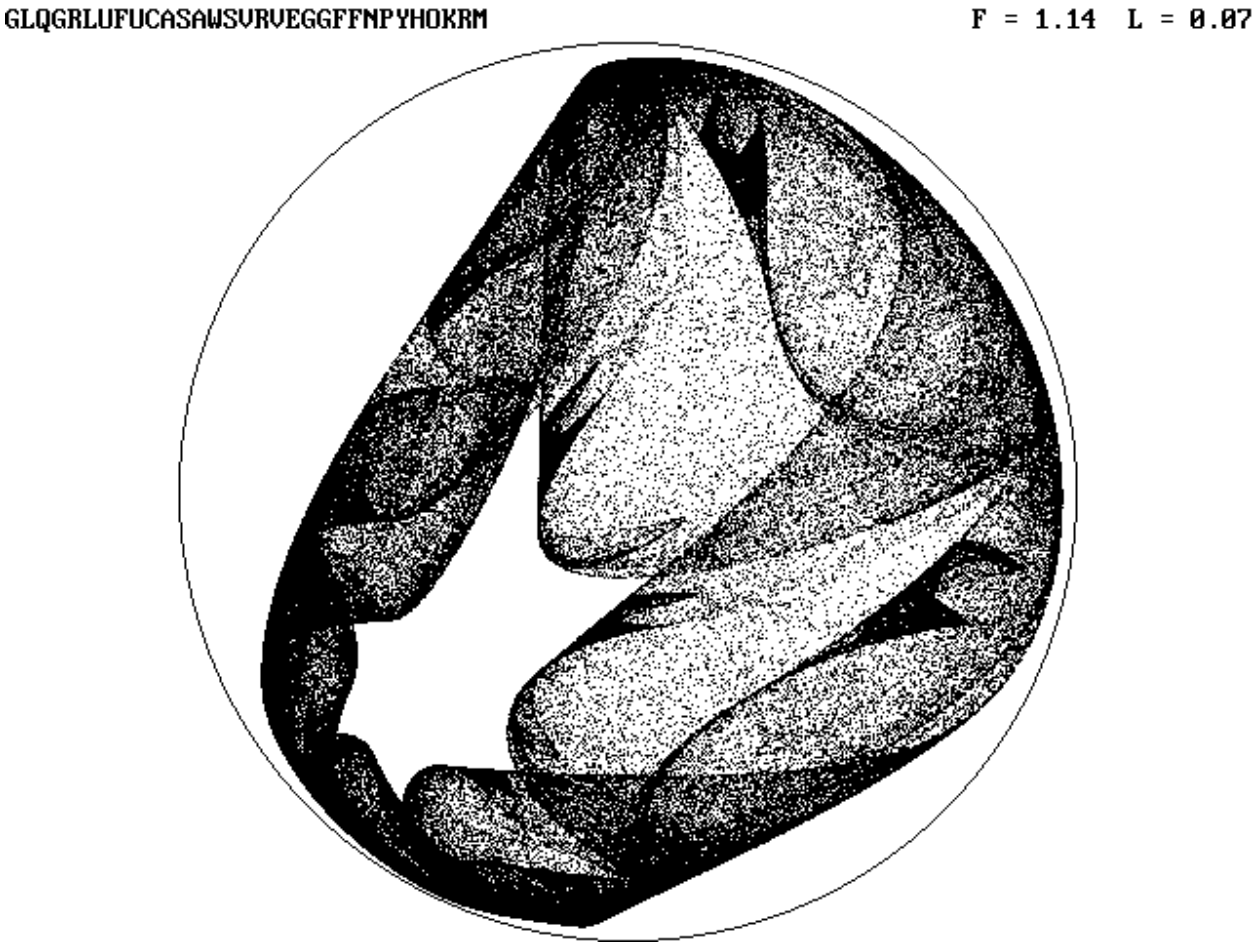


Figure 3-52. Two-dimensional quartic map projected onto a sphere

Figure 3-53. Two-dimensional quartic map projected onto a sphere

Figure 3-54. Two-dimensional quintic map projected onto a sphere

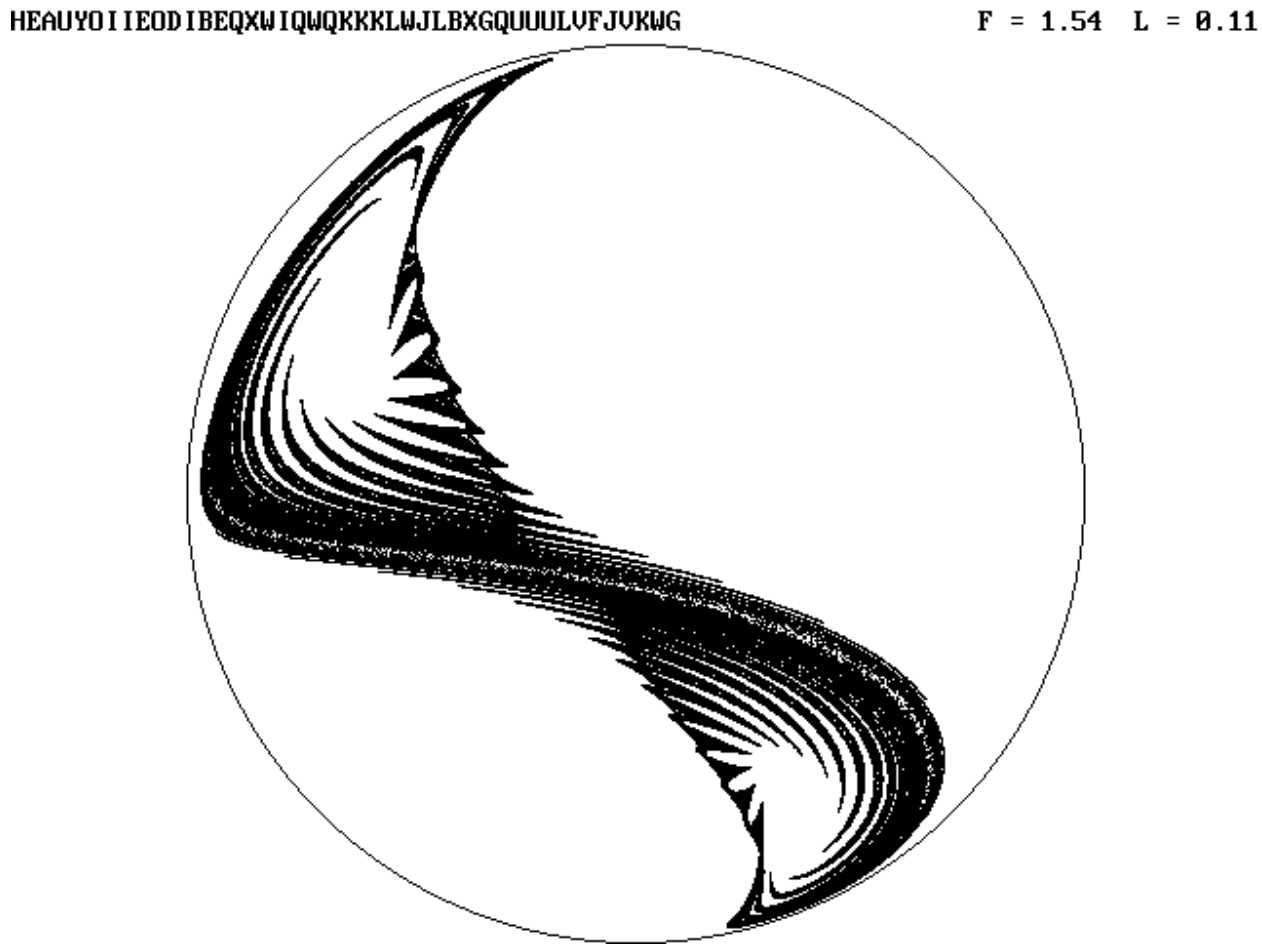
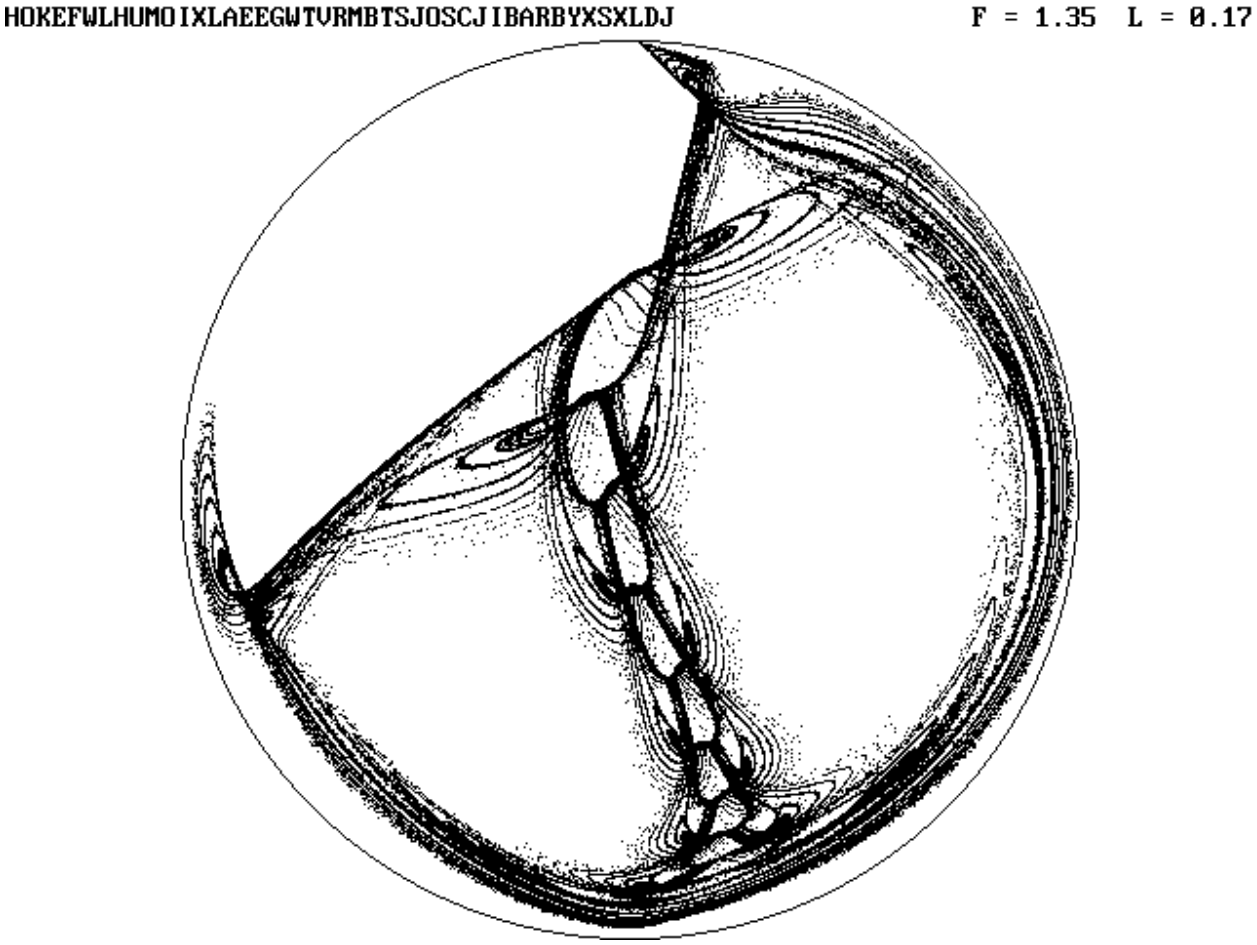


Figure 3-55. Two-dimensional quintic map projected onto a sphere

Figure 3-56. Two-dimensional quintic map projected onto a sphere

Figure 3-57. Two-dimensional quintic map projected onto a sphere



If you are using PowerBASIC or its predecessor, Turbo BASIC, and VGA graphics, you will notice a slight incompatibility with the CIRCLE command that causes the size of the circle that surrounds the attractor to vary from case to case. In these dialects of BASIC, the radius of the circle in SCREEN modes 11 and 12 is specified in units of the screen height rather than its width. If you encounter this problem, try replacing .36 * (*XH - XL*) in line 3320 with .5 * (*YH - YL*).

Planes and spheres are not the only two-dimensional surfaces onto which attractors can be projected. A cylinder is another possibility. The cylinder can be oriented with its axis either horizontal or vertical or tilted at some arbitrary angle. A torus is another possibility. You may be able to think of other more exotic surfaces onto which the attractors can be projected.

3.7 Designer Plaids

It is interesting that all the one-dimensional maps described in the previous chapter are included in the two-dimensional cases. One needs only to set the coefficients of the *Y* equation to zero. For example, a two-dimensional map equivalent to the logistic equation is given by the code EMu%M[9]. However, because *Y* doesn't change with successive iterations, a graph of *Y* versus *X* is simply a straight, horizontal line.

To display the logistic parabola, we need to replace *X* with the next iterate of *X* and *Y* with the second next iterate of *X*. Two successive iterations of a quadratic map requires a fourth-order equation. A code that accomplishes this is GMu%M[13]NHUIM[10].

There are other examples of two-dimensional maps that are really one-dimensional maps in disguise. Suppose X_{n+1} depends only on Y_n and Y_{n+1} depends only on X_n. Then X_{n+2} depends only on X_n, and we have a one-dimensional map for *X* in which *Y* is merely an intermediate value of *X*. The most general fifth-order polynomial example of such a case is

$$X_{n+1} = a_1 + a_{17}Y_n + a_{18}Y_n^2 + a_{19}Y_n^3 + a_{20}Y_n^4 + a_{21}Y_n^5$$

$$Y_{n+1} = a_{22} + a_{23}X_n + a_{24}X_n^2 + a_{25}X_n^3 + a_{26}X_n^4 + a_{27}X_n^5 \quad \text{(Equation 3H)}$$

This case can be achieved by setting the remaining 30 coefficients to zero in **PROG09** by adding the following line after line 2730:

```
2735    IF (I% > 1 AND I% < M% / 2 - O%) OR I% > M% / 2 + O% + 1 THEN MID$(CODE$,
I% + 1, 1) = "M"
```

The result is to produce a 25th-order, one-dimensional polynomial map displayed in two dimensions.

Figures 3-58 through 3-61 show sample attractors obtained in this way. Notice that they fill in rectangular regions resembling a plaid tartan, in sharp contrast to all the previous cases. These attractors are especially appropriate for projecting onto spheres because the features line up east-west along parallels and north-south along meridians. Figures 3-62 and 3-63 show some examples of plaid planetary attractors.

Figure 3-58. Two-dimensional quadratic plaid map

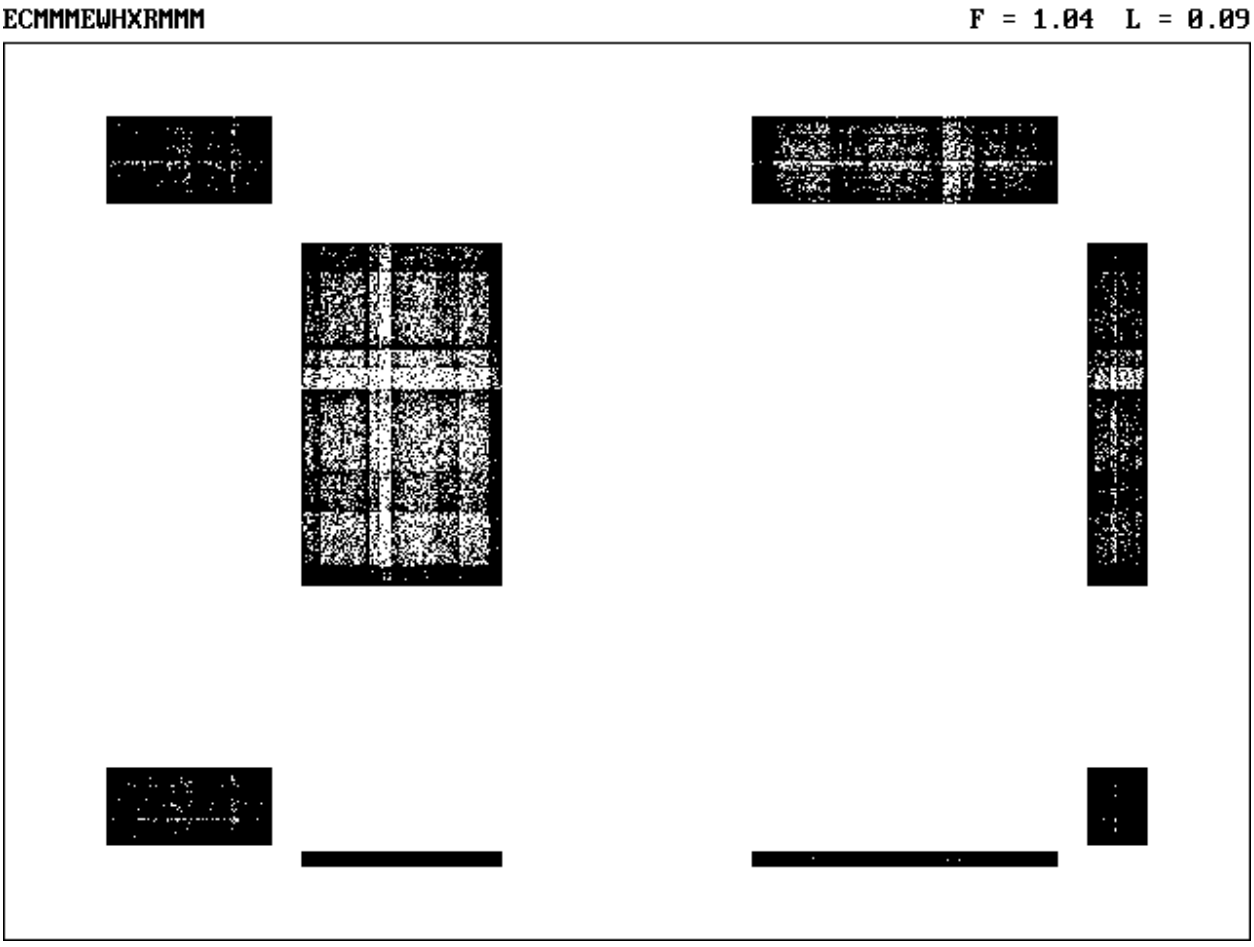


Figure 3-59. Two-dimensional cubic plaid map

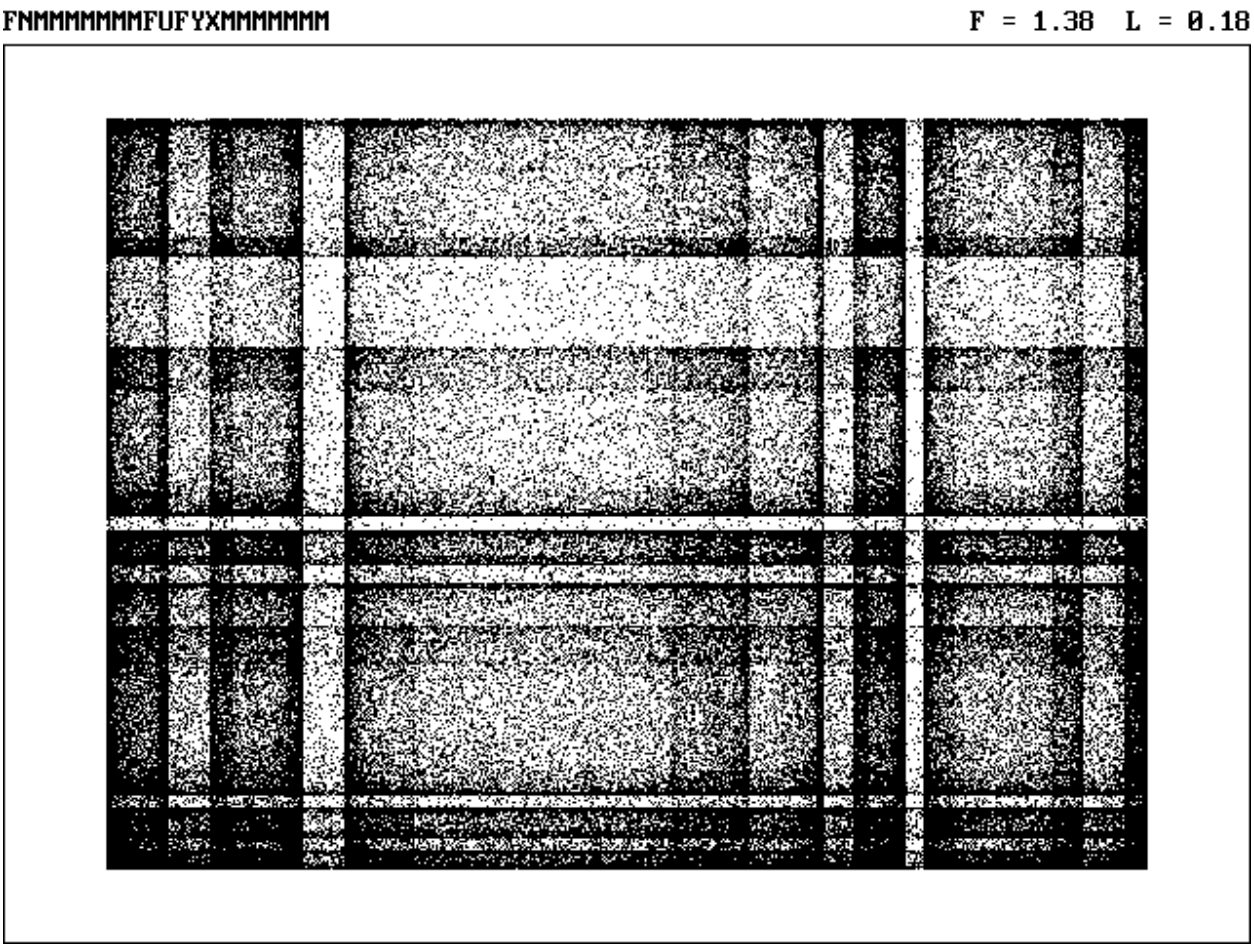


Figure 3-60. Two-dimensional quartic plaid map

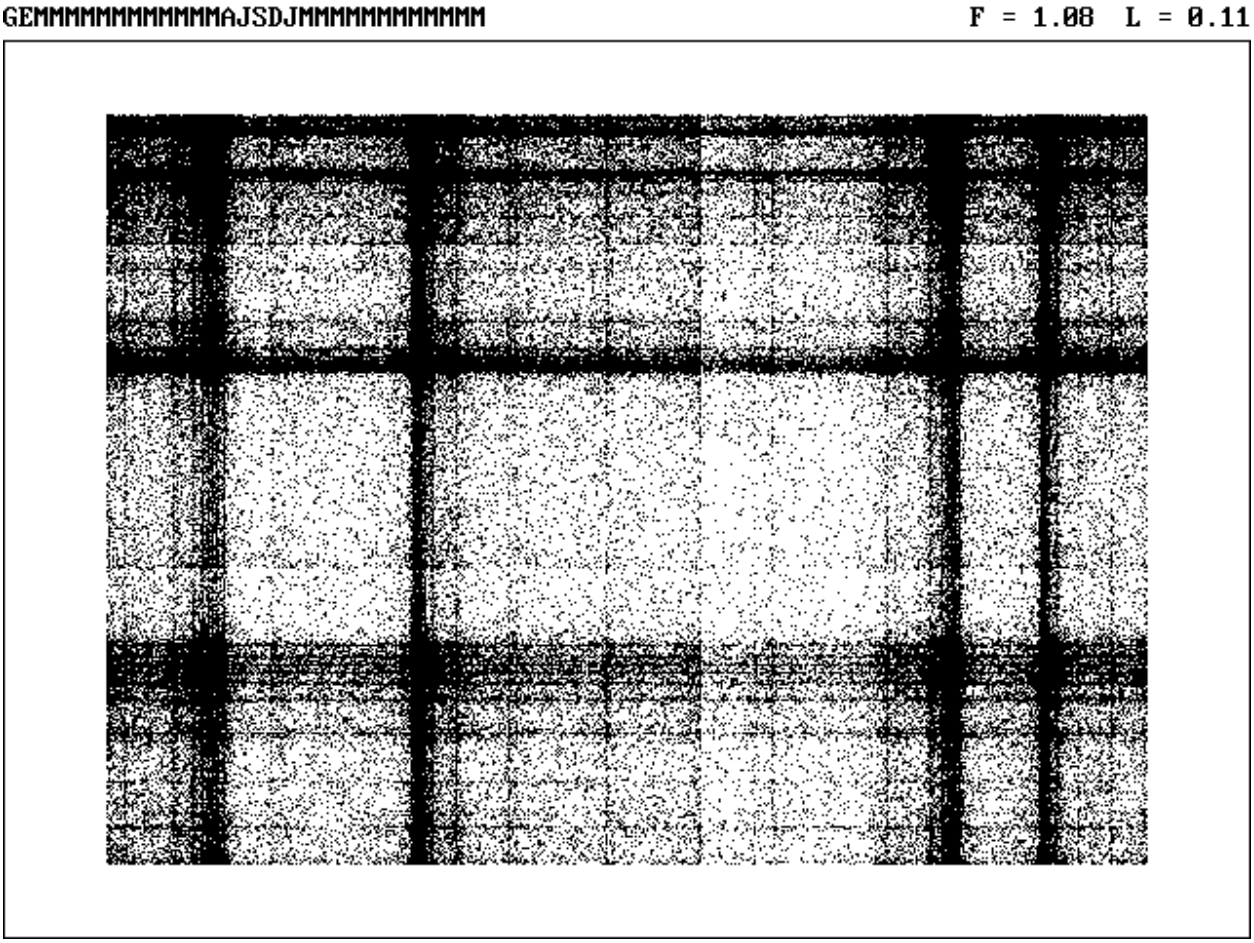


Figure 3-61. Two-dimensional quintic plaid map

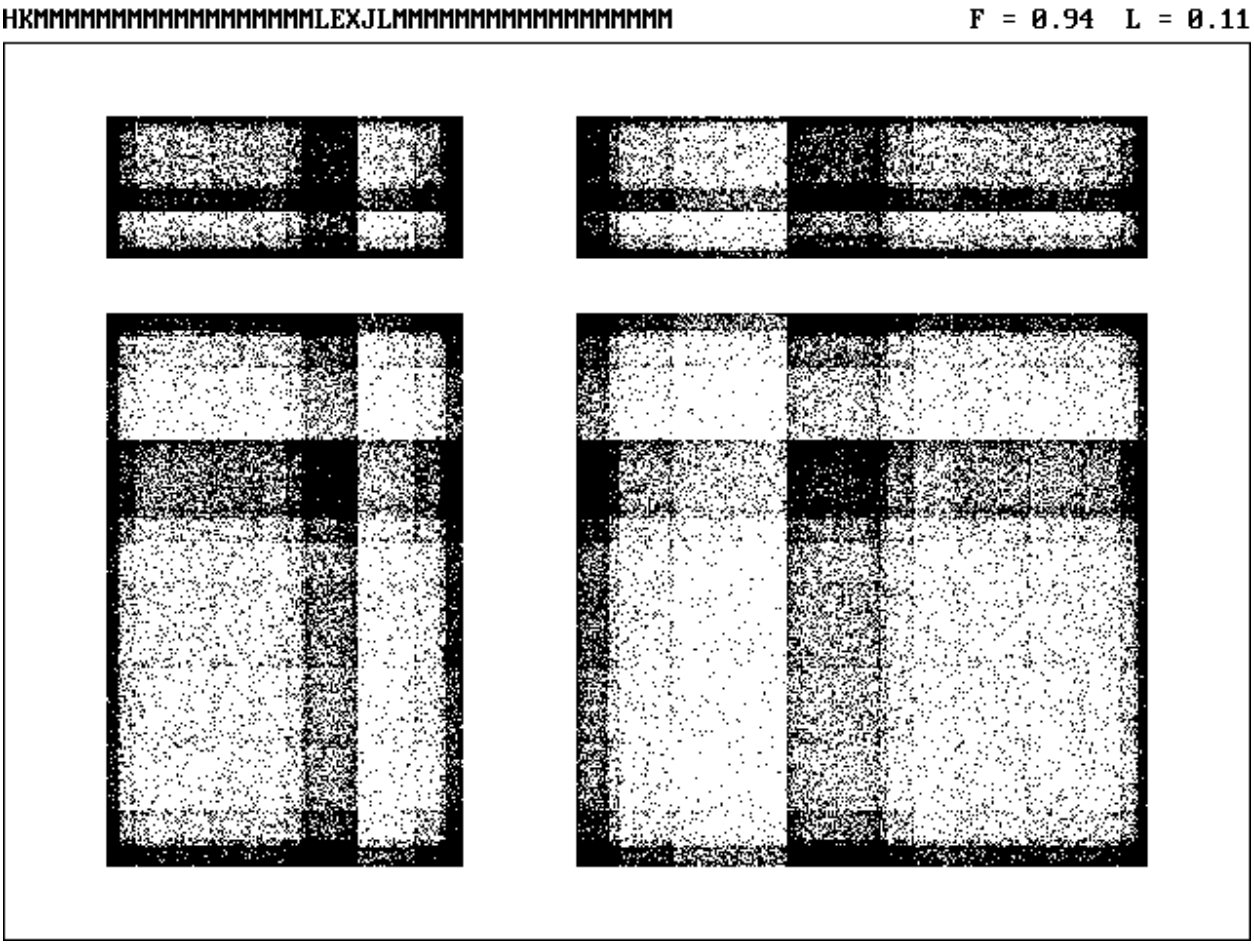


Figure 3-62. Two-dimensional quadratic plaid map on a sphere

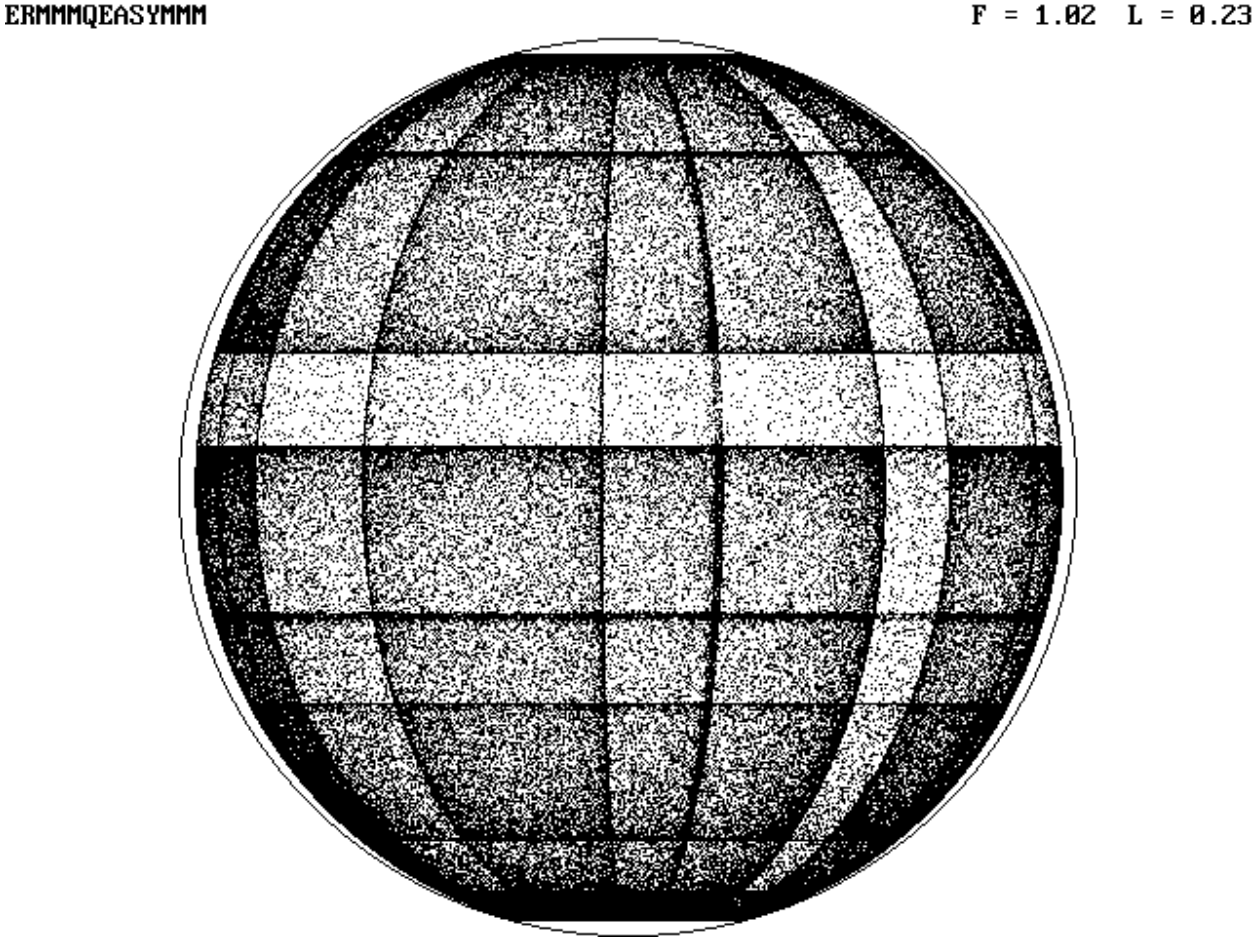
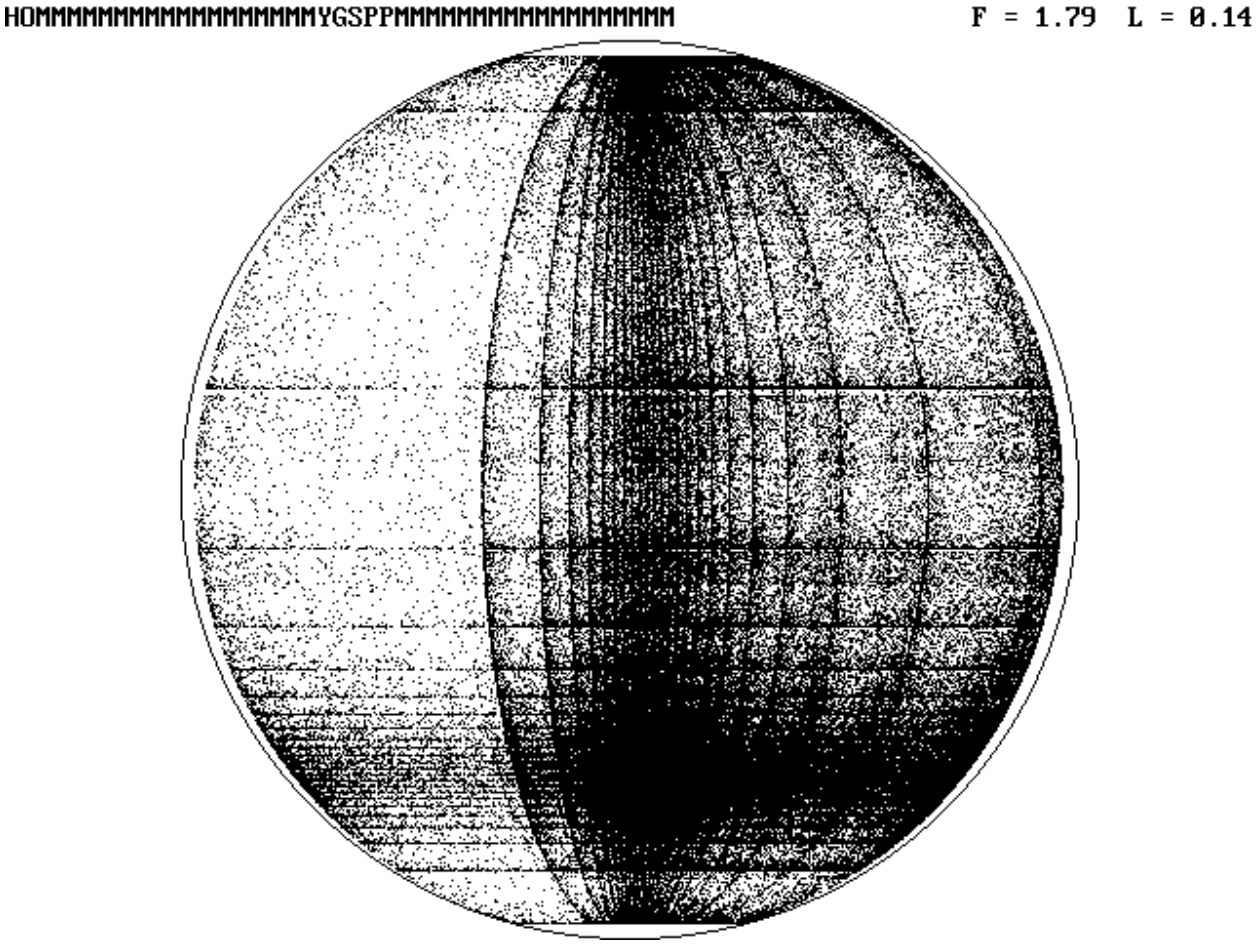


Figure 3-63. Two-dimensional quintic plaid map on a sphere



You might want to try adding colors to emulate a decorative cloth pattern. One way to do this is to color the pixels according to the number of times they are visited by the orbit. This is easily done by changing line 2300 in the program to

```
2300    PSET (XP, YP), (POINT (XP, YP) + 1) MOD 16
```

which causes the color of the existing point at (XP, YP) to be tested and then plotted with the next color in the palette of 16 colors. In Chapter 4, we discuss other ways to produce colorful attractors.

3.8 Strange Attractors that Don't

From the foregoing discussion, you might conclude that all chaotic equations produce strange attractors. Such is not the case. Under certain conditions, the successive iterates of an equation wanders chaotically throughout a region of the plane. There is no basin of attraction, and initial conditions near but outside the chaotic region are not drawn to the region but rather lie on closed curves. Although the chaotic region is not a strange attractor, it may have considerable beauty.

For a chaotic solution not to attract, the area occupied by a cluster of nearby initial conditions must remain the same with successive iterations. The cluster generally contracts in one direction and expands in the other, but the contraction and expansion just cancel, producing a long, thin filament of constant area. A characteristic of such a case is that the two Lyapunov exponents are equal in magnitude but of opposite signs. Such a system is area-preserving. An important class of area-preserving systems are Hamiltonian systems with their corresponding symplectic maps.

You might think that Hamiltonian systems are relatively rare in nature, because they require a special condition. However, there are many important examples of Hamiltonian chaos. They arise because there are quantities in nature such as energy and angular momentum that, in the absence of friction, remain accurately constant no matter how complicated the behavior of the system. We say these quantities are *conserved* or that they are *constants of the motion*. The motion of a planet orbiting a binary-star system or the motion of an electron near the null in a magnetic field exhibits Hamiltonian chaos. A more familiar example is the filamentation of milk when it is stirred into coffee, in which case the volume of the milk is conserved because liquids are nearly incompressible.

With equations such as those we have been using with randomly chosen coefficients, the chance of inadvertently finding an area-preserving solution is essentially zero. However, by placing appropriate conditions on the coefficients, we can guarantee such solutions. The following is an example of an area-preserving, two-dimensional polynomial map:

$$X_{n+1} = a_1 + a_2X_n + a_3X_n^2 + a_4X_n^3 + a_5X_n^4 + a_6X_n^5 \pm Y_n$$

$$Y_{n+1} = a_{22} \pm X_n \qquad \text{(Equation 3l)}$$

This map is fifth order to provide seven arbitrary coefficients that ensure a large number of solutions. The coefficient labels are consistent with the general two-dimensional fifth-order map, in which 33 of the coefficients have been set to zero.

The two terms preceded with ± have coefficients (a_{17} and a_{23}) of either +1 or -1, and this feature guarantees an area-preserving solution. If the signs are the same (both plus or both minus), chaotic solutions are not found. Hamiltonian chaos can occur when the signs are opposite. The negative product of these two coefficients is the *Jacobian* of the map ($J = -a_{17}a_{23}$). The Jacobian is a measure of the net contraction, and it must equal 1.0 for a Hamiltonian system.

Hamiltonian cases can be produced by adding the following lines to **PROG09** after line 2730:

```
2735    IF I% > O% + 1 AND I% <> M% / 2 + 1 THEN MID$(CODE$, I% +        1, 1)
= "M"

2736    MID$(CODE$, M% / 2 - O% + 2, 1) = "W": MID$(CODE$, M% / 2 +      3, 1)
= "C"
```

Sample chaotic symplectic maps are shown in Figures 3-64 through 3-71. Most of the cases resemble chains of islands in which each island contains a fixed point surrounded by closed contours that are not shown. These cases were produced using initial values of $X = Y = 0.05$. Other initial conditions would produce completely different pictures because there is no attractor.

Figure 3-64. Two-dimensional quintic symplectic map

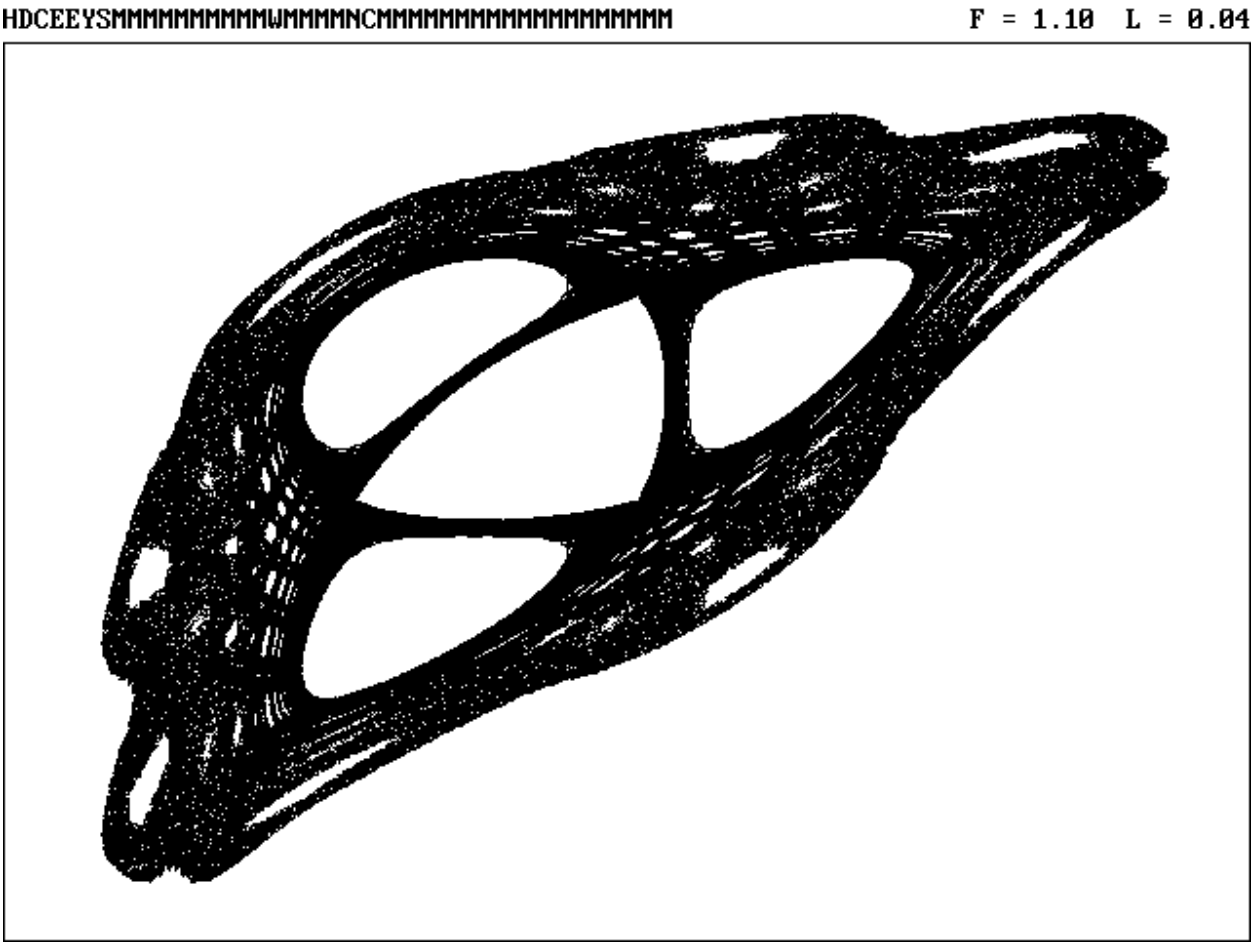


Figure 3-65. Two-dimensional quintic symplectic map

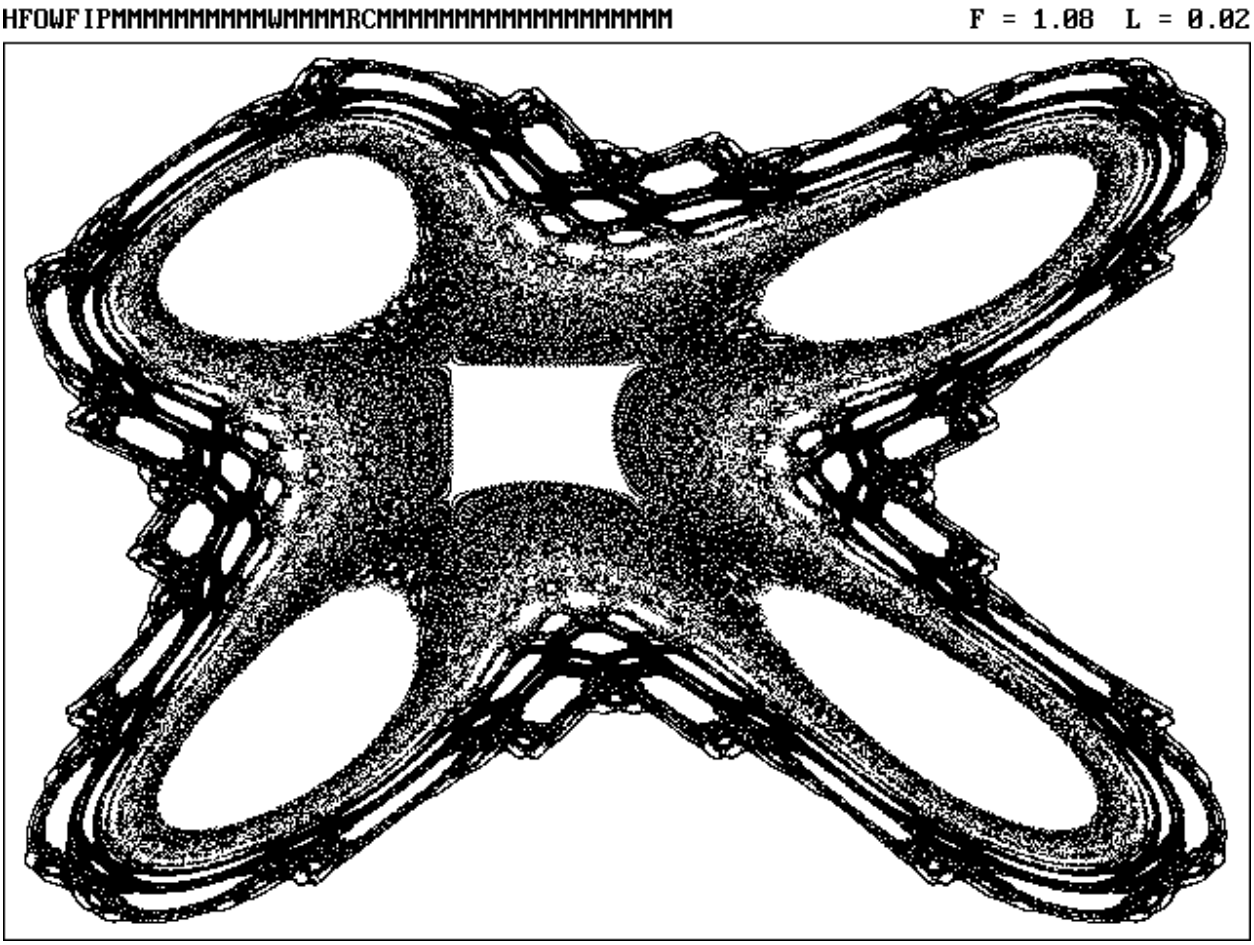


Figure 3-66. Two-dimensional quintic symplectic map

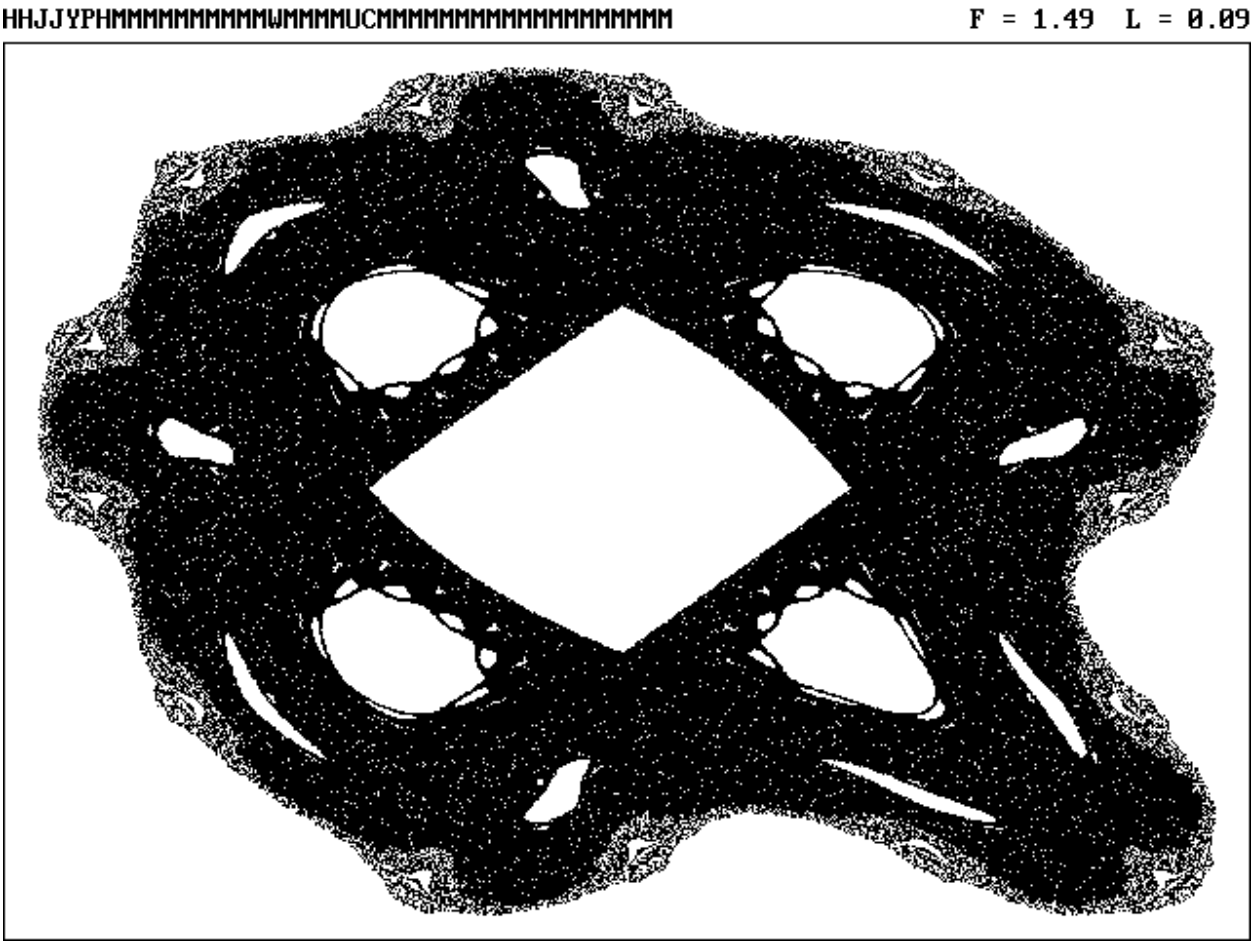


Figure 3-67. Two-dimensional quintic symplectic map

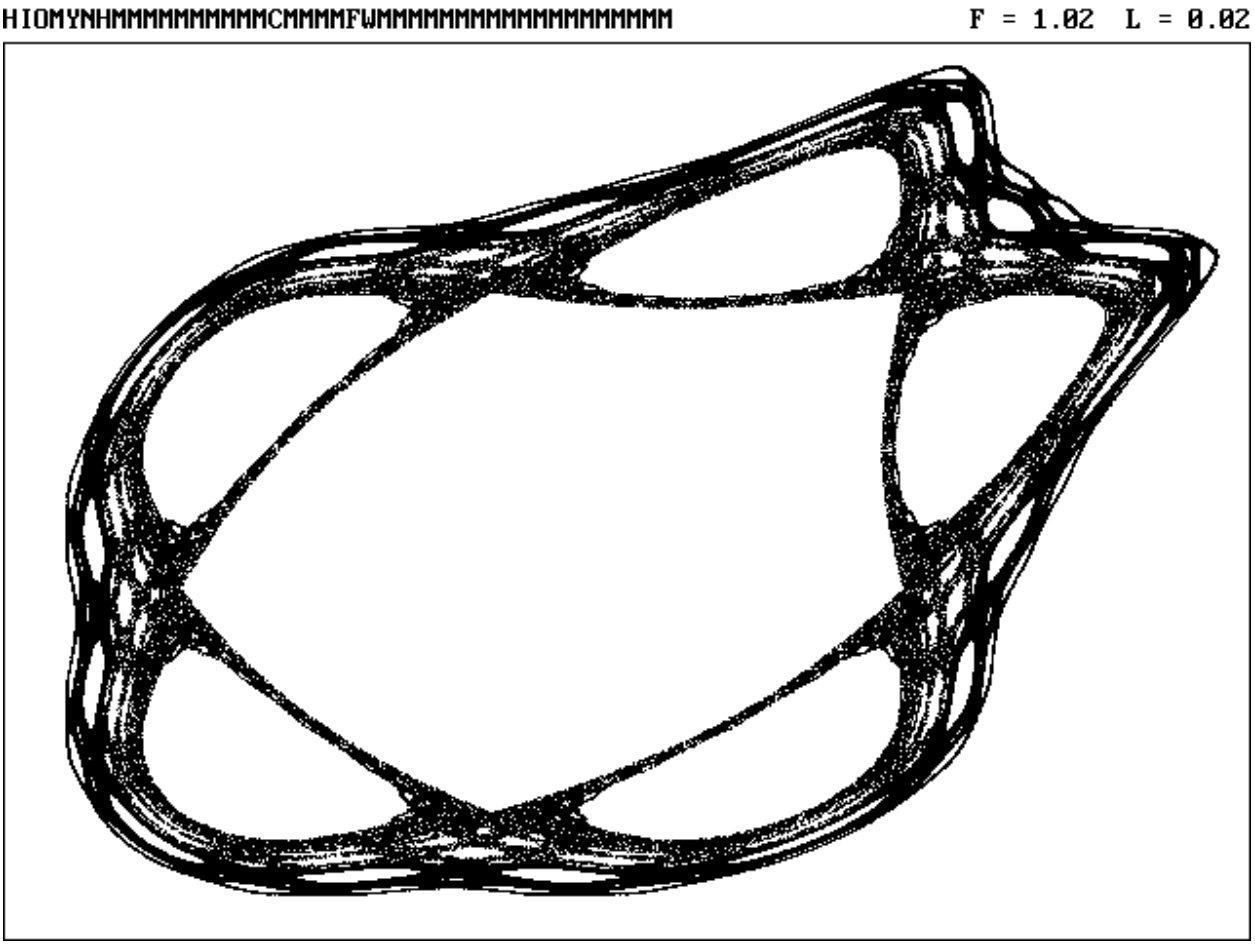


Figure 3-68. Two-dimensional quintic symplectic map

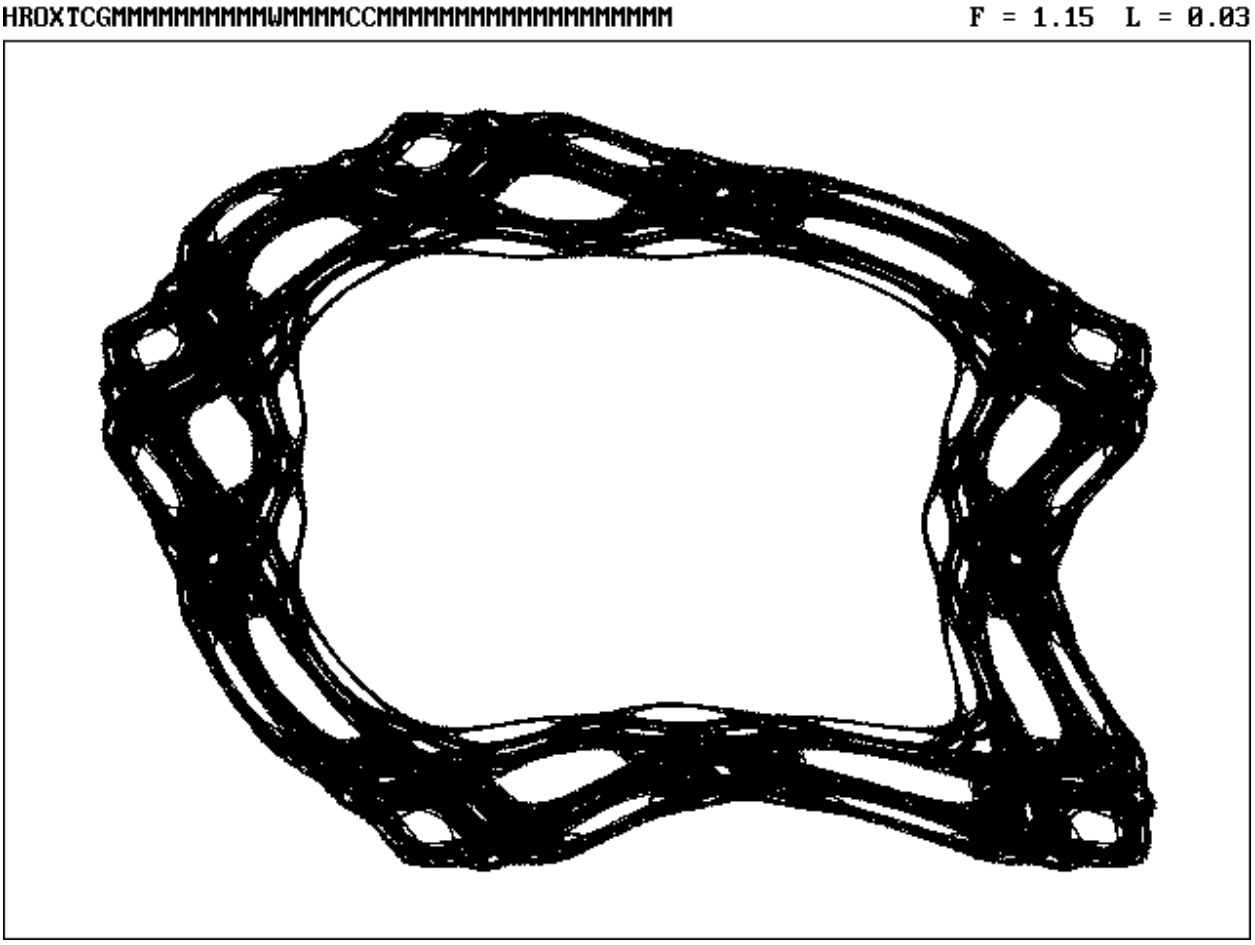


Figure 3-69. Two-dimensional quintic symplectic map

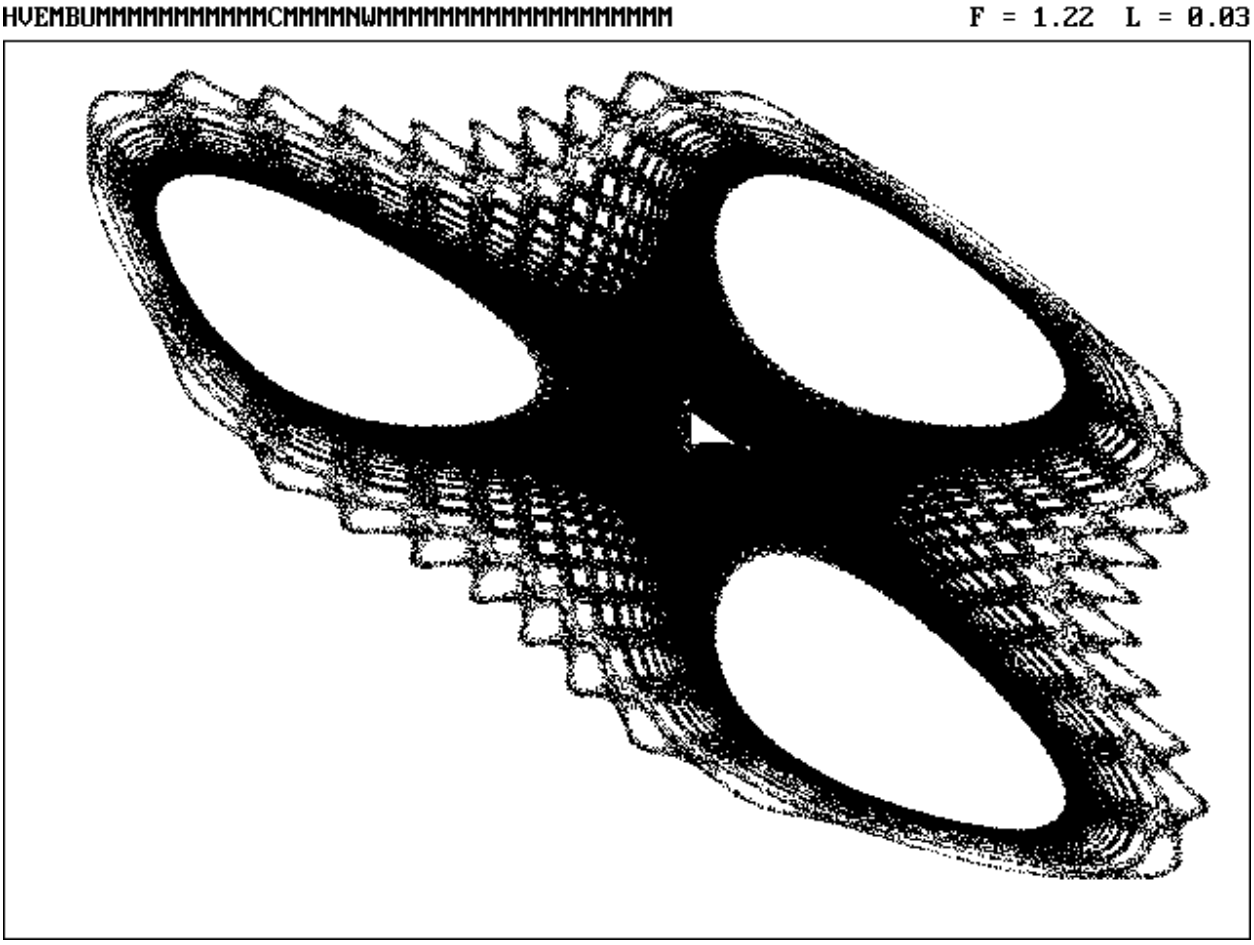


Figure 3-70. Two-dimensional quintic symplectic map

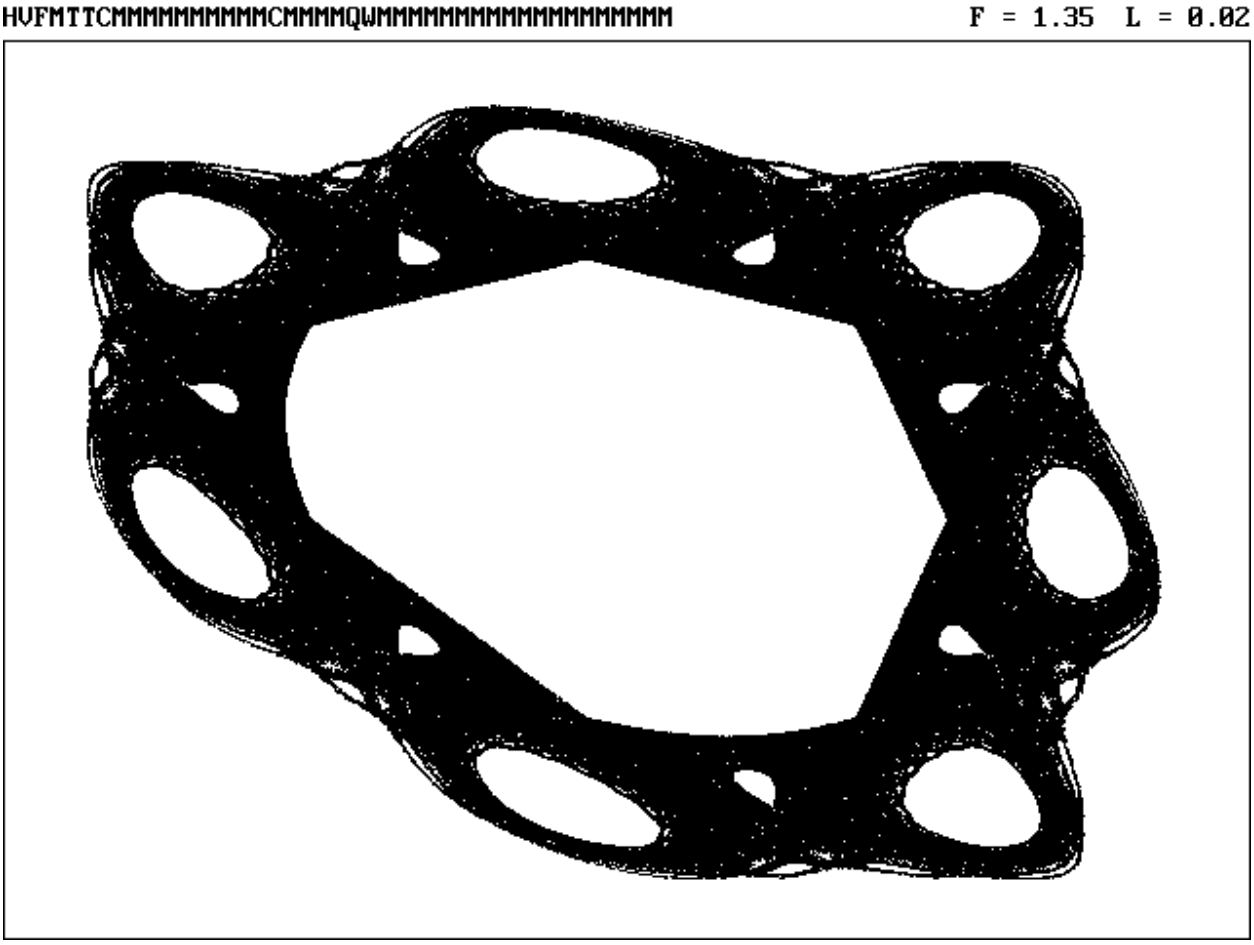
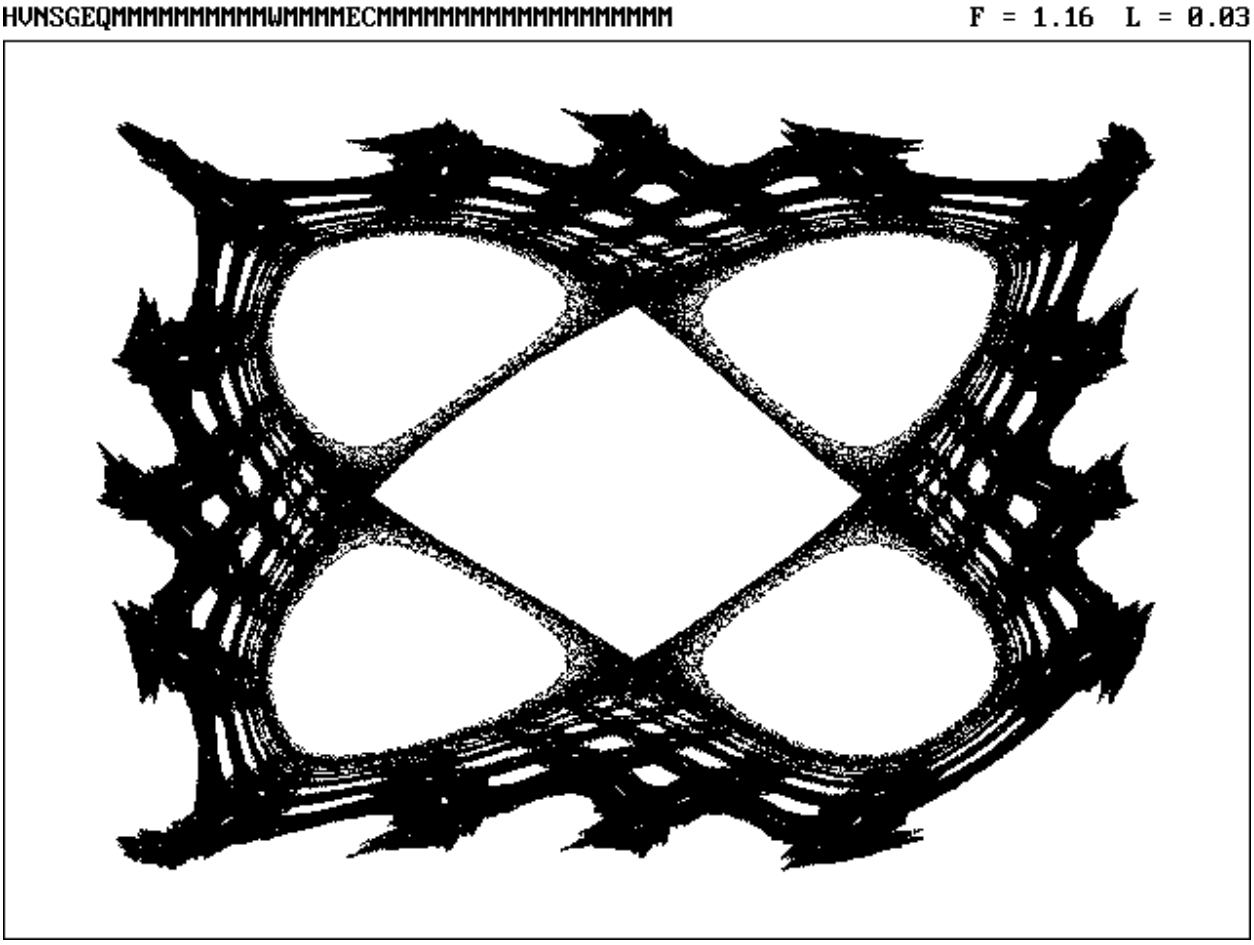


Figure 3-71. Two-dimensional quintic symplectic map



These cases have a different look from non-symplectic strange attractors. The difference is even more pronounced if you watch while they develop on the computer screen. Whereas the regions of a strange attractor tend to be visited uniformly and apparently randomly, the symplectic maps develop much more slowly. The points often wander over a small region for tens of thousands of iterations, and then they suddenly begin to fill in a new distinct region that has never been visited before. Consequently, many more iterations are required to determine the stability and chaotic nature of the solution. You need to be patient while the computer calculates.

The different time behavior of these cases raises an important issue. When you view any of the figures in this book, you are seeing a static object. However, it was produced by a dynamic process. Information about the sequence in which the points accumulated has been lost. This additional information is recovered when

you watch the attractors develop on your computer screen. Most of the attractors fill in uniformly. Their contrast gets progressively greater, much like a photographic print being developed.

However, the symplectic maps develop more slowly and in stages. If your computer has a color monitor, you might try exhibiting this sequence by plotting the points in color and changing the color every few thousand iterations. Some examples using this technique are shown in Section 7.5. If you try this for the non-symplectic attractors, the colors overlap and merge into a uniform gray, or you just see the most recent color. For the symplectic maps, beautiful color patterns can be produced. Otherwise, continue on to the next chapter, where various color techniques are discussed.

3.9 A New Dimension in Sound

With one-dimensional maps, we tried to make music by letting successive iterates control the pitch of the musical notes, all of which were of the same duration. The same procedure can be used with two-dimensional maps. However, we have a second variable at our disposal, so let's use it to control the duration of each note. With actual music, it turns out that there are many more notes of short duration than of long duration. There are roughly twice as many half notes as whole notes, and twice as many quarter notes as half notes, and so forth. This remarkable result seems to hold for all types of music from different composers and cultures. It is evidence of hidden determinism in music.

The program modification **PROG10** uses the *X* value to control the pitch and the *Y* value to control the duration of the notes. For convenience, we assume that the longest note is a whole note and the shortest note is a sixteenth note. Dotted notes and rests are not allowed.

PROG10 also adds to the program a menu screen that reminds you of the **S** command, which toggles the sound on and off, and the **P** command, which toggles the projection between planar and spherical. We also introduce an **A** command to initiate the search for attractors, a **D** command to toggle between one-dimensional and two-dimensional maps, an **I** command to let you input the code of an attractor that you know, and an **X** command to exit the program. Pressing any other key displays the menu screen.

PROG10. Changes required in PROG09 to produce chaotic music and provide a menu screen

```
1000 REM TWO-D MAP SEARCH (With Music and Menu Screen)

1100 SND% = 1                    'Turn sound on

1110 PJT% = 0                    'Projection is planar

1170 GOSUB 4200                  'Display menu screen

1180 IF Q$ = "X" THEN GOTO 1250 'Exit immediately on command


2450 IF QM% > 0 THEN GOTO 2490  'Skip tests when not in search mode


2640 IF QM% > 0 THEN GOTO 2730  'Not in search mode

2650    O% = 2 + INT((OMAX% - 1) * RND)

2660    CODE$ = CHR$(59 + 4 * D% + O%)

2680    GOSUB 4700               'Get value of M%


3530 IF D% > 1 THEN DUR = 2 ^ INT(.5 * (YH - YL) / (YNEW - 9 * YL / 8 + YH / 8))


3610 IF ASC(Q$) > 96 THEN Q$ = CHR$(ASC(Q$) - 32)   'Convert to upper case

3630 IF Q$ = "" OR INSTR("ADIPSX", Q$) = 0 THEN GOSUB 4200

3640 IF Q$ = "A" THEN T% = 1: QM% = 0

3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 2): T% = 1

3730 IF Q$ = "I" THEN IF T% <> 1 THEN SCREEN 0: WIDTH 80: COLOR 15, 1: CLS : LINE
INPUT "Code? "; CODE$: IF CODE$ = "" THEN Q$ = " ": CLS :  ELSE T% = 1: QM% = 1:
GOSUB 4700

3790 IF Q$ = "X" THEN T% = 0
```

```
4200 REM Display menu screen
4210 SCREEN 0: WIDTH 80: COLOR 15, 1: CLS
4220 WHILE Q$ = "" OR INSTR("AIX", Q$) = 0
4230    LOCATE 1, 27: PRINT "STRANGE ATTRACTOR PROGRAM"
4260    PRINT : PRINT
4270    PRINT TAB(27); "A: Search for attractors"
4300    PRINT TAB(27); "D: System is"; STR$(D%); "-D polynomial          map"
4370    PRINT TAB(27); "I: Input code from keyboard"
4400    PRINT TAB(27); "P: Projection is ";
4410        IF PJT% = 0 THEN PRINT "planar   "
4420        IF PJT% = 1 THEN PRINT "spherical"
4540    PRINT TAB(27); "S: Sound is ";
4550        IF SND% = 0 THEN PRINT "off"
4560        IF SND% = 1 THEN PRINT "on "
4600    PRINT TAB(27); "X: Exit program"
4610    Q$ = INKEY$
4620    IF Q$ <> "" THEN GOSUB 3600     'Respond to user command
4630 WEND
4640 RETURN

4700 REM Get dimension and order
4710 D% = 1 + INT((ASC(LEFT$(CODE$, 1)) - 65) / 4)
```

```
4740 O% = 2 + (ASC(LEFT$(CODE$, 1)) - 65) MOD 4

4750 M% = 1: FOR I% = 1 TO D%: M% = M% * (O% + I%): NEXT I%

4770 IF LEN(CODE$) = M% + 1 OR QM% <> 1 THEN GOTO 4810

4780    BEEP        'Illegal code warning

4790    WHILE LEN(CODE$) < M% + 1: CODE$ = CODE$ + "M": WEND

4800    IF LEN(CODE$) > M% + 1 THEN CODE$ = LEFT$(CODE$, M% + 1)

4810 RETURN
```

As you listen to the music produced by the various attractors, you may discover relations between the quality of the music and the appearance of the attractor. The cases that seem most musical tend to have certain visual characteristics, which are left for you to discover. Do attractors that appeal to the eye also appeal to the ear?

After you have generated some music of your own, you may want to try some of the cases in Table 3-1 using the **I** command to input them to the program. These cases have been selected for their musical quality and are limited to quadratic maps to simplify typing their codes. An interesting study would be to accumulate your own longer list of musical attractors and to see if they preferentially have certain fractal dimensions and Lyapunov exponents. If so, then it should be possible to program the computer to be a music critic as well as an art critic.

Table 3-1. List of some musical attractors and their characteristics

Code	F	L	Code	F	L
EDFLQJGDGMSJV	1.17	0.35	EPLKQNGALTVDD	1.03	0.20
EGITIKLJNSKAT	1.19	0.04	EQVHVRXREMJED	1.50	0.19
EHXJCQMYLONDK	0.95	0.12	ERKKCUNHERKAV	1.51	0.47
EJETCOHRSIQFN	1.56	0.25	ESHKBEWJFUOPJ	1.43	0.39
EKLVEVAOSGYJX	1.12	0.20	ETFJJNMKESAFX	0.97	0.30
ELLNJNEAMPLDX	1.11	0.64	EUFLXKIETROOO	0.90	0.40
ENIDATWFTPOSL	1.62	0.26	EVHEQLLDMMBFP	1.47	0.49
EOKYEVMDXXJUP	0.84	0.22	EXJNXAIFANNEN	1.60	0.17

After listening to the enormous variety of musical sequences that can be generated by this technique, you might wonder whether your favorite musical composition could be compressed into a short code and generated using iterated maps. After all, even the simple cases in Table 3-1 are chosen from among about 6×10^{16} different codes, and each code corresponds to a different piece of music.

However, a typical musical piece might have hundreds or thousands of notes, each of which can represent dozens of pitches and many durations. Thus we can be fairly confident using the principles of information theory that such extreme compression is unlikely, unless music has considerably more structure than is apparent. However, if you only want to generate a short tune with a few notes, there might well be a way to do so using this technique. If you are mathematically inclined, take it as a challenge to find a way to do this.

The generation of computer music using chaotic iterated maps is a promising technique still in its infancy. You may want to incorporate more sophisticated musical rules into the program to generate music that is much more pleasing than what results from this simple procedure. Furthermore, an interesting project would be to turn the process around and see if music written by humans resembles a strange attractor, and if so, to measure its fractal dimension and Lyapunov exponent. Perhaps music of different types or by different composers would have different values of these quantities.

Chapter 4

Attractors of Depth

A two-dimensional world is a mere shadow of reality. The techniques described in the previous chapters are easily extended to produce attractors embedded in the three-dimensional space in which we live. The challenge is in finding ways to exhibit and visualize such three-dimensional objects within the limitations of the computer screen and printed page. This chapter emphasizes new visualization techniques and provides many new examples of strange attractors that have depth as well as width and height.

4.1 Projections

The procedure for seeking attractors in three dimensions (which we might whimsically call *strange attractors of the third kind*) is just like the two-dimensional case, except that we introduce a third variable *Z* to accompany *X* and *Y*. You can think of *Z* as representing the position in a direction out of the screen or page on which the attractor is displayed. We assume the direction of positive *Z* is in front of the page and the direction of negative *Z* is behind the page, as is customary for a conventional *right-handed coordinate system*. The term *right-handed* comes from the convention that if you point the fingers of your right hand in the direction of the *X*-axis and curl them so that they point along the *Y*-axis, your thumb points in the *Z* direction. This choice is purely arbitrary but widely accepted.

The simplest system of equations that produces strange attractors embedded in a three-dimensional space is a set of coupled quadratic equations, the most general form of which is given by

$$X_{n+1} = a_1 + a_2X_n + a_3X_n^2 + a_4X_nY_n + a_5X_nZ_n + a_6Y_n + a_7Y_n^2 + a_8Y_nZ_n + a_9Z_n + a_{10}Z_n^2$$

$$Y_{n+1} = a_{11} + a_{12}X_n + a_{13}X_n^2 + a_{14}X_nY_n + a_{15}X_nZ_n + a_{16}Y_n + a_{17}Y_n^2 + a_{18}Y_nZ_n + a_{19}Z_n + a_{20}Z_n^2$$

$$Z_{n+1} = a_{21} + a_{22}X_n + a_{23}X_n^2 + a_{24}X_nY_n + a_{25}X_nZ_n + a_{26}Y_n$$

$$+ a_{27}Y_n^2 + a_{28}Y_nZ_n + a_{29}Z_n + a_{30}Z_n^2 \qquad \text{(Equation 4A)}$$

These equations have 30 coefficients, which allow an enormous variety of attractors. The extension to equations with orders higher than two is straightforward. Three-dimensional cubic equations have 60 coefficients, quartic equations have 105 coefficients, and quintic equations have 168 coefficients. The number of coefficients for order *O* is given by *(O + 1)(O + 2)(O + 3) / 2*. We will code the second-order through fifth-order systems in three dimensions with the initial letters I, J, K, and L, respectively.

Note that 168 coefficients allows 25^{168} or about 10^{234} combinations. This is a truly astronomical number. Even if only a small fraction of them correspond to distinct strange attractors, their number enormously exceeds the number of electrons, protons, and neutrons in the entire universe—a mere 10^{79}. Thus the number of fifth-order three-dimensional strange attractors is essentially infinite. You can have a large collection of your own, none of which are likely to be reproduced by anyone else unless you give them the code you used to produce them. The code is like a combination lock with 168 settings that all must be entered correctly and in the proper order.

Now we must confront the issue of how best to display an object composed of points in a three-dimensional space. Such problems are in the domain of a new specialty called *visualization*, which we may define as the use of computer imagery to gain insight into complex phenomena. The need for improved visualization techniques has emerged from the rapidly growing use of computers as the primary tool for scientific calculation and modeling. As computers become more powerful, it is increasingly important to devise methods of dealing with large quantities of data. The eye and brain are very efficient at discerning visual patterns, and these patterns permit an intuitive understanding of complicated processes in a way that equations often cannot. Scientists have recently developed impressive visualization techniques, simple versions of which are presented here.

The simplest method is to ignore one of the coordinates and to plot the points in the remaining two dimensions. This method is equivalent to looking at the shadow cast by an object when illuminated from directly above by a point-source of light a large distance away. If the light source is on the *Z*-axis, we say the attractor is *projected* onto the *XY* plane. The screen used in conjunction with a slide projector is such a plane. Of course, considerable information about the attractor is lost in such a projection, but the method is a convenient starting point, and it is simple to program.

PROG11 provides the changes that must be made in **PROG10** to extend the

attractor search to three dimensions with order up to five. Since the search slows down considerably in three dimensions with such a large number of coefficients, especially if you don't have a compiled version of BASIC and a fast computer, the program saves, for each case found, the code, fractal dimension, and Lyapunov exponent in a disk file with the name **SA.DIC** (Strange Attractor DICtionary). This feature allows you to run the program unattended and to collect the attractors it finds. We will later modify the program to let you examine the cases that you collect.

PROG11. Changes required in PROG10 to search for strange attractors in three dimensions

```
1000 REM THREE-D MAP SEARCH

1020 DIM XS(499), YS(499), ZS(499), A(504), V(99), XY(4), XN(4)

1070 D% = 3                    'Dimension of system

1100 SND% = 0                  'Turn sound off



1530 Z = .05

1550 XE = X + .000001: YE = Y: ZE = Z

1600 ZMIN = XMIN: ZMAX = XMAX



1720 M% = 1: XY(1) = X: XY(2) = Y: XY(3) = Z

2010 M% = M% - 1: XNEW = XN(1): YNEW = XN(2): ZNEW = XN(3)



2160    IF Z < ZMIN THEN ZMIN = Z

2170    IF Z > ZMAX THEN ZMAX = Z

2210 XS(P%) = X: YS(P%) = Y: ZS(P%) = Z



2410 IF ABS(XNEW) + ABS(YNEW) + ABS(ZNEW) > 1000000! THEN T% = 2
```

```
2460 IF N >= NMAX THEN T% = 2: GOSUB 4900   'Strange attractor found
2470 IF ABS(XNEW - X) + ABS(YNEW - Y) + ABS(ZNEW - Z) < .000001 THEN T% = 2
2530 Z = ZNEW


2910 XSAVE = XNEW: YSAVE = YNEW: ZSAVE = ZNEW
2920 X = XE: Y = YE: Z = ZE: N = N - 1
2950 DLZ = ZNEW - ZSAVE
2960 DL2 = DLX * DLX + DLY * DLY + DLZ * DLZ
3010    ZE = ZSAVE + RS * (ZNEW - ZSAVE)
3020    XNEW = XSAVE: YNEW = YSAVE: ZNEW = ZSAVE


3140 IF ZMAX - ZMIN < .000001 THEN ZMIN = ZMIN - .0000005: ZMAX = ZMAX + .0000005
3400 LOCATE 1, 1: IF LEN(CODE$) < 62 THEN PRINT CODE$
3410 IF LEN(CODE$) >= 62 THEN PRINT LEFT$(CODE$, 57) + "..."


3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 3): T% = 1


3920 IF N = 1000 THEN D2MAX = (XMAX - XMIN) ^ 2 + (YMAX - YMIN) ^ 2 + (ZMAX - ZMIN)
^ 2
3940 DX = XNEW - XS(J%): DY = YNEW - YS(J%): DZ = ZNEW - ZS(J%)
3950 D2 = DX * DX + DY * DY + DZ * DZ


4760 IF D% = 3 THEN M% = M% / 2
```

```
4900 REM Save attractor to disk file SA.DIC

4910 OPEN "SA.DIC" FOR APPEND AS #1

4920 PRINT #1, CODE$; : PRINT #1, USING "##.##"; F; L

4930 CLOSE #1

4940 RETURN
```

Some examples of the attractors produced by **PROG11** are shown in Figures 4-1 through 4-16. Note that the fractal dimension shown for each case is the dimension of the actual attractor and not the dimension of its projection. Thus the fractal dimension can be as large as 3 even though the projection has dimension of at most 2. The projection of a point (zero dimensions) onto a surface is a point, the projection of a line (one dimension) is a line, the projection of a surface (two dimensions) is a surface, but the projection of a solid (three dimensions) onto a surface is only a surface (two dimensions).

Figure 4-1. Projection of three-dimensional quadratic map

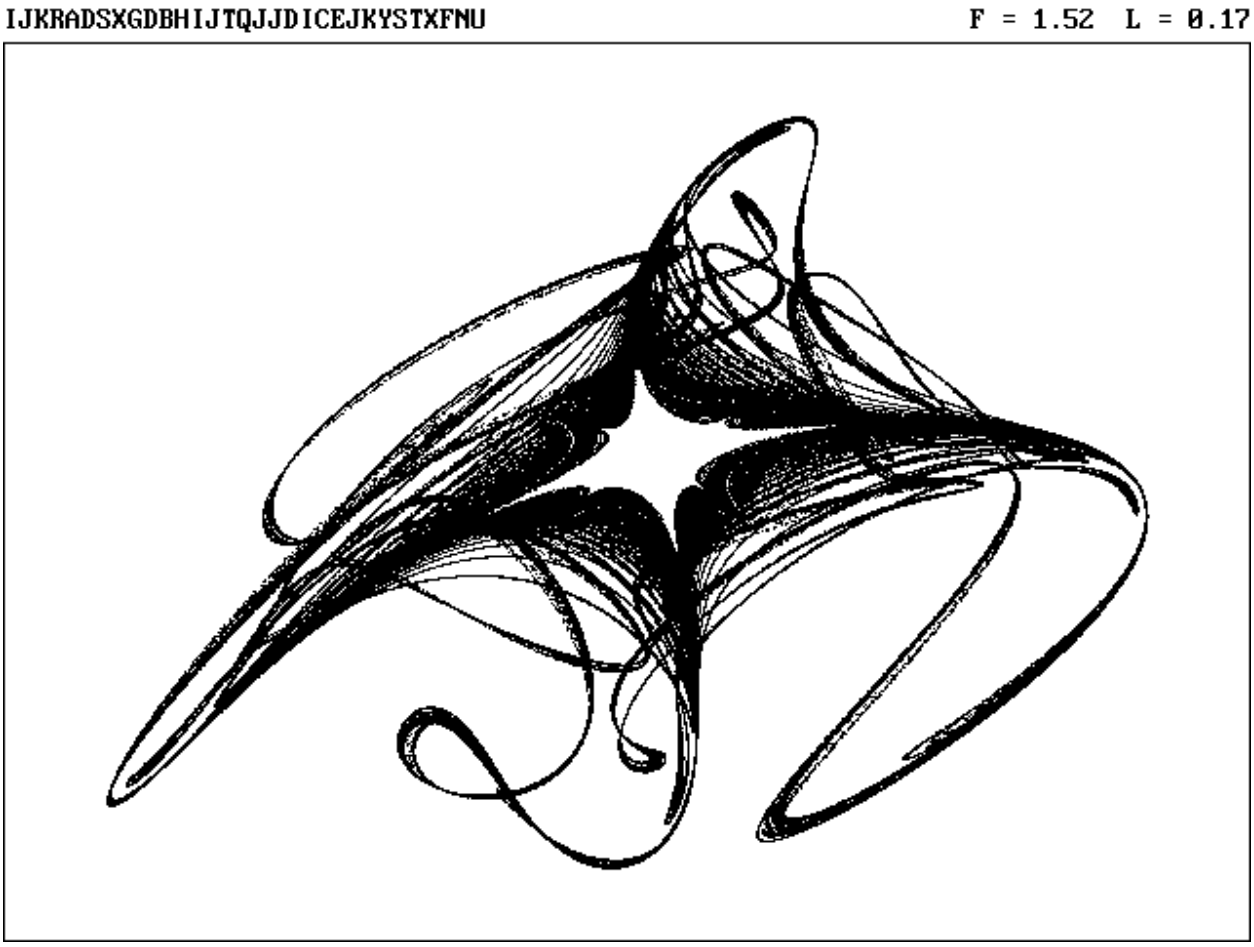


Figure 4-2. Projection of three-dimensional quadratic map

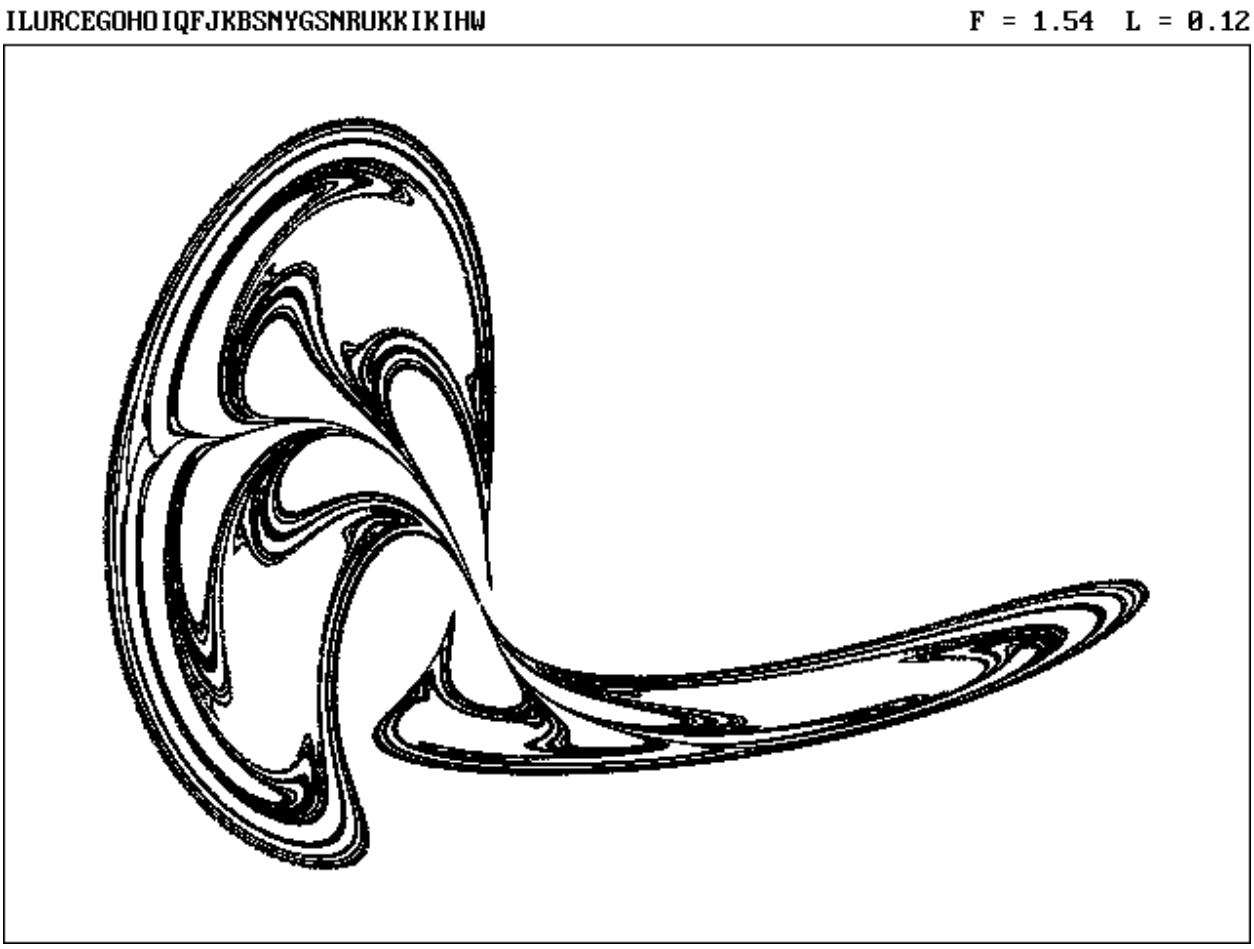


Figure 4-3. Projection of three-dimensional quadratic map

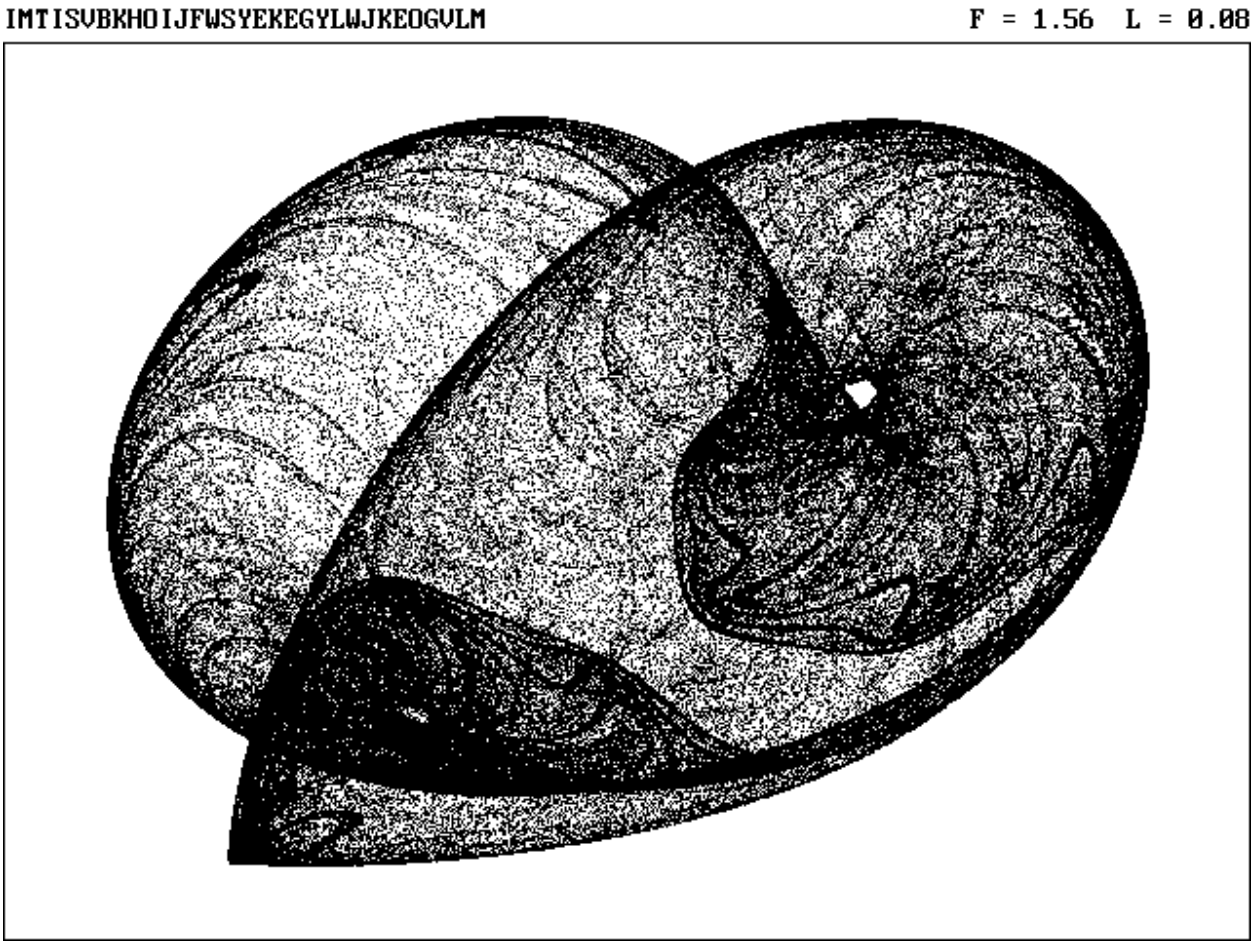


Figure 4-4. Projection of three-dimensional quadratic map

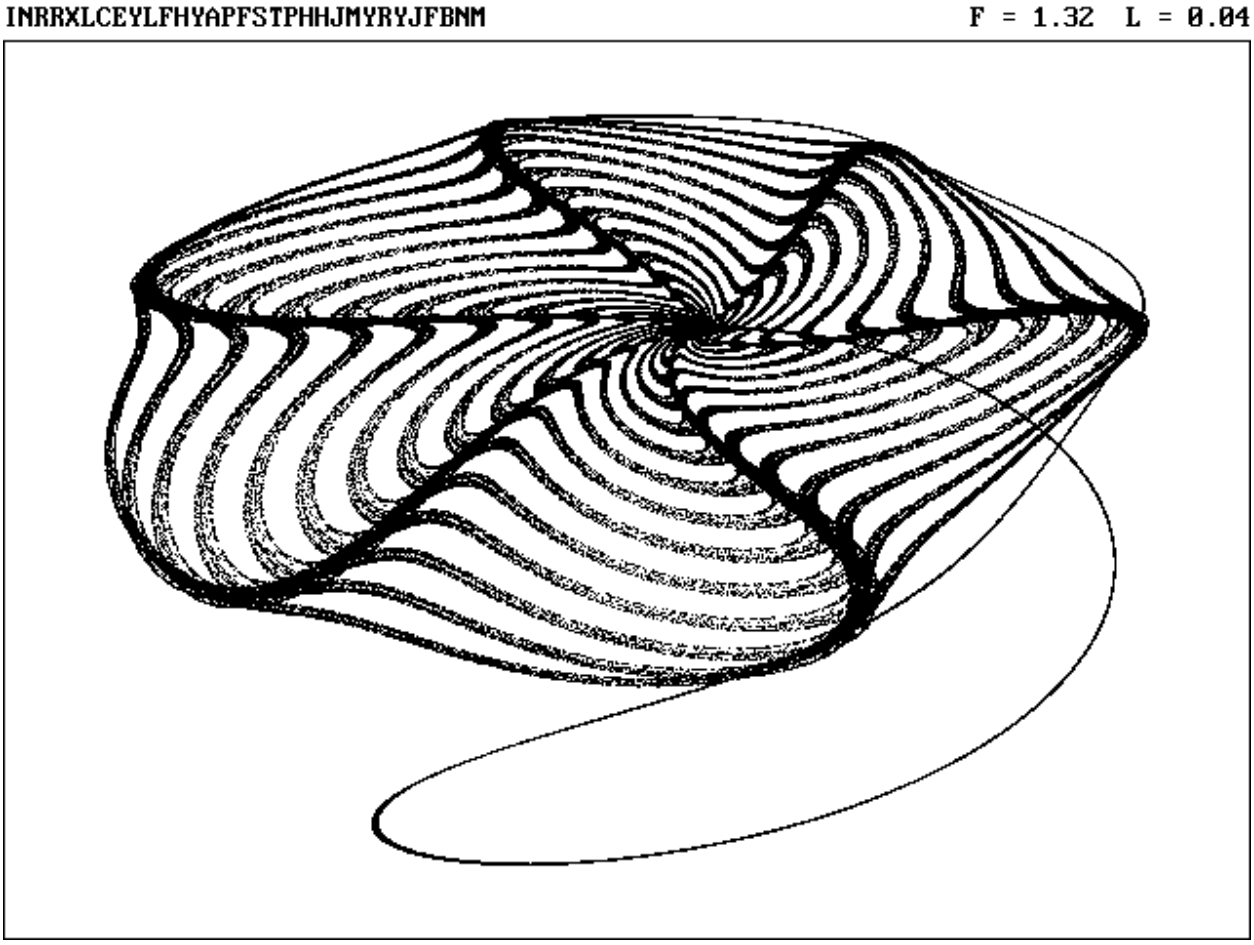


Figure 4-5. Projection of three-dimensional quadratic map

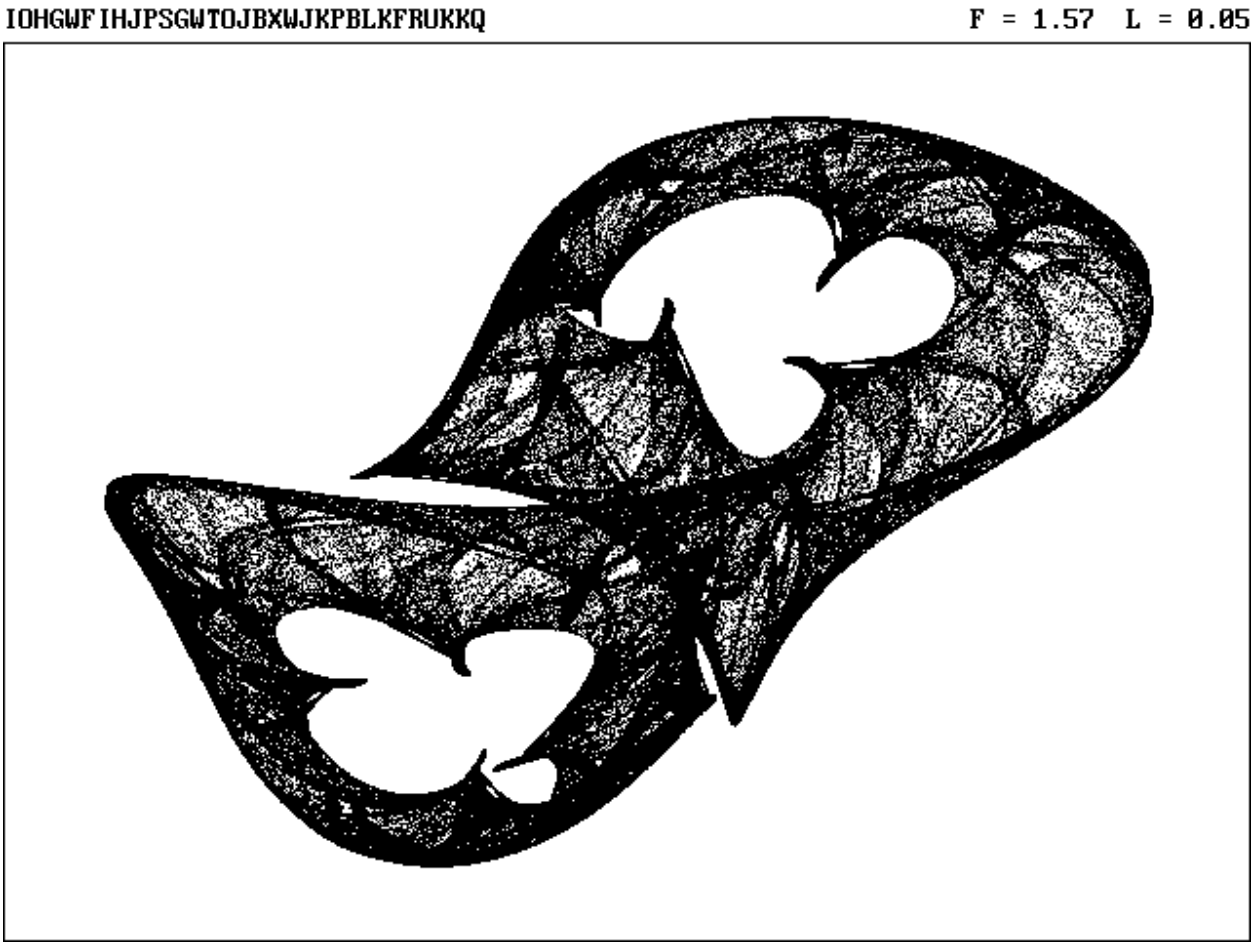


Figure 4-6. Projection of three-dimensional quadratic map

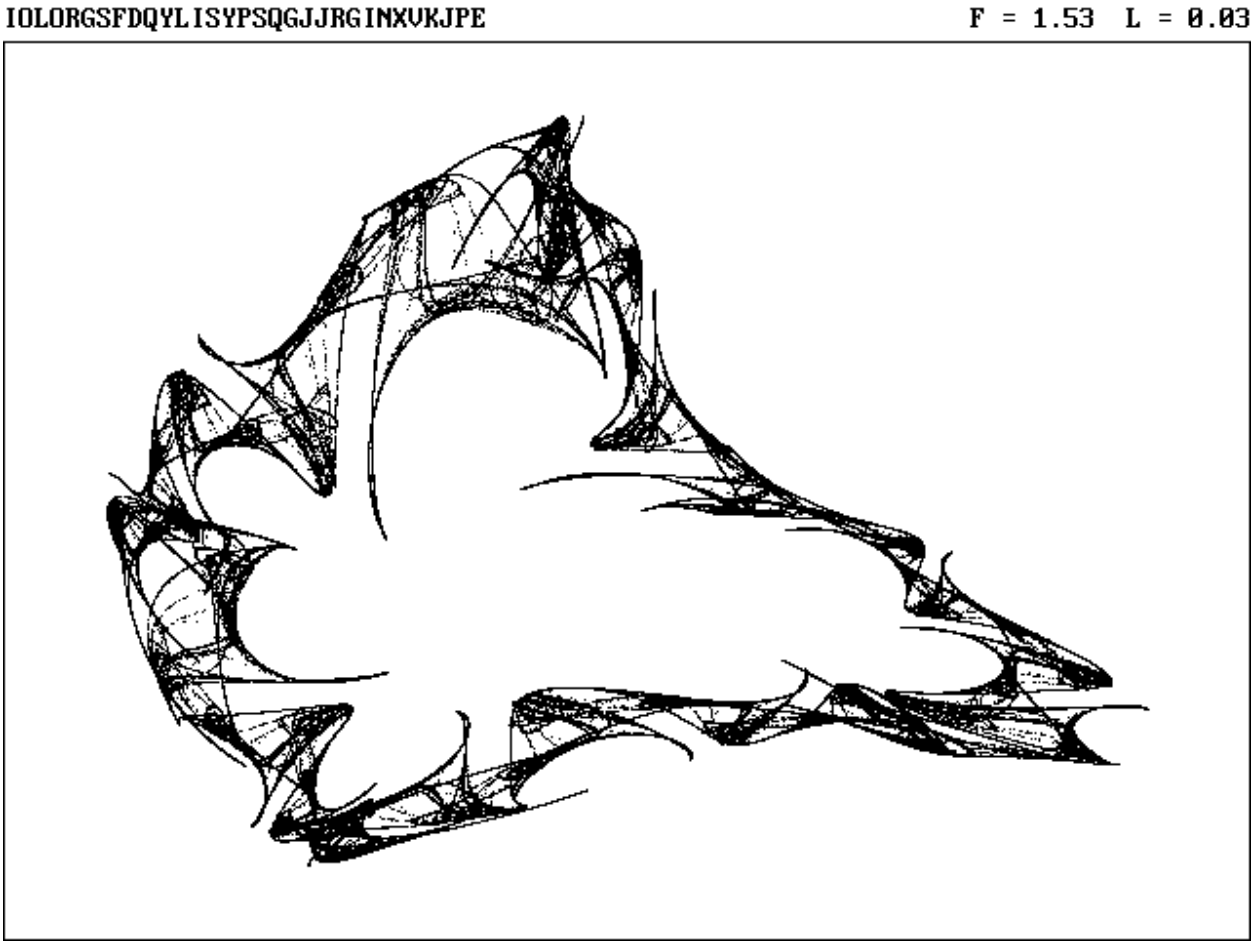


Figure 4-7. Projection of three-dimensional quadratic map

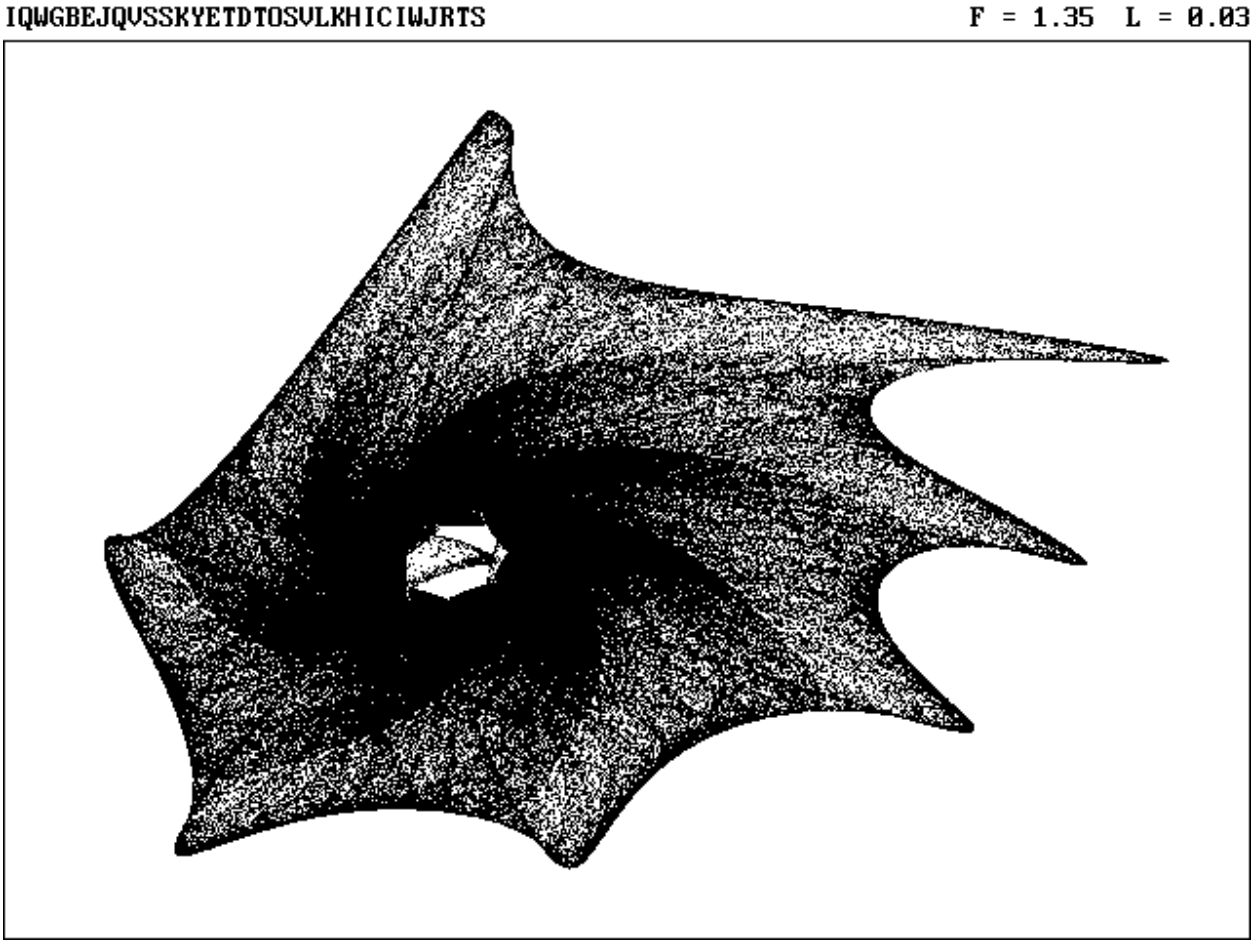


Figure 4-8. Projection of three-dimensional quadratic map

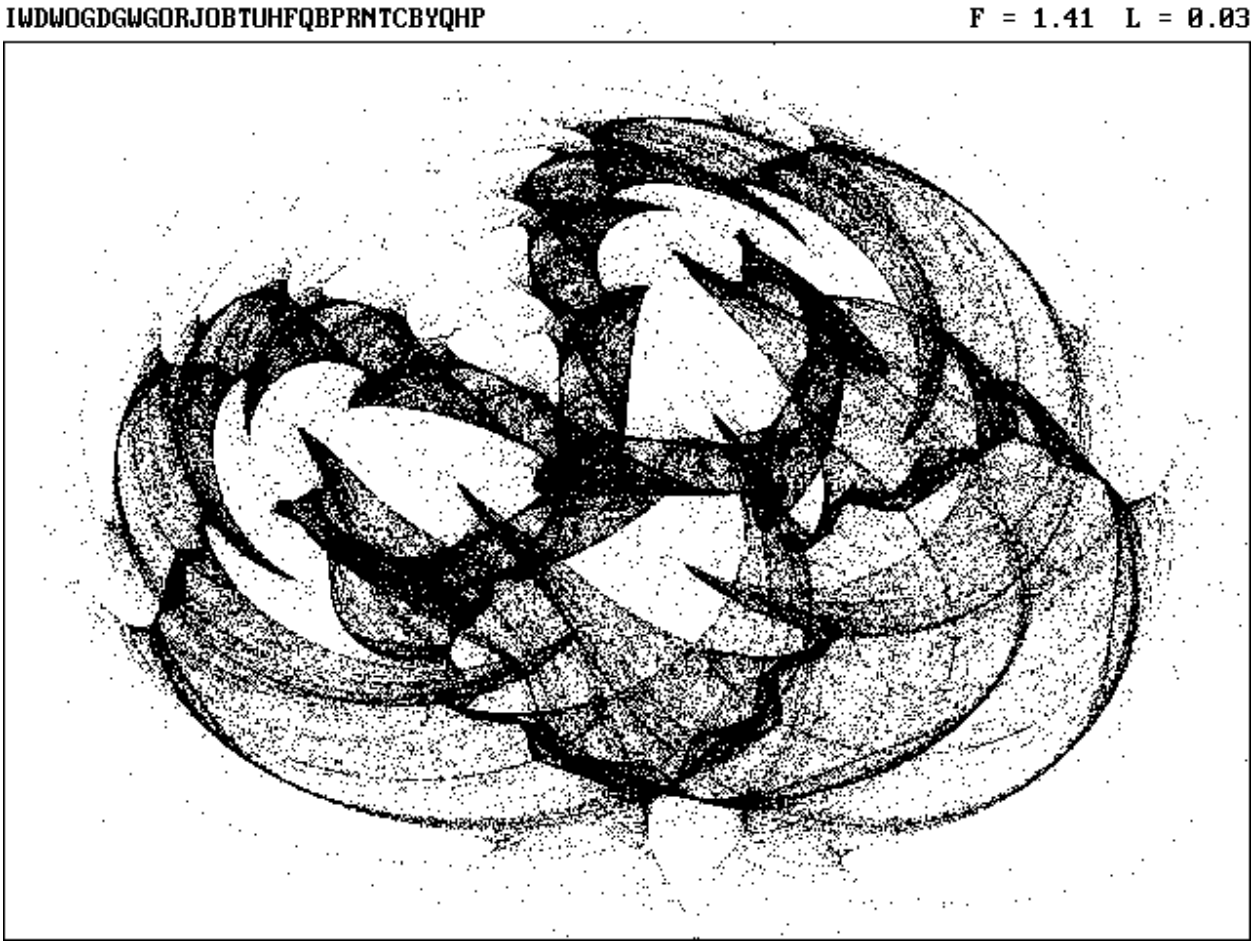


Figure 4-9. Projection of three-dimensional cubic map

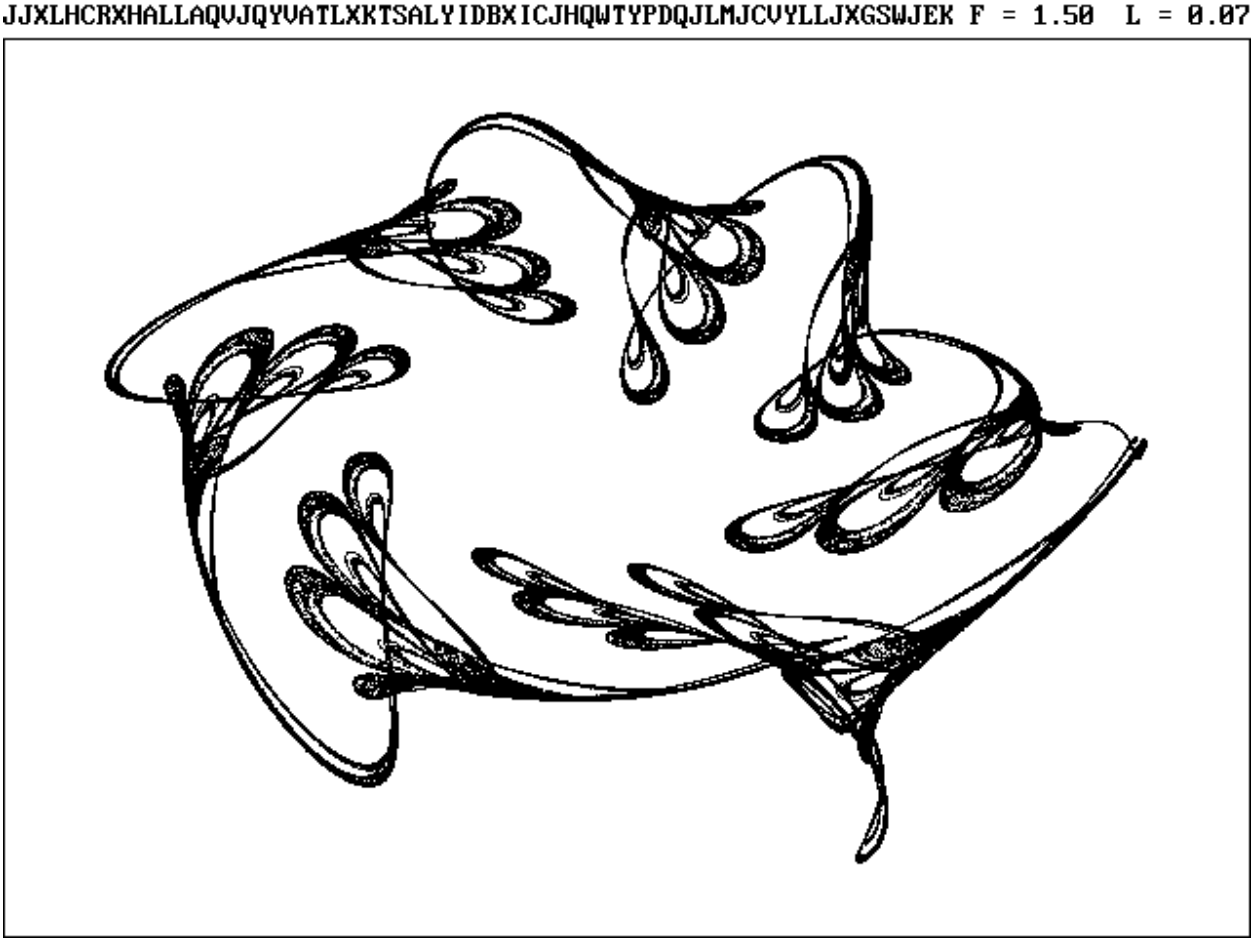


Figure 4-10. Projection of three-dimensional cubic map

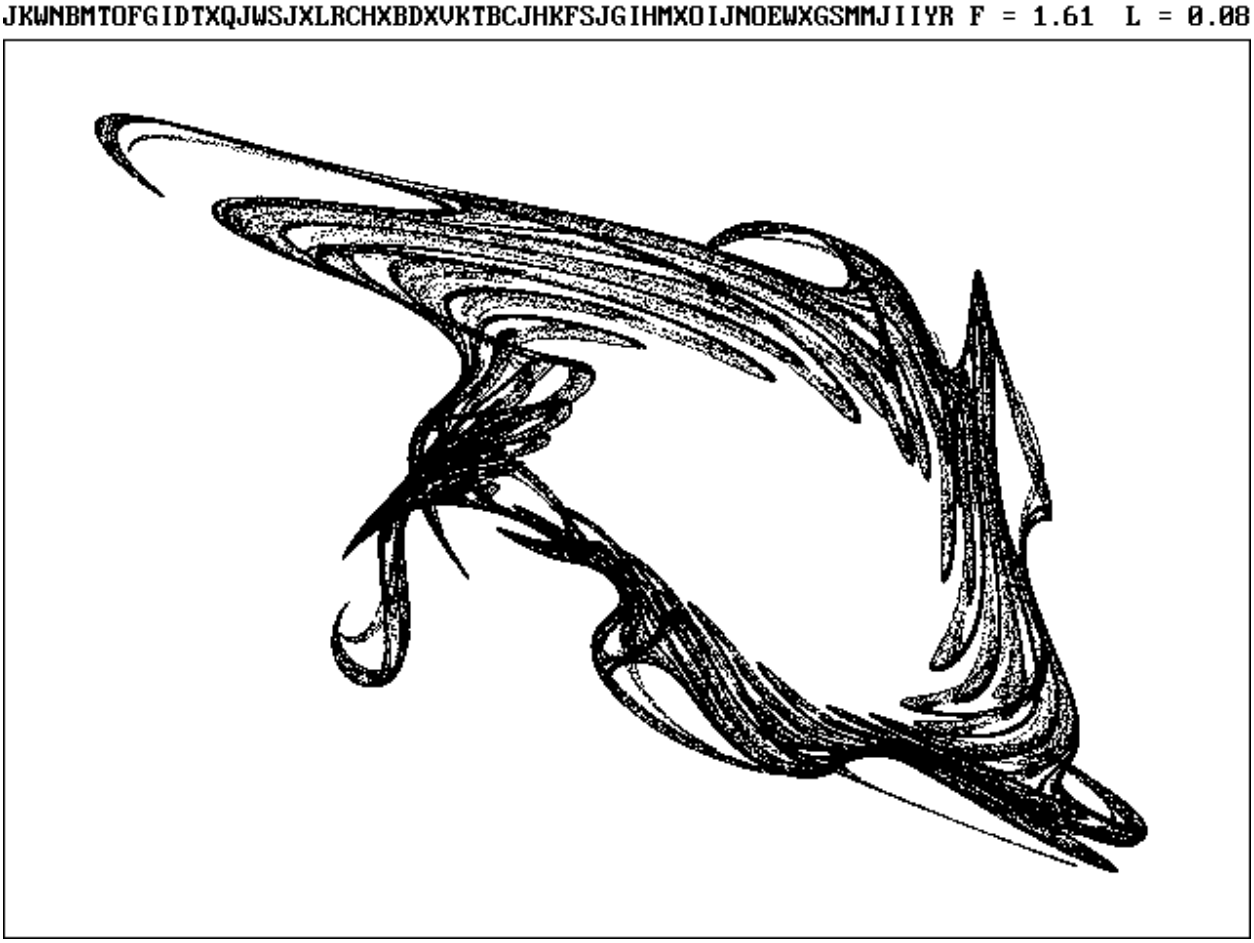


Figure 4-11. Projection of three-dimensional cubic map

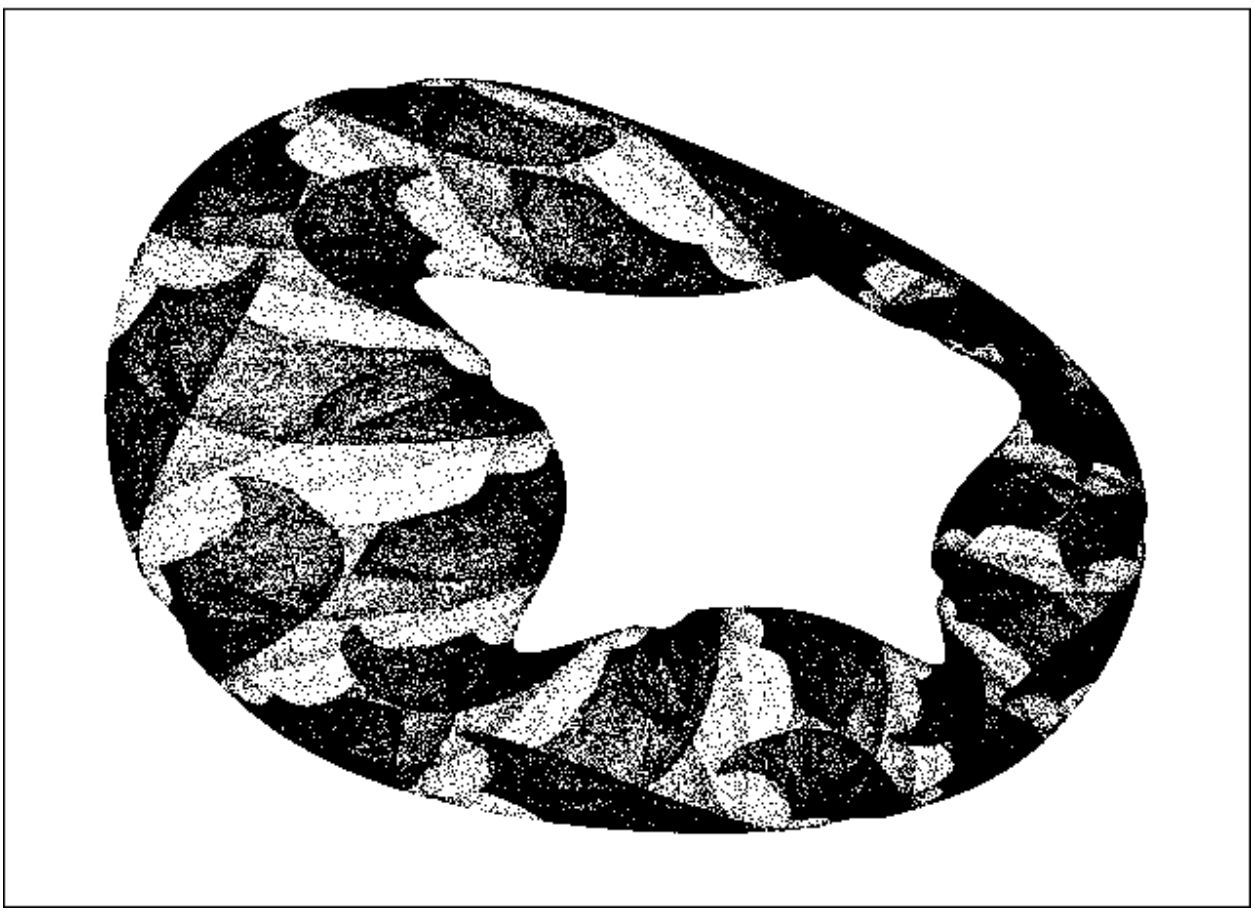


Figure 4-12. Projection of three-dimensional quartic map

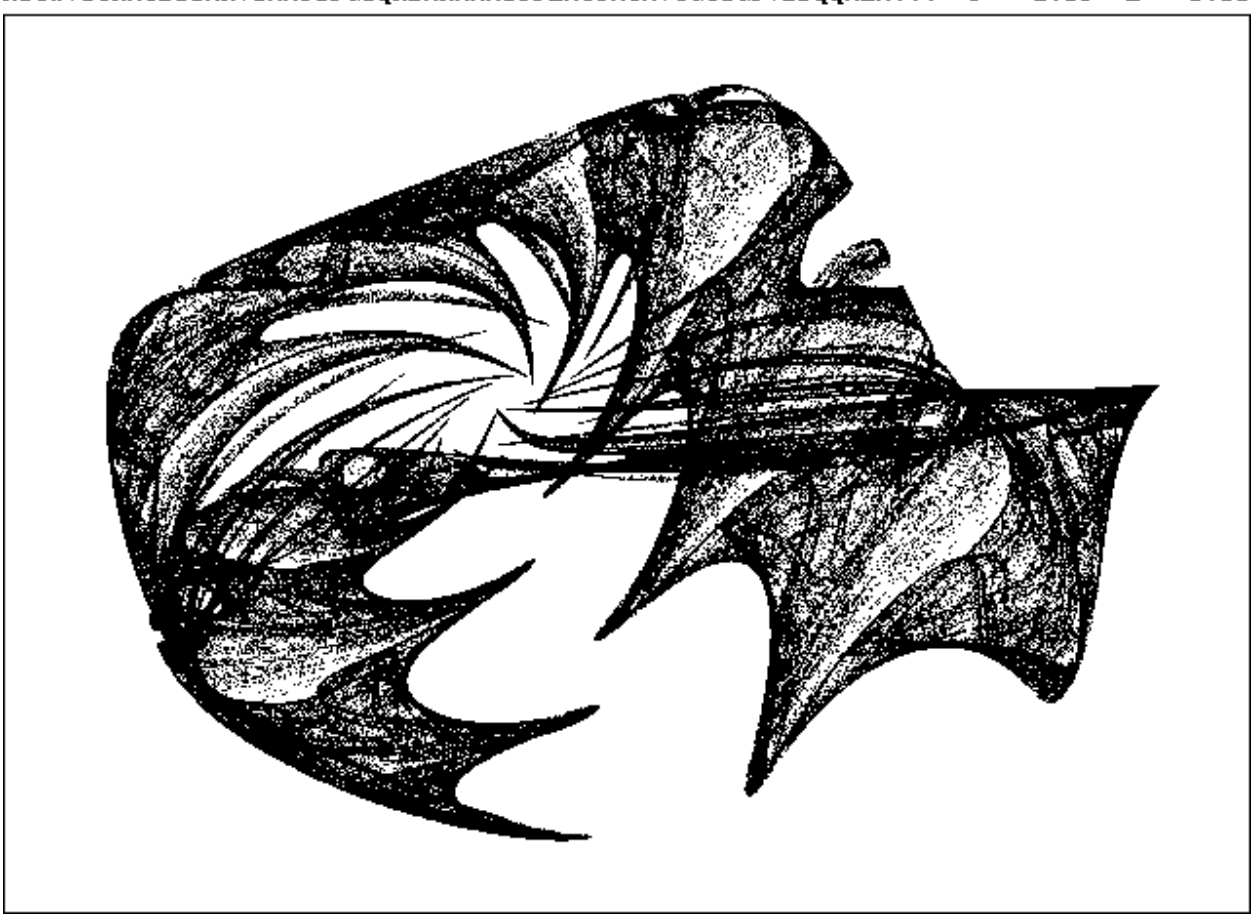


Figure 4-13. Projection of three-dimensional quartic map

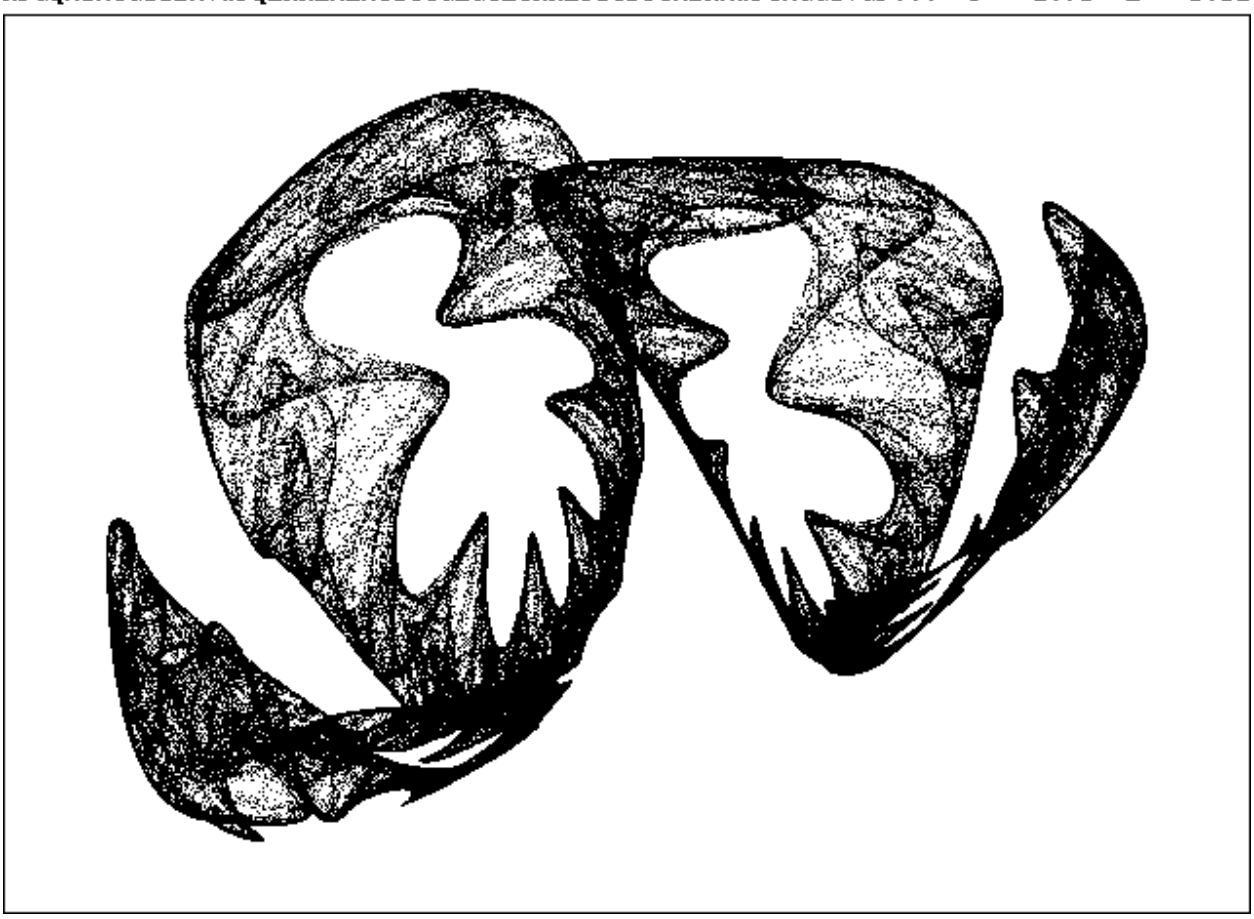


Figure 4-14. Projection of three-dimensional quartic map

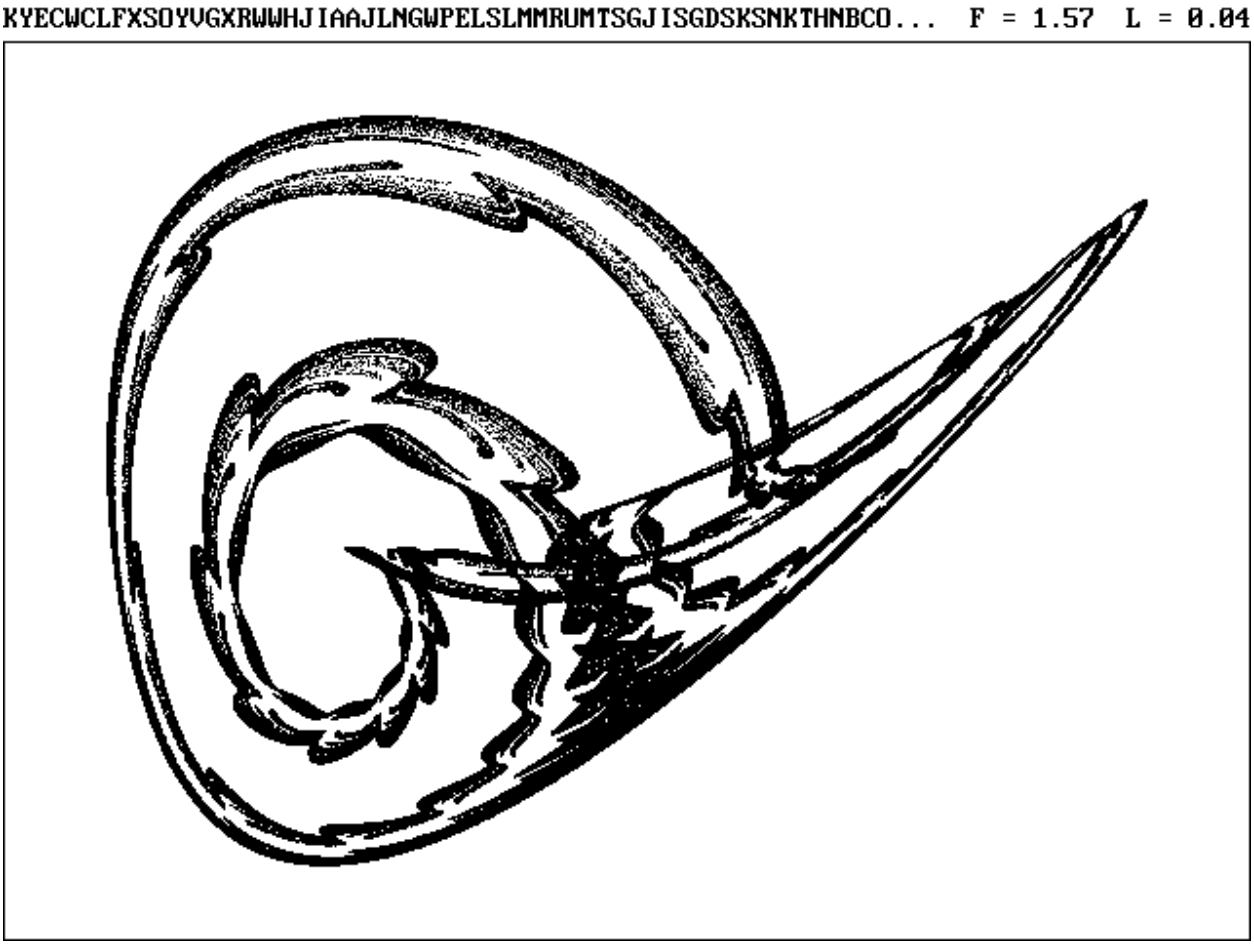


Figure 4-15. Projection of three-dimensional quintic map

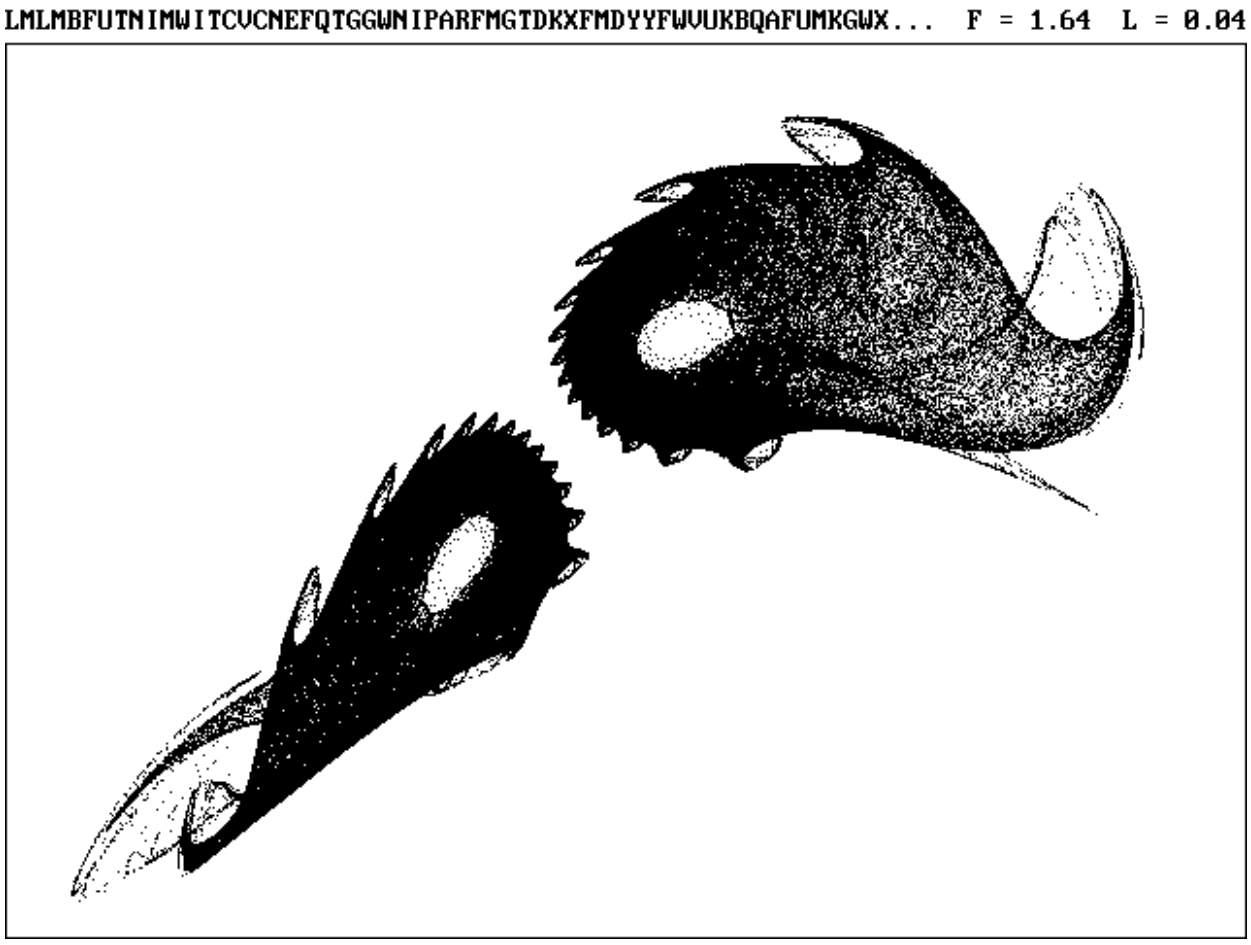
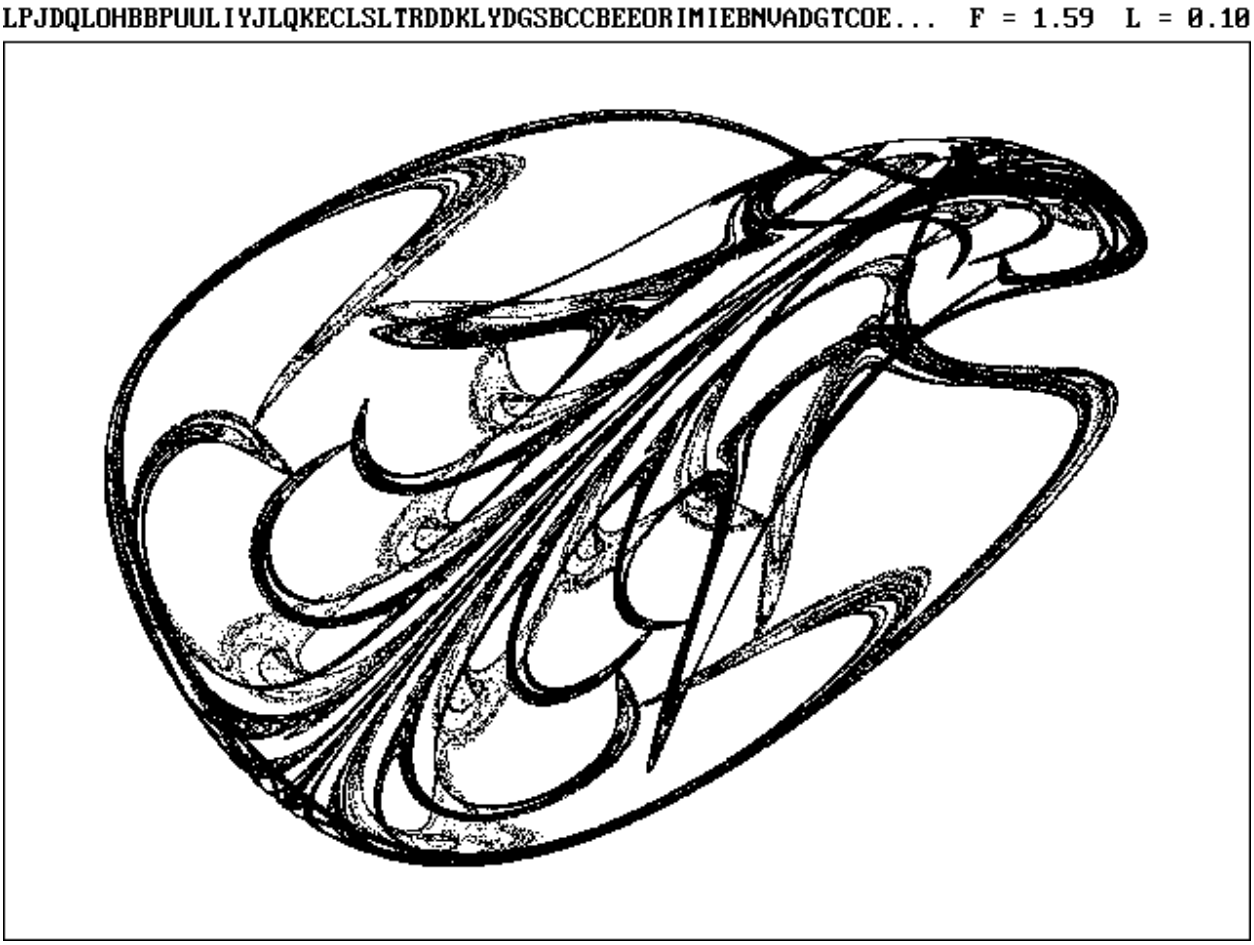


Figure 4-16. Projection of three-dimensional quintic map



On the whole, attractors in three dimensions projected onto a plane are not particularly different or better than the two-dimensional examples of the previous chapter. Ones with high fractal dimensions (near and above 2) tend to be uninteresting when projected onto two dimensions because they are too filled-in. Note also that all the two-dimensional cases are included as special cases of the three-dimen sional ones and that they can be recovered by setting the appropriate coefficients to zero. For example, the Hénon map can be reproduced in three dimensions using the code IWM?M^2PM^5WM^{18}. You may want to try entering this case into the program using the **I** command. Be sure to count the number of Ms very carefully and to use capital letters.

The attractors displayed in the previous figures are projected onto the *XY* plane. They could equally well be projected onto the *YZ* or *ZX* plane. With a bit more effort it would be possible to project them onto a plane inclined at an arbitrary angle. Attractors are most visually appealing when viewed from a particular

direction. The formulas that transform a point with coordinates (*X, Y, Z*) into a two-dimensional projection (X_p, Y_p) with viewing angles (θ, φ) in spherical coordinates are

$$X_p = - X \sin \_ + Y \cos \_$$

$$Y_p = - X \sin \_ \cos \_ - Y \sin \_ \cos \_ + Z \sin \_ \qquad \text{(Equation 4B)}$$

With a sufficiently powerful computer, you could rotate the attractor to produce an animated display. You may want to experiment with these ideas.

The program can be modified in a number of ways to change the orientation of the projection. The simplest (though not very practical) way is just to wait until the search turns up the same attractor viewed from a different angle. Note in Equation 4A, for example, that if you interchange the coefficients in an appropriate way, the result is to replace *X* with *Y*, *Y* with *Z*, and *Z* with *X*. If the attractor code were `IABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^`, the code `IKPQRNSTOLMUZ[\X]^YVWAFGHDIJEBC` would produce a projection of the same attractor onto the *YZ* plane, and the code `IU]^Y\VWXZ[AIJEHBCDFGKSTORLMNPQ` would produce a projection of the attractor onto the *ZX* plane. Figures 4-17 and 4-18 show the result of applying these transformations to the attractor in Figure 4-4.

Figure 4-17. Attractor in Figure 4-4 projected onto the YZ plane

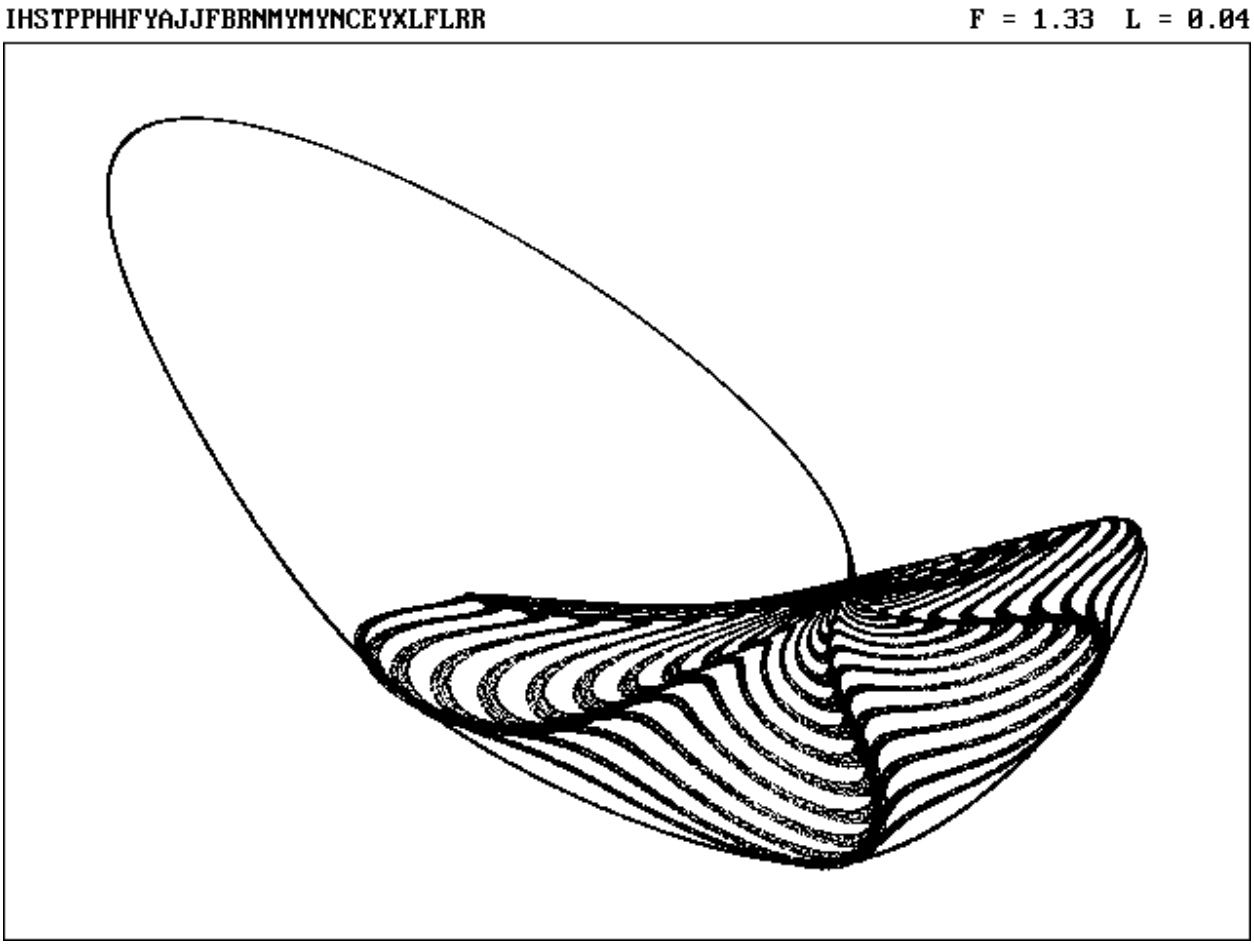
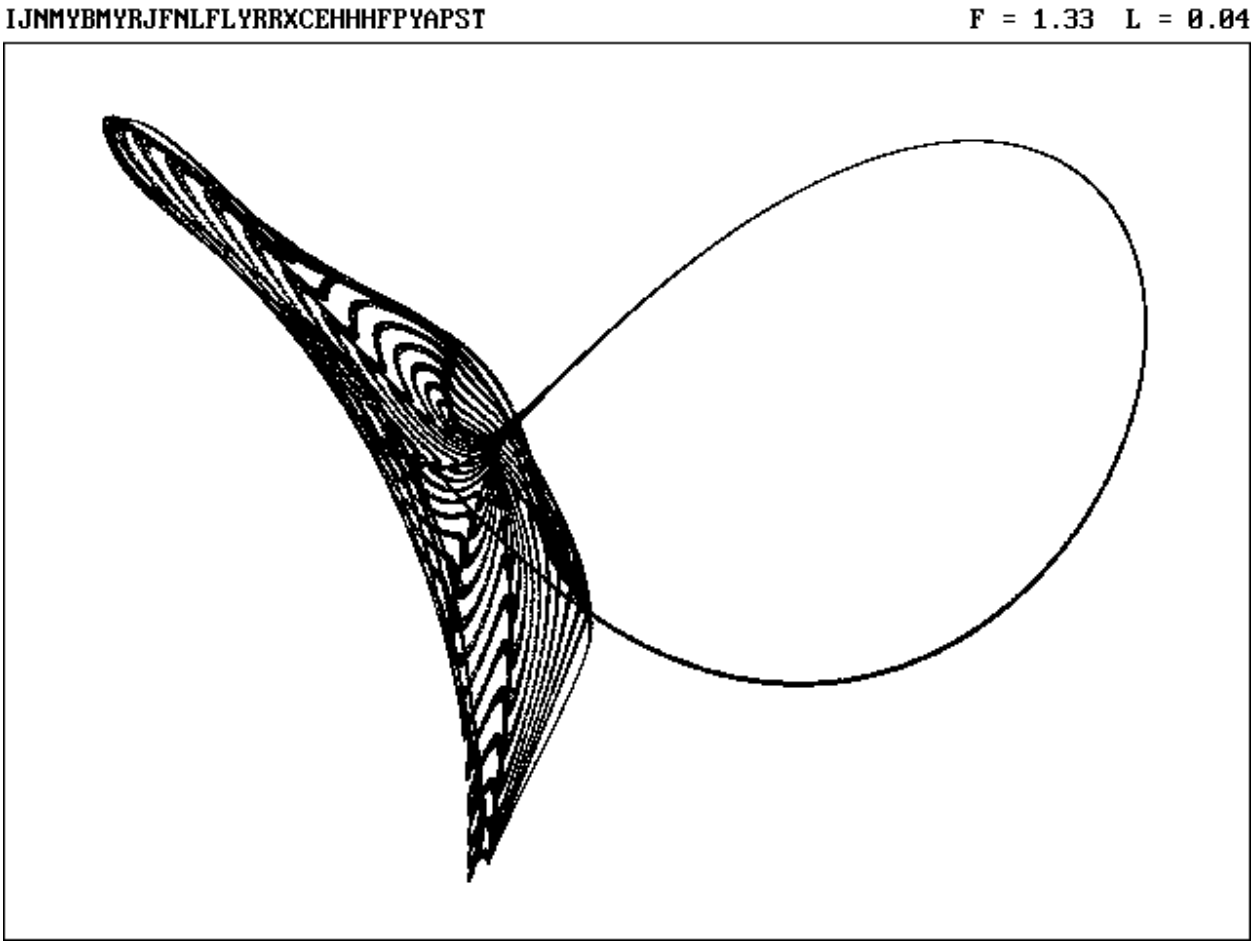


Figure 4-18. Attractor in Figure 4-4 projected onto the ZX plane



4.2 Shadows

In the previous figures, the attractors were projected as if illuminated from directly along the line of sight. Now suppose the point source of illumination is moved slightly off to one side and you observe the attractor against a background screen. Each point making up the attractor appears as a dot of reflected light above the background plane and produces a shadow point opposite the illumination a distance proportional to the position of the point above the plane. For this purpose, we assume the most distant point of the attractor is touching the screen and the nearest point is out of the screen a distance equal to its width. We'll assume it is illuminated from above your left shoulder so that the shadow is below and to the right, in keeping with the Microsoft style guidelines as exhibited in recent versions of Windows.

To produce a shadow, we need a background shade of gray intermediate between the black and white that we have been using so far. If your computer has at least EGA graphics, this poses no difficulty. There are two grays, COLOR 8, which is 25% illuminated, and COLOR 7, which is 75% illuminated. We'll use COLOR 8, which is the darker of the two. For convenience, Table 4-1 lists the 16 default colors provided with SCREEN modes 7 through 13.

Table 4-1. Default EGA and VGA colors for SCREEN modes 7 to 13

Number	Color	Number	Color
0	Black	8	Gray
1	Blue	9	Bright blue
2	Green	10	Bright green
3	Cyan	11	Bright cyan
4	Red	12	Bright red
5	Magenta	13	Bright magenta
6	Brown	14	Yellow
7	White	15	Bright white

If you have CGA graphics, you might try plotting the points in white (COLOR 3) and their shadow in black (COLOR 0) on a magenta (COLOR 2) background using SCREEN 1 (320 by 200 resolution) and PALETTE 1. In any case, it may help to adjust the intensity control on the monitor for an easily visible shadow.

Since we have another shade of gray available, we can use it to control the brightness of the points plotted. The first time a screen pixel is illuminated by a point on the attractor, we will use low-intensity white (COLOR 7), and subsequent times we will use high-intensity white (COLOR 15). If your computer has a monochrome graphics monitor that maps the other colors to various shades of gray, you can extend this technique to provide additional gray levels, producing an attractor whose brightness corresponds to the frequency its various regions are visited. This trick helps to compensate for the limited spatial resolution of the computer screen.

It also helps to draw a grid on the background to make it more obvious that the attractor is sitting above the screen. The grid is drawn in black (COLOR 0), the same as the shadow.

If your computer has at least EGA capability, **PROG12** produces the desired shadow display. It allows you to toggle between projections and shadows by pressing the **R** key.

PROG12. Changes required in PROG11 to display shadows

```
1000 REM THREE-D MAP SEARCH (With Shadow Display)

1020 DIM XS(499), YS(499), ZS(499), A(504), V(99), XY(4), XN(4), COLR%(15)

1120 TRD% = 1                    'Display third dimension as shadow



1370 GOSUB 5600                  'Set colors



2300    GOSUB 5000                       'Plot point on screen



3210 IF D% < 3 THEN GOTO 3310
```

```
3230    IF TRD% = 1 THEN LINE (XL, YL)-(XH, YH), COLR%(1), BF: GOSUB 5400

3430 TIA = .05               'Tangent of illumination angle

3440 XZ = -TIA * (XMAX - XMIN) / (ZMAX - ZMIN)

3450 YZ = TIA * (YMAX - YMIN) / (ZMAX - ZMIN)


3630 IF Q$ = "" OR INSTR("ADIPRSX", Q$) = 0 THEN GOSUB 4200

3760 IF Q$ = "R" THEN TRD% = (TRD% + 1) MOD 2: T% = 3: IF N > 999 THEN N = 999


4460    PRINT TAB(27); "R: Third dimension is ";

4470        IF TRD% = 0 THEN PRINT "projection"

4480        IF TRD% = 1 THEN PRINT "shadow    "


5000 REM Plot point on screen

5060 IF TRD% = 0 THEN PSET (XP, YP)

5070 IF TRD% <> 1 THEN GOTO 5130

5090    C% = POINT(XP, YP)

5100   IF C% = COLR%(2) THEN PSET (XP, YP), COLR%(3) ELSE IF C% <> COLR%(3) THEN
PSET (XP, YP), COLR%(2)

5110    XP = XP - XZ * (Z - ZMIN): YP = YP - YZ * (Z - ZMIN)

5120    IF POINT(XP, YP) = COLR%(1) THEN PSET (XP, YP), 0

5130 RETURN


5400 REM Plot background grid

5410 FOR I% = 0 TO 15          'Draw 15 vertical grid lines
```

```
5420    XP = XMIN + I% * (XMAX - XMIN) / 15
5430    LINE (XP, YMIN)-(XP, YMAX), 0
5440 NEXT I%
5450 FOR I% = 0 TO 10           'Draw 10 horizontal grid lines
5460    YP = YMIN + I% * (YMAX - YMIN) / 10
5470    LINE (XMIN, YP)-(XMAX, YP), 0
5480 NEXT I%
5490 RETURN


5600 REM Set colors
5620 COLR%(0) = 0: COLR%(1) = 8: COLR%(2) = 7: COLR%(3) = 15
5720 RETURN
```

The angle of illumination is determined by the .05 in line 3430. You might try different values. The value of .05 is the tangent of both the horizontal and vertical angle that the source of illumination makes with the perpendicular to the plane. The angle is about 3 degrees toward the left and 3 degrees toward the top of the figure. Sample attractors produced by **PROG12** are shown in Figures 4-19 through 4-34.

Figure 4-19. Three-dimensional quadratic map with shadows

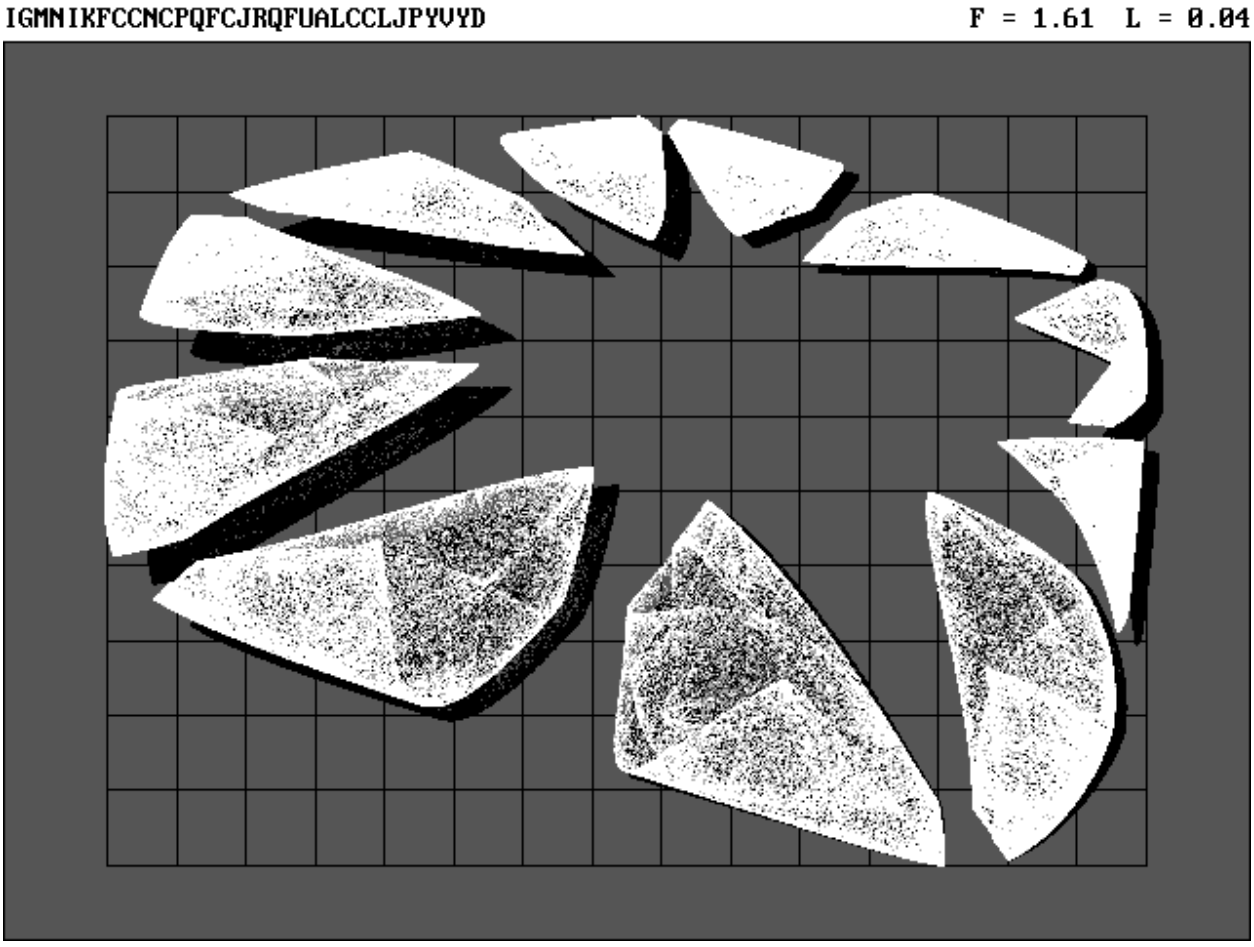


Figure 4-20. Three-dimensional quadratic map with shadows

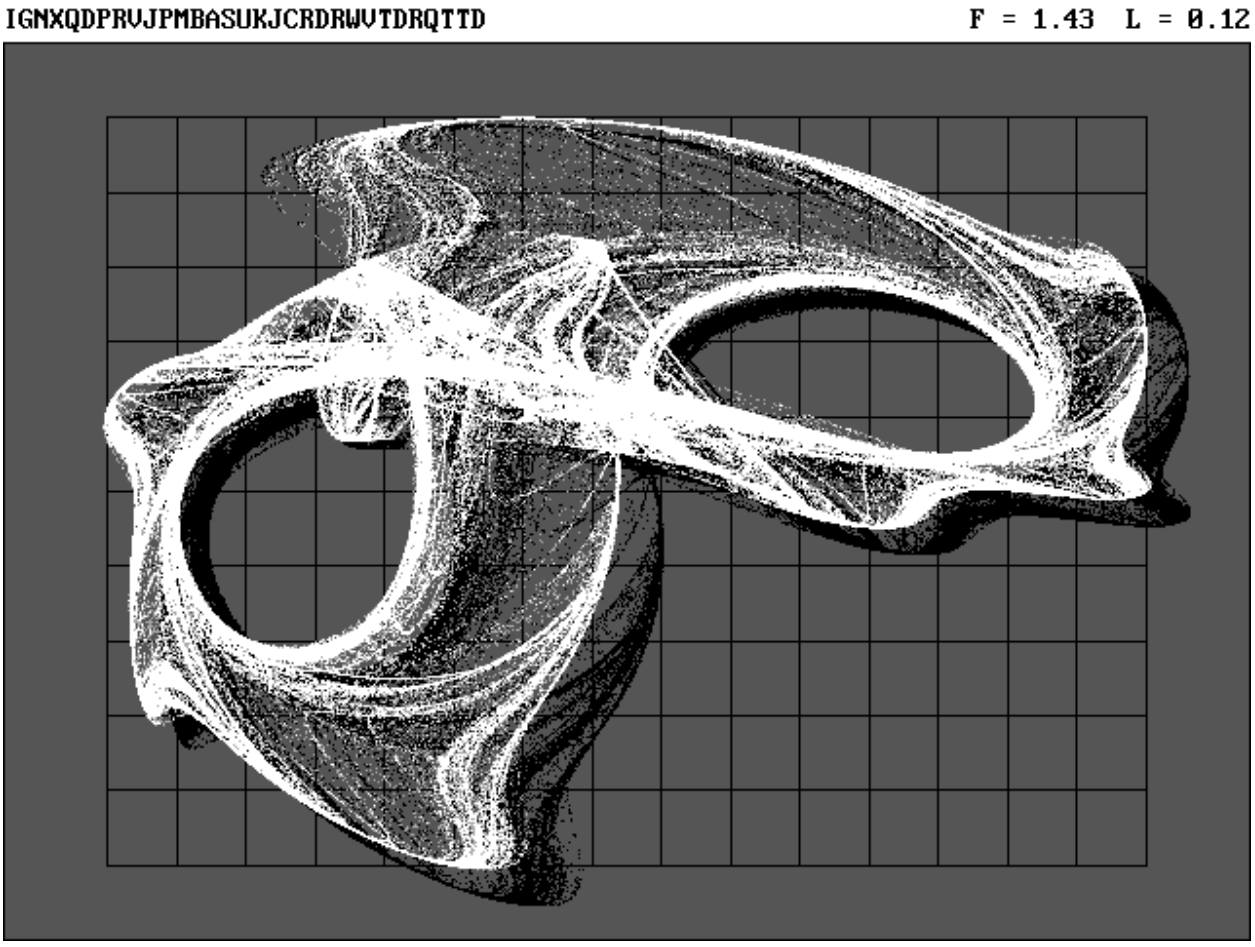


Figure 4-21. Three-dimensional quadratic map with shadows

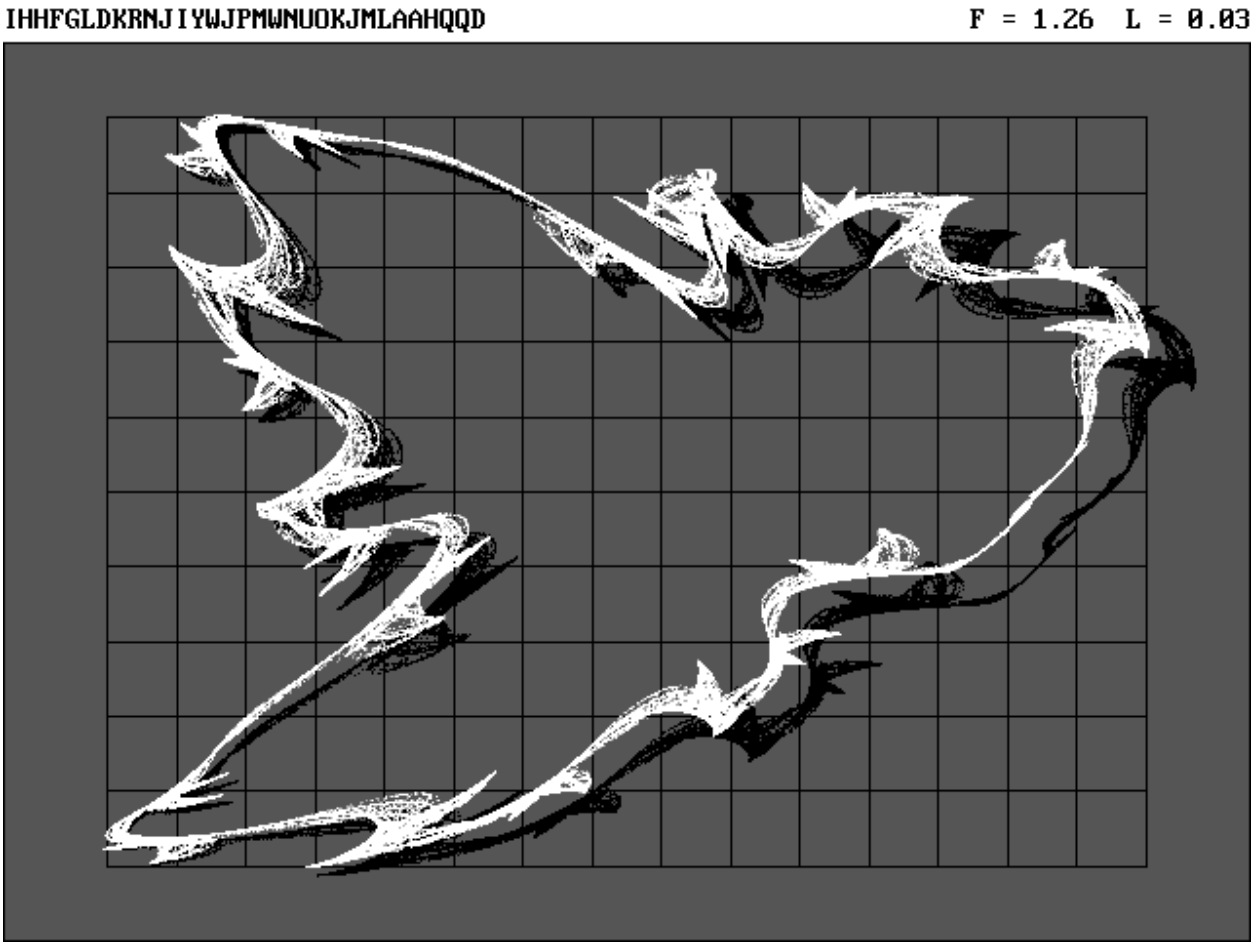


Figure 4-22. Three-dimensional quadratic map with shadows

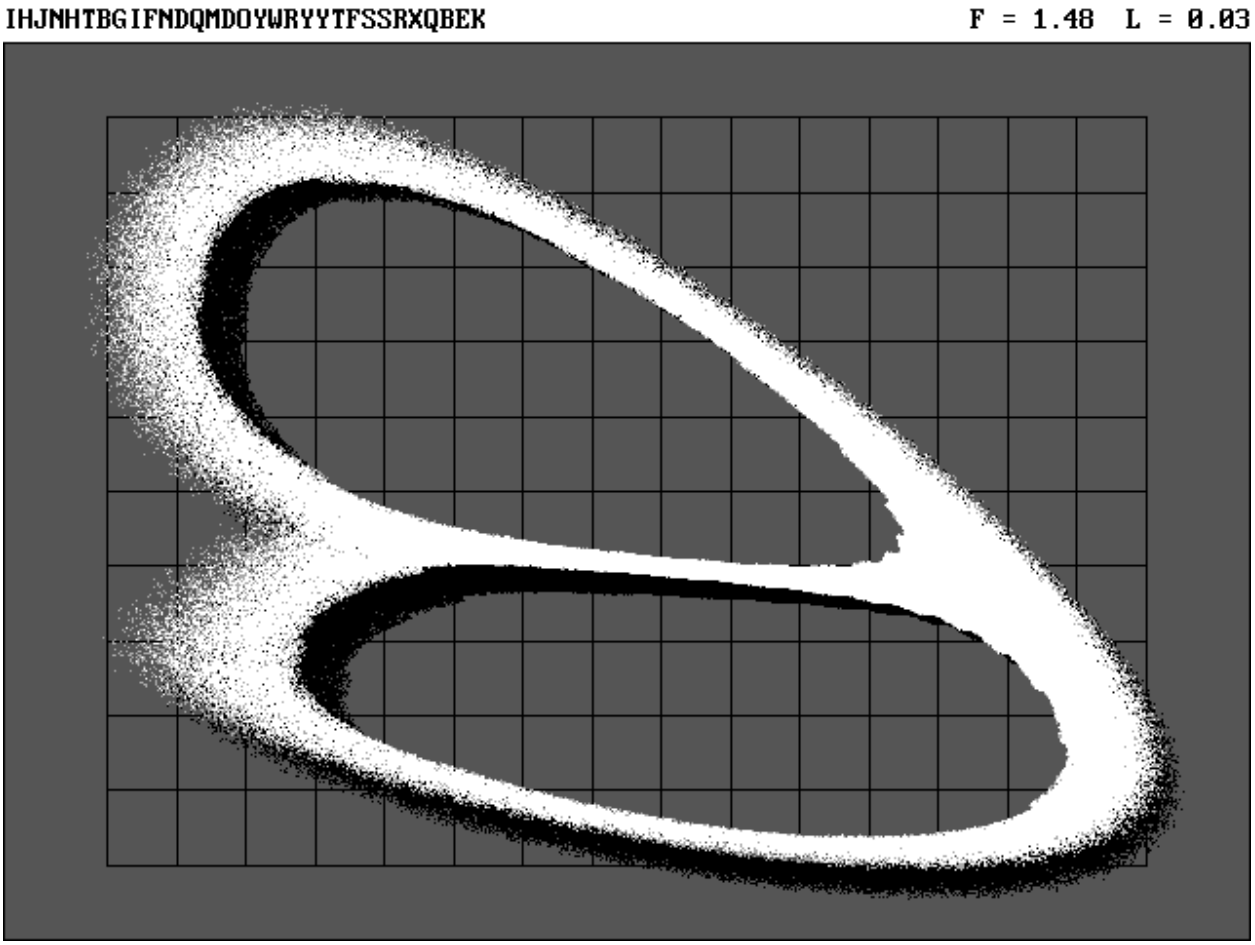


Figure 4-23. Three-dimensional quadratic map with shadows

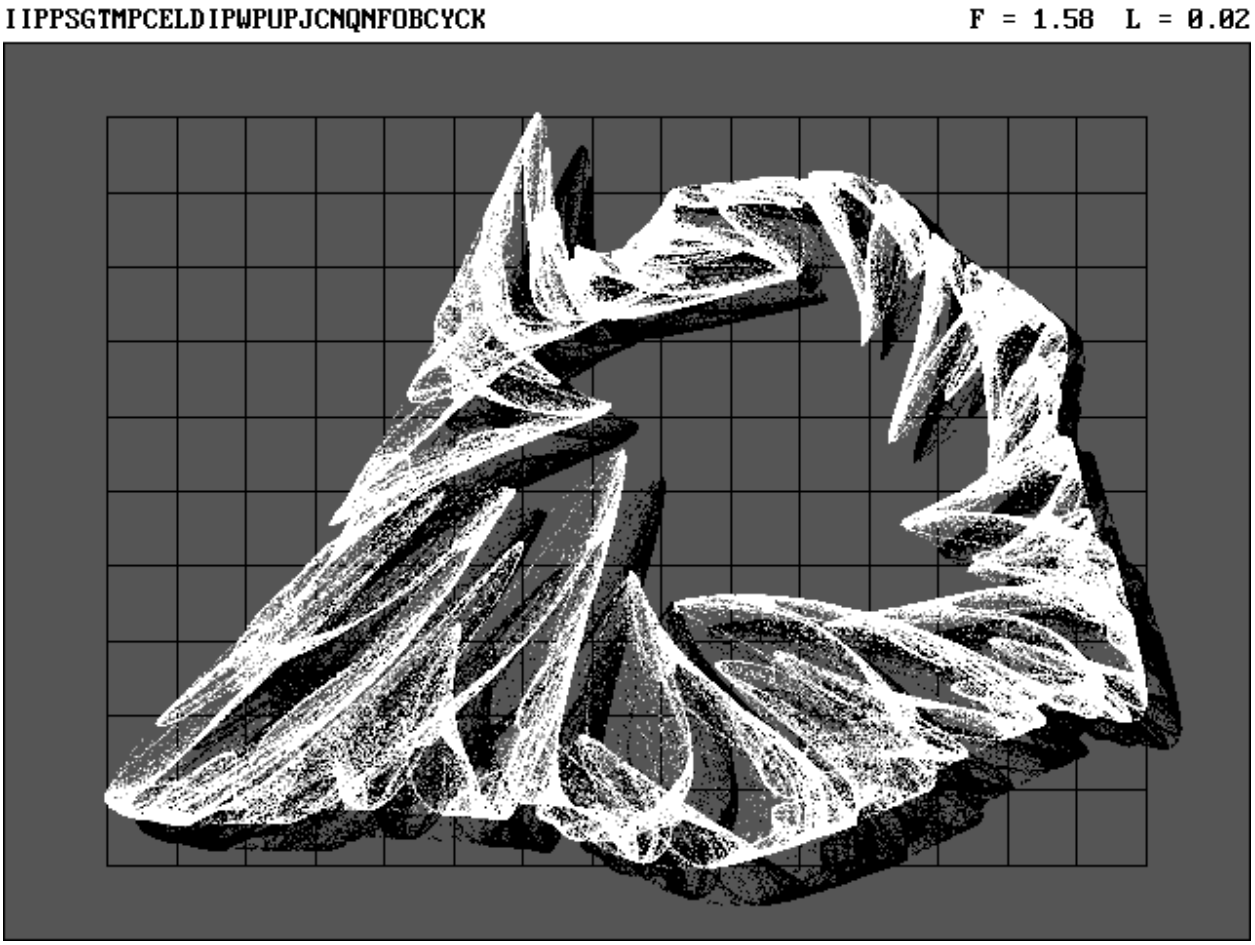


Figure 4-24. Three-dimensional quadratic map with shadows

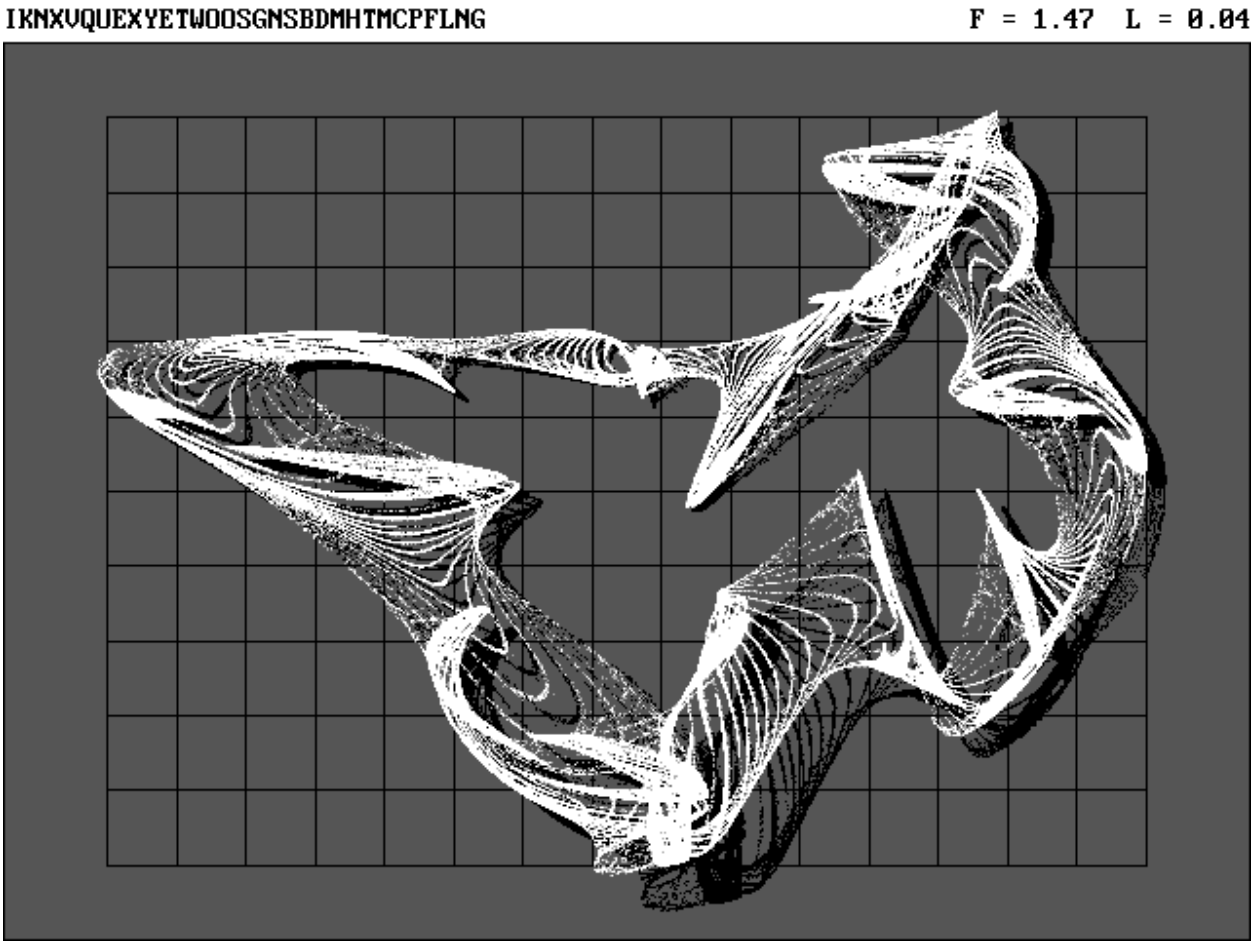


Figure 4-25. Three-dimensional quadratic map with shadows

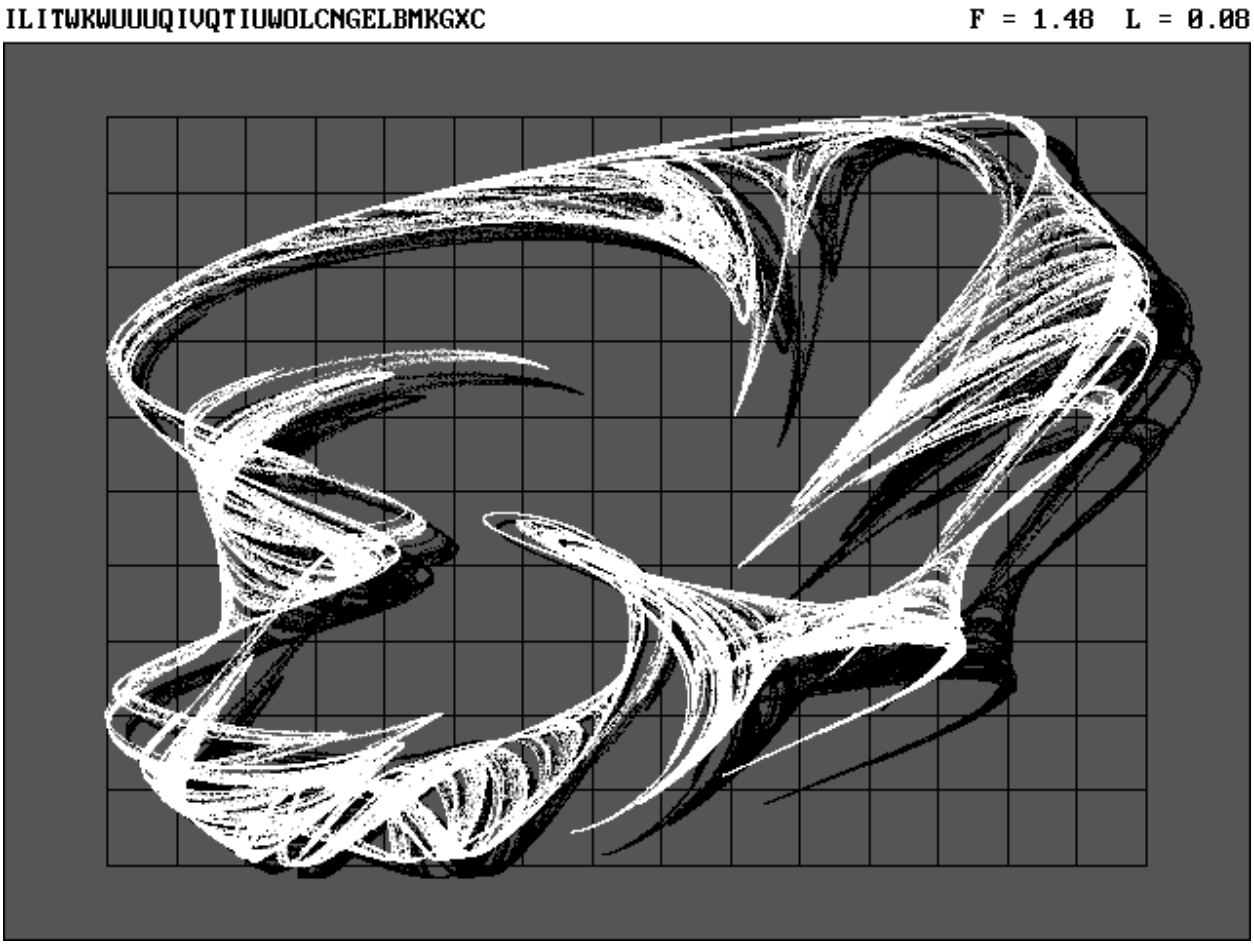


Figure 4-26. Three-dimensional quadratic map with shadows

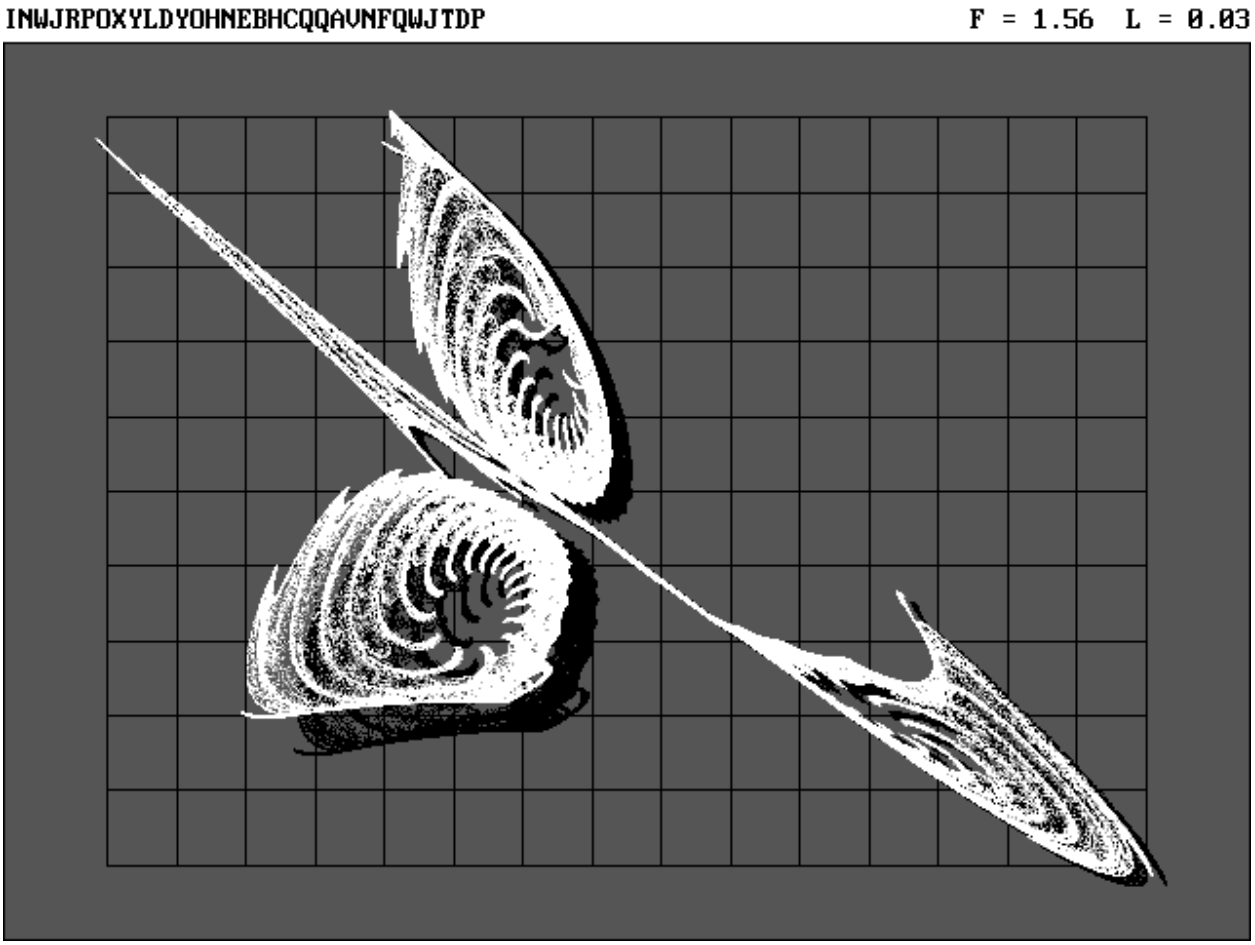


Figure 4-27. Three-dimensional quadratic map with shadows

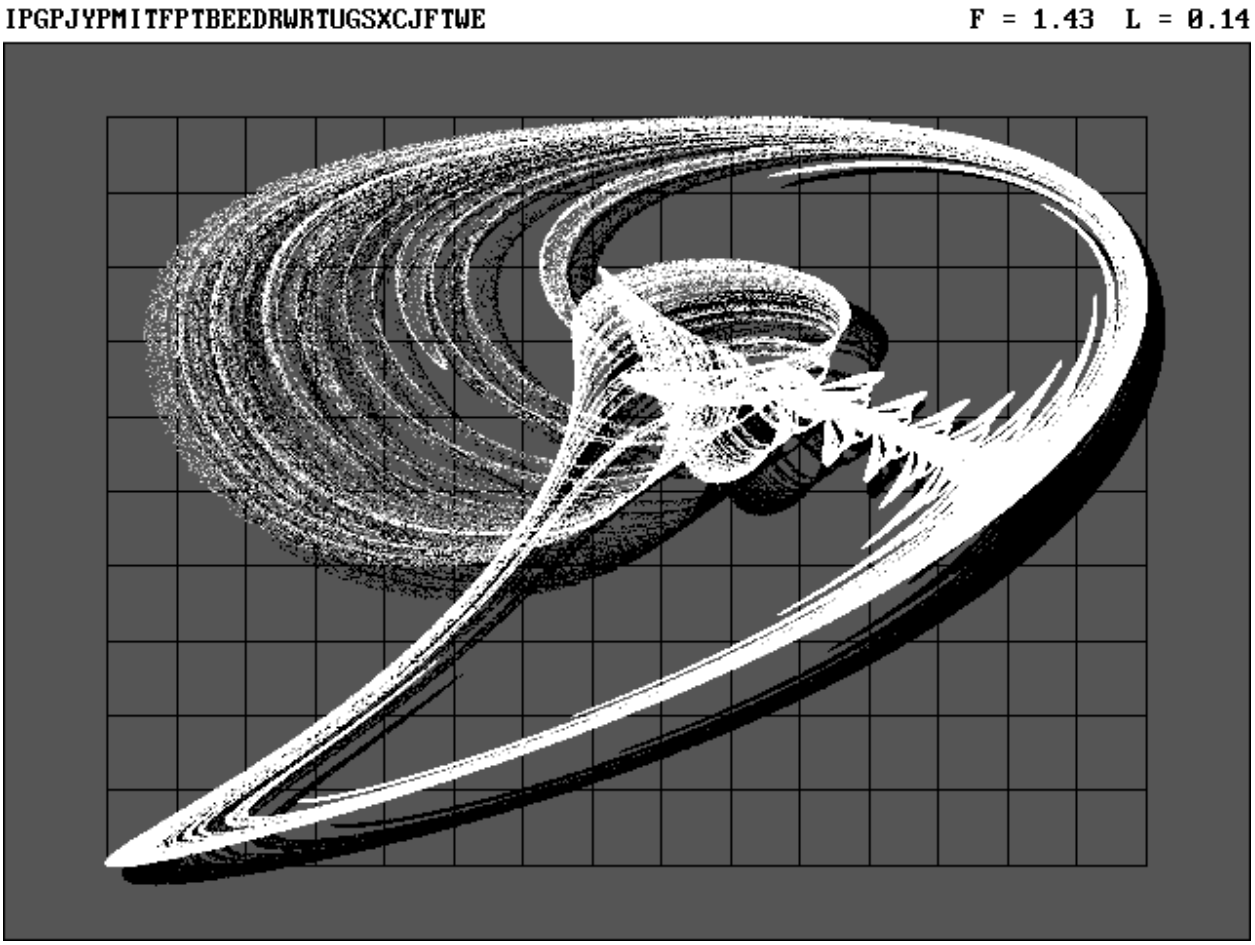


Figure 4-28. Three-dimensional quadratic map with shadows

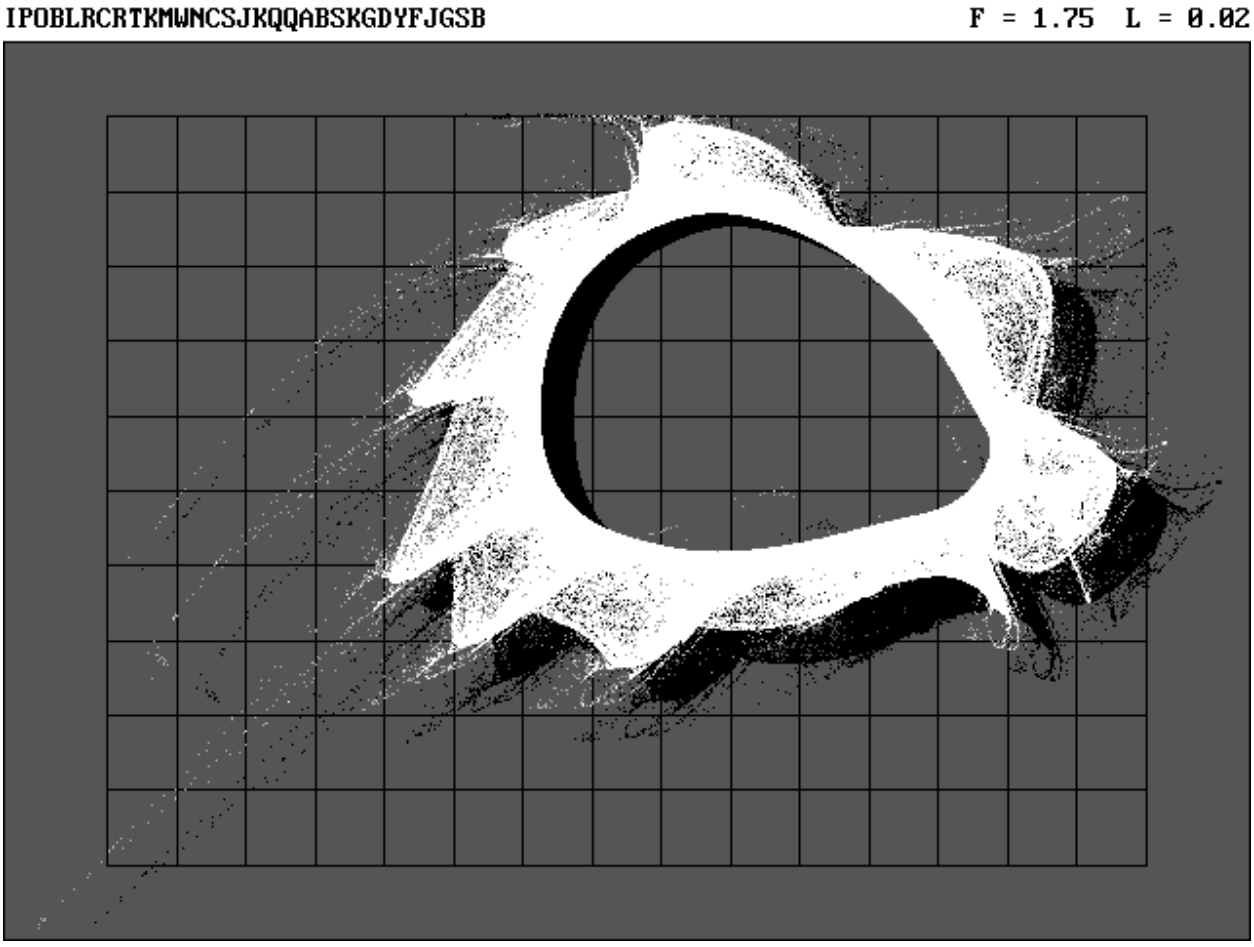


Figure 4-29. Three-dimensional quadratic map with shadows

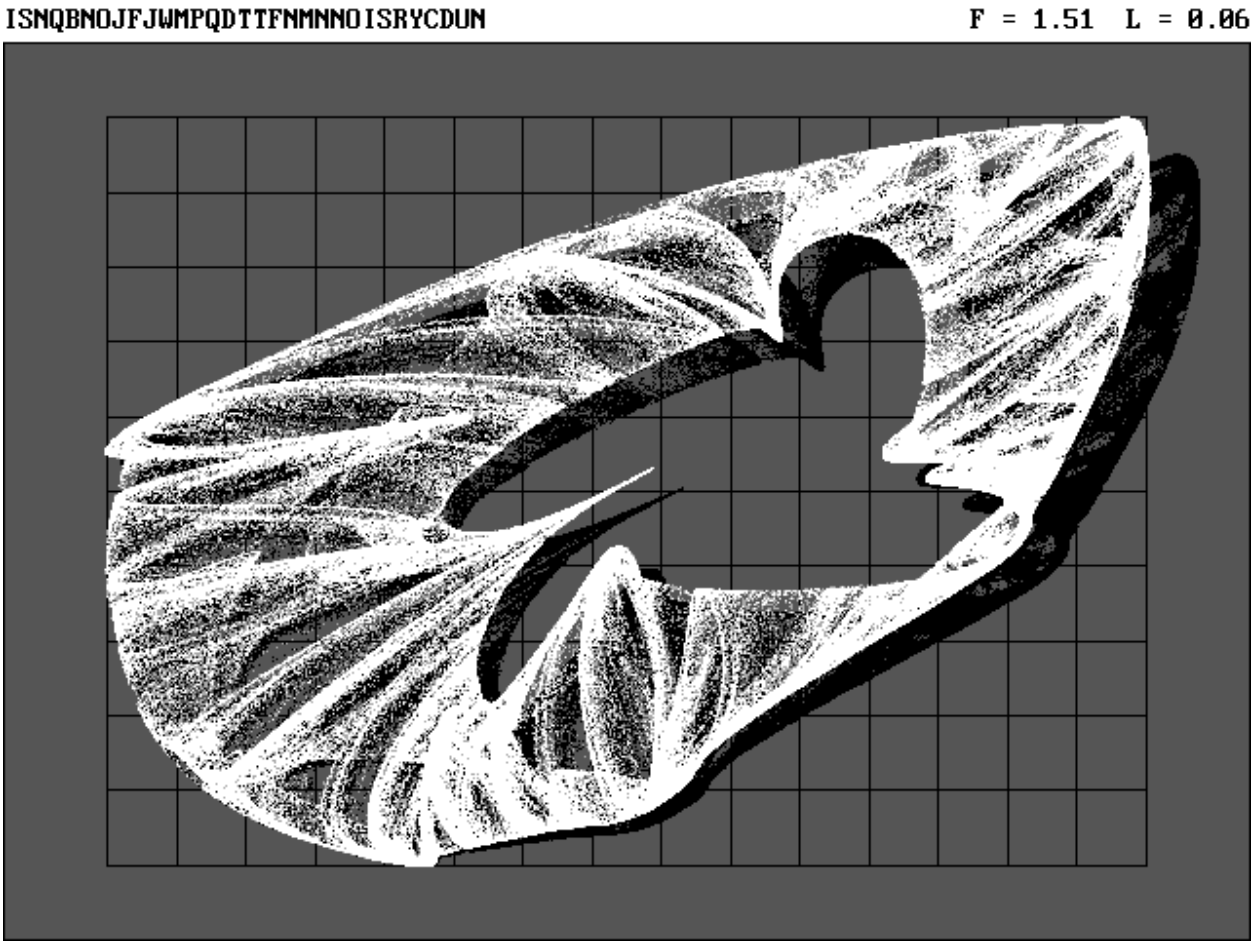


Figure 4-30. Three-dimensional cubic map with shadows

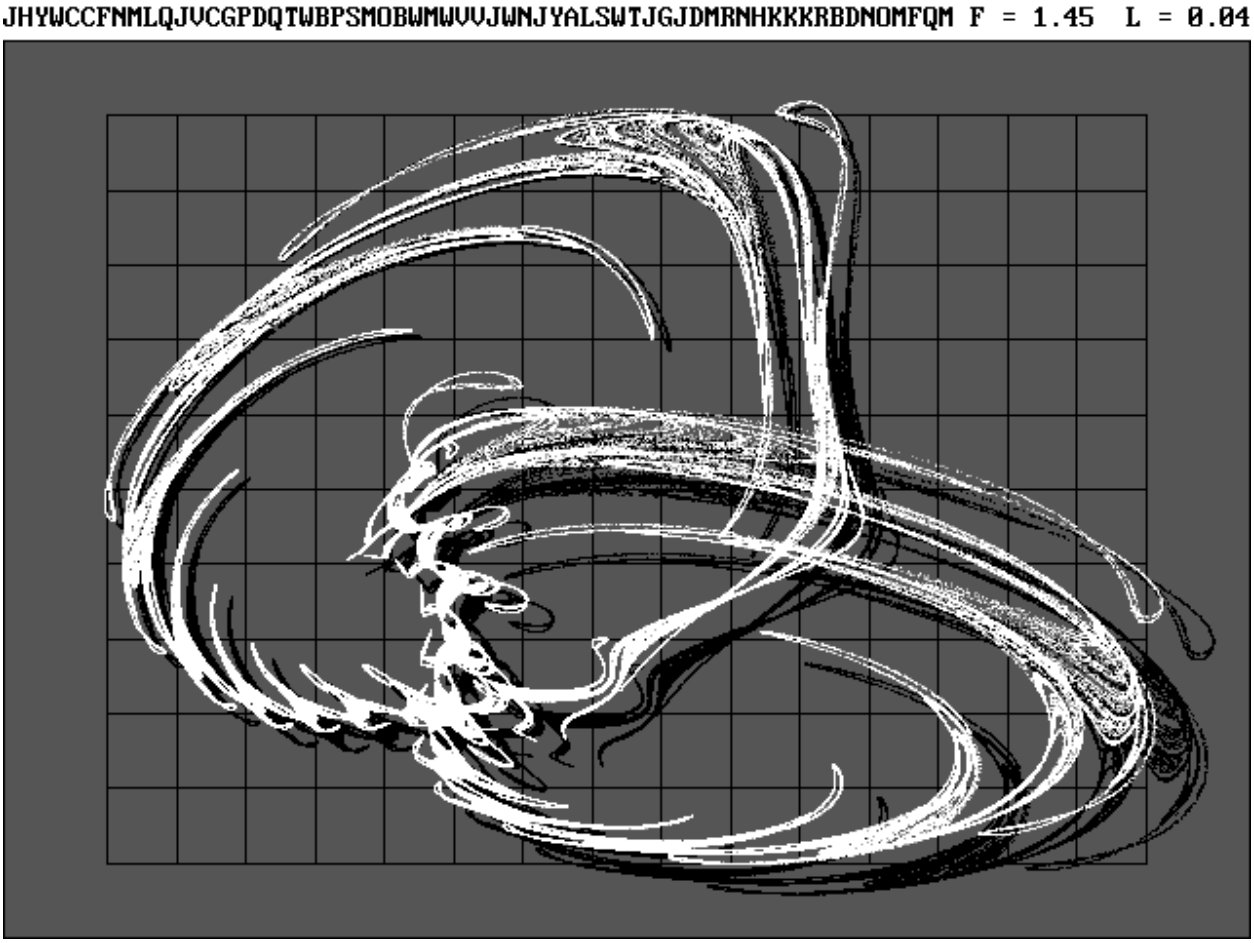


Figure 4-31. Three-dimensional quartic map with shadows

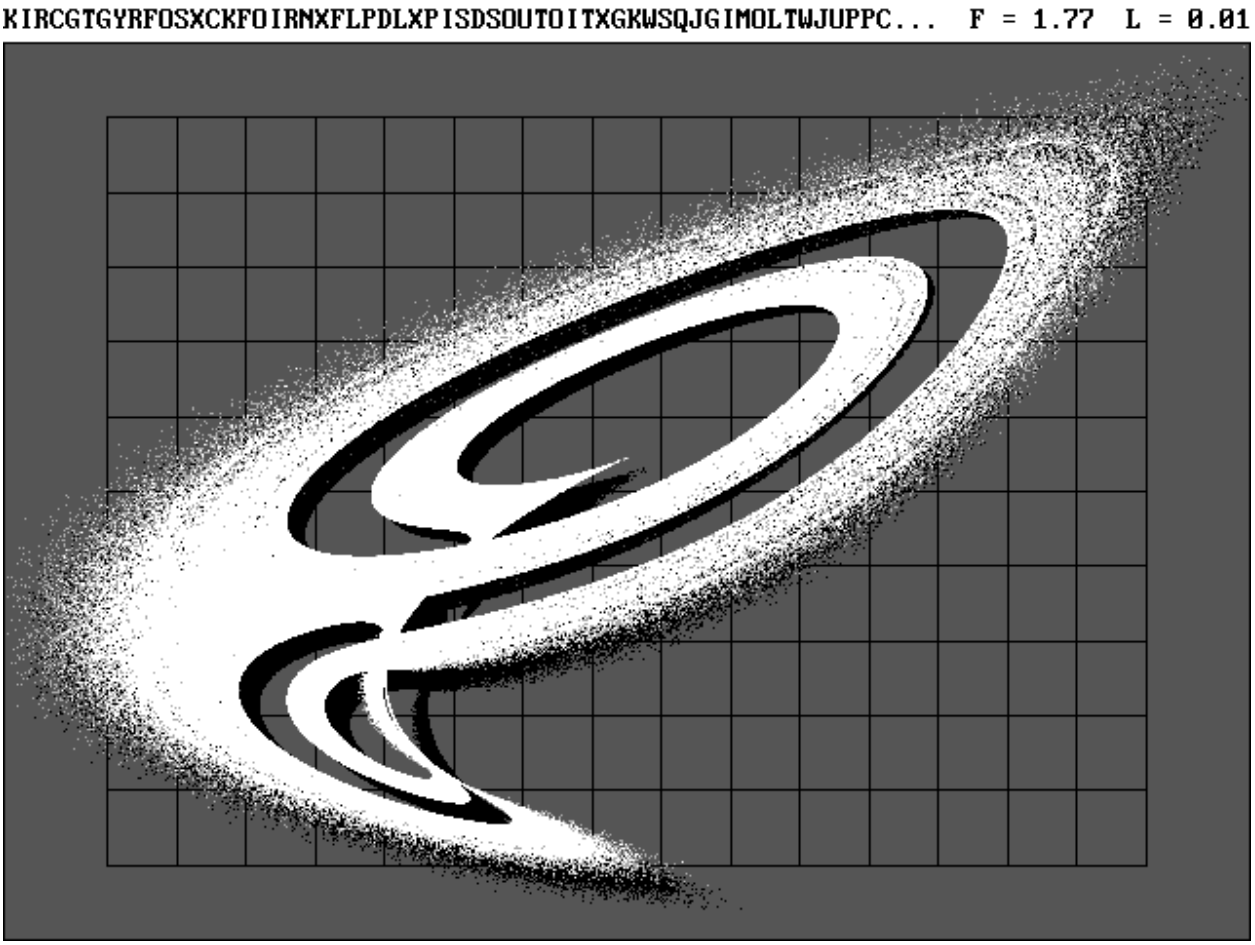


Figure 4-32. Three-dimensional quartic map with shadows

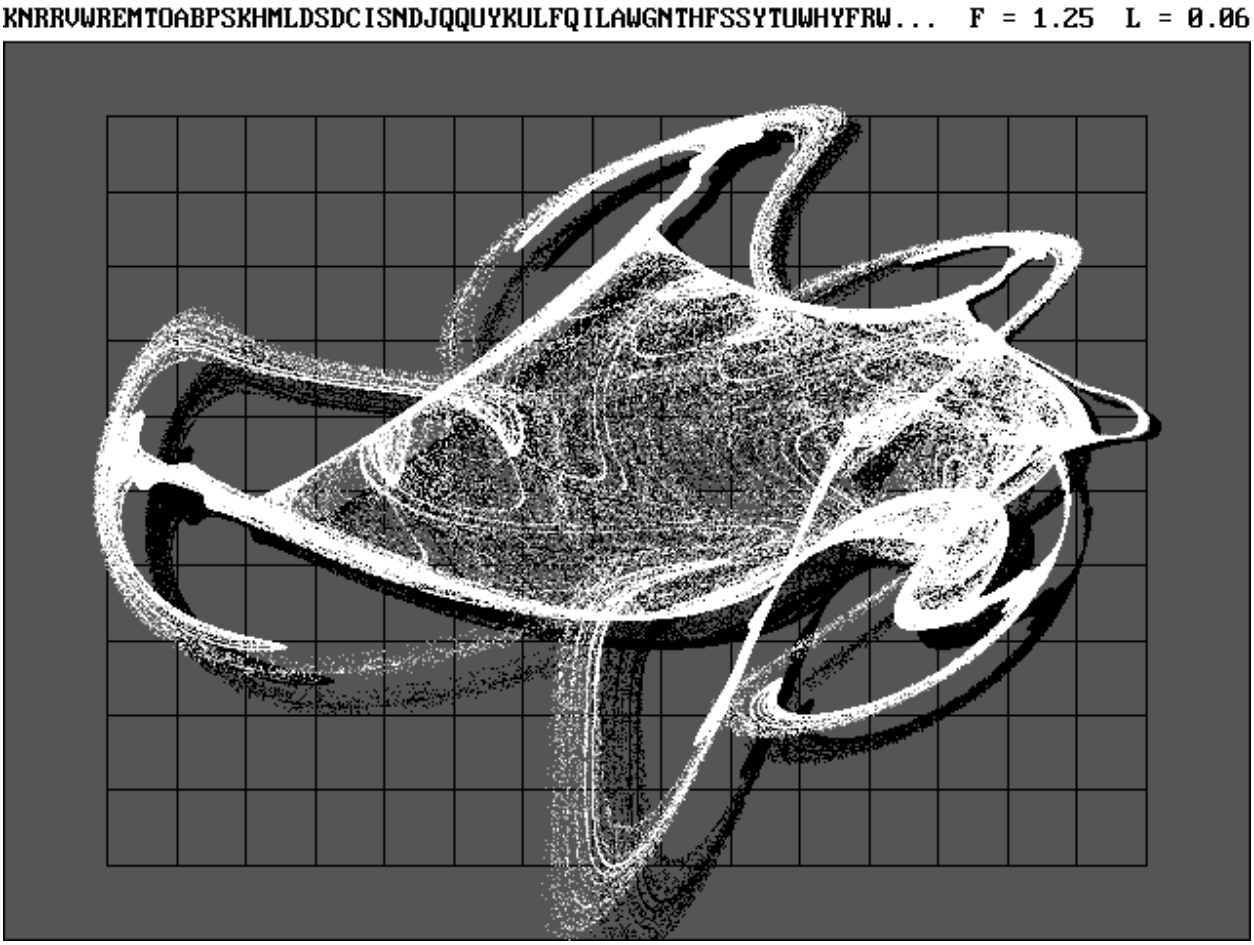


Figure 4-33. Three-dimensional quintic map with shadows

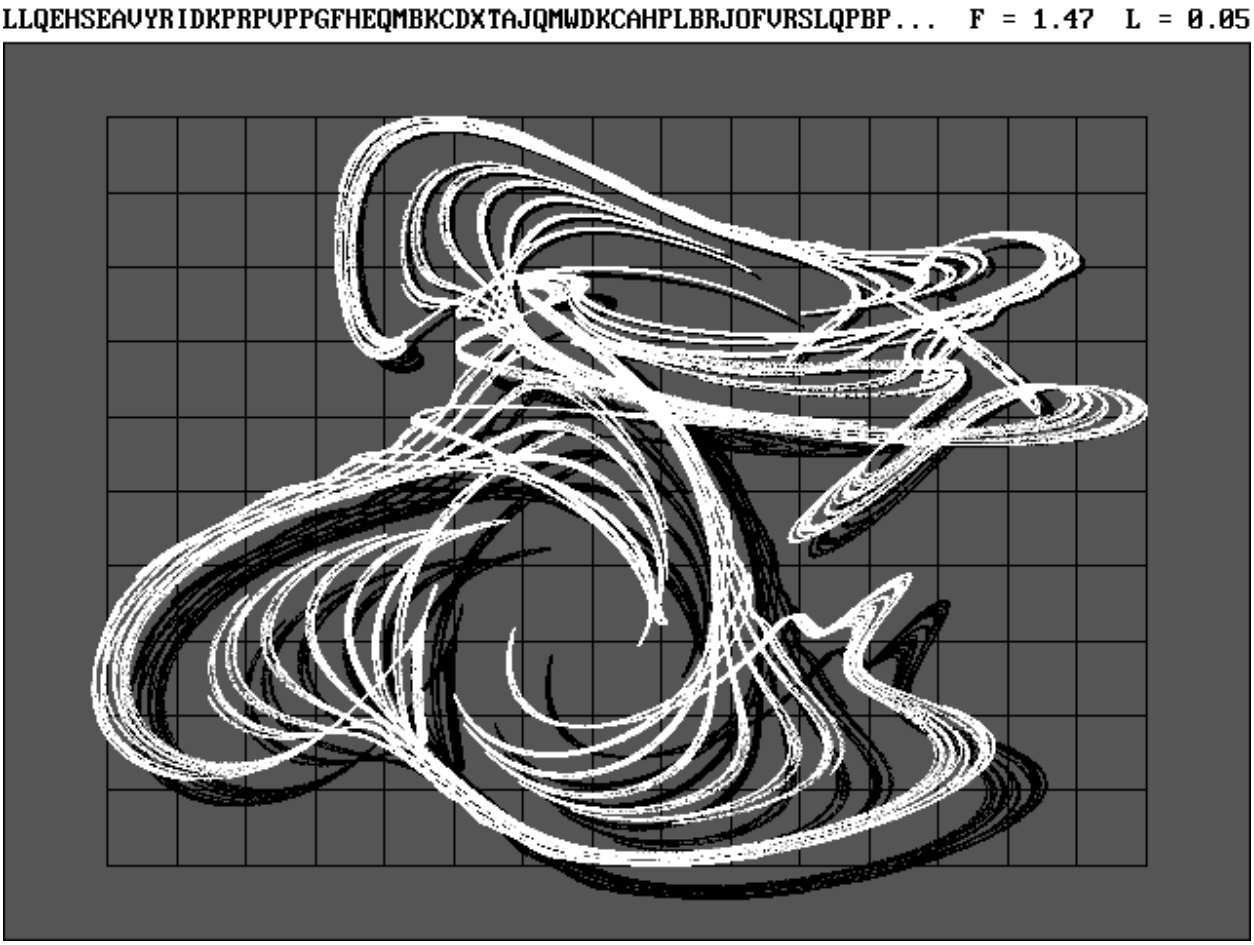
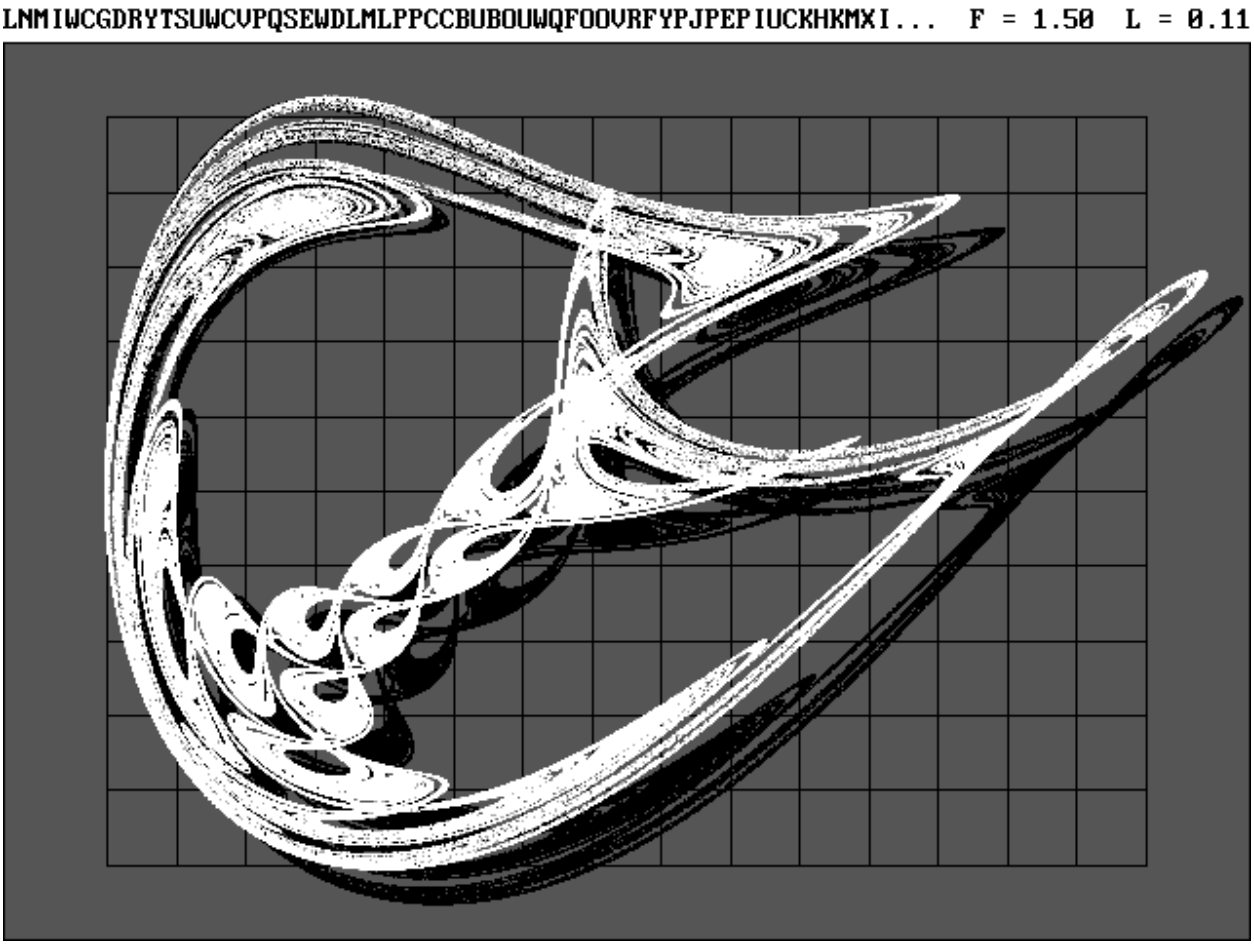


Figure 4-34. Three-dimensional quintic map with shadows



Look closely at the figures with shadows and you can see that it is hard to tell whether one portion of the attractor lies above or below another portion. One reason for this is that we have not allowed the closer portion of the attractor to cast a shadow on the more distant portion. To do so requires a complicated program, which will be left as a challenge for you.

If you attempt to improve the shadow display in this way, you must store in an array the largest *Z* value corresponding to each screen pixel. With VGA (640 by 480), you need 600 kilobytes (K), even if you convert the *Z* values into integers. Most versions of BASIC limit the size of arrays to 64 K, and the disk operating system usually limits the total program size to about 600 K. Thus you probably need to use a lower screen resolution or devise a more compact coding scheme. For example, if you use only 16 values of *Z*, you can store two screen pixels per byte, which is four times better than storing the *Z* value of each pixel as a two-byte integer. Alternately, you

might store the *Z* values of only those pixels that are illuminated, but then you must devise a quick way to locate the proper element in the array corresponding to each pair of screen coordinates.

Before each point on the attractor is plotted, you must be sure it doesn't fall in the shadow of a previously plotted point. If it doesn't, then it can be plotted, but then you have to determine whether it occludes any previously plotted point. Alternatively, you can first plot all the points and then scan the image starting from the side toward the illumination, blocking out any points that fall in the shadow of another point.

4.3 Bands

Another way to display the third dimension is with elevation contours such as those found on topographic maps. With enough points, you could plot only those that have specific values of *Z*. Of course the chance that a point has any particular exact value of *Z* is negligibly small, and so the points would accumulate on the screen very slowly. To make the method work, you have to plot all the points that lie within bands centered on the desired values.

You have freedom to choose the width of the bands. With narrow bands, the contours resemble distinct lines, but they form very slowly. With wide bands, the gaps between the bands are hard to see. By making the bands half as wide as the spacing between the contours, the bright and dark spaces are equal in width, and they form rapidly and are easy to see.

You also need to decide how many contours to use. You need at least several to make the method work, but if you use too many, they begin to run together at modest screen resolution and number of iterations. For the cases shown here, we use 15 bands as a reasonable compromise.

Since we used a four-level gray scale to produce the shadows in the previous displays, we will also use it here to give the bands a softer shading. Of course, this requires a computer with at least EGA graphics. If your computer has CGA graphics, you will see only two shades (black and white) with 30 bands in SCREEN mode 2.

The changes required in the program to produce contour bands are shown in **PROG13**.

PROG13. Changes required in PROG12 to display contour bands

```
1000 REM THREE-D MAP SEARCH (With Contour Bands)

1120 TRD% = 2                    'Display third dimension as bands



3760 IF Q$ = "R" THEN TRD% = (TRD% + 1) MOD 3: T% = 3: IF N > 999 THEN N = 999



4490        IF TRD% = 2 THEN PRINT "bands     "



5130 IF TRD% = 2 THEN PSET (XP, YP), COLR%(INT(60 * (Z - ZMIN) / (ZMAX - ZMIN)
+ 4) MOD 4)

5240 RETURN
```

Some sample attractors with contour bands produced by **PROG13** are shown in Figures 4-35 through 4-50.

Figure 4-35. Three-dimensional quadratic map with contour bands

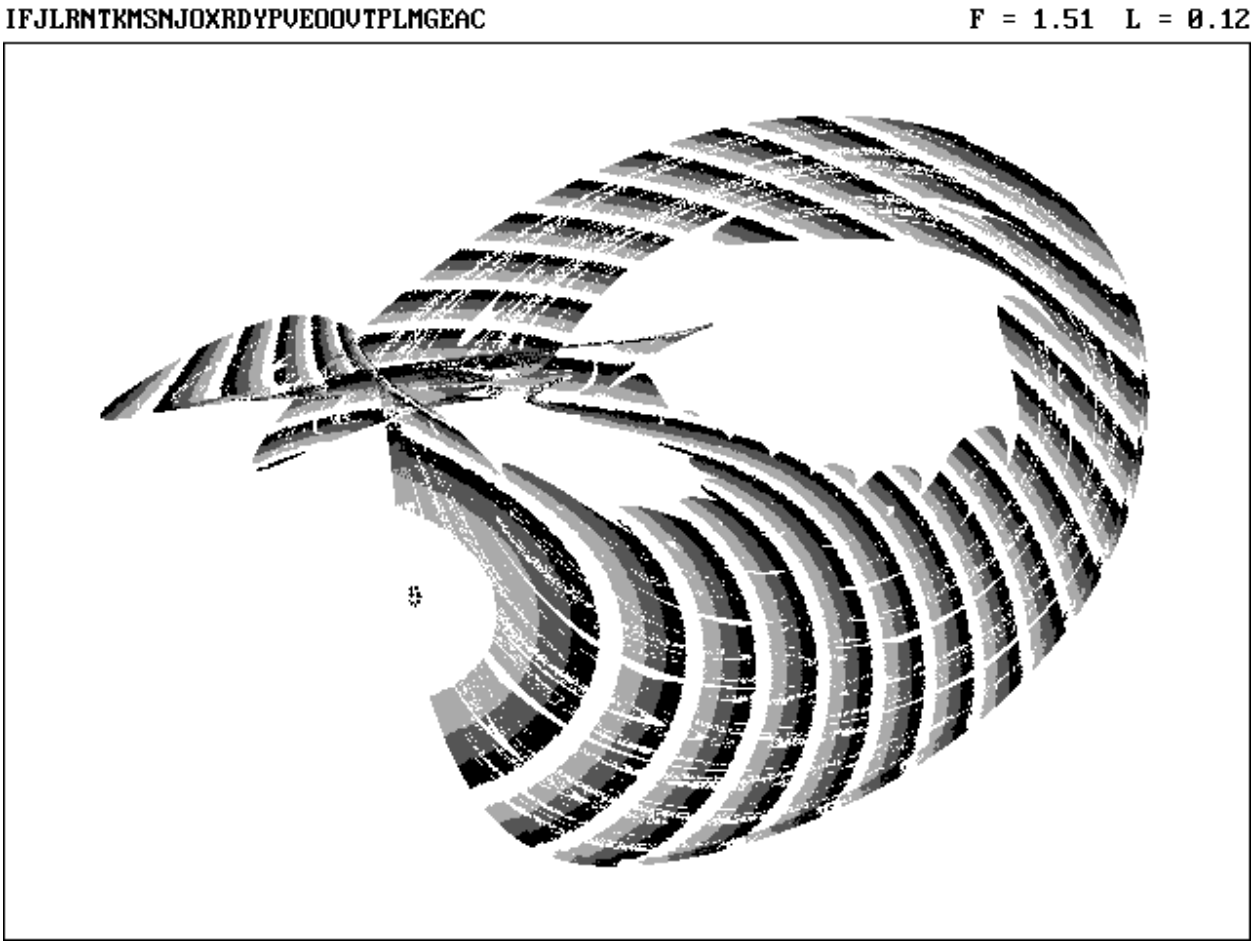


Figure 4-36. Three-dimensional quadratic map with contour bands

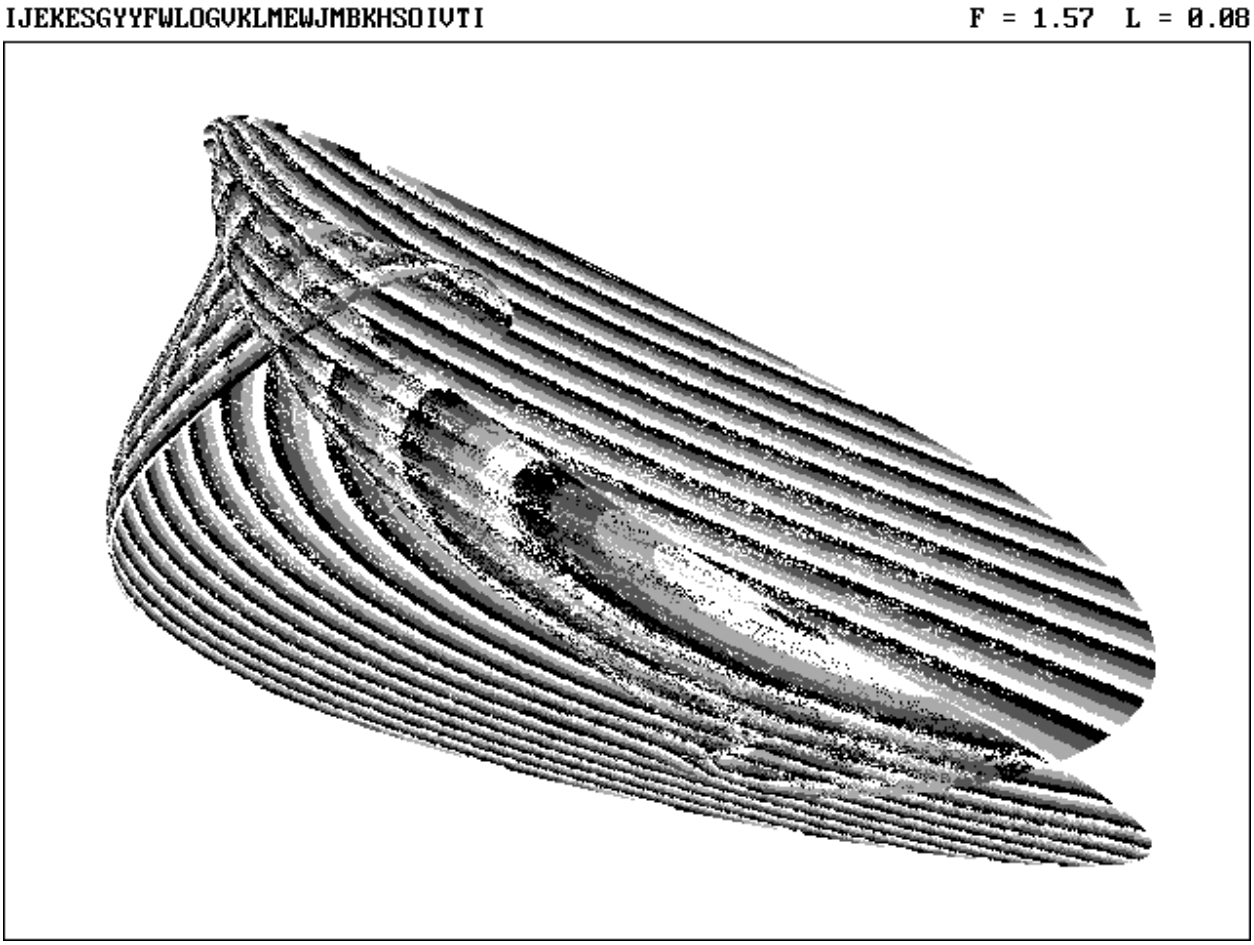


Figure 4-37. Three-dimensional quadratic map with contour bands

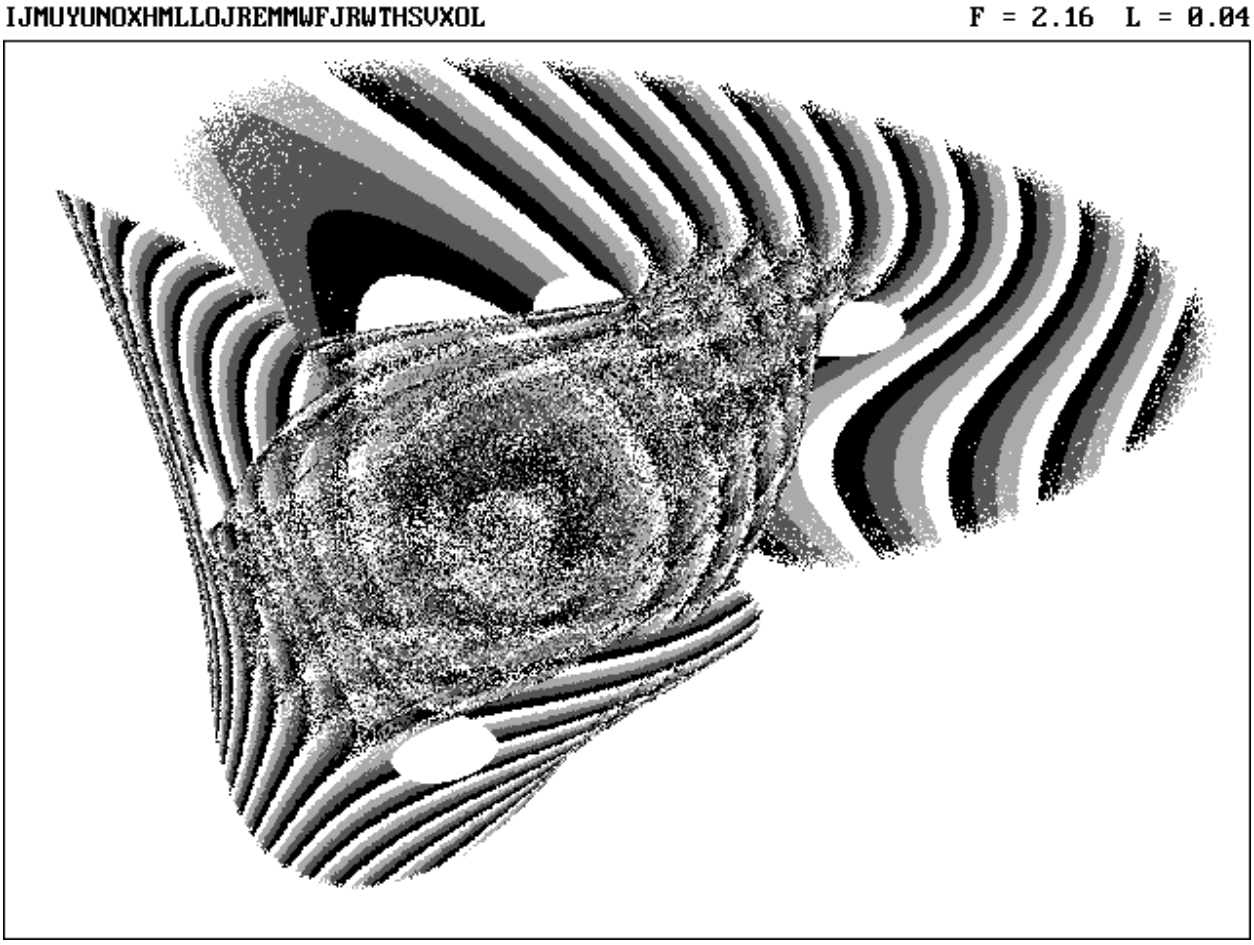


Figure 4-38. Three-dimensional quadratic map with contour bands

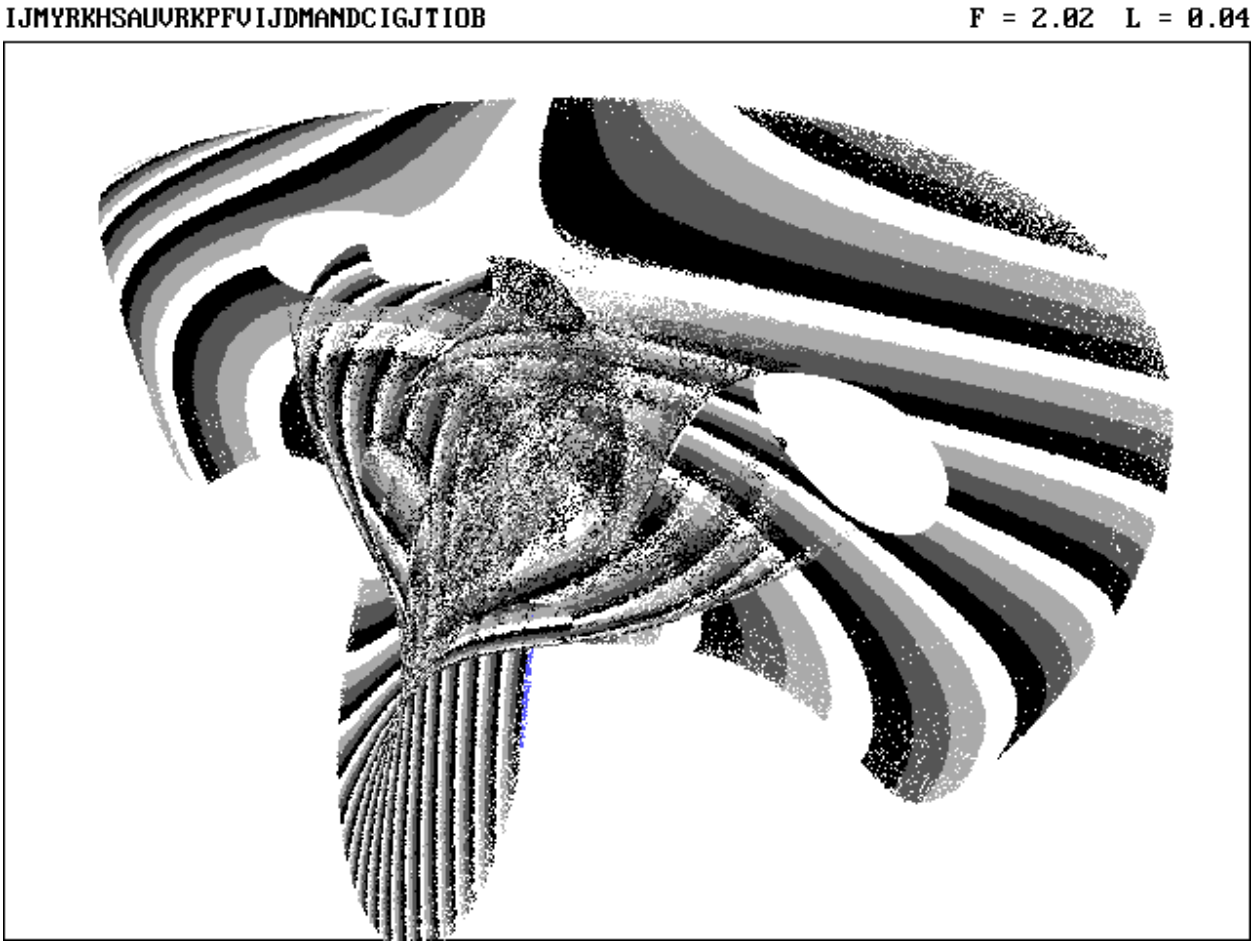


Figure 4-39. Three-dimensional quadratic map with contour bands

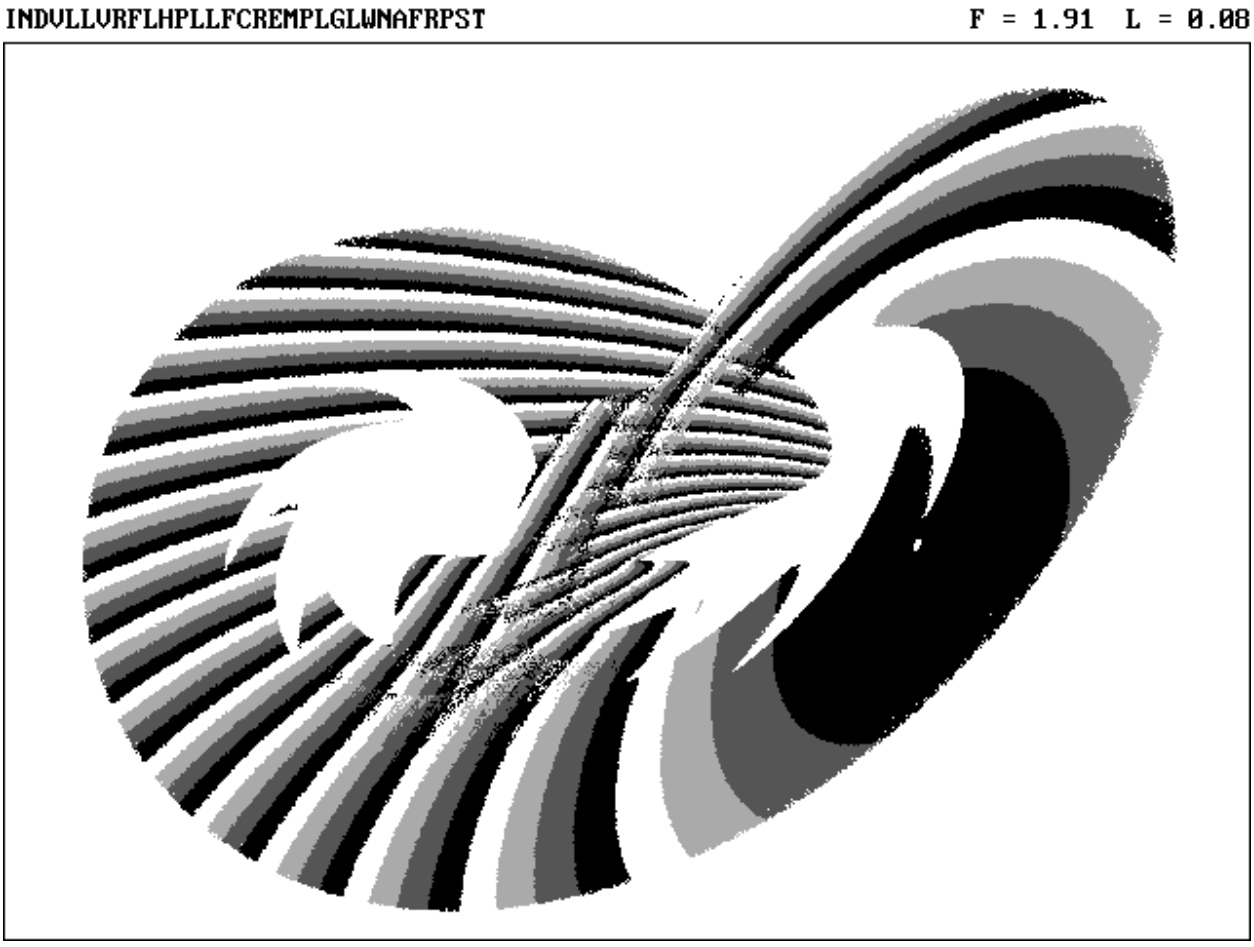


Figure 4-40. Three-dimensional quadratic map with contour bands

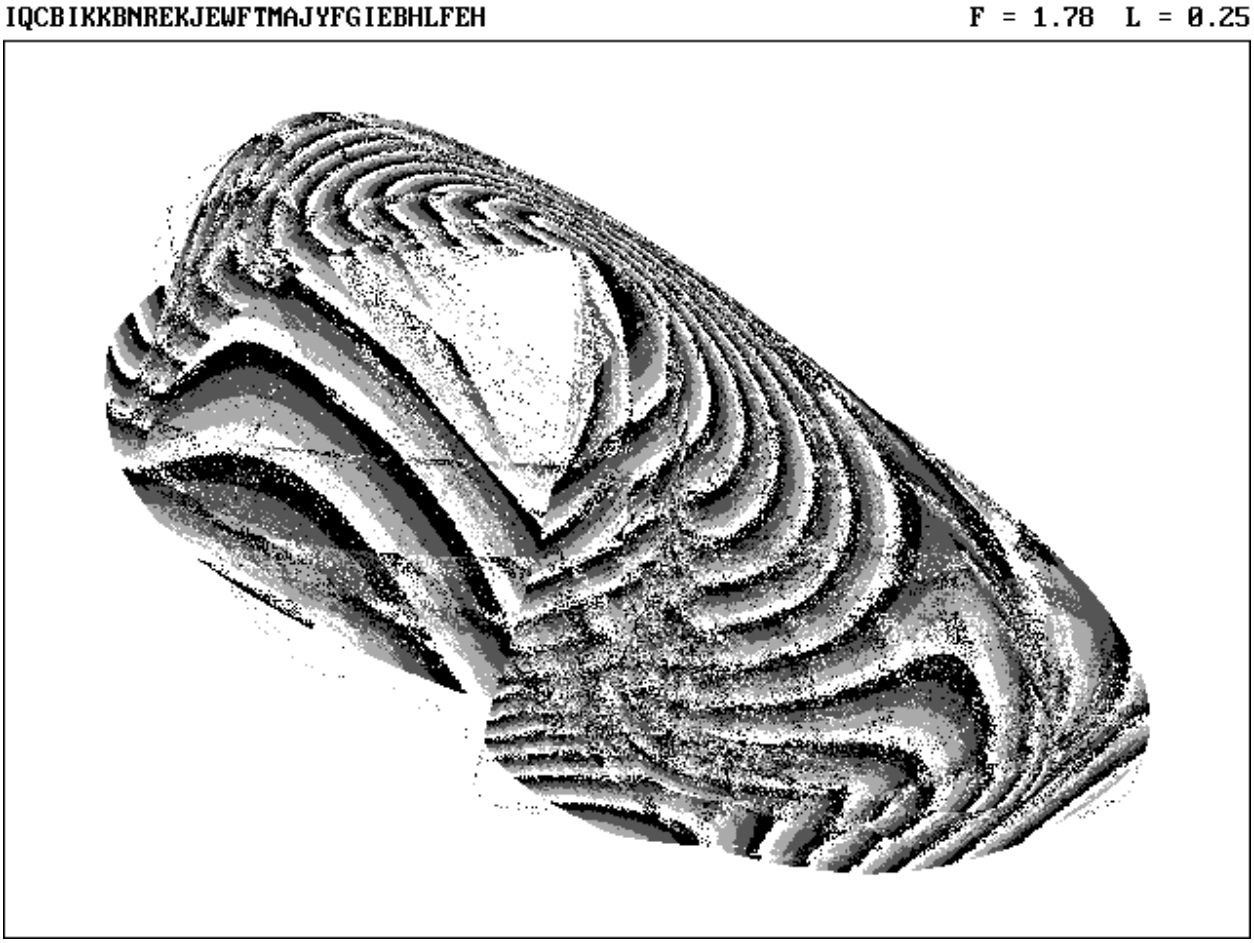


Figure 4-41. Three-dimensional quadratic map with contour bands

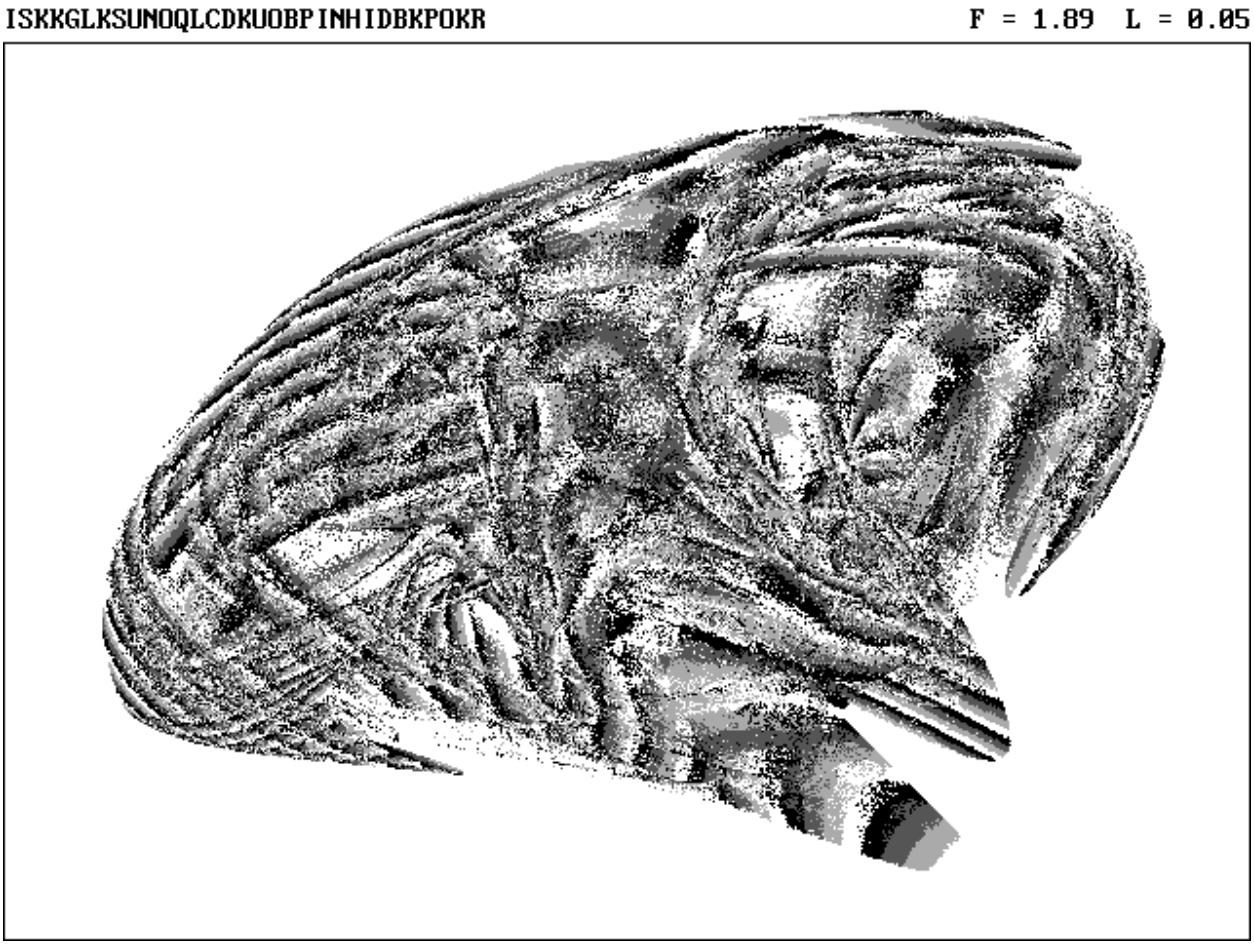


Figure 4-42. Three-dimensional cubic map with contour bands

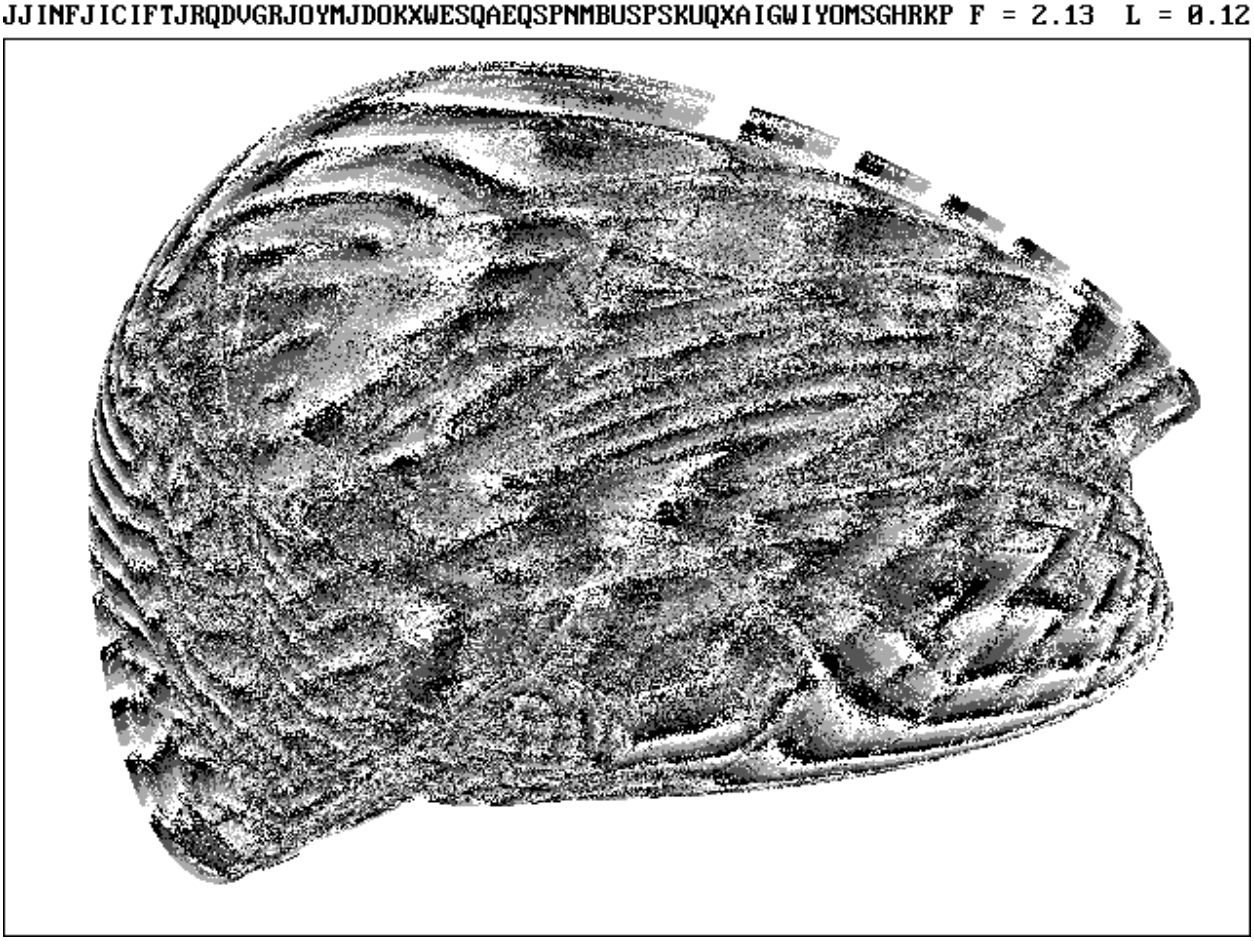


Figure 4-43. Three-dimensional cubic map with contour bands

Figure 4-44. Three-dimensional cubic map with contour bands

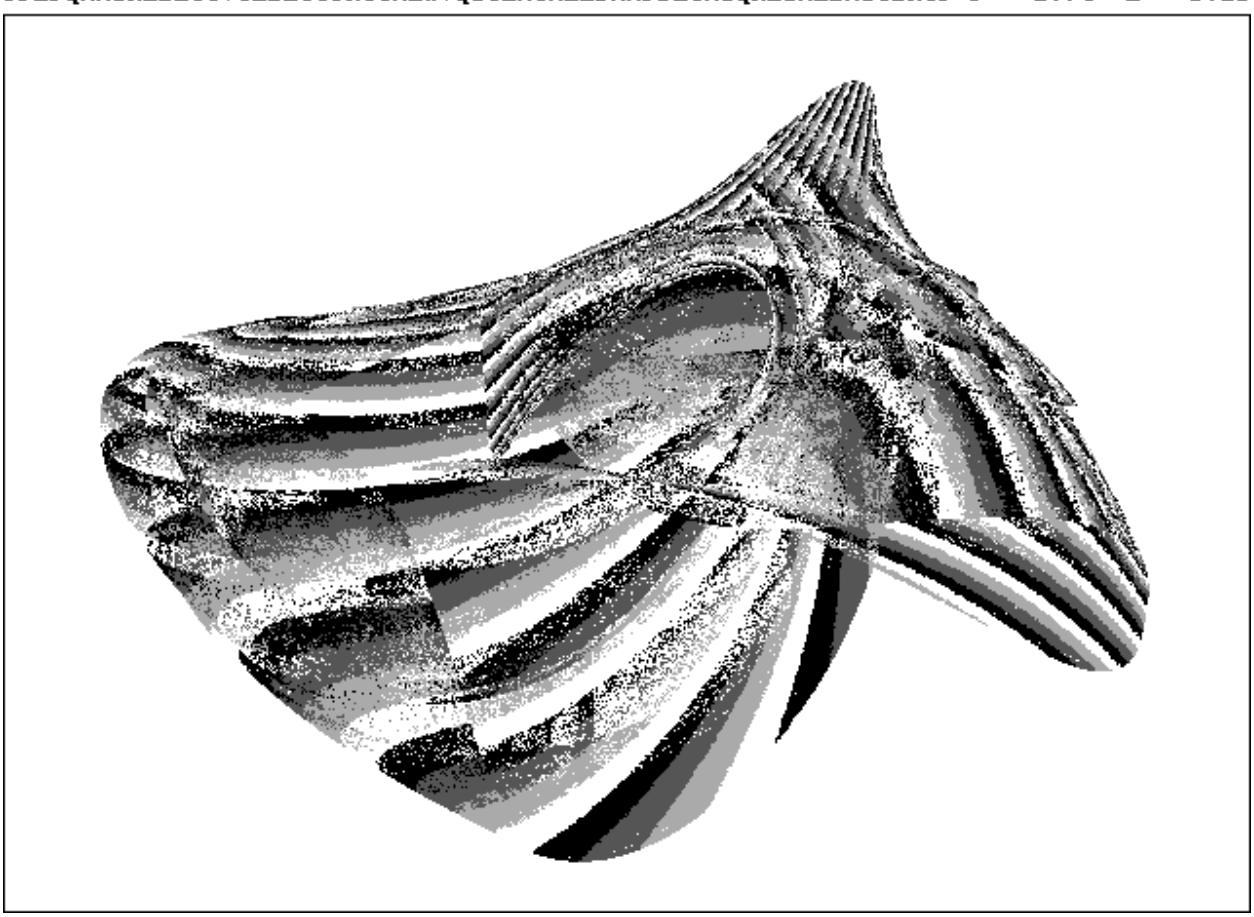


Figure 4-45. Three-dimensional quartic map with contour bands

Figure 4-46. Three-dimensional quartic map with contour bands

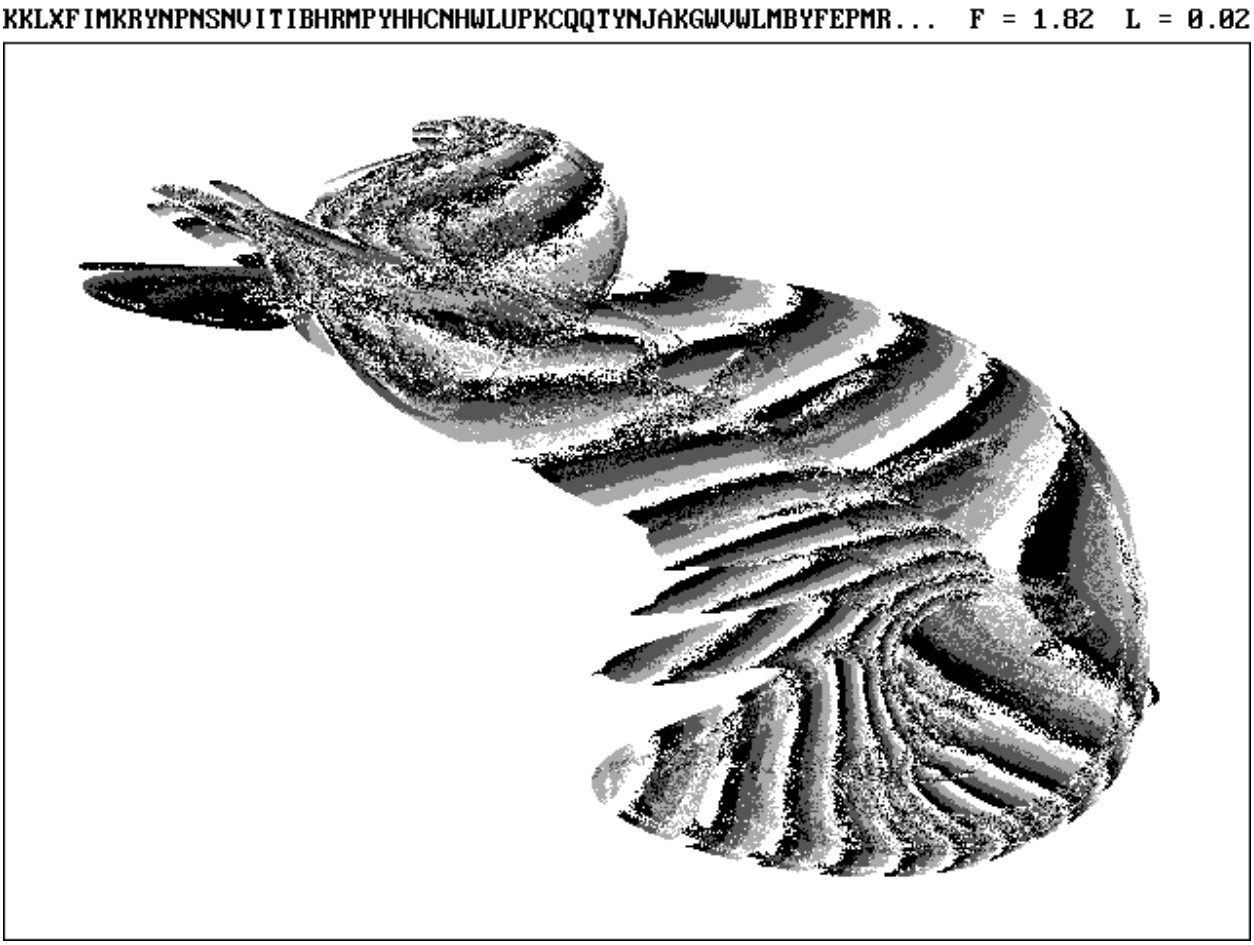


Figure 4-47. Three-dimensional quartic map with contour bands

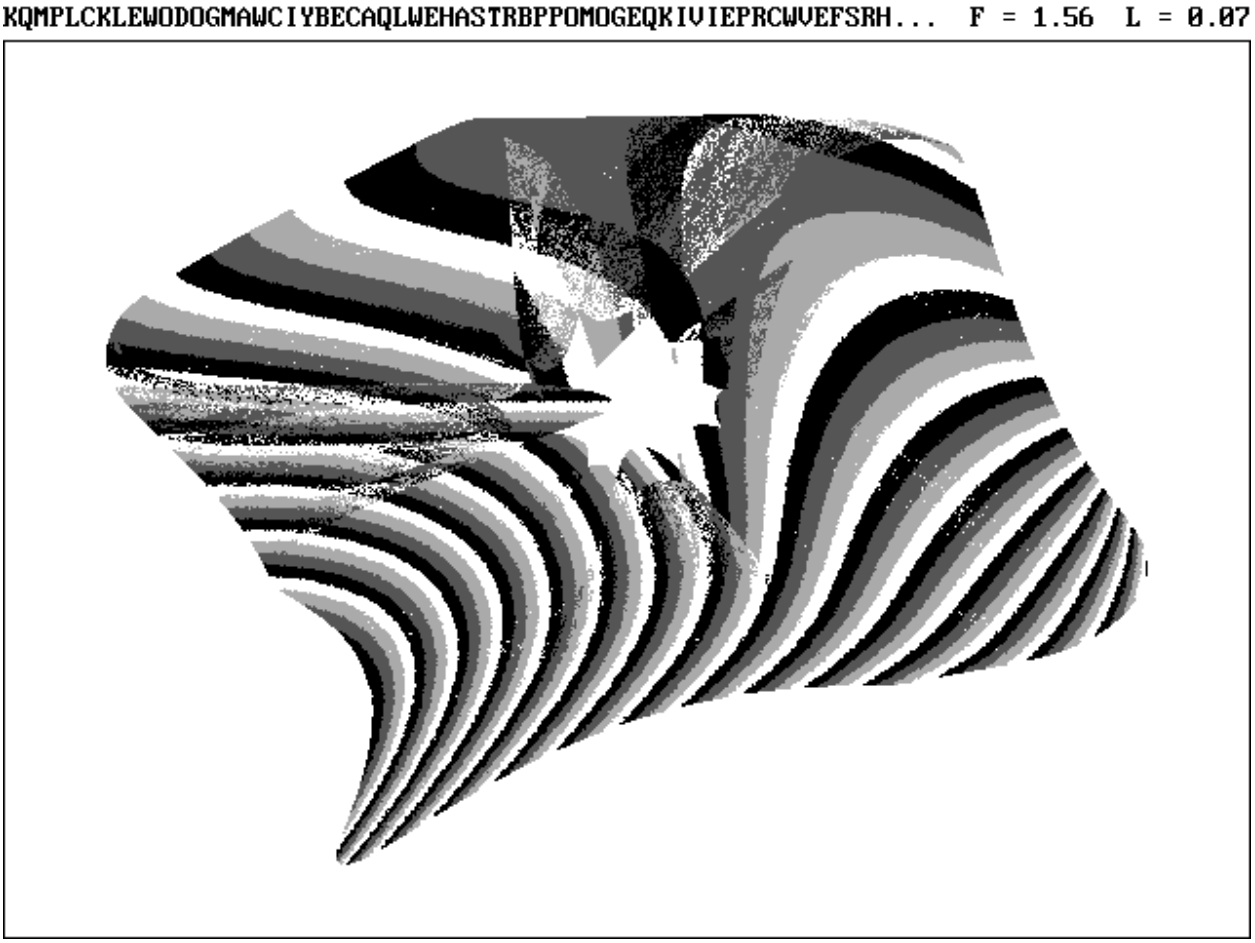


Figure 4-48. Three-dimensional quintic map with contour bands

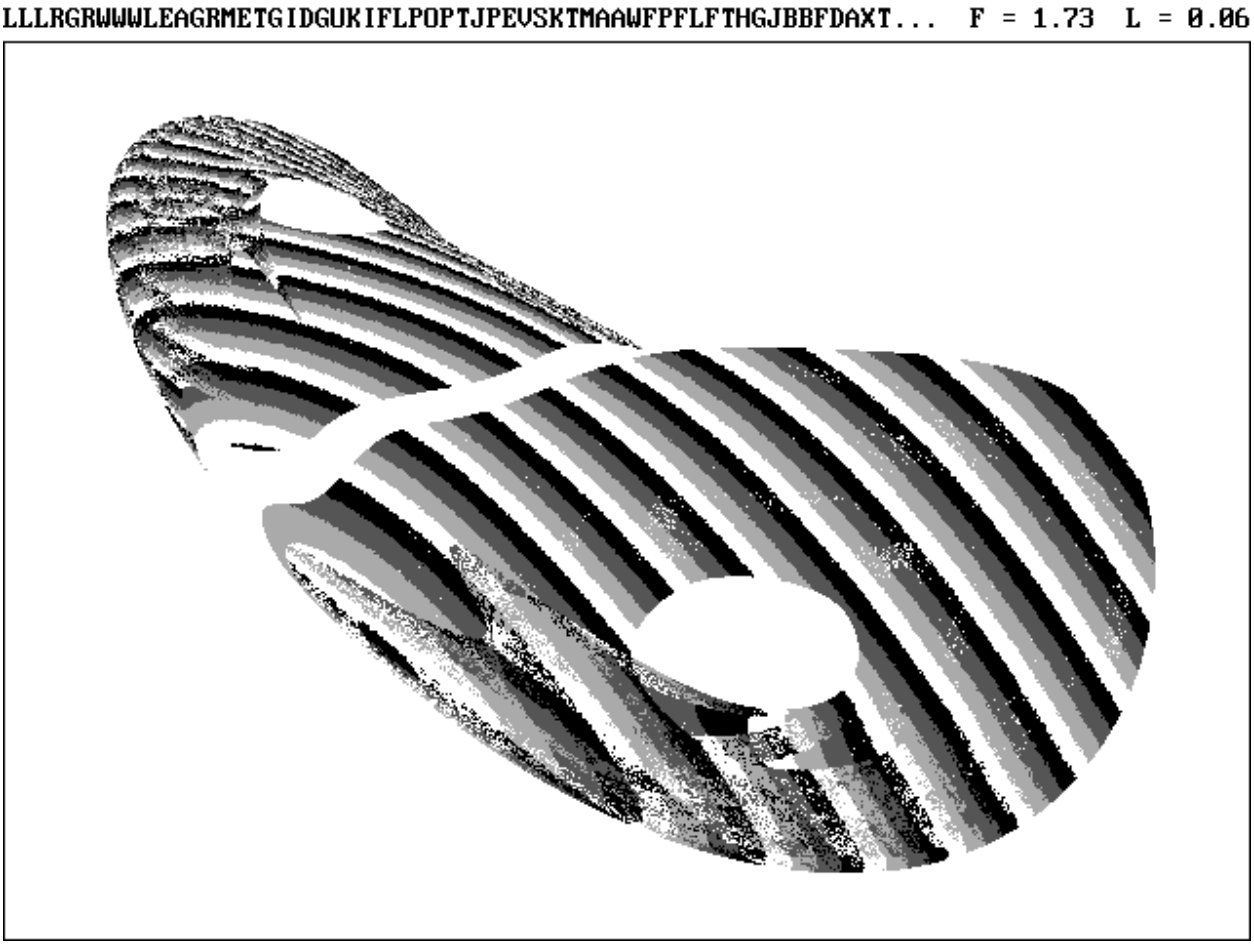


Figure 4-49. Three-dimensional quintic map with contour bands

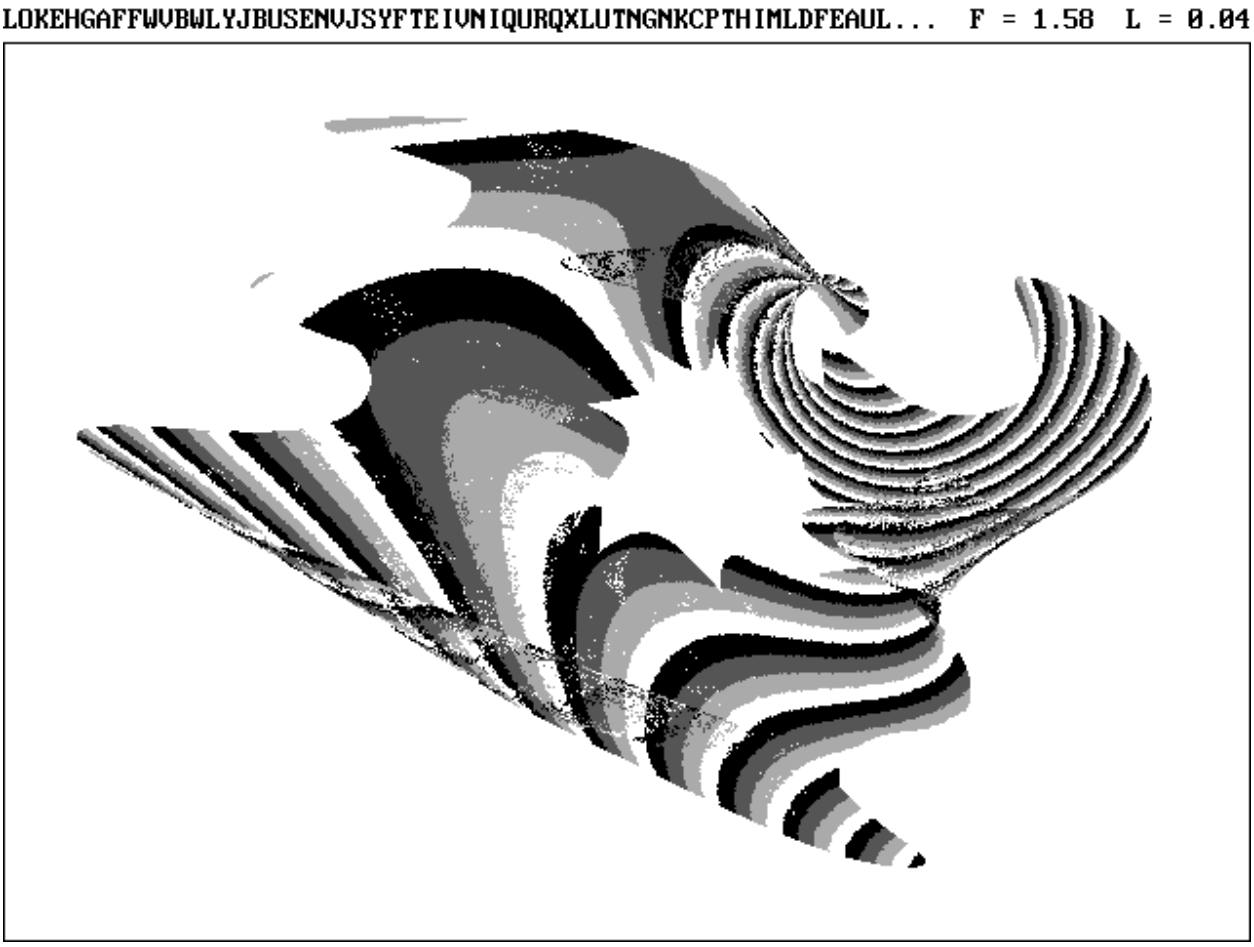
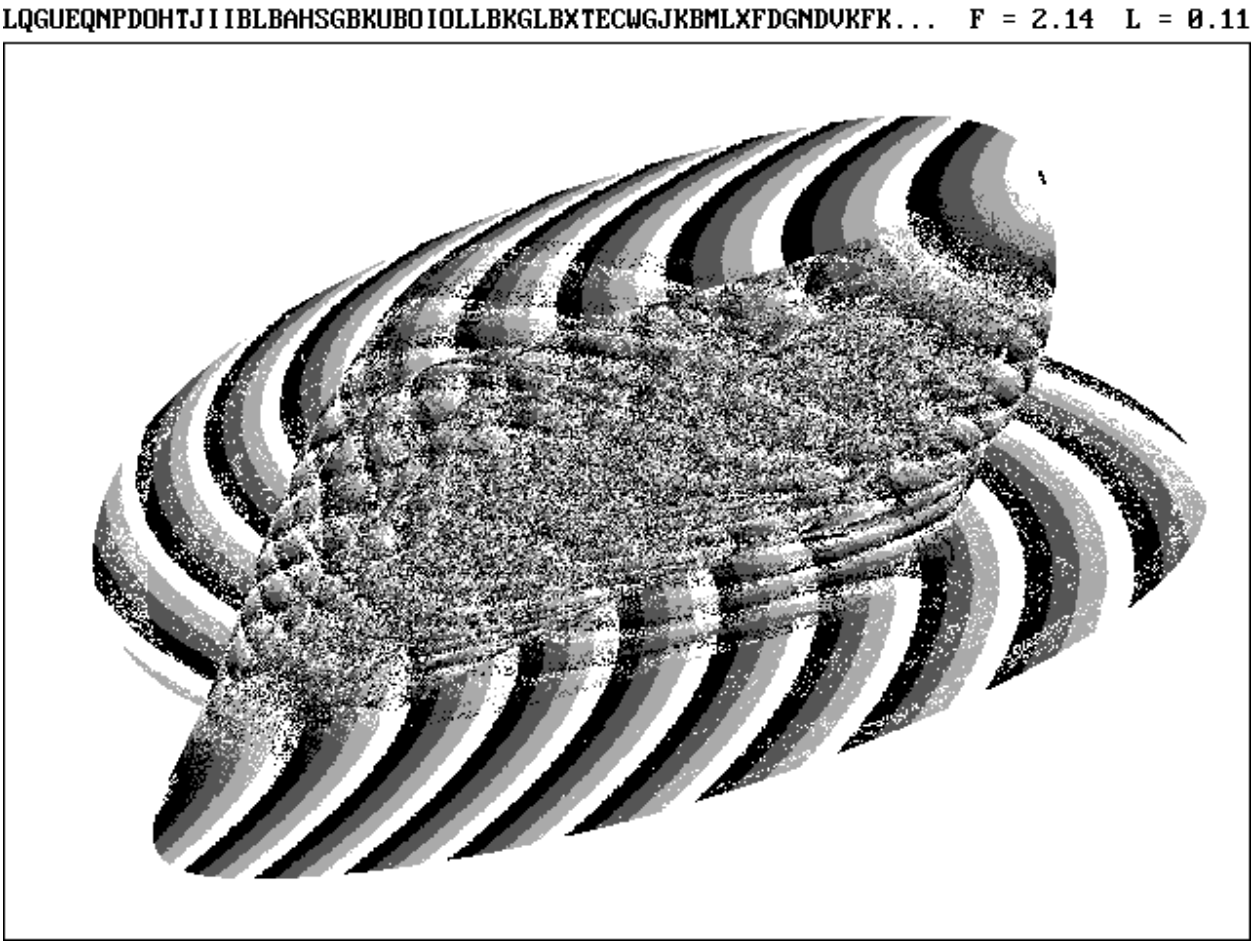


Figure 4-50. Three-dimensional quintic map with contour bands



4.4 Colors

It's not hard to guess that the next logical step is to use the full array of colors available on a computer with a color monitor. In SCREEN mode 9 (EGA) and SCREEN mode 12 (VGA), 16 colors can be displayed simultaneously from a palette of 64 (EGA) or 262,144 (VGA). SCREEN mode 13 (VGA), which is supported by some BASIC versions, allows 256 colors, but the screen resolution of 320 by 200 is inadequate for our purposes, and thus it will not be used. In SCREEN mode 1 (CGA) only four colors can be displayed from one of two palettes. We assume your computer has EGA or VGA capabilities, but the program also works with CGA if you use *SM%* = 1 in line 1030. The program is written to simplify extending the technique to future new graphics modes with more colors and higher resolution, provided they are supported by your BASIC compiler.

We will convert the Z values into 15 different colors (COLOR 1 through COLOR 15). The 16th (COLOR 0) is the background color and will not be used. The default values of the colors are given in Table 4-1. The changes required to the program are shown in PROG14.

PROG14. Changes required in PROG13 to display colors

```
1000 REM THREE-D MAP SEARCH (With Color Display)

1120 TRD% = 3                    'Display third dimension in color



3760 IF Q$ = "R" THEN TRD% = (TRD% + 1) MOD 4: T% = 3: IF N > 999 THEN N = 999:
GOSUB 5600



4500        IF TRD% = 3 THEN PRINT "colors    "



5160 IF TRD% = 3 THEN PSET (XP, YP), COLR%(INT(NC% * (Z - ZMIN) / (ZMAX - ZMIN)
+ NC%) MOD NC%)



5610 NC% = 15                    'Number of colors

5630 IF TRD% = 3 THEN FOR I% = 0 TO NC%: COLR%(I%) = I% + 1: NEXT I%
```

Some sample color attractors produced by **PROG14** are shown in Plates 1 through 8. Some of these examples are projected onto a sphere. Note that the interposition of dots of different colors in some of the figures gives the impression that there are many more than 16 colors. The addition of color usually enhances the appearance of the attractors. More such cases could have been included in this book, but then its cost would have been considerably higher. Henceforth you will probably want to view your three-dimensional attractors in color.

Notice that, where one part of the attractor lies behind another, you can see

the more distant portion through the closer portion. Thus the attractor appears transparent, which enables you to see its interior but tends to diminish the perception of depth. You might want to modify the program so that the closer portion occludes the region behind it. It is relatively easy to do so using the BASIC POINT function to test the existing color of the pixel before plotting the new point and plotting it only if its color is higher in the sequence than the existing one. Thus each pixel eventually is colored according to the closest part of the attractor. This effect can be accomplished by changing line 5160 of **PROG14** to

```
5160 IF TRD% = 3 THEN C% = INT(NC% * (Z - ZMIN) / (ZMAX - ZMIN) + NC%) MOD NC%:
IF POINT(XP, YP) < C% THEN PSET (XP, YP), C%
```

You can also alter the sequence of colors by changing the values stored in the array *COLR%* in line 5630. For example, a sequence that mimics the rainbow would advance from red (12) through yellow (14), green (10), cyan (11), and blue (9) to magenta (13). With *aerial perspective*, brilliant, warm colors such as red appear closer to the viewer than lighter, less brilliant, cool colors such as blue, which we associate with the distant sky. Thus assigning red to the large *Z* values and blue to the small *Z* values enhances the illusion of depth.

Most dialects of BASIC include a PALETTE command that allows you to change the screen colors without replotting the data, but this command works differently with different versions of BASIC and in different graphics modes, so we will not try to provide a program that takes advantage of it. However, a challenging programming exercise is to add the capability of rotating the color palette by pressing a key while a color attractor is being displayed to produce a psychedelic animated display. You might use the + key to rotate in one direction and the - key to rotate in the opposite direction. For most of the figures in this book, the PALETTE command was used to interchange black and white before printing them to save ink and to improve the appearance of the attractors when they are displayed on a white background.

You may also want to experiment with combining the various display techniques. Clearly there is nothing to preclude displaying a color attractor with shadows and contour bands. Such combinations offer interesting possibilities that are exploited for the four-dimensional cases in the next chapter.

4.5 Characters

Many computer monitors and printers lack color capabilities. However, it is often possible to produce a similar effect using a gray scale. In some cases, the various colors are mapped automatically into a shade of gray. Another technique that works on almost any computer and offers interesting display possibilities is to map the *Z* values into different ASCII characters and print them as a block of text. Such text files are easily manipulated by word processors, transported to different computers, displayed on almost any monitor, and printed with any printer on paper of various sizes.

Perhaps the simplest method is to map the *Z* values into consecutive ASCII characters, thereby producing a type of gray scale with bands whose darkness depends on the density of the character. A more reasonable approach is to order the characters so that the more dense ones correspond to larger values of *Z*. The ordering depends on the font, typeface, and size of the characters. For example, 10-point Letter Gothic bold can be ordered as shown in Table 4-2. The table uses 32 characters, which is about the maximum for this technique because many of the characters have the same density. The eye can distinguish about 500 levels of gray. This sequence is only one of many that are equally good. To see the gray scale, you should view the table from at least six feet away. Squinting and removing your glasses, if you wear them, might also help.

Table 4-2. Gray scale produced by ordering 10-point Letter Gothic bold characters

```
.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM
```

```
.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM

.:,;"=>!/+?icl7IjvJL64VOASUDXEBM
```

The character densities for this table were determined by writing each of the ASCII characters from 33 to 127 (see Table 2-1) to the screen of a Macintosh computer and then counting the illuminated pixels using the POINT command in BASIC. Bold characters are chosen for their increased density. Avoid lowercase characters whenever possible because they often don't extend the full height and thus leave a wide blank band between their top and the bottom of the row above. Be sure to use a monospaced font rather than a proportional one. For the default font on most IBM computers in VGA SCREEN mode 12, a better sequence is

```
.-,;=+>i%lI?v7zuCjTFSVGAEUDHBWQ
```

We will not develop a computer program for implementing this technique because the resolution is too poor for useful display on a computer screen or page of a book. Furthermore, the program would be dependent on the fonts available and the capabilities of the printer. The programming is not difficult and closely parallels the example in **PROG14**.

However, Table 4-3 provides an indication of what is possible using 5-point Letter Gothic bold characters with 78 lines of 103 characters each. The character sequence ordered by density for this case is given in Table 4-2. This case is a three-dimensional quadratic map with a code of `ILRRHAEYWNTPWFLHTCSLYLFAKQITQTW`.

At such a low resolution, much of the detail is lost. However, if you have a printer or plotter capable of printing small fonts on a large piece of paper, you can recover the resolution and produce figures of considerable artistic quality. You can divide the text into many segments on separate pages and tape them together, or print them on a paper roll or fanfold paper to make a very skinny attractor many feet long. With ordinary objects such as the text in this book, extreme stretching in one dimension just produces sticklike figures. However, strange attractors are fractals,

and they have detail on all scales, which ensures that they look interesting however much they are stretched.

Table 4-3. Three-dimensional quadratic map with character scale

```
::::..

; ::::

;;;;;;;

"""""""""""""";

================

>>>==>>>>>>>!!!!!!

!>>>>!!!!!!!!!!!!!

JJ !!!!!!///+++++++++++++++

J //////++++++??iiiiiiiii??????

JJ +///++???? icccccccciccciiiiiiii

JJv ++++??ii? ccllllllllllllllccccccccii

JJv ??+??iiii lll77777777777lllllllllccccc

JJJj ????iicci 777777IIIII77777777777lllllllccc

JJv ii?iiccc IIIIIIIIIIIIIIIIIIIIII7777777llllllc

JJvj ciiiclll IIjjjjjjjjjjjjjjjjjjjIIIIIIII77777llllll

JLvjj ccciccllll jjjjjjjvvvvvvvvvvvvjjjjjjjjjjjjjIIIIII77777lll

LLvjI llcccl777 jvvvvvvvvvvvvvvvvvvvvvvvvvvvvvjjjjjjjjIIIIII7777l

LLvjjI77lllcll777 vvvvvvvJJJJJJJJJJJJJJJJvvvvvvvvvvvvvjjjjjjjIIIII777

LLvvjjI77llll77I7 vJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJvvvvvvvvjjjjjjIIIII7
```

JLLvjjII77lll7III JJJJJJJJJJLLLLLLLLLLLLLLLLJJJJJJJJJJJJJvvvvvvvjjjjjIIII7

LLvvjjII77777IjI JJJJJLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLJJJJJJJJJvvvvvvjjjjjII

LLvvjjjII7777Ijj JJLLLLLLLLLLL LLLLLLLLLLLL6LLLLLLLLLLLLLLLLJJJJJJJJvvvvvjjjjI

JLLvvjjIII77IIjjj LLLLLLL 6666666666666LLLLLLLLLLJJJJJJJvvvvvjj

LLvvvjjIIIIIIjvj LLLLL 66666666666666LLLLLLLLJJJJJJvvvvj

LLJvvjjjIIIIjjvv LLLLL 66666666666666LLLLLLLJJJJJJvv

LLvvvjjjjIIjjvv LLL 664446666666666LLLLLLJJJJ

LLJvvvjjjjjjjvvv L66 4444444466666666LLLLLLJ

JLJvvvvjjjjjvvJJ 6666 4444444444466666666LLL

LLJvvvvvjjjvvJJ 666 4VV444444444466666

JLJJvv vvvvvvJJJ 66 VVVVVVV44444444

LLJJvv vvvvvJJJ 66 VVVVVVVVVVV444

JLJJJv vvvvJJJJ OOOOOVVVVVVVV

LLJJJv vJJJJL OOOOOOOOOOOOOOOVV

LJJJJ JJJJLL AAAAAAAAAAAAA0000000000V

JJJJJJ JJJLL OOOOOOOOO AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA00

JJJJJ LLLL OOOOOOOOOOOOOAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

JJJJJ LLL OOOOOAAAAAAAAAAAAAAAAAAAAAAAASSASAASSSSSSSSSAAAAAAA

JJJJJJ LLL AAAAAAAAAAAAASSSSSSSSSSSSSSSSSSSSSSSSSSS

JJJJJJ LLL AAAAAASSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

JJJJJJ L66 AASSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSSS

JJJJJJ 66 ASSSSSSSSSSSSSSSUUUUUUUUUUUUUUUUUUU

JJJJJJ 66 SSSSSSSSSSUUUUUUUUUUUUUUUUUUUUUU

LJJJJLL 6 SSSSSSUUUUUUUUUUUUUUUUUUUUUUUUU

LJJJLLL SSUUUUUUUUUUUUUUUUUUUUDDDDDDDDDDD

LLJLLLLL SUUUUUUUUUUUUUUDDDDDDDDDDDDDDDD

LLJLLLLL UUUUUUUUUUDDDDDDDDDDDDDDDDDDD

LLLLLLLL UUUUUUDDDDDDDDDDDDDDDDDDDDD

LLLLLLLLL UUUDDDDDDDDDDDDDDDDXXXXXXXXXX

LLLLLLLLL UDDDDDDDDDDDDDDXXXXXXXXXXXXXX

LLLLLLLLL DDDDDDDDDDXXXXXXXXXXXXXXXXX

LLLLLLLL66 SS DDDDDDDXXXXXXXXXXXXXXXXXX

LLLL6666666 SSSS DDDDDXXXXXXXXXXXXXXXXEEEEEE

LLL666666444444 SSSS DXXXXXXXXXXXXXEEEEEEEEEEE

LLL6666664444444VV SSSSS XXXXXXXXXXXXEEEEEEEEEEEEE

LLL666664446644 SSSSS XXXXXXXXXEEEEEEEEEEEEEEE

LLL666666 SSSUU XXXXXXEEEEEEEEEEEEEEBBBB

LLL6666 SUUUUU XXXXEEEEEEEEEEEEEBBBBBBBB

LLL66 O SUUUUUUU XXEEEEEEEEEEEEBBBBBBBBB

LLLLL OO UUUUUUUUD EEEEEEEEEEEBBBBBBBBBBBB

LLL OOO UUUUUUUUDDD EEEEEEEEBBBBBBBBBBBBB

LLLL OOOO SUUUUUUUDDDDDDD EEEEEEBBBBBBBBBBBBMMM

LLLLLLL VOOOO SUUUUUUUDDDDDDDXX EEEEEBBBBBBBBBBBBMMMMM

LLLLL6666664444444VVVVVOOOO SUUUUUUUUDDDDDDDXXXXXX EEEBBBBBBBBBBBMMMMMMM

LLLLL6666664444444VVVVVOOO SSUUUUUUUUDDDDDDDXXXXXXXXEEE EEBBBBBBBBBBBMMMMMMMM

LLLL66666664444444VVVVVVOOSSUUUUUUUDDDDDDDXXXXXXXEEEEEEEEEBBBBBBBBBBMMMMMMMM

```
LLLLL666666444444VVVVVVV00  SSSUUUUUUUUDDDDDDDDXXXXXXXXEEEEEEEEEBBBBBBBBBMMMMMMMM

LLLL666666444444VVVVVV000  SSSSUUUUUUUUDDDDDDDDXXXXXXXEEEEEEEEEBBBBBBBBBMMMMM

LL666666444444VVVVVV000  SSSSUUUUUUUUDDDDDDDXXXXXXXXEEEEEEEEBBBBBBBBB

L66666644444VVVVVV0000  SSSSSUUUUUUUDDDDDDDXXXXXXXXEEEEEEEEBBB

6666444444VVVVVV0000  SSSSSSUUUUUUUDDDDDDDXXXXXXXXEEEEEEE

444444VVVVVV00000  ASSSSSSUUUUUUUDDDDDDDXXXXXXXEEEE

4VVVVVV000000  AAASSSSSSUUUUUUUDDDDDDDXXXXXXXX

VVV0000000AAAAAASSSSSSUUUUUUUDDDDDDDXXXXX

V000000AAAAAASSSSSSUUUUUUUDDDDDDDXXX

000AAAAAASSSSSSSUUUUUUDDDDDDDD

AAAAAASSSSSSSUUUUUUDDDDD

AAASSSSSSSUUUUUUDDD

SSSSSUUUUUU
```

4.6 Anaglyphs

An alternative approach for displaying objects in three dimensions is the binocular stereogram in which the parallax produced by separate images in each eye creates the illusion of depth. The idea dates back to Socrates in the fourth century BC, and the earliest stereograms were produced in the mid-1800s by Sir Charles Wheatstone and Sir David Brewster. The inception of motion pictures in the early 1900s was accompanied by 3-D movies using overlapping red-green images that were viewed through red-green glasses to produce a black-and-white image in what is called the *anaglyphic process*. Anaglyphs have also been widely used in comic books. Color 3-D movies using cross-Polaroid glasses were briefly popular in the 1950s. They require a special screen to reflect the polarized light from the projectors without allowing it to depolarize.

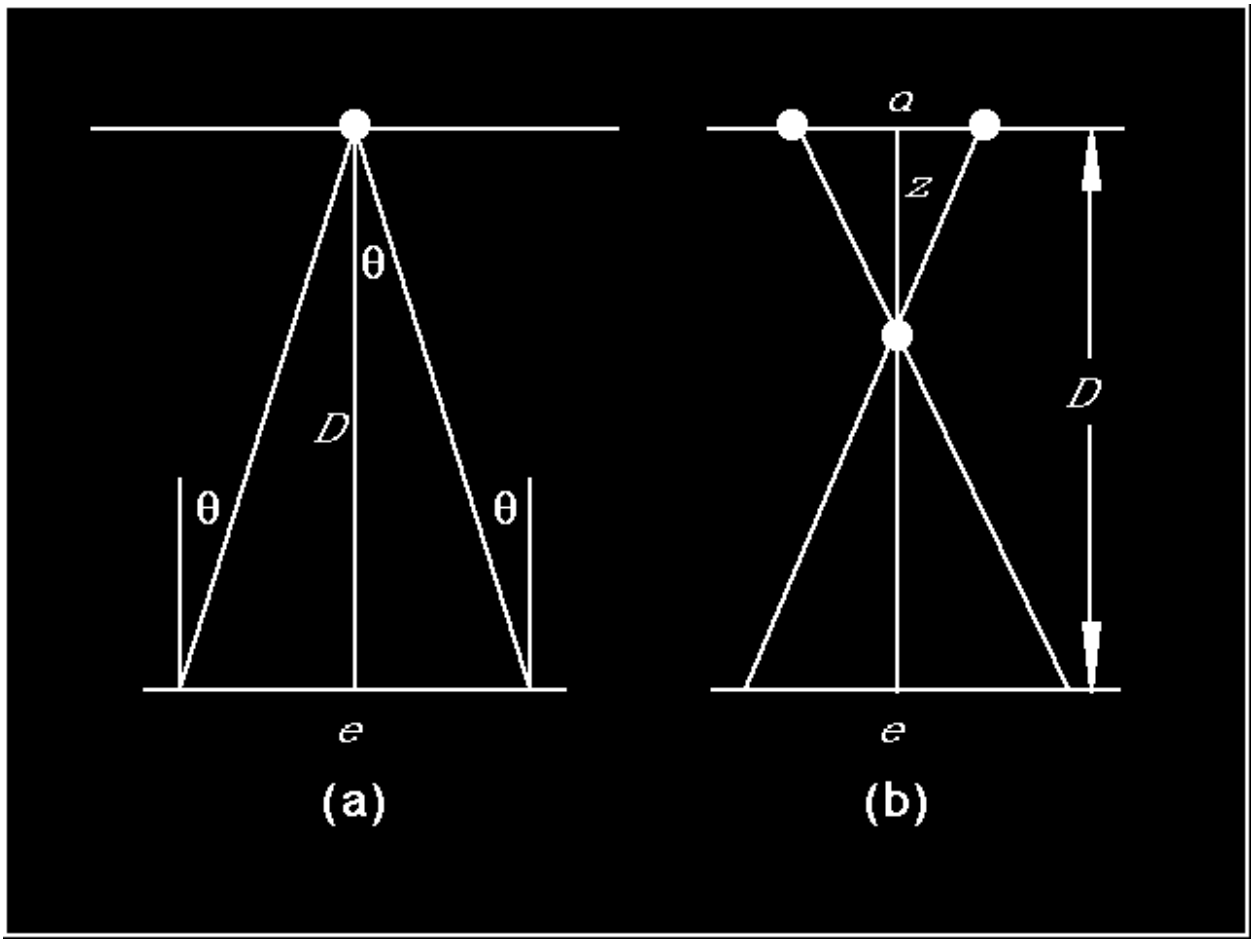
The anaglyphic process offers distinct advantages in computer visualization. The hardware requirements are minimal (a color monitor and a pair of 50-cent

glasses), the programming is surprisingly simple, and the results can be impressive. The main drawback is that the images produced are usually monochromatic, although a gray scale and some limited coloration are possible.

Our perception of depth arises from a number of psychological and physiological processes. Many of these processes are induced by visual cues that don't depend on binocular vision, such as the relative size and motion of objects, interposition, illumination, shadows, and focal accommodation. Others require the parallax attendant to stereoscopic vision. When some of the usual visual cues are absent or contradictory, a rivalry ensues that demands time and mental effort for our brains to resolve. It is remarkable that, with just the single cue of binocular stereopsis, most people can quickly perceive a vivid three-dimensional image.

Consider a point at a distance D from the midpoint of your eyes whose separation we take to be e (typically 6.5 cm), as shown in Figure 4-51(a). Assume the point is a single illuminated pixel on the computer screen. Each eye must swivel inward through an angle θ in order for the two images to fuse into a single point, where θ is the angle whose tangent is $e / 2D$. It is this muscular response of the eyes that provides the brain with the relevant depth information.

Figure 4-51. Line of sight of each eye when viewing anaglyphs



Now suppose you are to perceive the point to be at a distance *D - Z* from your eyes (a distance *Z* in front of the computer screen) as shown in Figure 4-51 (b). We must then plot two points on the computer screen separated horizontally by a distance *d*. From the similarity of the two triangles, we calculate

$$d = eZ / (D-Z) \qquad \text{(Equation 4C)}$$

The formula works also for negative *Z*.

To achieve the proper *linear perspective*, we should plot the closer points a

little farther apart than the more distant points, but with unfamiliar objects such as strange attractors, there is little reason to do so. Usually *D* is much greater that *Z*, and we can approximate the right-hand side of Equation 4C by *e Z / D*. This approximation causes some expansion of the image for negative *Z* (behind the screen) and compression of the image for positive *Z* (in front of the screen). This compression can be desirable to keep the image always in front of your face rather than to let it pass behind your head.

The length of most people's arms is almost exactly ten times the distance between their eyes. Therefore, a value of *D / e = 10* is appropriate for a computer screen viewed at arm's length. In practice, the viewing distance is not very critical. The perceived depth of the image is enhanced by viewing from a greater distance, but it usually takes longer for the brain to accommodate, so it is often best to view first from close up. It sometimes speeds the adjustment to move your head from side to side.

A computer display optimized for viewing at arm's length is very effective when projected on a large screen and viewed in an auditorium. Were this not the case, three-dimensional movies could not be shown to theater audiences. In such a case, the brain perceives a scaled version of the image at a closer distance. The same effect occurs when viewing 2-D movies. The characters on the screen are not perceived as giants a large distance from the viewer. Similarly, the brain is able to compensate almost without limit to other distortions if the objects are familiar. A movie viewed from the rightmost seat in the front row appears normal after a short period of adjustment.

It is important to maintain a somewhat limited depth and field of view. Leonardo da Vinci recommended that a painting be optimally viewed from a distance equal to three times its width. Most computer screens approximately satisfy this criterion when viewed at arm's length. An object as deep as it is wide thus requires that the two images be separated by up to about an inch, requiring that the eyes toe-in by about three degrees.

The computational task, therefore, is to plot each point that makes up the attractor twice, with a horizontal separation proportional to the distance the point is to appear in front of or behind the screen, and to arrange that one set of points be visible only to the left eye and the other only to the right eye. In the anaglyphic process, this is done by plotting one set of points in red and the other in the complementary cyan (blue-green) and viewing through appropriate color-filtered glasses. By convention, the left eye should only respond to the red and the right eye only to the cyan.

Note that individuals who are color blind should experience no difficulty because it is unnecessary (and indeed undesirable) to perceive the individual colors; it is only necessary that the eyes be sensitive to them. Certain other eye defects, particularly those resulting in ocular asymmetry, are more problematic.

You can plot the points on either a black or a white background. With a black background, the images fuse into white (additive process), and with a white background, the images fuse into black (subtractive process). The sense of *Z* is reversed with the choice of background. With a black background, the red is seen through the red filter on the left eye, while for a white background, red is seen through the cyan filter on the right eye. In practice the white background is usually more satisfactory, but you may want to try it both ways to see which works best for you. Wherever a red and cyan point overlap, they should be plotted as a single black point if the background is white or as a single white point if the background is black. The changes required to the program to produce such anaglyphs are shown in **PROG15**.

PROG15. Changes required in PROG14 to produce anaglyphs

```
1000 REM THREE-D MAP SEARCH (With Anaglyphic Display)

1120 TRD% = 4                   'Display third dimension as anaglyph



3220    ZA = (ZMAX + ZMIN) / 2

3240    IF TRD% = 4 THEN LINE (XL, YL)-(XH, YH), WH%, BF



3760 IF Q$ = "R" THEN TRD% = (TRD% + 1) MOD 5: T% = 3: IF N > 999 THEN N = 999:
GOSUB 5600



4510        IF TRD% = 4 THEN PRINT "anaglyph  "



5170 IF TRD% <> 4 THEN GOTO 5240
```

```
5180    XRT = XP + XZ * (Z - ZA): C% = POINT(XRT, YP)

5190    IF C% = WH% THEN PSET (XRT, YP), RD%

5200    IF C% = CY% THEN PSET (XRT, YP), BK%

5210    XLT = XP - XZ * (Z - ZA): C% = POINT(XLT, YP)

5220    IF C% = WH% THEN PSET (XLT, YP), CY%

5230    IF C% = RD% THEN PSET (XLT, YP), BK%


5640 WH% = 15: BK% = 8: RD% = 12: CY% = 11
```

PROG15 assumes EGA or VGA graphics and a color monitor. If you have CGA graphics, you can obtain satisfactory results in SCREEN mode 1, PALETTE 1 by changing the colors in line 5640 to *WH%* = 3: *BK%* = 0: *RD%* = 2: *CY%* = 1.

Some sample anaglyphs are shown in Plates 9 through 16. Use the special glasses included with the book. If these glasses are missing, you can probably find a suitable pair at a comic book store. If you have difficulty acclimating to the anaglyphs, try viewing them from close-up and then back away once you see the effect. You may need some practice, especially because the attractors you are viewing are unfamiliar objects and they lack other depth clues. You might also try reversing the glasses (red over right eye), which reverses in and out.

Because of the large variation of computer monitor colors and spectacle filters, ghost images are common. Manipulation of the computer color palette is of limited use because the monitor ultimately constructs its colors from three distinct phosphors (red, green, and blue). The usual problem is inadequate rejection of the green by the red filter, resulting in a red ghost image when viewed against a white background. Suppression of the green by using only red and blue on a magenta background eliminates this problem but yields poor contrast of the resulting image. In some cases, the ghost images can be suppressed by viewing through multiple pairs of glasses. You may want to adjust the intensity of the red, green, and blue, so the images seen by each eye through the glasses have similar intensities.

4.7 Stereo Pairs | Stereo Pairs

Believe it or not, with a bit of practice, you can learn to view attractors in stereoscopic 3-D without special glasses. For this purpose, we print the two images side-by-side in the same color instead of superimposed on one another in different colors as we did with the anaglyphs. This technique permits full-color displays, and we will exploit this capability in the next chapter. For the moment, let's consider only monochrome images.

First we develop the computer program necessary to produce the images. The images should not be separated more than the distance between your eyes, which for most people is about 6.5 cm. If the images are separated by a larger distance, the eyes have to rotate outward beyond the normal parallel position, which at best is uncomfortable and at worst impossible. Such images are described as being *walleyed*. However, if we reduce the images to a sufficiently small size on the computer screen, the resolution is poor. Therefore, we will plot the images as large as possible and rely on the printer to reduce the size for comfortable viewing. If you prefer to sacrifice the resolution and view the attractors directly on the screen, the program is written to make it easy for you to do so. Alternately, your monitor may have an adjustment that allows you to shrink the width of the image.

PROG16 shows the changes required to produce such stereo pairs.

PROG16. Changes required in PROG15 to produce stereo pairs

```
1000 REM THREE-D MAP SEARCH (With Stereo Display)

1120 TRD% = 5                'Display third dimension as stereogram



3250    IF TRD% = 5 THEN LINE (XA, YL)-(XA, YH)

3320 IF PJT% = 1 AND TRD% < 5 THEN CIRCLE (XA, YA), .36 * (XH - XL)



3760 IF Q$ = "R" THEN TRD% = (TRD% + 1) MOD 6: T% = 3: IF N > 999 THEN N = 999:
GOSUB 5600
```

```
4520        IF TRD% = 5 THEN PRINT "stereogram"


5240 IF TRD% <> 5 THEN GOTO 5280

5250    HSF = 2                 'Horizontal scale factor

5260    XRT = XA + (XP + XZ * (Z - ZA) - XL) / HSF: PSET (XRT, YP)

5270    XLT = XA + (XP - XZ * (Z - ZA) - XH) / HSF: PSET (XLT, YP)

5280 RETURN
```

If you want to shrink the image for direct viewing from the computer screen, change the horizontal scale factor (*HSF*) in line 5250 from 2 to a larger value that separates the images by about 6.5 centimeters, or less if you are having trouble adapting to them.

Sample stereo pairs produced by this technique are shown in Figures 4-52 through 4-67. You should hold them exactly horizontally, directly in front of your face at a normal reading distance, and gaze into the distance until you see three images. The one in the middle should appear in 3-D, and the ones on each side, which you must train yourself to ignore, should be in 2-D.

Figure 4-52. Stereo pair of three-dimensional quadratic map

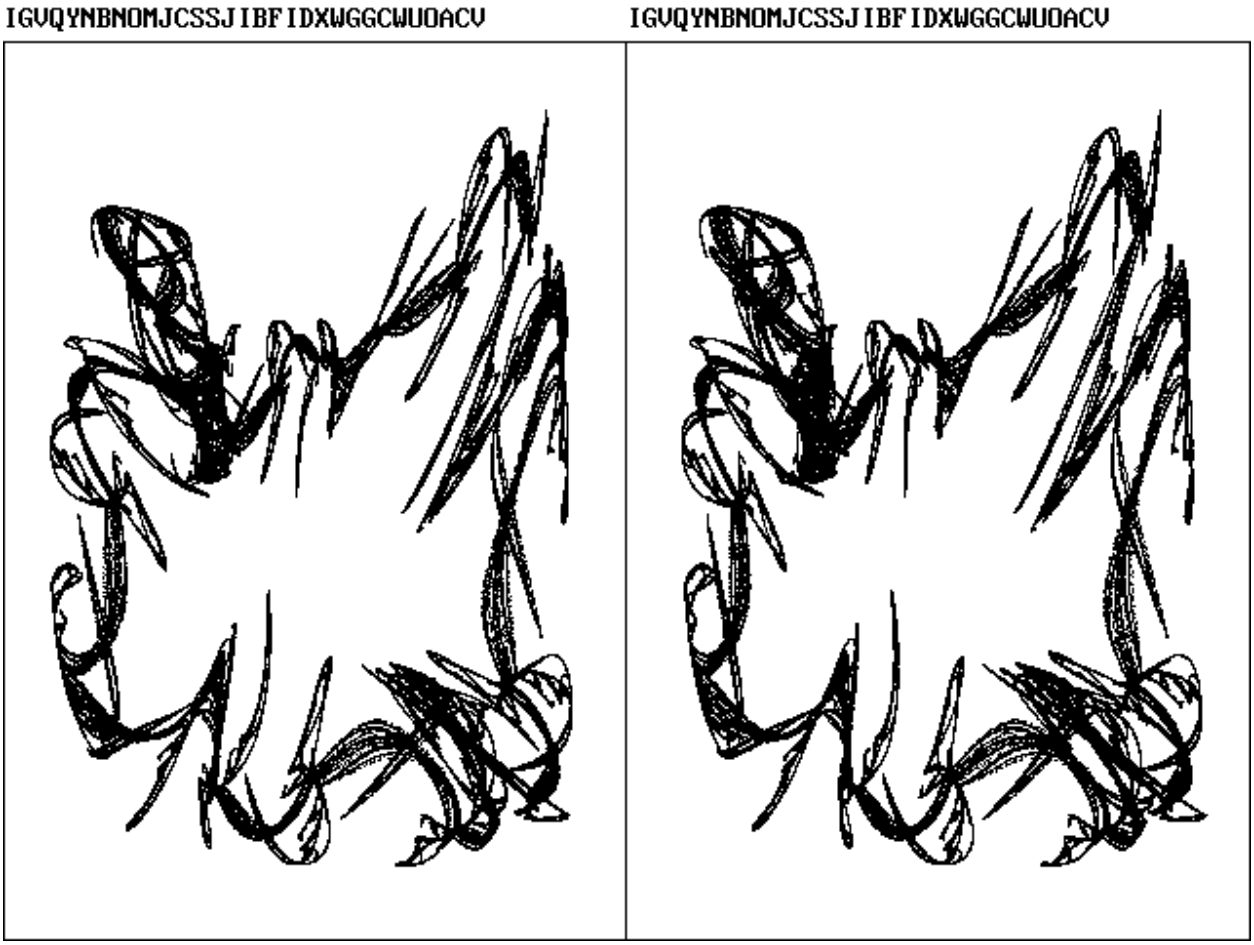


Figure 4-53. Stereo pair of three-dimensional quadratic map

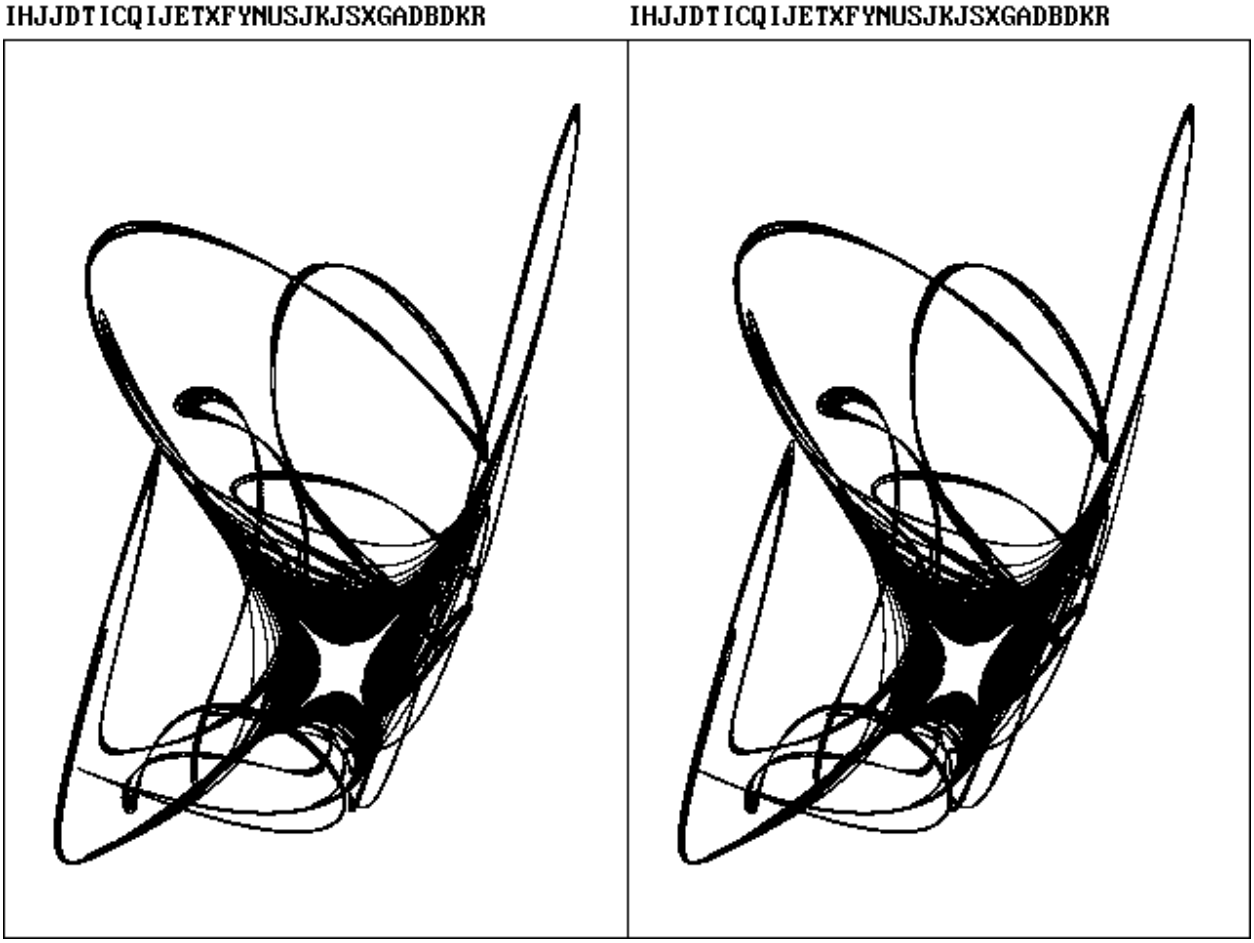


Figure 4-54. Stereo pair of three-dimensional quadratic map

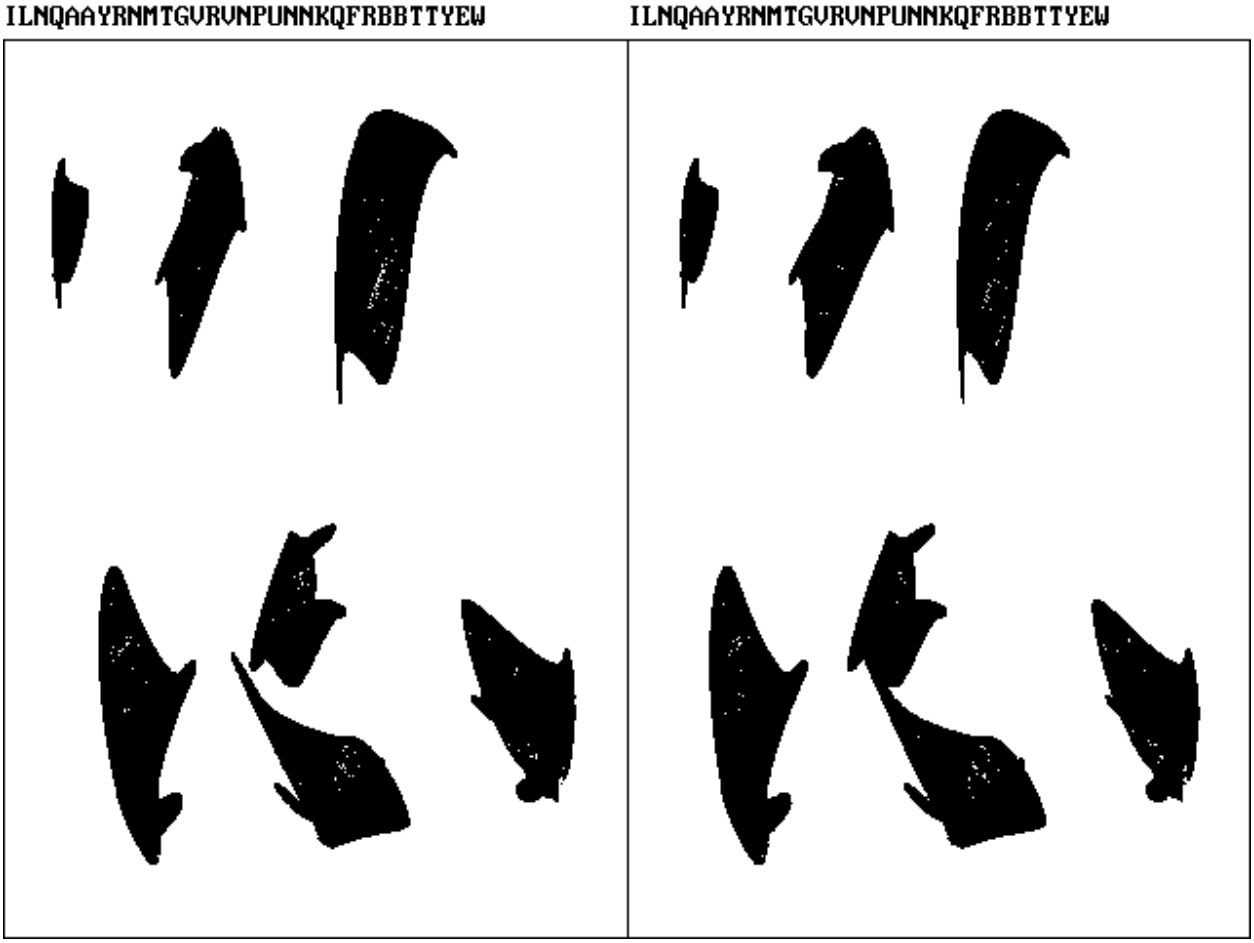


Figure 4-55. Stereo pair of three-dimensional quadratic map

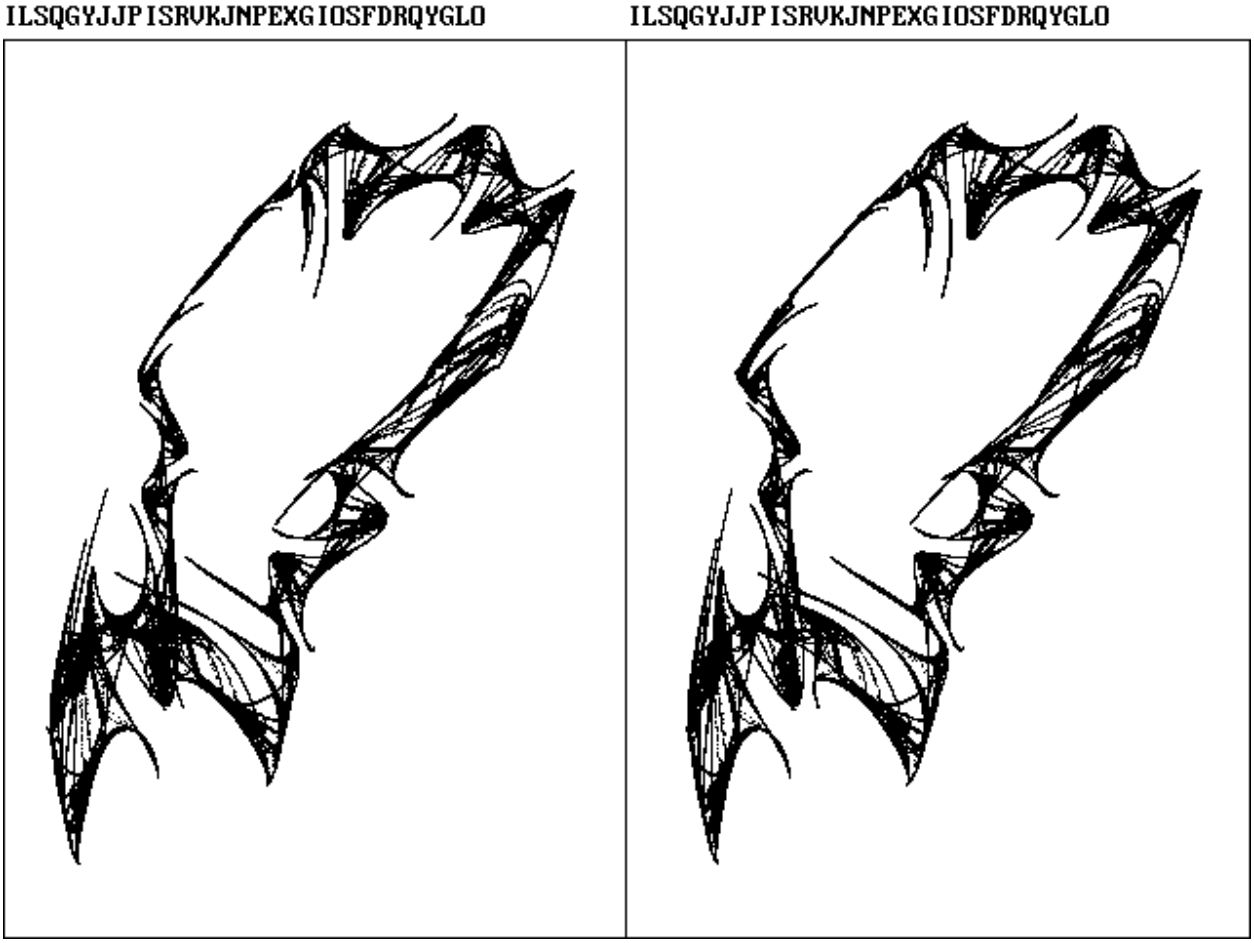


Figure 4-56. Stereo pair of three-dimensional quadratic map

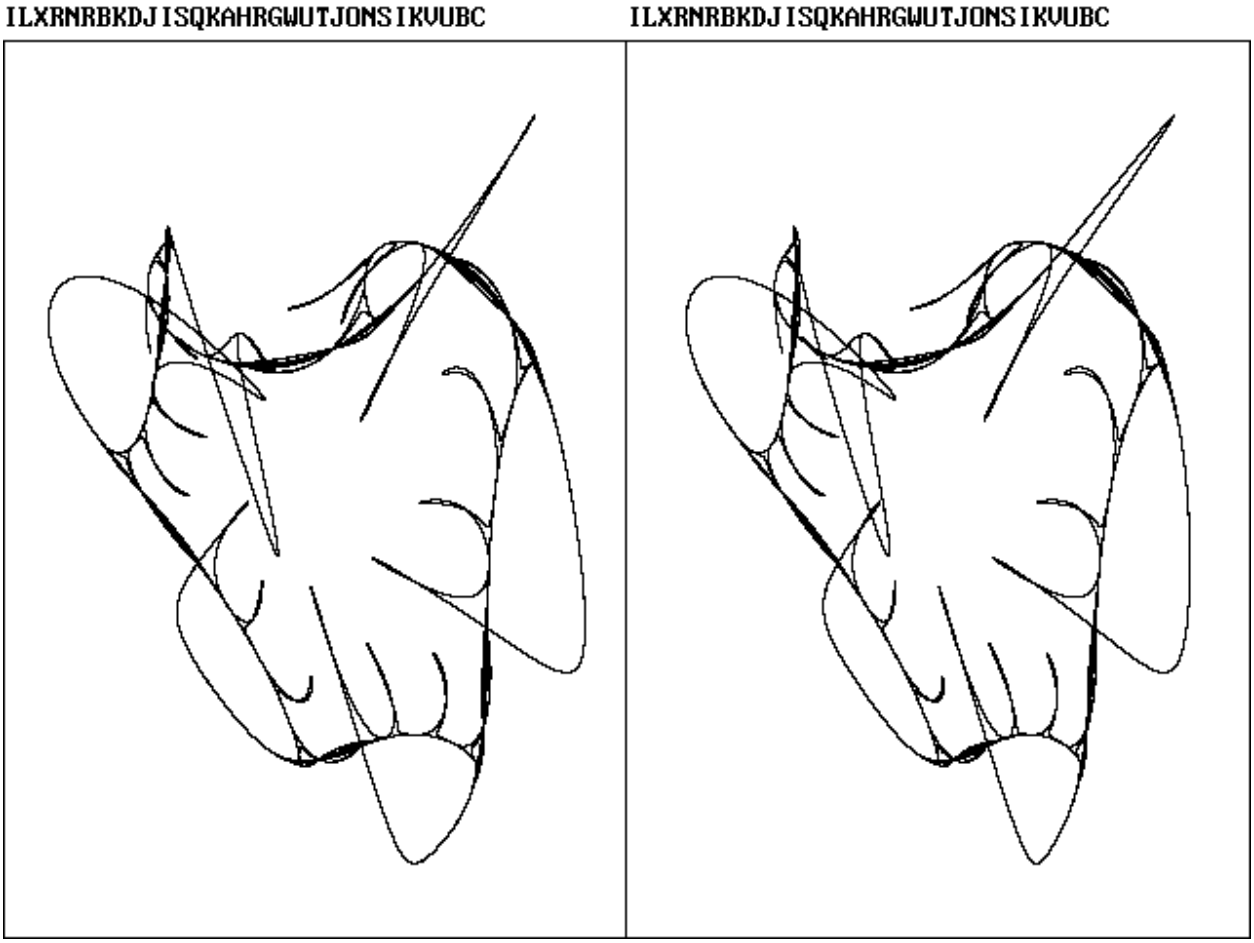


Figure 4-57. Stereo pair of three-dimensional quadratic map

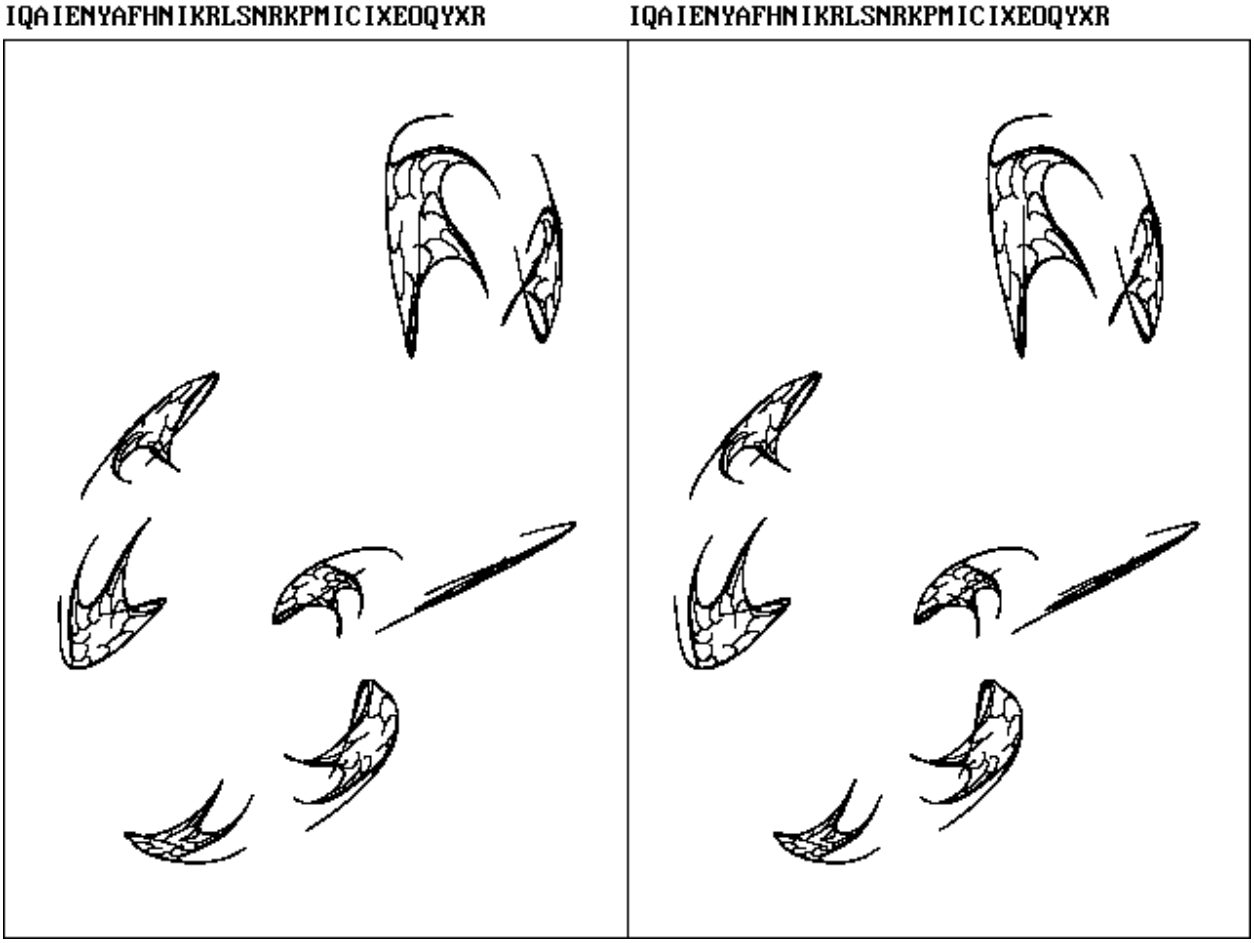


Figure 4-58. Stereo pair of three-dimensional quadratic map

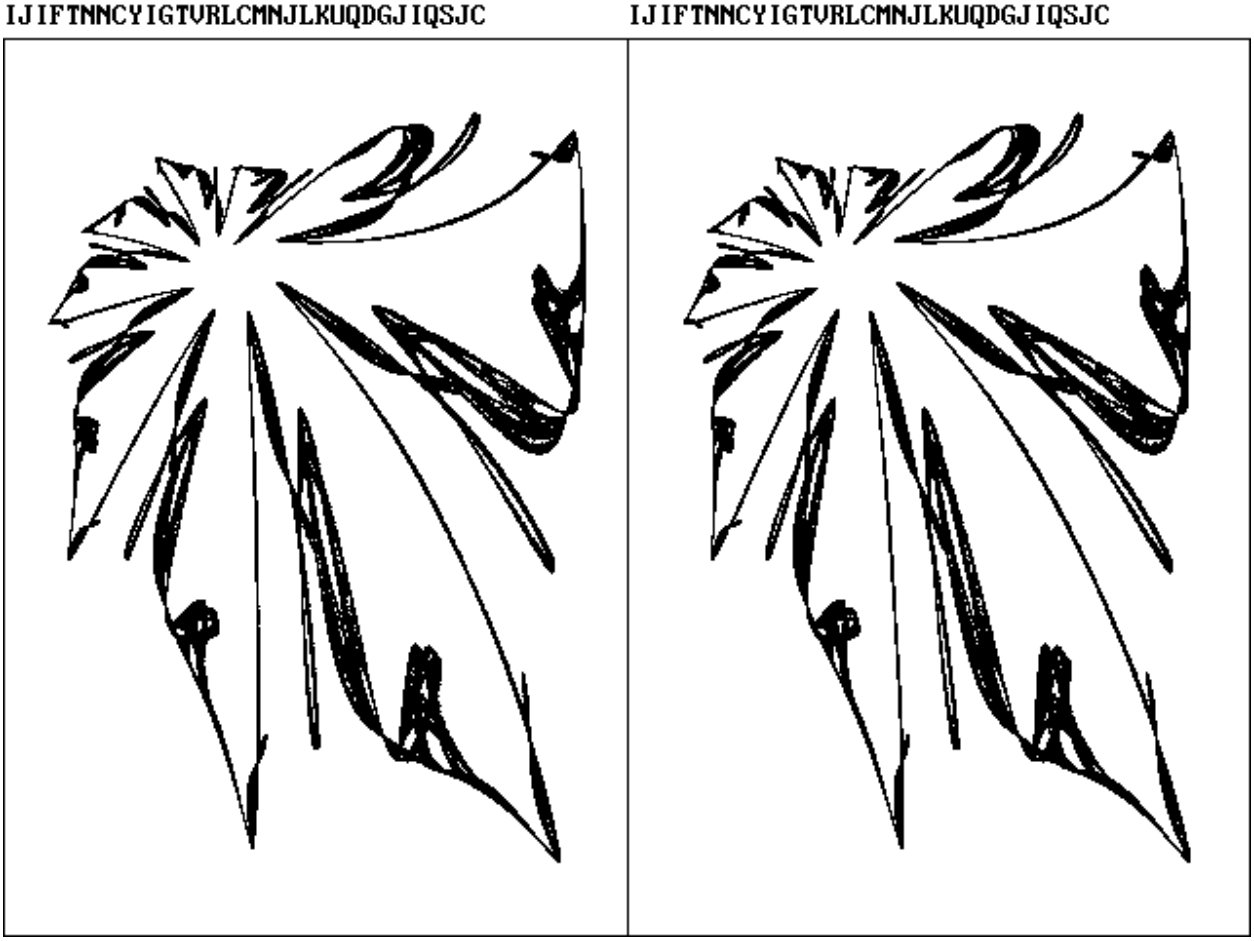


Figure 4-59. Stereo pair of three-dimensional quadratic map

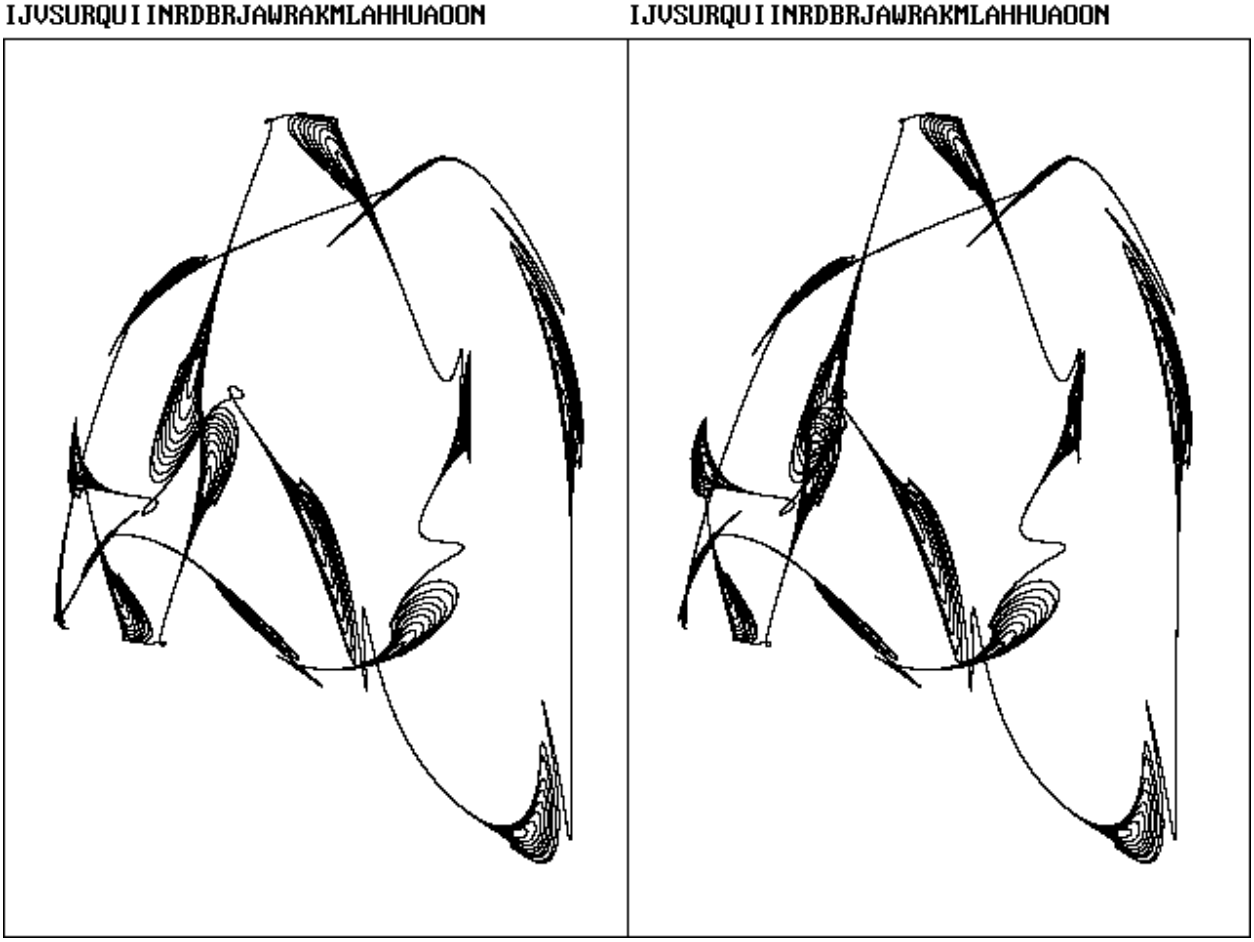


Figure 4-60. Stereo pair of three-dimensional quadratic map

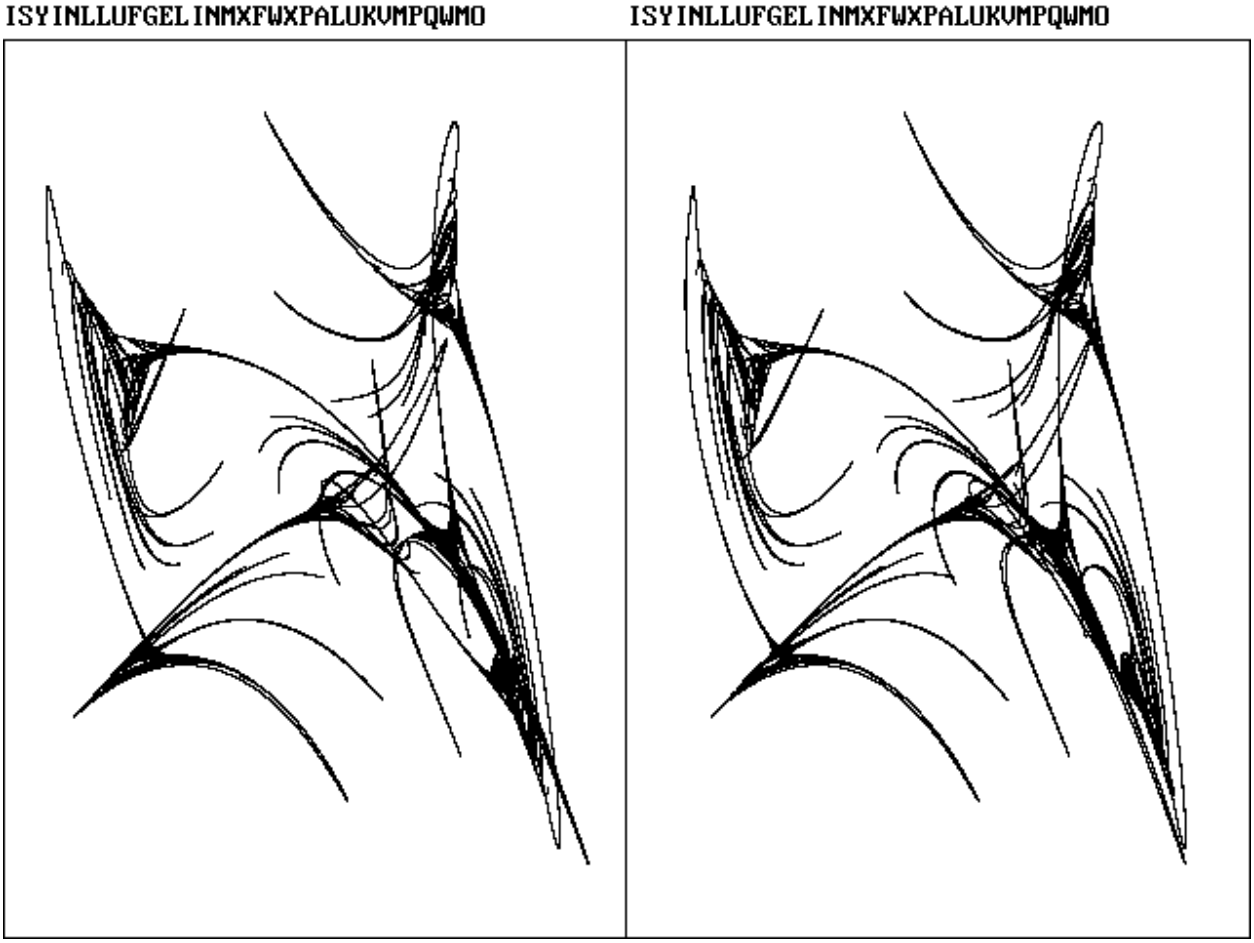


Figure 4-61. Stereo pair of three-dimensional quadratic map

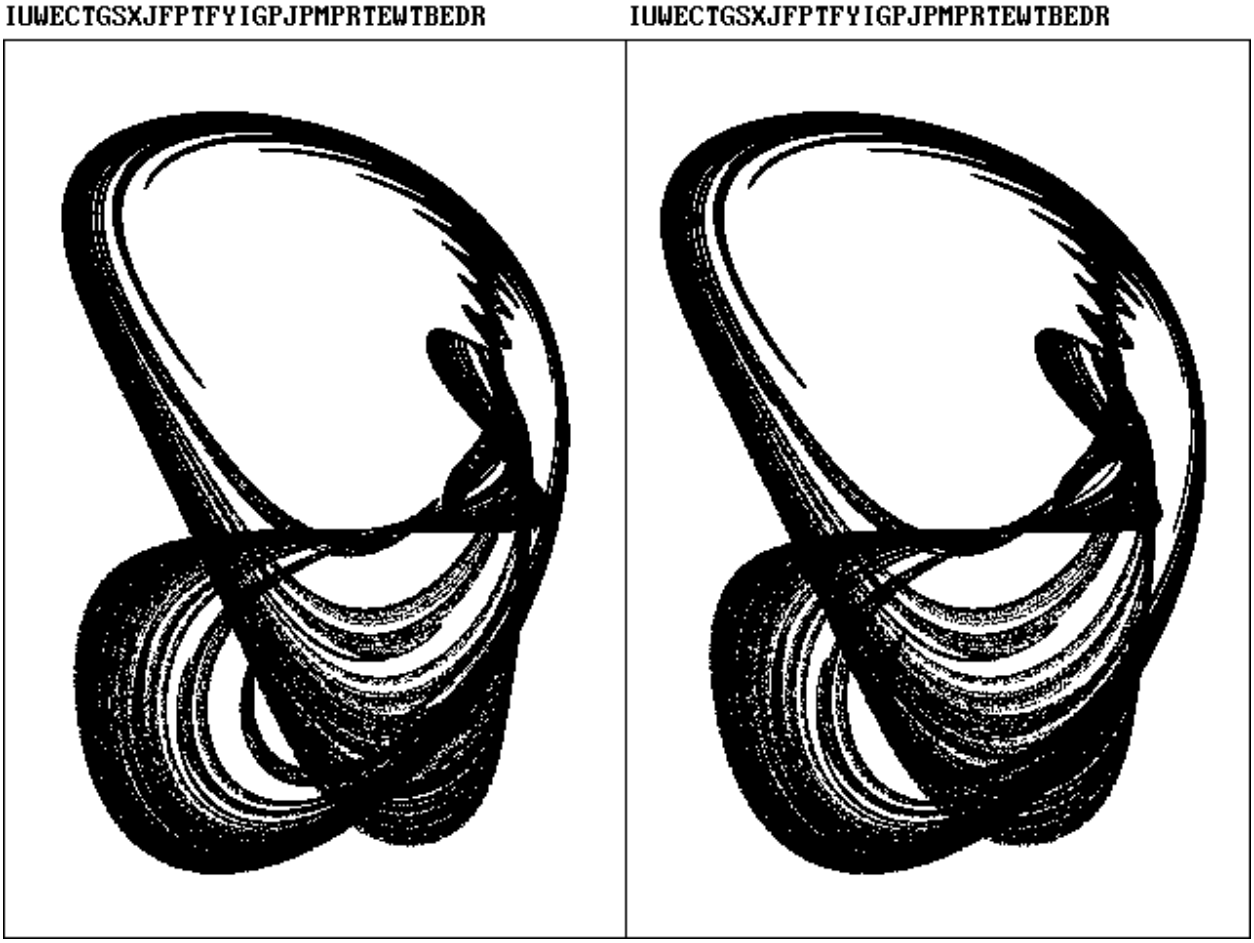


Figure 4-62. Stereo pair of three-dimensional quadratic map

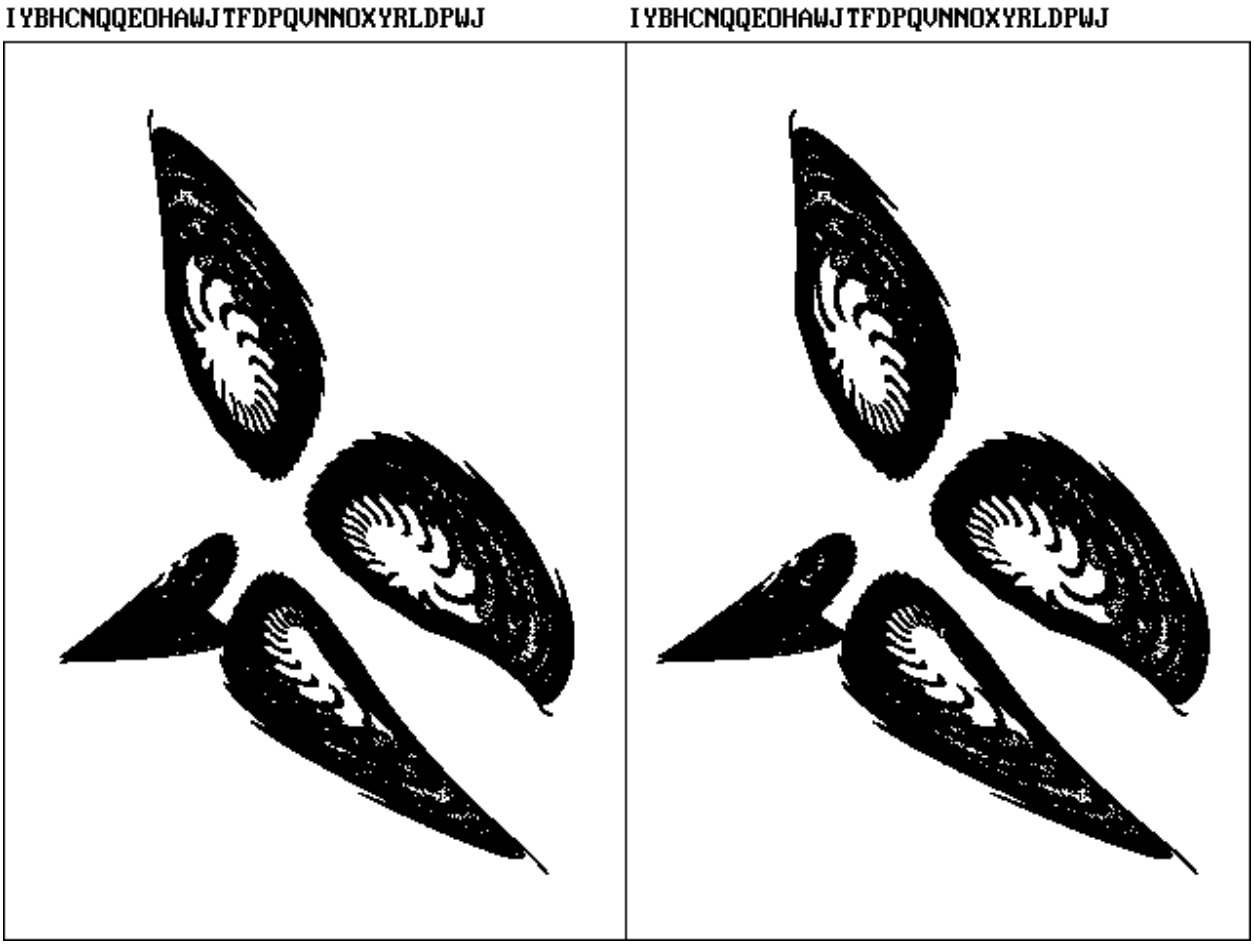


Figure 4-63. Stereo pair of three-dimensional cubic map

JJICKAFXIOXFVGOCIDNIVRPSFYPFGABXKKON... JJICKAFXIOXFVGOCIDNIVRPSFYPFGABXKKON...

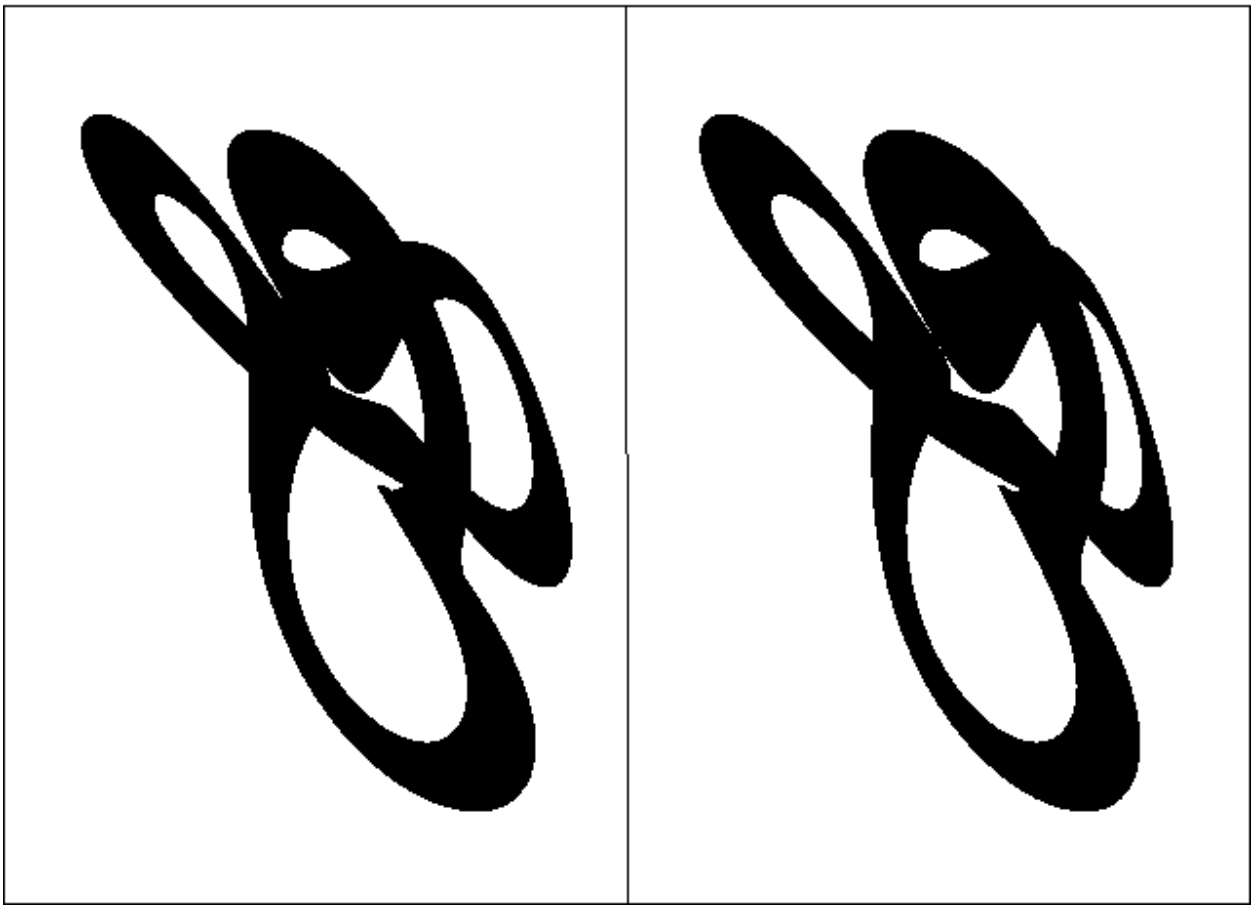


Figure 4-64. Stereo pair of three-dimensional cubic map

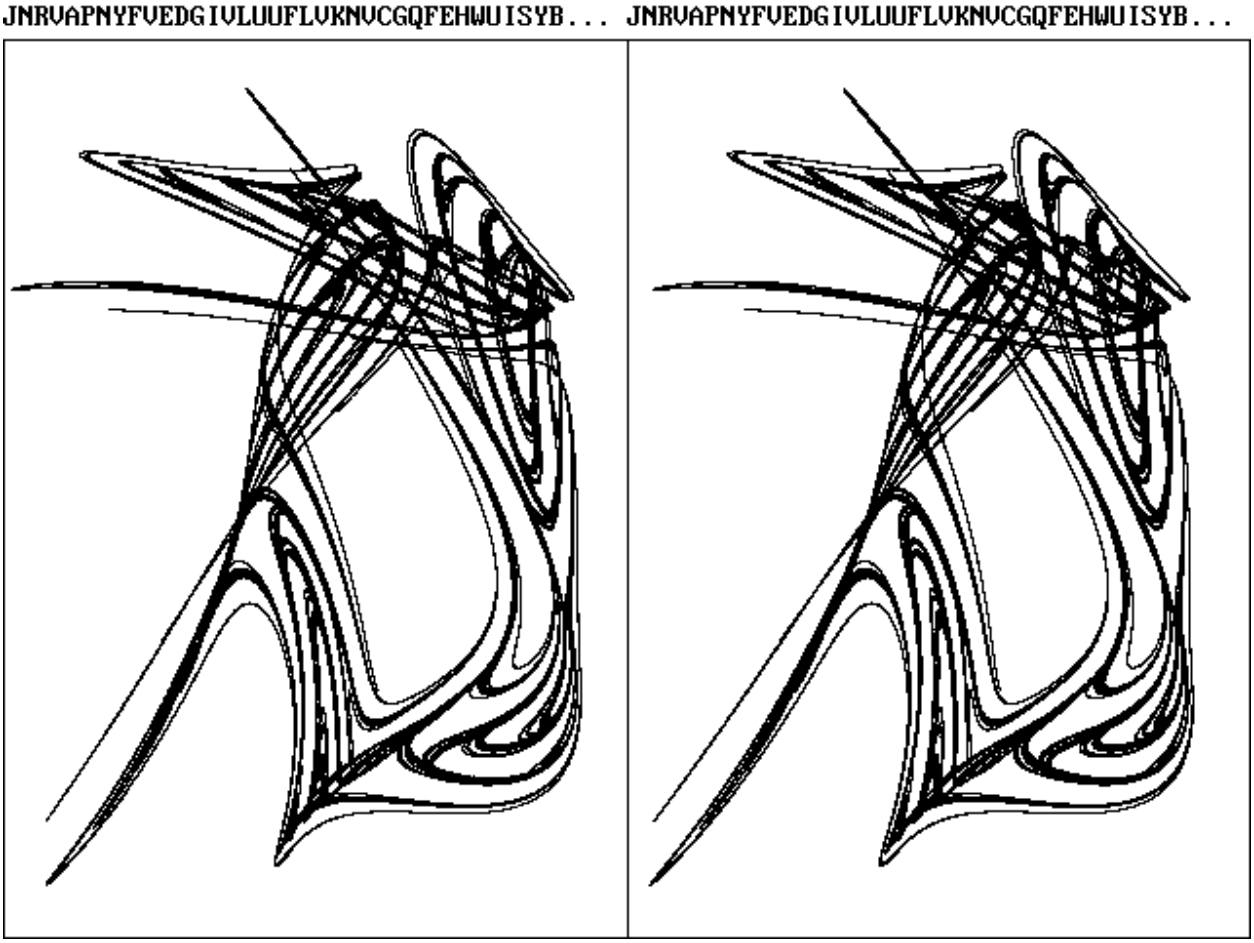


Figure 4-65. Stereo pair of three-dimensional quartic map

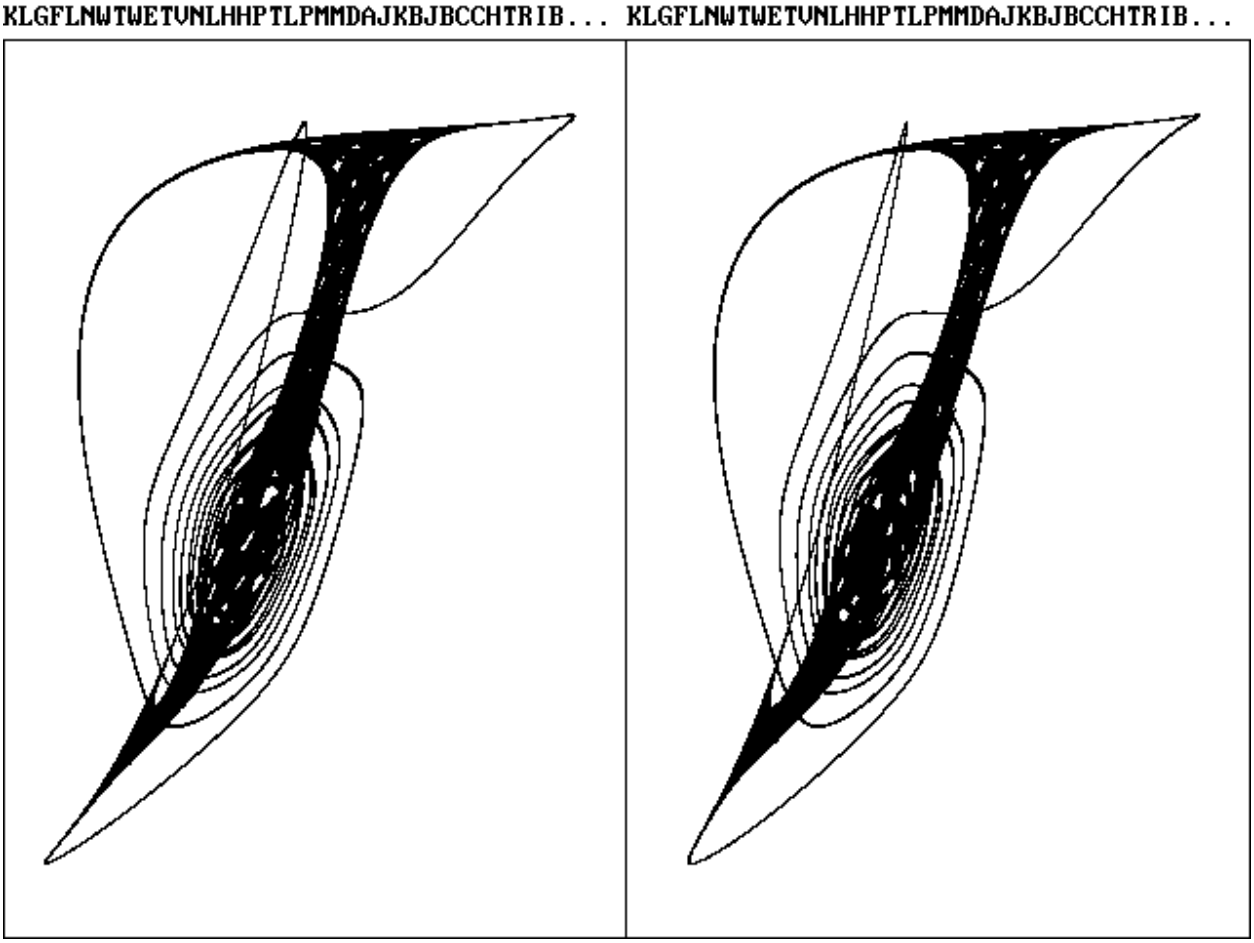


Figure 4-66. Stereo pair of three-dimensional quartic map

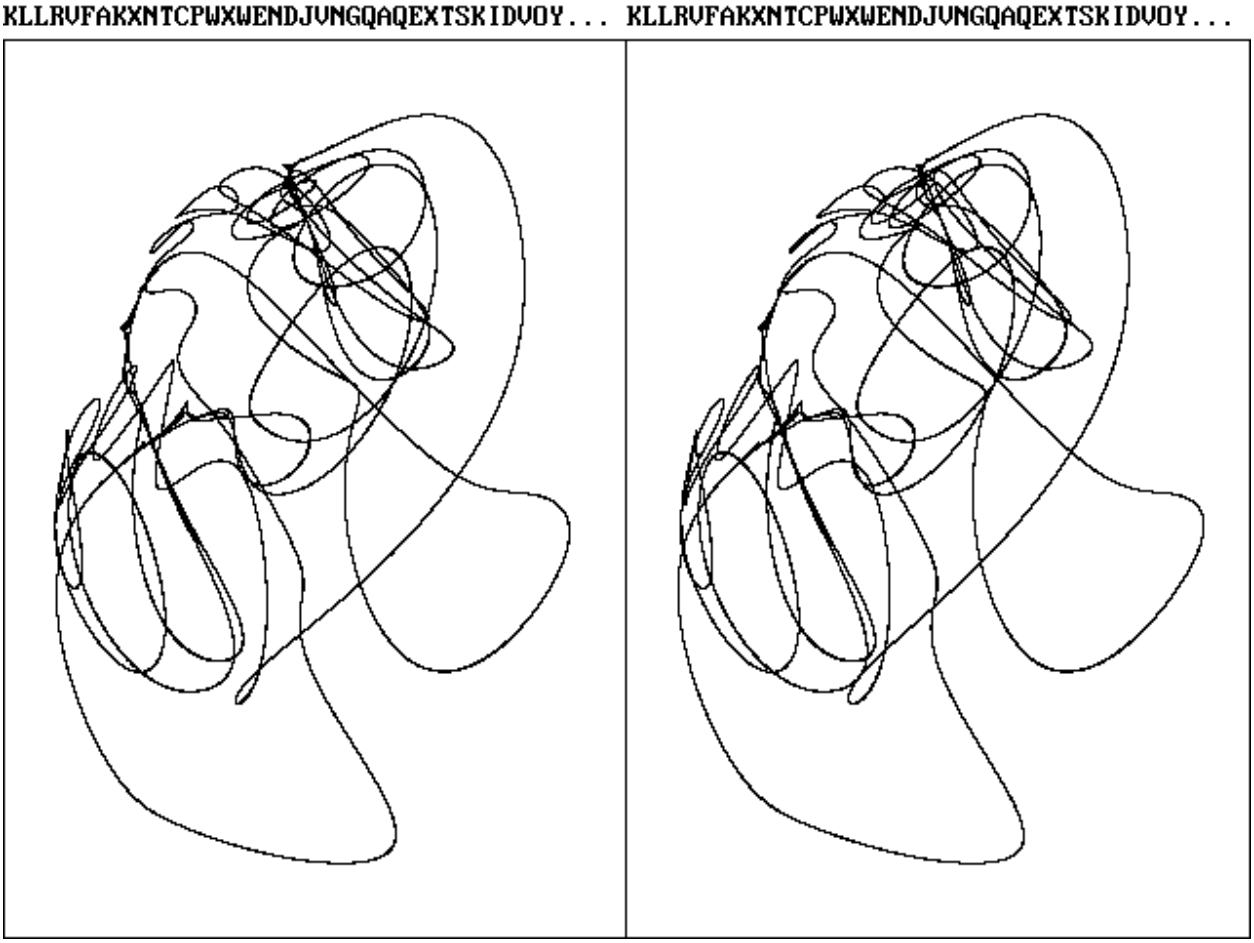
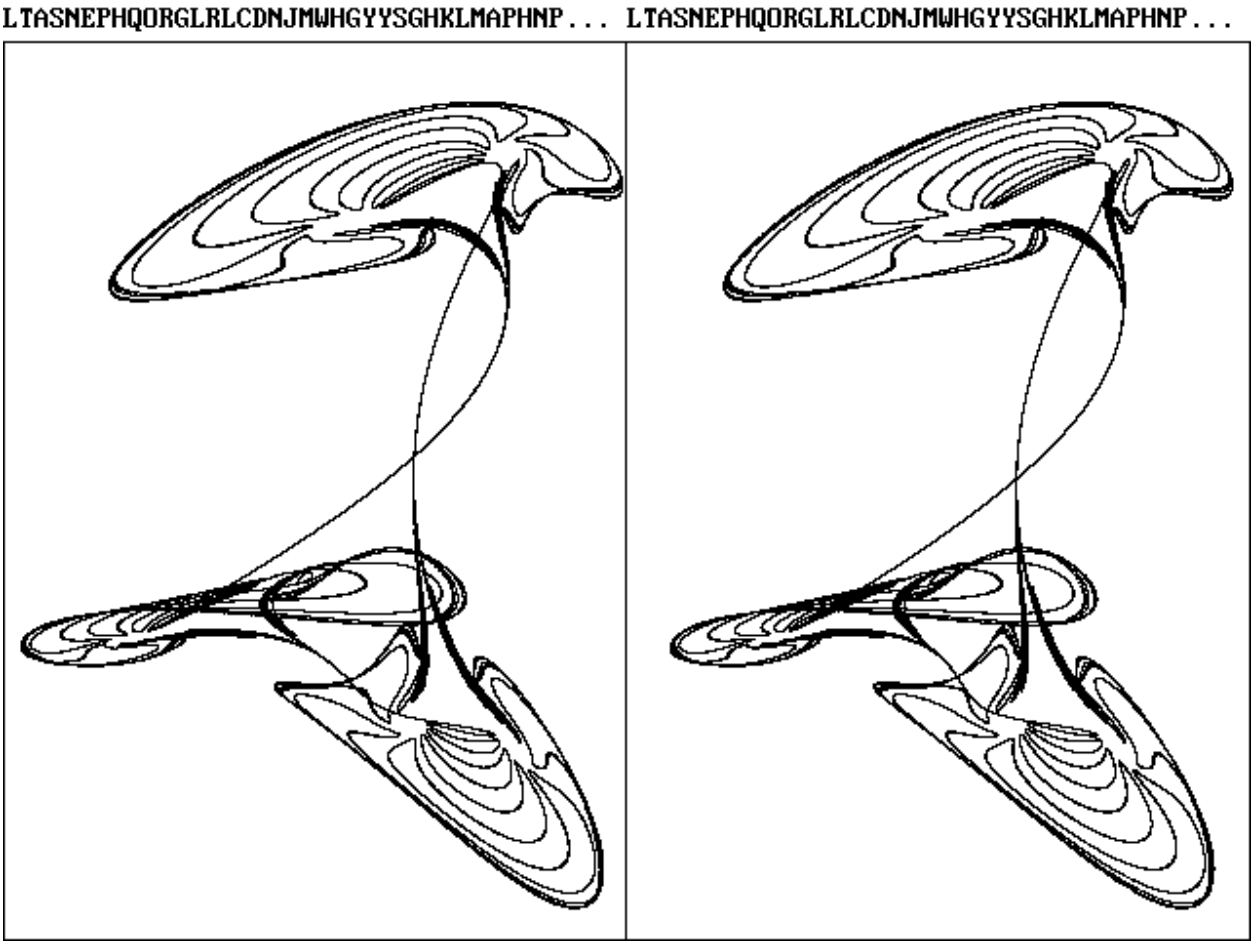


Figure 4-67. Stereo pair of three-dimensional quintic map



You may find it difficult to adjust to the images at first, but with practice you should be able to see them almost instantly. Viewing them should be relaxing, with a sensation resembling a blank stare. Effort is required to return to normal viewing, much like returning to the words on this page after gazing into the distance.

It might help to close your eyes momentarily and then reopen them if you are having trouble adapting. You can also buy an inexpensive hand *stereoscope* containing prisms that separate and magnify the images so that you can view them from a closer distance. Such viewing forces the side images out of your field of view and eases the adjustment to the middle image.

In the 1950s, the View-Master stereo viewer was very popular for home use, and many stereoscopic photographs were produced. Their popularity has waned, and the View-Master is no longer made, but similar inexpensive models can still be

found in toy stores, often with images of cartoon characters. Stereo images are usually photographed with a dual camera whose separation can be increased beyond the normal eye separation to enhance the depth sensation in what is called *hyperstereo*.

Stereoscopic images are used extensively by geologists and cartographers to determine terrain elevation from aerial photographs. As an airplane or satellite travels across the Earth, photographs are taken at two positions separated by a distance much greater than the distance between the eyes. When viewed through a stereoscope, the Earth appears as a scaled model viewed from just a foot or so above, and it is easy to discern the elevation changes.

The preceding technique is called *free* viewing. An alternate and more difficult technique, called *short-focus viewing*, can also be used to view the stereo pairs. Here the procedure is to place the figure at arm's length but to look at a point about halfway to the figure. It may help to hold your finger at the halfway point and to cross your eyes until you see a single image of your finger. You should then see the three images of the figure float up off the page in the plane of your finger. The middle image should be three-dimensional. It may be difficult to keep the image from wavering and returning to the plane of the page. Squinting sometimes helps.

Some people find short-focus viewing easier than free viewing. If you are instinctively short-foucus viewing, you will find that the right side of Figure 4-67 is farther from you than the left side. With free viewing, it will be closer.

An advantage of short-focus viewing is that the images can be separated by a much larger distance, and so it works for projection on a large screen in a classroom or auditorium. However, note that the image is in-out reversed from what it is with free viewing. This reversal is called *pseudostereo*. With anaglyphs, pseudostereo can be obtained by reversing the glasses. Pseudostereo images usually would be very undesirable and would lead to all kinds of visual contradictions, but with our strange attractors, which have no other visual depth cues, it makes little difference. It can even be an advantage to be able to view the objects in either of these ways. Can you guess what you will see if you turn the figures upside down? Think about it, and then give it a try.

Note that with free viewing, the left image disappears when you close your right eye and the right image disappears when you close your left eye. With short-focus viewing, the opposite occurs. However, it is incorrect to assume that each eye sees only one of the images. Both eyes see both images, and the images fuse into one when your eyes are aimed in the proper direction.

4.8 Slices

We will discuss one final way to view attractors resulting from three-dimensional maps. Low-dimensional attractors are like loosely wound balls of string, whereas high-dimensional ones are more like loaves of bread filled with holes. Anaglyphs and stereo pairs are effective for cases of low dimension, but as the dimension increases, the attractor becomes too opaque, and the illusion of depth is lost.

Carrying the loaf-of-bread analogy a bit further, you could imagine slicing the loaf into a large number of very thin slices. The result is to decrease the dimension of the object by one. For example, an object with a fractal dimension of 2.5 would become an object of dimension of 1.5 in each slice. This is an example of what is called a Poincaré section (as in "cross section").

Perhaps it's easier to consider a specific case. Suppose the attractor were a loosely wound ball of very thin string. The attractor would then be essentially one-dimensional. The string would cross the slices at a number of points. Thus the slices would contain dots wherever the string pierced them. A set of a finite number of dots is an object of zero dimension.

If the attractor were a loosely crumpled piece of paper, its dimension would be close to two. If you were to cut a thin slice through the crumpled paper, you would be left with a handful of wormlike paper strings, which are one-dimensional objects. You should now set the book down, get a piece of paper and a pair of scissors, and try it for yourself. Be sure to make the slice as thin as possible.

With our maps, which contain only a finite number of points, we cannot make the slices too thin, lest they contain so few points as to be invisible. Furthermore, it is impractical to look at all the slices if they are very thin because there are too many of them. As always, we have to compromise. We'll use 16 slices and lay them out in a 4 by 4 array so that we can see them all at once. On the computer screen, this method entails a serious sacrifice in resolution, but it does illustrate the principle. You might want to experiment with using a larger number of slices but displaying only a fraction of them. For example, try using 64 slices, and display every fourth one.

The modifications that are required to make the program produce a sliced display are shown in PROG17.

PROG17. Changes required in PROG16 to produce slices

```
1000 REM THREE-D MAP SEARCH (With Sliced Display)

1120 TRD% = 6                    'Display third dimension as slices



3260    IF TRD% <> 6 THEN GOTO 3310

3270      FOR I% = 1 TO 3

3280          XP = XL + I% * (XH - XL) / 4: LINE (XP, YL)-(XP, YH)

3290          YP = YL + I% * (YH - YL) / 4: LINE (XL, YP)-(XH, YP)

3300        NEXT I%



3760 IF Q$ = "R" THEN TRD% = (TRD% + 1) MOD 7: T% = 3: IF N > 999 THEN N = 999:
GOSUB 5600



4530        IF TRD% = 6 THEN PRINT "slices    "



5280 IF TRD% <> 6 THEN GOTO 5330

5290    DZ = (15 * (Z - ZMIN) / (ZMAX - ZMIN) + .5) / 16

5300    XP = (XP - XL + (INT(16 * DZ) MOD 4) * (XH - XL)) / 4 + XL

5310    YP = (YP - YL + (3 - INT(4 * DZ) MOD 4) * (YH - YL)) / 4 + YL

5320    PSET (XP, YP)

5330 RETURN
```

Figures 4-68 through 4-83 show some sample attractors displayed as a succession of slices. The succession is from left to right and top to bottom in the same

way you read (in most Western languages, at least). In these cases attractors with dimensions greater than two have been chosen; otherwise, the dimension of the slices would be too small to be interesting.

Figure 4-68. Slices of a three-dimensional quadratic map

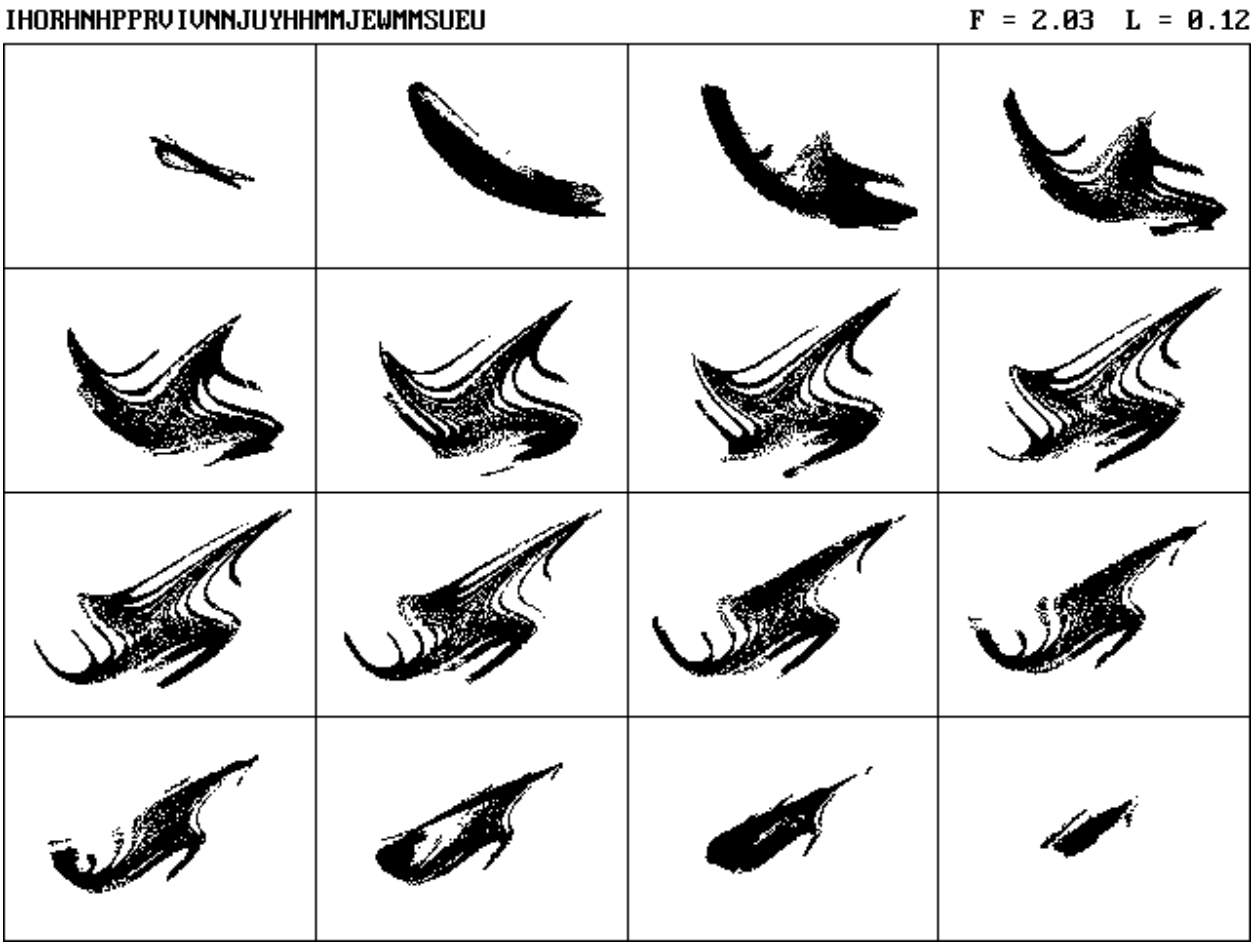


Figure 4-69. Slices of a three-dimensional quadratic map

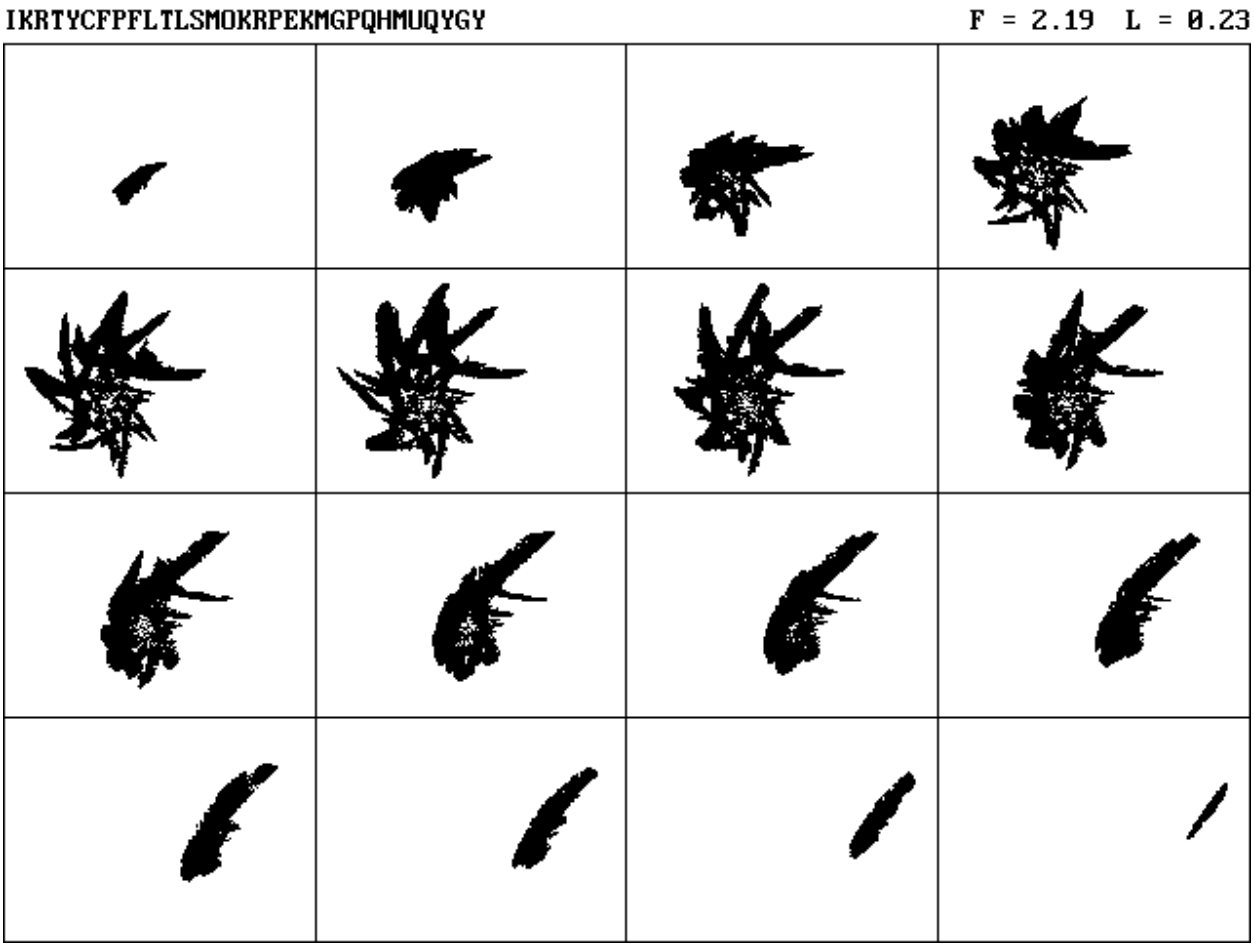


Figure 4-70. Slices of a three-dimensional quadratic map

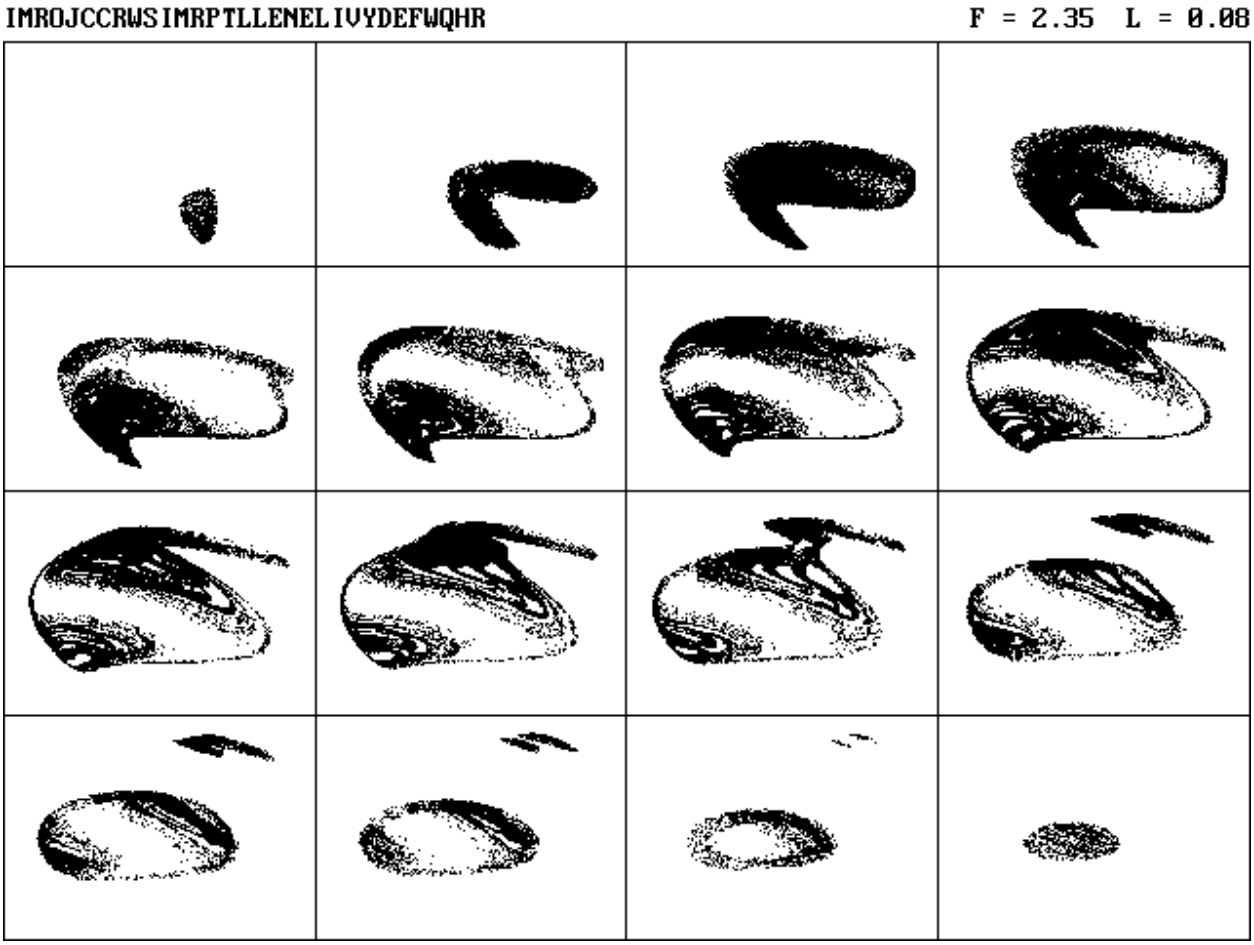


Figure 4-71. Slices of a three-dimensional quadratic map

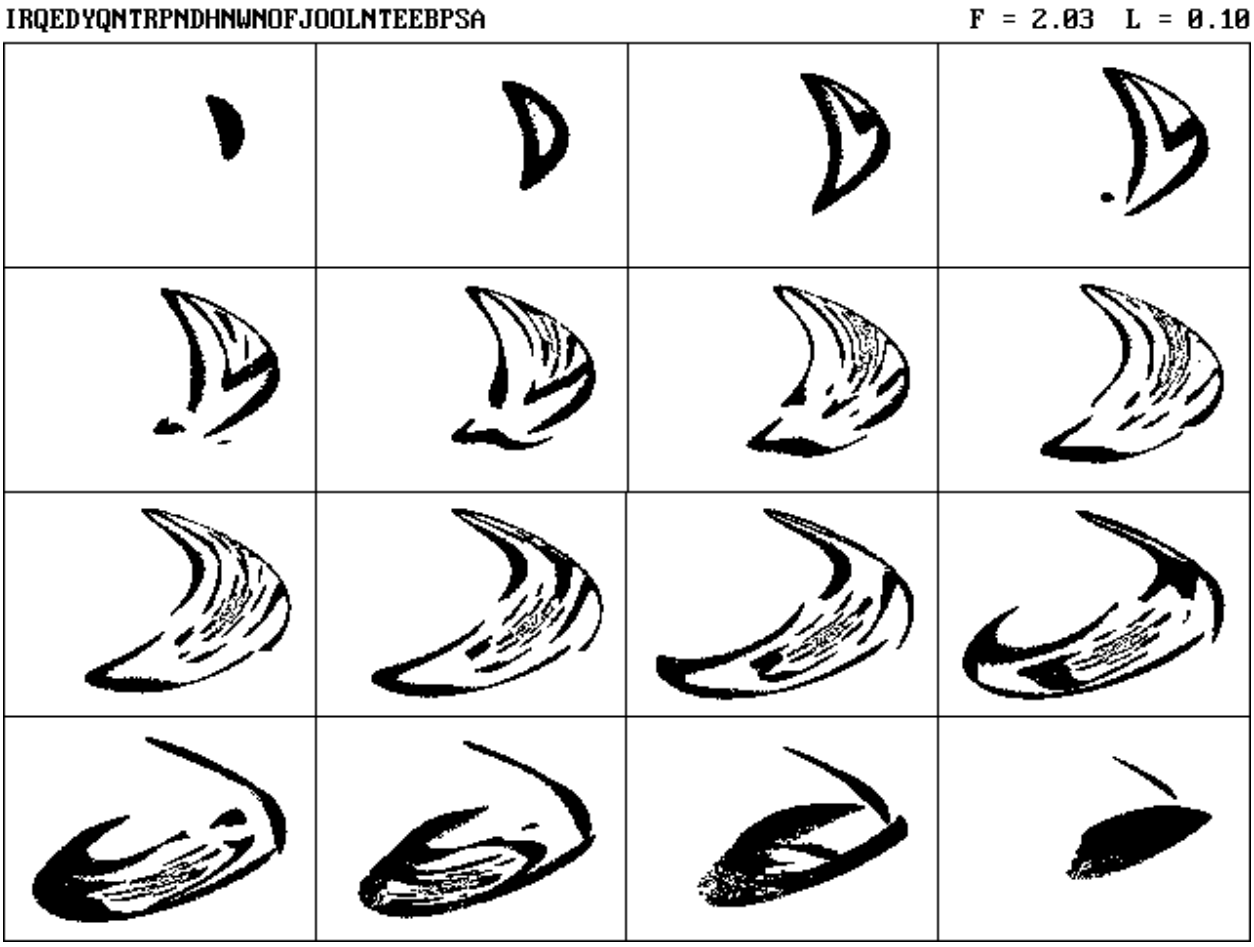


Figure 4-72. Slices of a three-dimensional cubic map

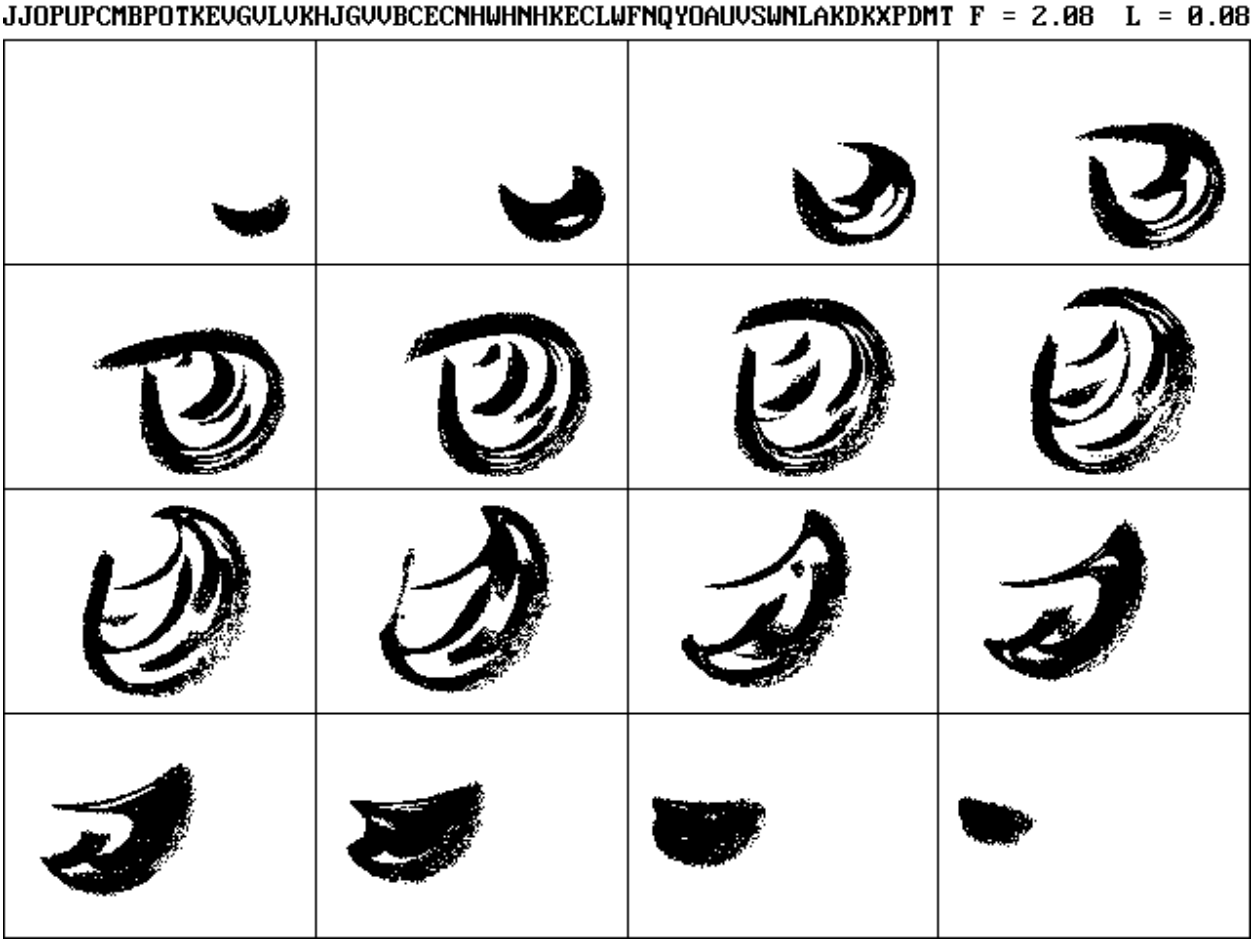


Figure 4-73. Slices of a three-dimensional cubic map

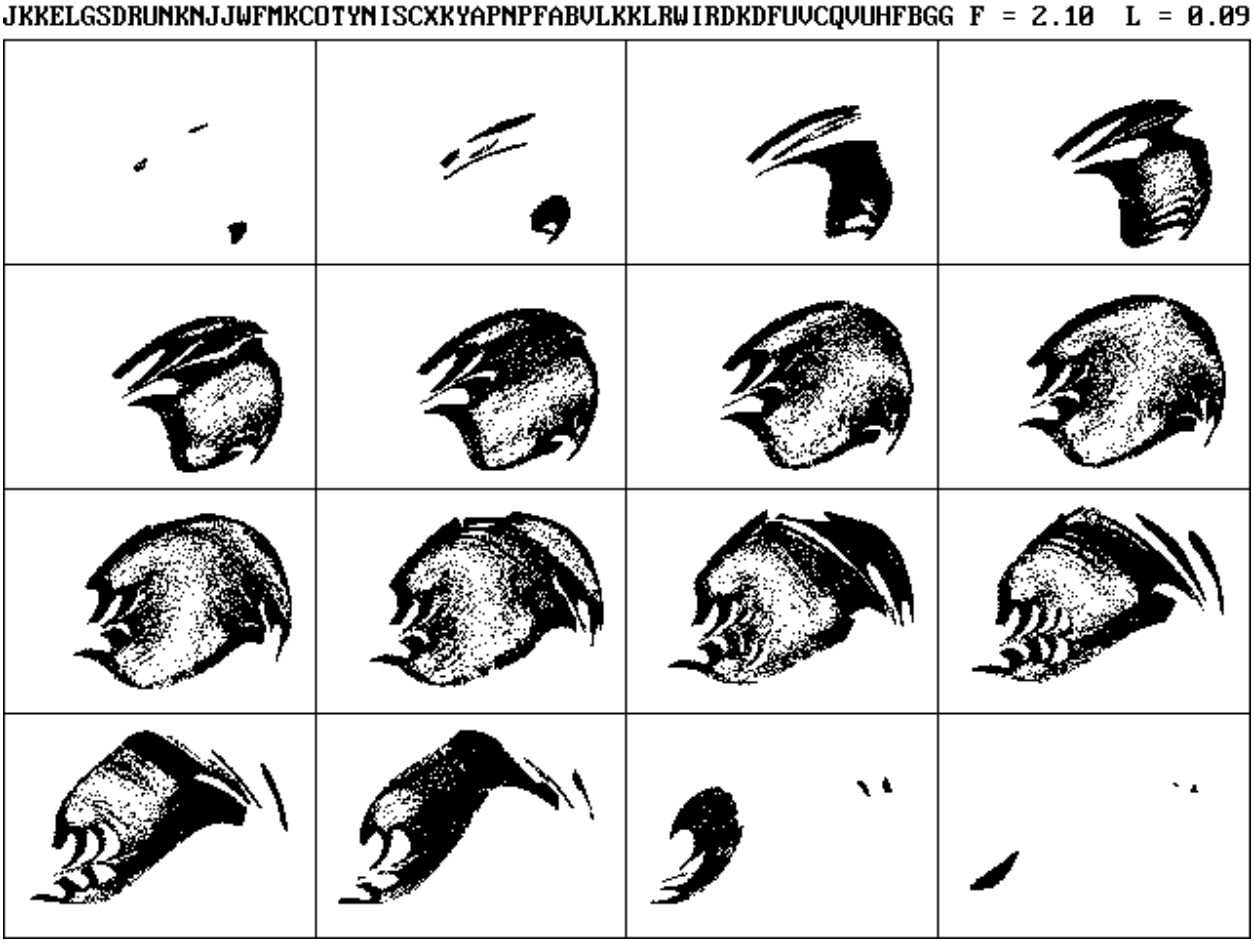


Figure 4-74. Slices of a three-dimensional cubic map

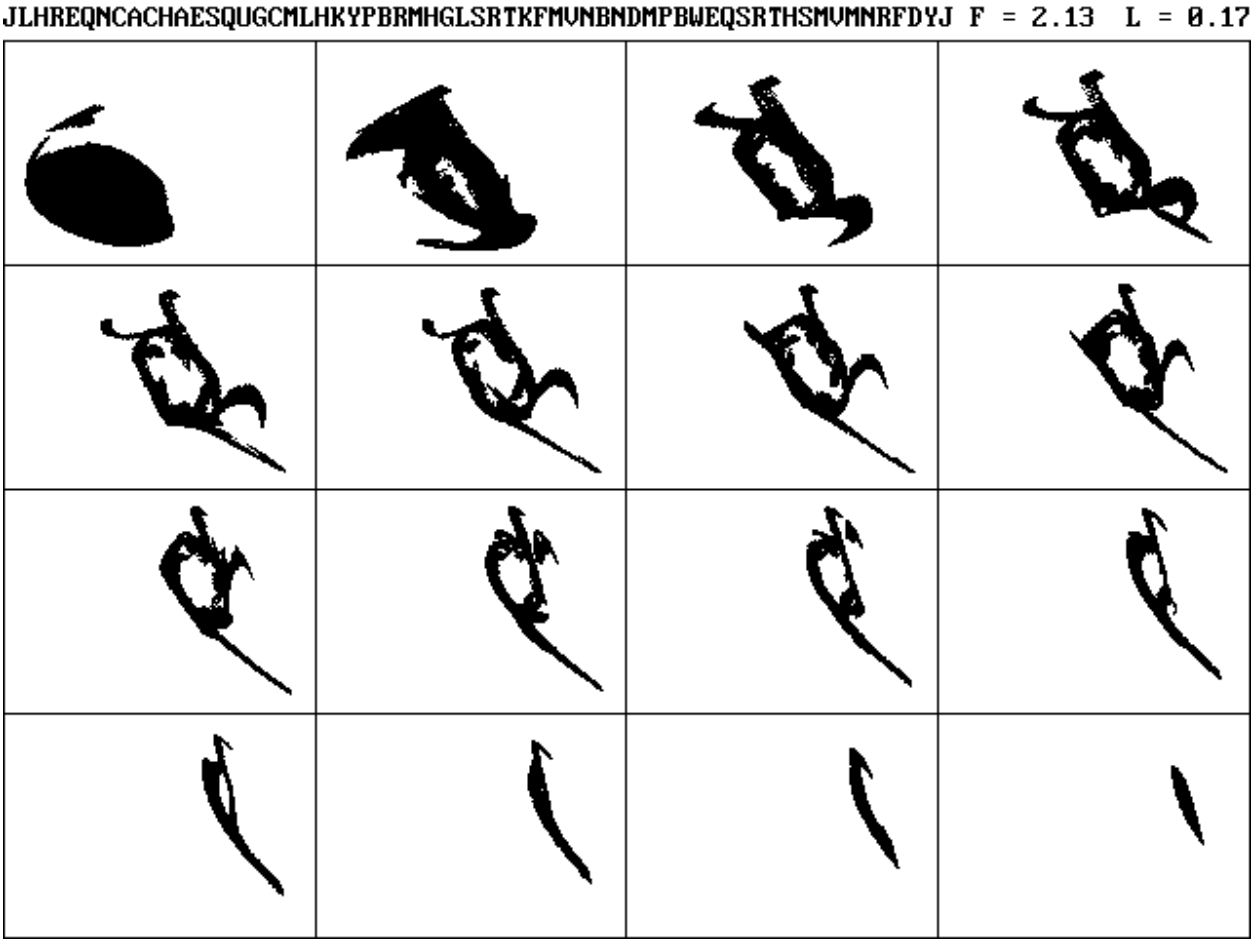


Figure 4-75. Slices of a three-dimensional cubic map

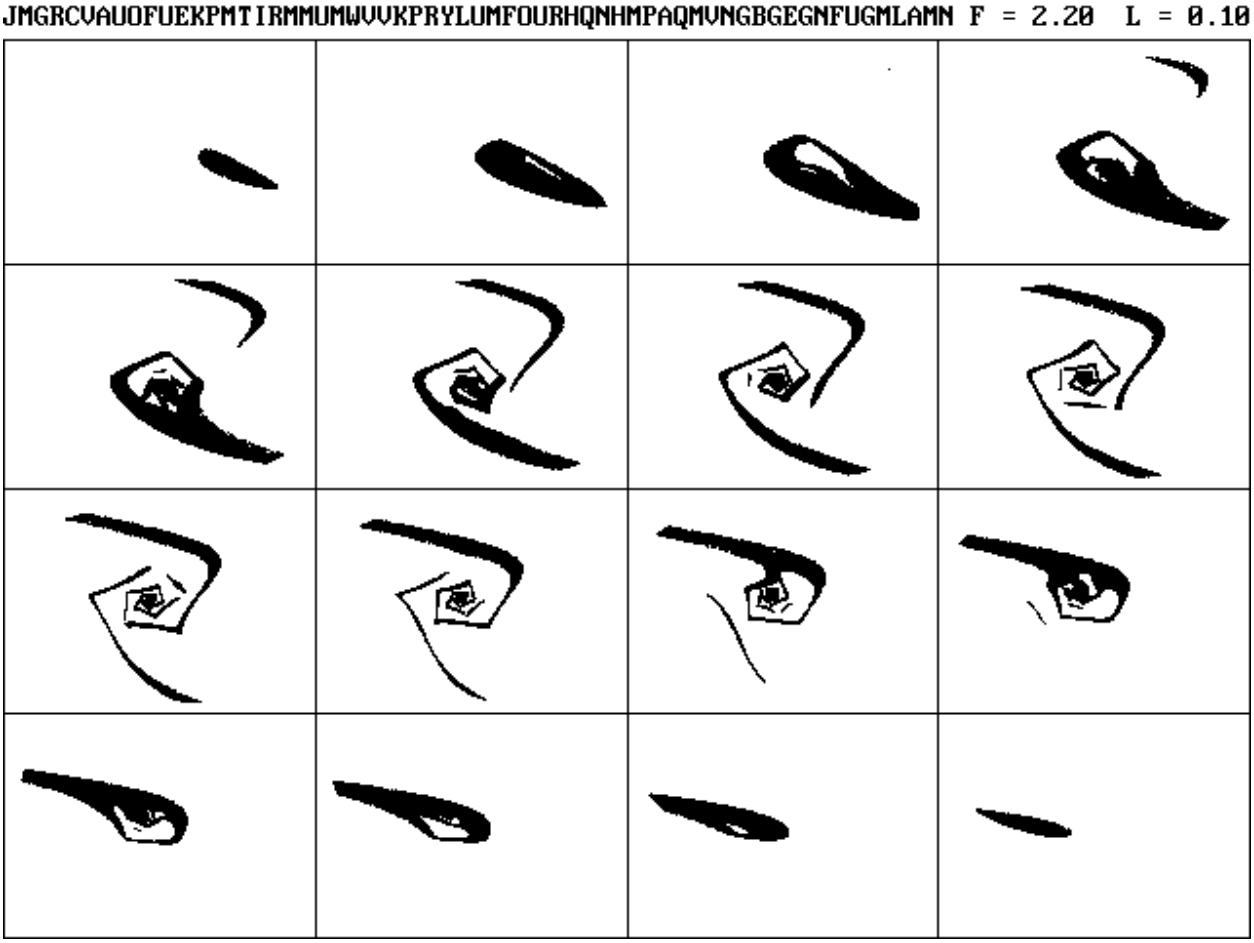


Figure 4-76. Slices of a three-dimensional quartic map

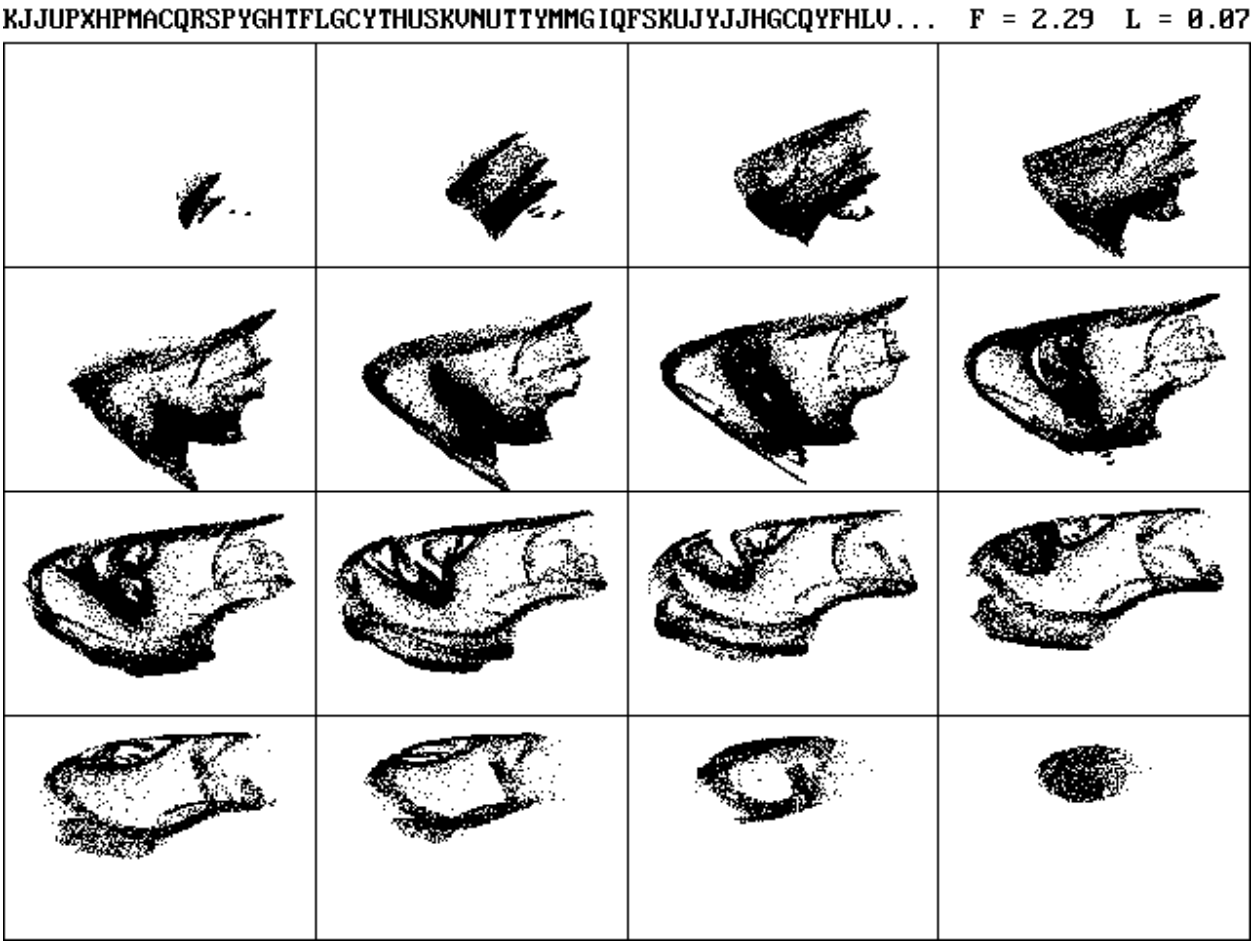


Figure 4-77. Slices of a three-dimensional quartic map

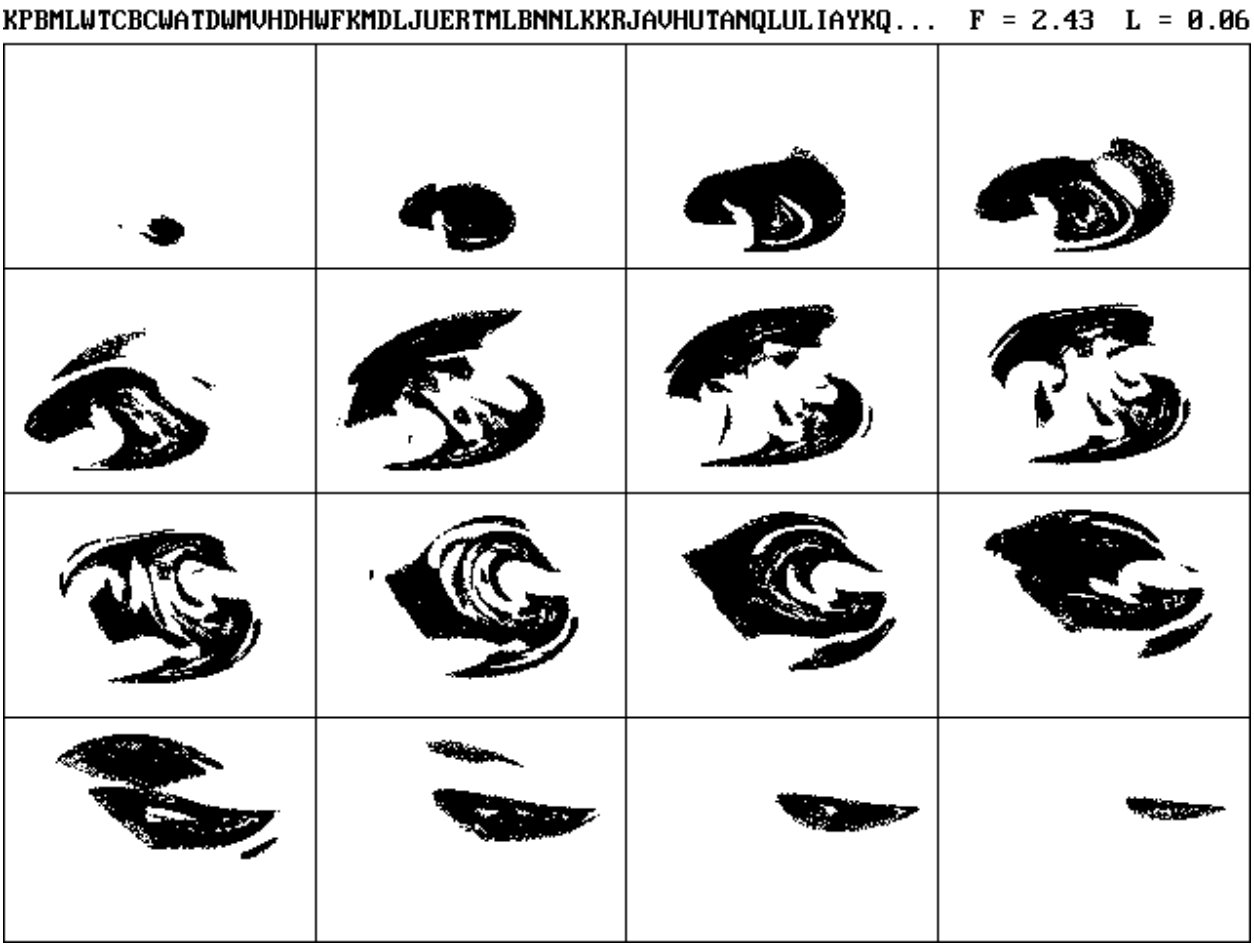


Figure 4-78. Slices of a three-dimensional quartic map

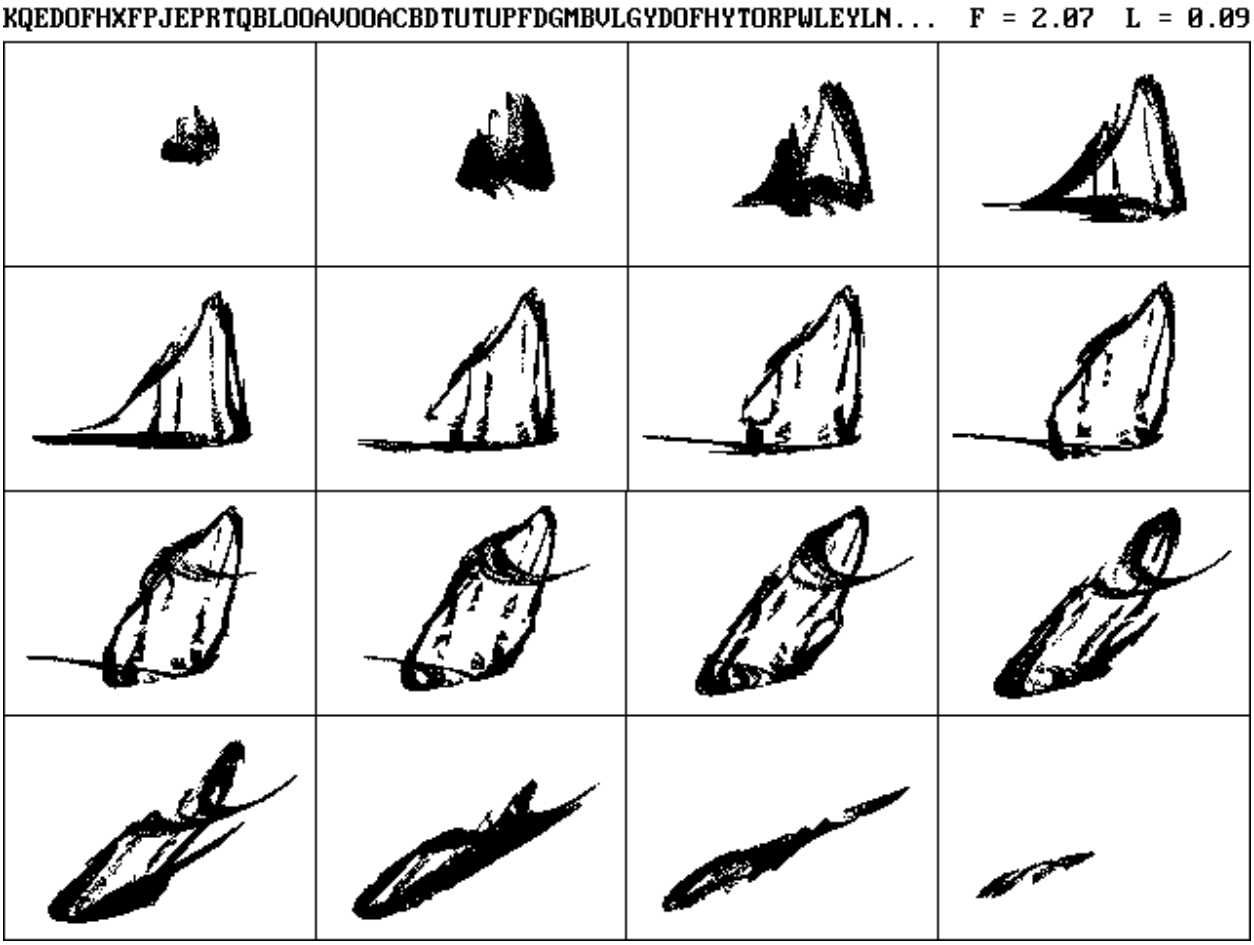


Figure 4-79. Slices of a three-dimensional quartic map

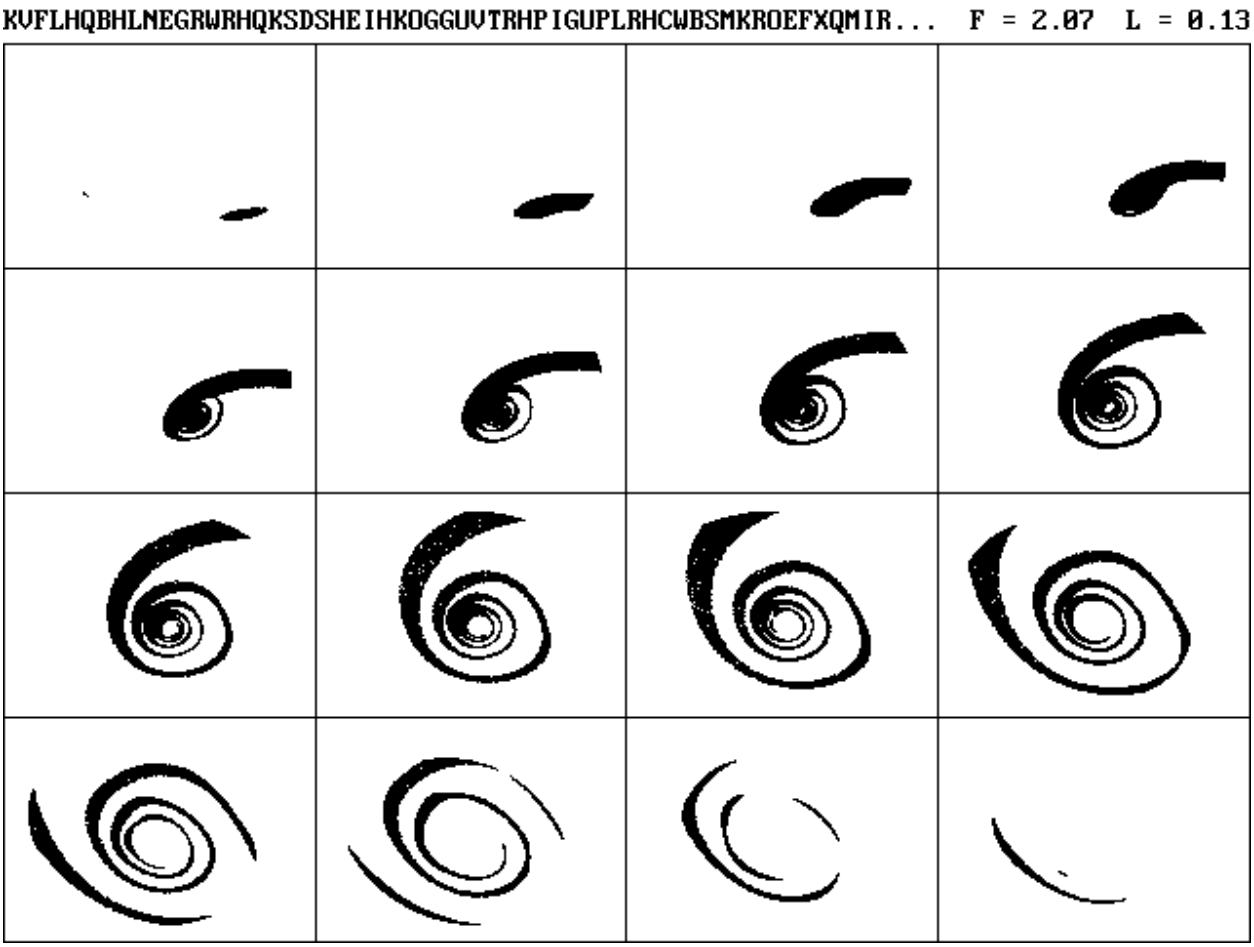


Figure 4-80. Slices of a three-dimensional quintic map

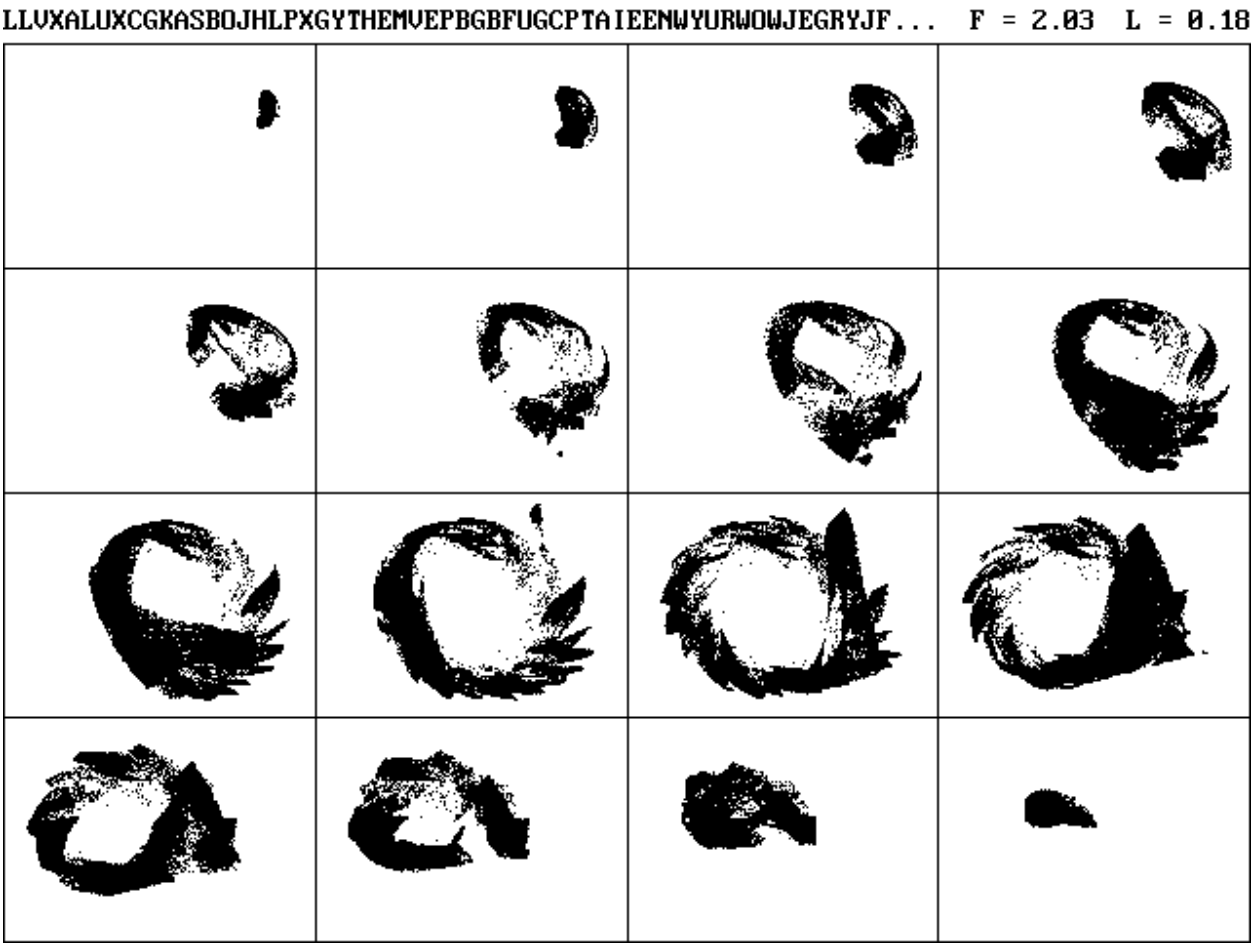


Figure 4-81. Slices of a three-dimensional quintic map

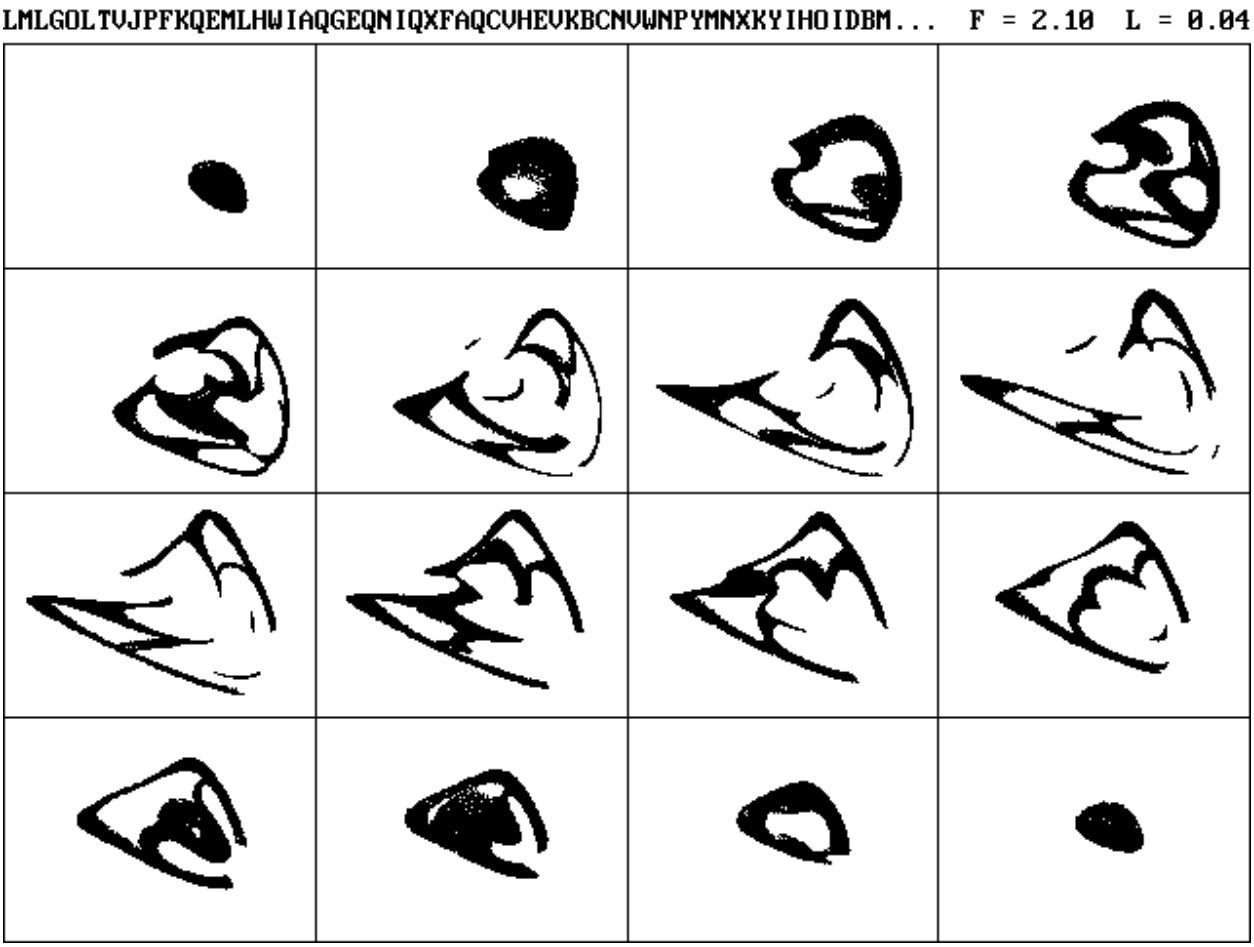


Figure 4-82. Slices of a three-dimensional quintic map

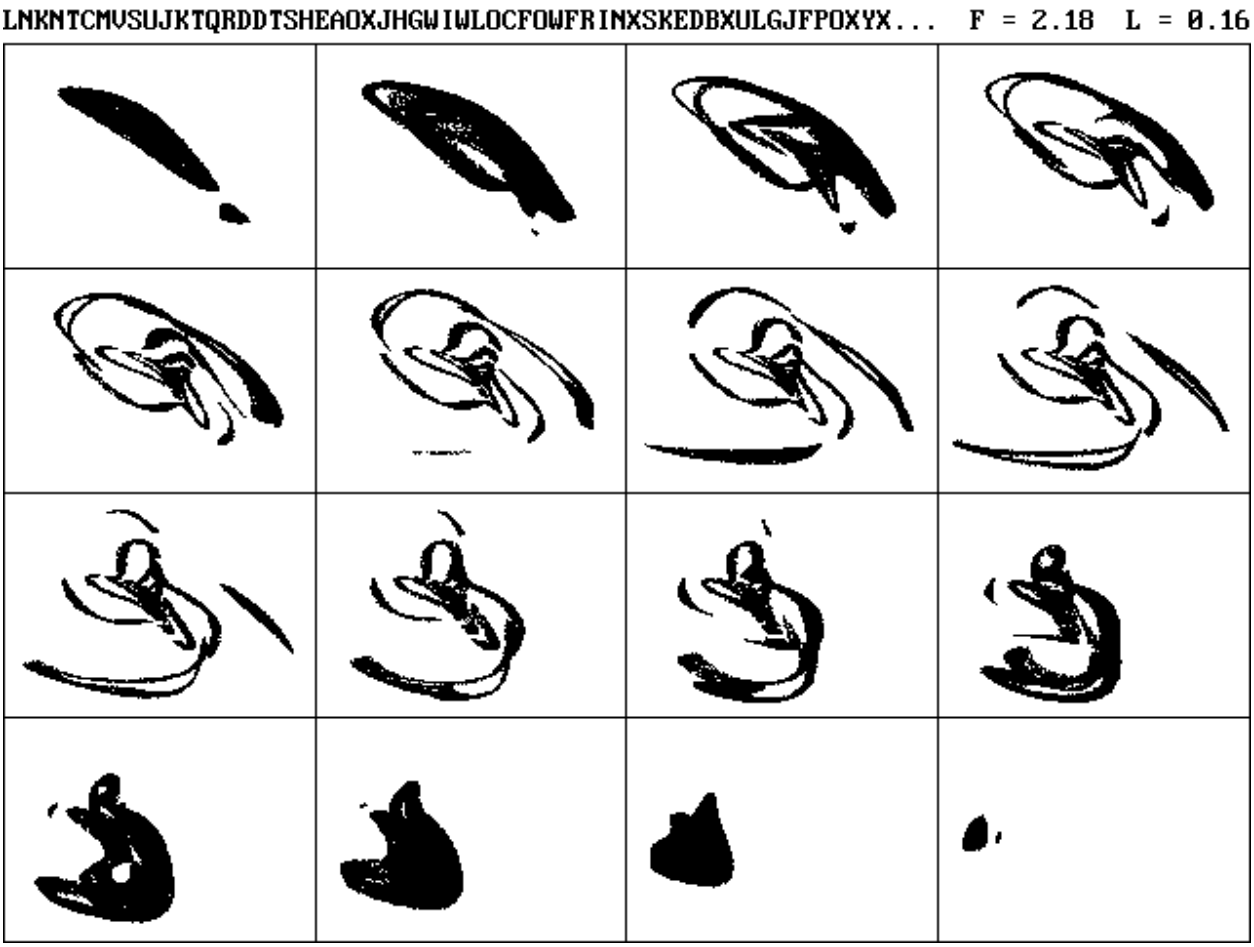
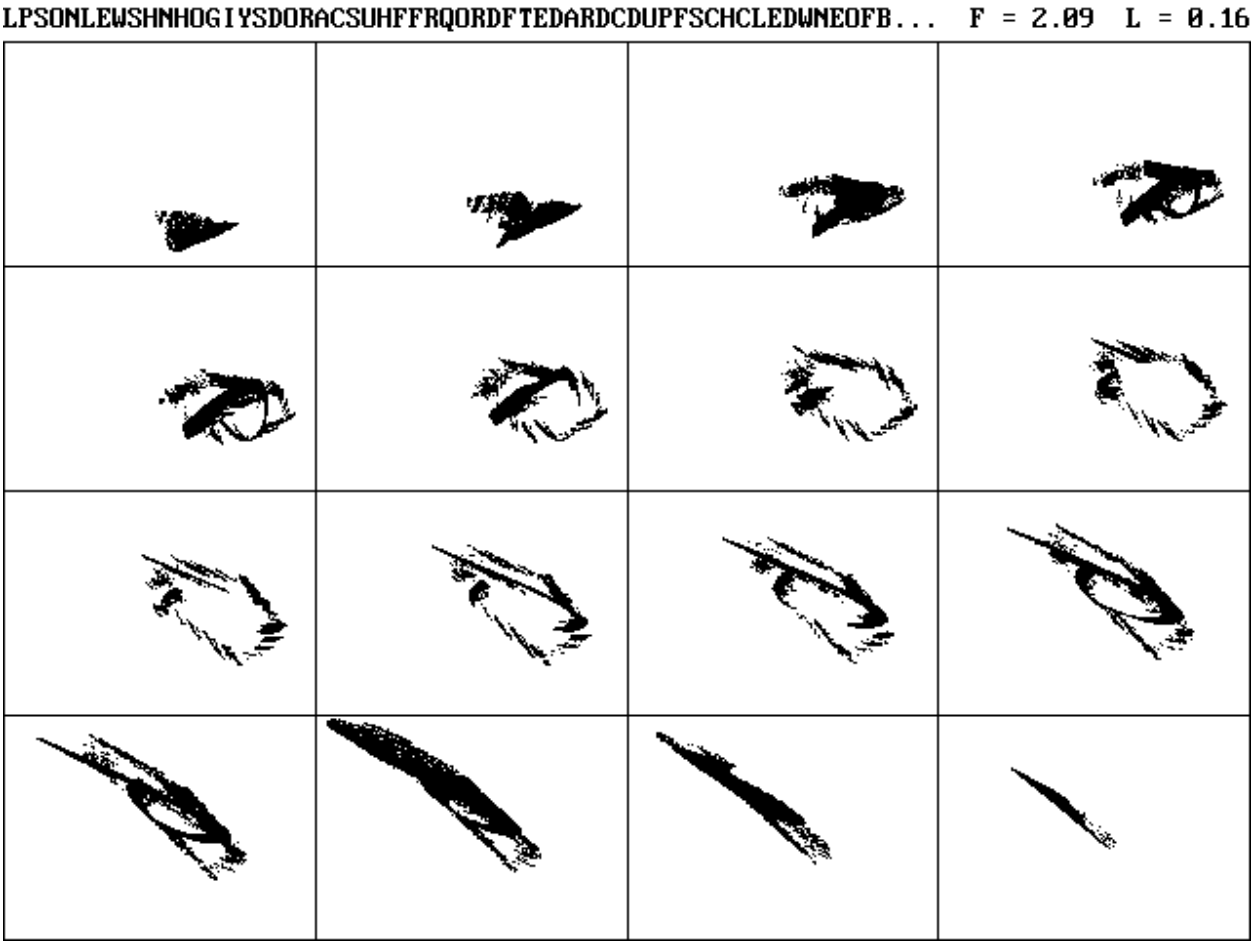


Figure 4-83. Slices of a three-dimensional quintic map



In this chapter we have described a number of techniques whereby three-dimensional information can be exhibited on a computer screen or printed page. However, none of these displays is truly three-dimensional. You have seen the term "3-D" used loosely in advertisements for computer graphics, often meaning little more than a perspective drawing or a view from an oblique angle. In a true three-dimensional display, the viewer must be able to see behind an object by moving his or her head from side to side. A holographic display allows this, but most so-called 3-D displays do not. Anaglyphs and stereo pairs are probably better described as *stereoscopic* displays. They merely provide the illusion of 3-D, as do shadows. Techniques using bands, colors, and slices deserve even less to be called 3-D, however useful they are for conveying information about the third dimension. You should be appropriately discerning when confronted with graphics claimed to be 3-D.

Chapter 5

The Fourth Dimension

Although we normally think of space as three-dimensional, mathematics is not so constrained. Strange attractors can be embedded in space of four and even higher dimensions. Their calculation is a straightforward extension of what we have done before. The challenge is to find ways to visualize such high-dimensional objects. This chapter exploits a number of appropriate visualization techniques after a digression to explain why dimensions higher than three are useful for describing the world in which we live.

5.1 Hyperspace

Ordinary space is three-dimensional. The position of any point relative to an arbitrary origin can be characterized by a set of three numbers—the distance forward or back, right or left, up or down. An object, such as a solid ball, in this space may itself be three-dimensional, or perhaps, like an eggshell of negligible thickness, it may be two-dimensional. You can also imagine an infinitely fine thread, which is one-dimensional, or the period at the end of this sentence, which is essentially zero-dimensional. Although we can easily visualize objects with dimensions less than or equal to three, it is hard to envision objects of higher dimension.

Before discussing the fourth dimension, it is useful to clarify and refine some familiar terms. Perhaps the best example of a one-dimensional object is a straight line. The line may stretch to infinity in both directions, or it may have ends. A line remains one-dimensional even if it bends, in which case we call it a curve.

When we say that a curve is one-dimensional, we are referring to its *topological dimension*. By contrast the *Euclidean dimension* is the dimension of the space in which the curve is embedded. If the line is straight, both dimensions are one, but if it curves, the Euclidean dimension must be higher than the topological dimension in order for it to fit into the space. Both dimensions are integers. One definition of a fractal is an object whose Hausdorff-Besicovitch (fractal) dimension exceeds its topological dimension. For example a coastline on a flat map has a topological dimension of one, a Euclidean dimension of two, and a fractal dimension between one and two. It is an infinitely long line. On a globe, its Euclidean dimension would be three.

A special and important example of a curve is a *circle*—a curve of finite length but without ends, every segment of which lies at a constant distance from a point at the center. Every circle lies in a *plane*, which is a flat, two-dimensional entity. Like a line, the plane may stretch to infinity in all directions, or it may have *edges*. If a plane has an edge, we call it a *disk*. Note the distinction between a circle, which is a one-dimensional object that does not include its interior, and a circular disk, which is a two-dimensional object that includes the interior.

Just as not all lines are straight, not all two-dimensional objects are flat. A sheet of paper of negligible thickness remains two-dimensional if it is curled or even crumpled up, in which case it is no longer a plane but is still a *surface*. A curved surface has a Euclidean dimension of at least three. A surface can be finite but without edges. An example is a *sphere*, every segment of which is at a constant distance from its center.

Note that just as a circle doesn't include its interior, neither does a sphere. When we want to refer to the three-dimensional region bounded by a sphere, we call it a *ball*. This terminology is universal among mathematicians, but not among physicists, who sometimes consider the dimension of circles and spheres to be the minimum Euclidean dimension of the space in which they can be embedded (two and three, respectively).

Another example of a finite surface without edges is a *torus*, most familiar as the surface of a doughnut or inner tube. Such curved spaces without edges are useful whenever one of the variables is periodic. Spaces of arbitrary dimensions, whether flat or curved, are called *manifolds*. The branch of mathematics that deals with these shapes is called *topology*.

If we could describe the world purely by specifying the position of objects, three dimensions would suffice. However, if you consider the motion of a baseball, you are interested not only in where it is, but in how fast it is moving and in what direction. Six numbers are needed to specify both its position and its velocity. This six-dimensional space is called *phase space*. Furthermore, if the ball is spinning, six more dimensions are needed, one to specify the angle and another to specify the angular velocity about each of three perpendicular axes through the ball.

If you have *two* spinning balls that move independently, you need a phase space with *twice* as many (24) dimensions, and so forth. Contemplate the phase-space dimension required to specify the motion of more than 10^{25} molecules in every cubic meter of air! Sometimes physicists even find it useful to perform calculations in an infinite-dimensional space, called *Hilbert space*.

You might also be interested in other properties of the balls, such as their temperature, color, or radius. Thus the state of the balls as time advances can be described by a curve, or *trajectory*, in some high-dimensional space called *state space*, in which the various perpendicular directions correspond to the quantities that describe the balls. The trajectory is a curve connecting temporally successive points in state space.

You have probably heard of *time* referred to as the *fourth dimension* and associate the idea with the theory of relativity. Long before Einstein, it was obvious that to specify an *event*, as opposed to a location, it is necessary to specify not only where the event occurred (*X*, *Y*, and *Z*) but also when (*T*). We can consider events to be points in this four-dimensional space.

Note that the spatial coordinates of a point are not unique. An object four feet in front of one observer might be six feet to the right of a second and two feet above a third. The values of *X*, *Y*, and *Z* of the position depend on where the coordinate system is located and how it is oriented. However, we would expect the various observers to agree on the separation between any two locations. Similarly we expect all observers to agree on the time interval between two events.

The special theory of relativity asserts that observers usually do not agree on either the separation or the time interval between two events. Events that are simultaneous for one observer will not be simultaneous for a second moving relative to the first. Similarly, two successive events at the same position as seen by one observer will be seen at different positions by the other.

You have probably heard that, according to the special theory of relativity, moving clocks run slow and moving meter sticks are shortened. (It is also true that the effective mass of an object increases when it moves, leading to the famous $E = mc^2$, but that's another story.) These discrepancies remain even after the observers correct for their motion and for the time required for the information about the events to reach them traveling at the speed of light. It is important to understand that these facts have nothing to do with the properties of clocks and meter sticks and that they are not illusions; they are properties of space and time, neither of which possess the absolute qualities we normally ascribe to them.

What is remarkable is that all observers agree on the separation between the events in four-dimensional *space-time*. This separation is called the *proper length*, and it is calculated from the Pythagorean theorem by taking the square root of the sum of the squares of the four components after converting the time interval (ΔT) to a distance by multiplying it by the speed of light (c). The only subtlety is that the square of the time enters as a *negative* quantity:

$$\text{Proper length} = [\Delta X^2 + \Delta Y^2 + \Delta Z^2 - c^2 \Delta T^2]^{1/2} \qquad \text{(Equation 5A)}$$

Because of the minus sign in Equation 5A, time is considered to be an *imaginary* dimension; an imaginary number is one whose square is negative. Note, however, that the word "imaginary" does not mean it is any less real than the other dimensions, only that its square combines with the others through subtraction rather than addition. If you are unfamiliar with imaginary numbers, don't be put off by the name. They aren't really imaginary; they are just the other part of certain quantities that require a pair of numbers rather than a single number to specify them.

The minus sign also means that proper length, unlike ordinary length, may be imaginary. If the proper length is imaginary, we say the events are separated in a *timelike*, as opposed to a *spacelike*, manner. Timelike events can be causally related (one event can influence the other), but spacelike events cannot, because information about one would have to travel faster than the speed of light to reach the other, which is impossible. Events separated in a timelike manner are more conveniently characterized by a *proper time*:

$$\text{Proper time} = [\Delta T^2 - \Delta X^2/c^2 - \Delta Y^2/c^2 - \Delta Z^2/c^2]^{1/2} \qquad \text{(Equation 5B)}$$

In this case, time is real, but space is imaginary. Proper length is the length of an object as measured by an observer moving with the same velocity as the object, and proper time is the time measured by a clock moving with the same velocity as the observer.

Quantities such as proper length and proper time on which all observers agree, independent of their motion, are called *invariants*. The speed of light itself is an invariant. There are many others, and they all involve four components that combine by the Pythagorean theorem.

Thus the theory of relativity ties space and time together in a very fundamental way. One person's space is another person's time. Since space and time can be traded back and forth, there is no reason to call time the fourth dimension any more than we call width the second dimension. It is better just to say that space-time is

four-dimensional, with each dimension on an equal footing. The apparent asymmetry between space and time comes from the large value of c (3×10^8 meters per second, or about a billion miles per hour) and the fact that time moves in only one direction (past to future). It is also important to understand that, although special relativity is called a "theory," it has been extensively verified to high accuracy by many experiments, most of which involve particle accelerators.

The foregoing discussion explains why it might be useful to consider four-dimensional space and four-dimensional objects, but it is probably fruitless to waste too much time trying to visualize them. However, we can describe them mathematically as extensions of familiar objects in lower dimensions.

For example, a *hypercube* is the four-dimensional extension of the three-dimensional cube and the two-dimensional square. It has 16 corners, 32 edges, 24 faces, and contains 8 cubes. Its *hypervolume* is the fourth power of the length of each edge, just as the volume of a cube is the cube of the length of an edge and the area of a square is the square of the length of an edge.

A *hypersphere* consists of all points at a given distance from its center in four-dimensional space. Its *hypersurface* is three-dimensional and consists of an infinite family of spheres, just as the surface of an ordinary sphere is two-dimensional and consists of an infinite family of circles. We have reason to believe that our Universe might be a hypersurface of a very large hypersphere, in which case we could see ourselves if we peered far enough into space, except for the fact that we are also looking backward to a time before Earth existed. We would also need an incredibly powerful telescope to see Earth in this way. Thus our perception that space is three-dimensional could be analogous to the ancient view that Earth was flat, a consequence of experience limited to a small portion of its surface.

5.2 Projections

The previous section was intended to motivate your consideration of strange attractors embedded in four-dimensional space, but most of the discussion is not essential to what follows. We will now describe the computer program necessary to produce attractors in four dimensions and then develop methods to visualize them.

The mathematical generalization from three to four dimensions is straightforward. Whereas before we had three variables—*X*, *Y*, and *Z*—we now have a fourth. Having used up the three letters at the end of the alphabet, we must back up and use *W* for the fourth dimension, but remember that all the dimensions are on an

equal footing. We use the first letters M, N, O, and P to code 4-D attractors of second through fifth orders, respectively. The number of coefficients for these cases is 60, 140, 280, and 504, respectively. The number of coefficients for order *O* is *(O + 1)(O + 2)(O + 3)(O + 4) / 6*. The number of four-dimensional fifth-order codes is 25^{504}, a number too large to compare to anything meaningful; it might as well be infinite.

The program modifications required to add a fourth dimension are shown in **PROG18**.

PROG18. Changes required in PROG17 to add a fourth dimension

```
1000 REM FOUR-D MAP SEARCH

1020 DIM XS(499), YS(499), ZS(499), WS(499), A(504), V(99), XY(4), XN(4), COLR%(15)

1070 D% = 4                    'Dimension of system

1120 TRD% = 0                  'Display third dimension as projection



1540 W = .05

1550 XE = X + .000001: YE = Y: ZE = Z: WE = W

1610 WMIN = XMIN: WMAX = XMAX



1720 M% = 1: XY(1) = X: XY(2) = Y: XY(3) = Z: XY(4) = W

2010 M% = M% - 1: XNEW = XN(1): YNEW = XN(2): ZNEW = XN(3): WNEW = XN(4)



2180    IF W < WMIN THEN WMIN = W

2190    IF W > WMAX THEN WMAX = W

2210 XS(P%) = X: YS(P%) = Y: ZS(P%) = Z: WS(P%) = W
```

```
2410 IF ABS(XNEW) + ABS(YNEW) + ABS(ZNEW) + ABS(WNEW) > 1000000! THEN T% = 2

2470 IF ABS(XNEW - X) + ABS(YNEW - Y) + ABS(ZNEW - Z) + ABS(WNEW - W) < .000001
THEN T% = 2

2540 W = WNEW



2910 XSAVE = XNEW: YSAVE = YNEW: ZSAVE = ZNEW: WSAVE = WNEW

2920 X = XE: Y = YE: Z = ZE: W = WE: N = N - 1

2950 DLZ = ZNEW - ZSAVE: DLW = WNEW - WSAVE

2960 DL2 = DLX * DLX + DLY * DLY + DLZ * DLZ + DLW * DLW

3010    ZE = ZSAVE + RS * (ZNEW - ZSAVE): WE = WSAVE + RS * (WNEW - WSAVE)

3020    XNEW = XSAVE: YNEW = YSAVE: ZNEW = ZSAVE: WNEW = WSAVE



3150 IF WMAX - WMIN < .000001 THEN WMIN = WMIN - .0000005: WMAX = WMAX + .0000005



3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 4): T% = 1



3920 IF N = 1000 THEN D2MAX = (XMAX - XMIN) ^ 2 + (YMAX - YMIN) ^ 2 + (ZMAX - ZMIN)
^ 2 + (WMAX - WMIN) ^ 2

3940 DX = XNEW - XS(J%): DY = YNEW - YS(J%): DZ = ZNEW - ZS(J%): DW = WNEW - WS(J%)

3950 D2 = DX * DX + DY * DY + DZ * DZ + DW * DW



4760 IF D% > 2 THEN FOR I% = 3 TO D%: M% = M% / (I% - 1): NEXT I%
```

If you run **PROG18** under certain old versions of BASIC, such as BASICA and GW-BASIC, you are likely to get an error in line 2710 when the program attempts to construct a code for the fourth-order and fifth-order maps as a result of the string-length limit of 255 characters. In such a case, you may need to restrict the search

to second and third orders by setting *OMAX%* = 3 in line 1060. Alternatively, it's not difficult to modify the program to store the code in a pair of strings or to replace the string with a one-dimensional array of integers containing the numeric equivalents of each character in the string, perhaps with a terminating zero to signify the end of the string. For example, after dimensioning *CODE%*(504) in line 1020, line 2710 would become

```
2710        CODE%(I%) = 65 + INT(25 * RAN)
```

and line 2740 would become

```
2740    A(I%) = (CODE%(I%) - 77) / 10
```

Also notice that the search for attractors is painfully slow unless you have a very fast computer and a good compiler. Refer back to Table 2-2, which lists some options for increasing the speed. The search can be made faster by limiting it to second order by setting *OMAX%* = 2 in line 1060.

We have another trick we can use to increase dramatically the rate at which four-dimensional strange attractors are found without sacrificing variety. It turns out that most of these attractors have their constant terms near zero. The reason presumably has to do with the fact that the origin ($X = Y = Z = W = 0$) is then a fixed point, and the initial condition is chosen near the origin ($X_0 = Y_0 = Z_0 = W_0 = 0.05$). If the fixed point is unstable, then we have one of the conditions necessary for chaos. It is easy to accomplish this by adding after line 2730 a statement such as

```
2735    IF I% MOD M% / D% = 1 THEN MID$(CODE$, I% + 1, 1) = "M"
```

This increases the rate of finding attractors by about a factor of 50. Many of the attractors illustrated in this chapter were produced in this way. This change also increases the rate for lower-dimensional maps, but by a much smaller factor. This

improvement suggests that there is yet room to optimize the search routine by a more intelligent choice of the values of the other coefficients.

Note that **PROG18** does not attempt to display the fourth dimension but projects it onto the other three, for which all the visualization techniques of the last chapter are available. Don't waste too much time trying to understand what it means to project a four-dimensional object onto a three-dimensional space. It is just a generalization of projecting a three-dimensional object onto a two-dimensional surface. In the program, it simply involves plotting *X*, *Y*, and *Z* and ignoring the variable *W*.

Some examples of four-dimensional attractors projected onto the two-dimensional *XY* plane are shown in Figures 5-1 through 5-20. They don't look particularly different from those obtained by projecting three-dimensional attractors onto the plane or, indeed, by just plotting two-dimensional attractors directly. Note that most of these attractors have fractal dimensions less than or about 2.0, so perhaps it is not too surprising that their projections resemble those produced by equations of lower dimension. It is rare to find attractors with fractal dimensions greater than 3.0 produced by four-dimensional polynomial maps, as will be shown in Section 8.1.

Figure 5-1. Projection of four-dimensional quadratic map

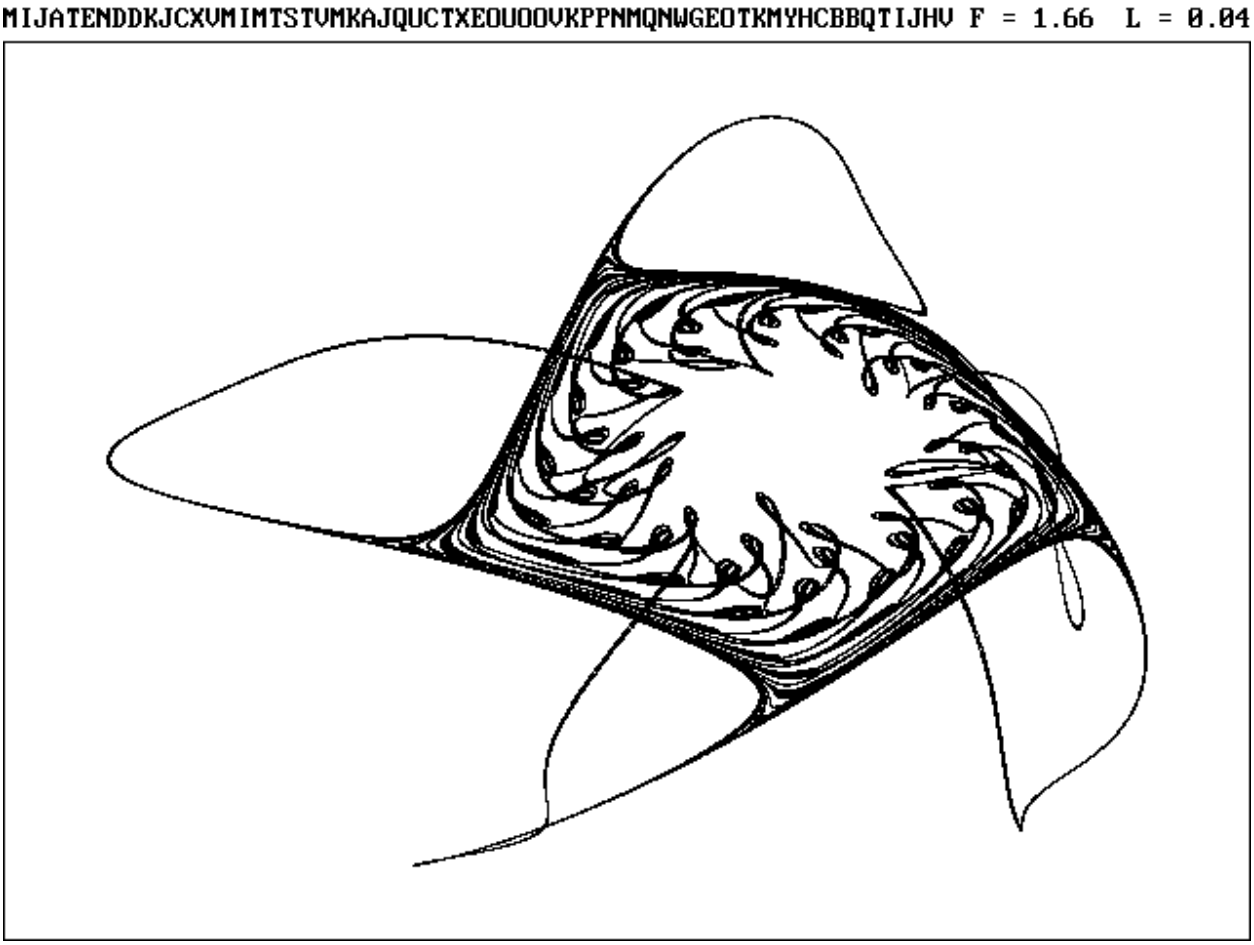


Figure 5-2. Projection of four-dimensional quadratic map

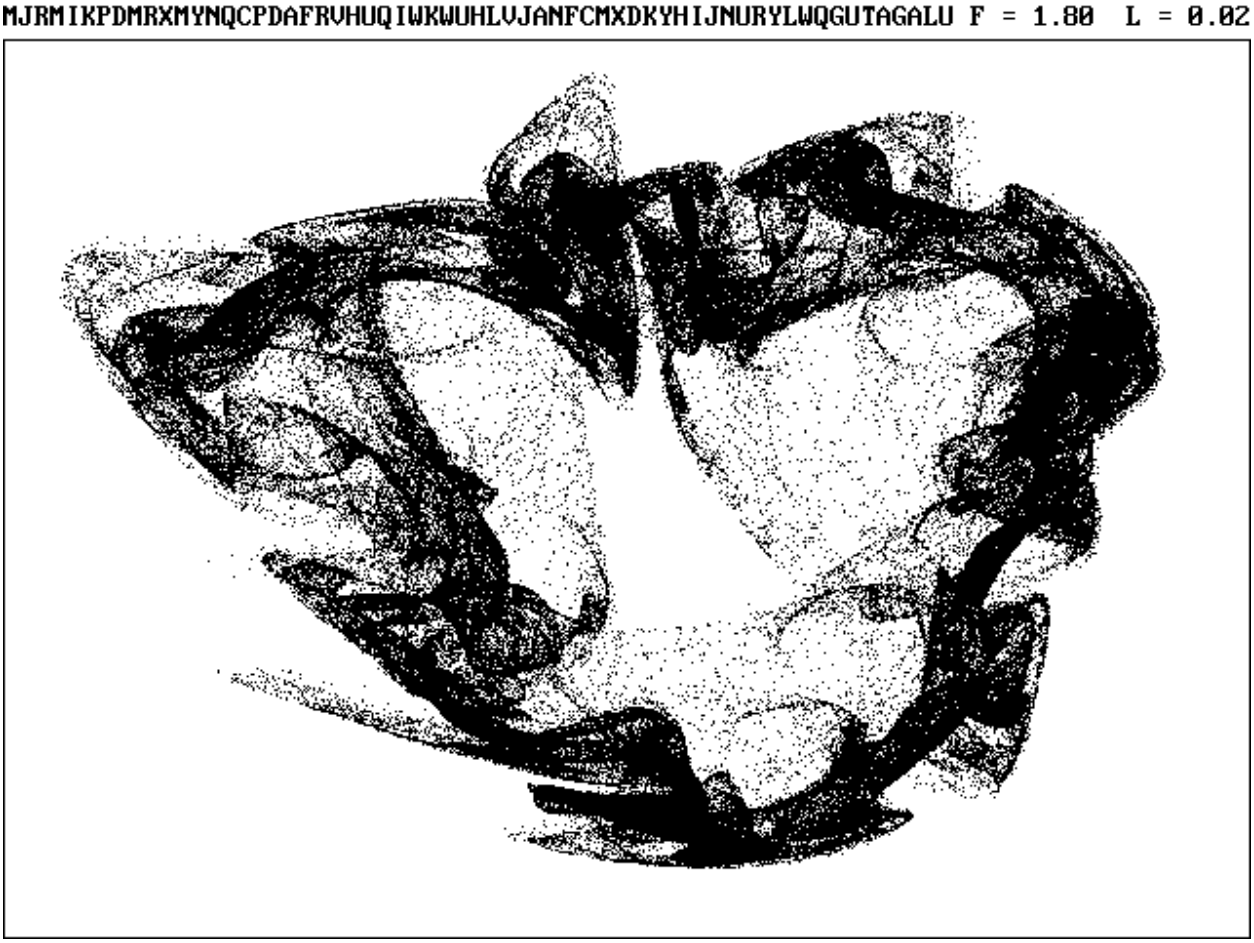


Figure 5-3. Projection of four-dimensional quadratic map

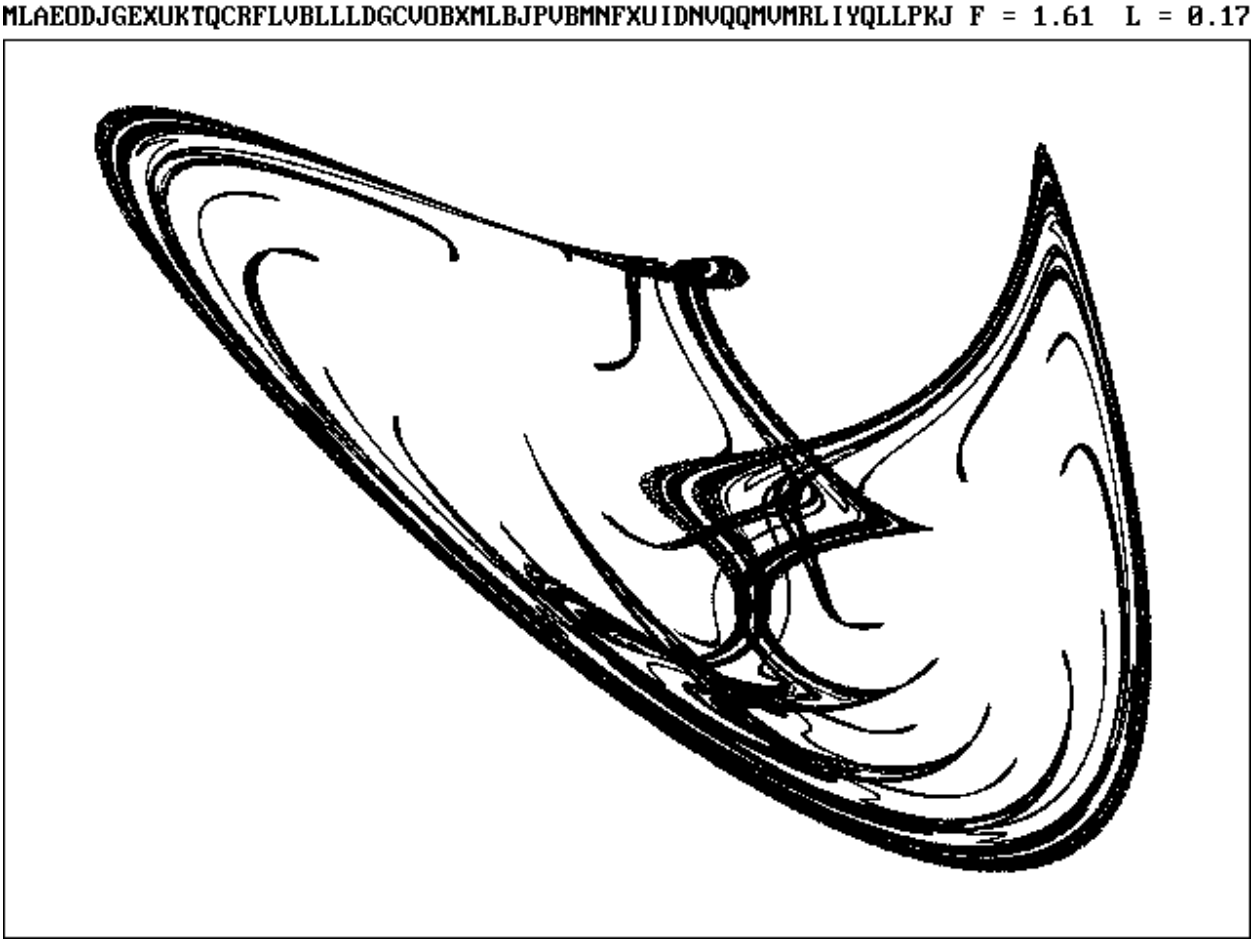


Figure 5-4. Projection of four-dimensional quadratic map

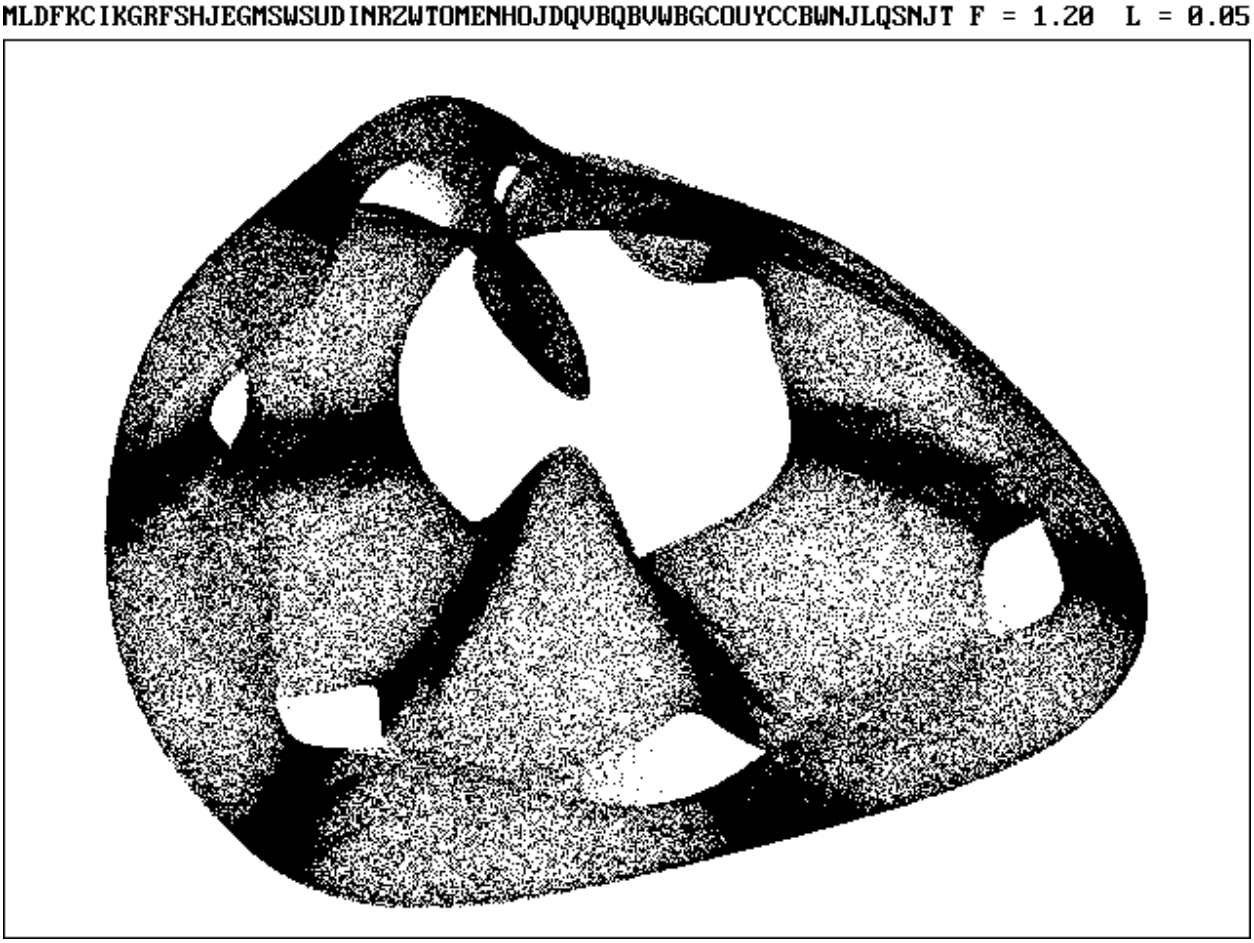


Figure 5-5. Projection of four-dimensional quadratic map

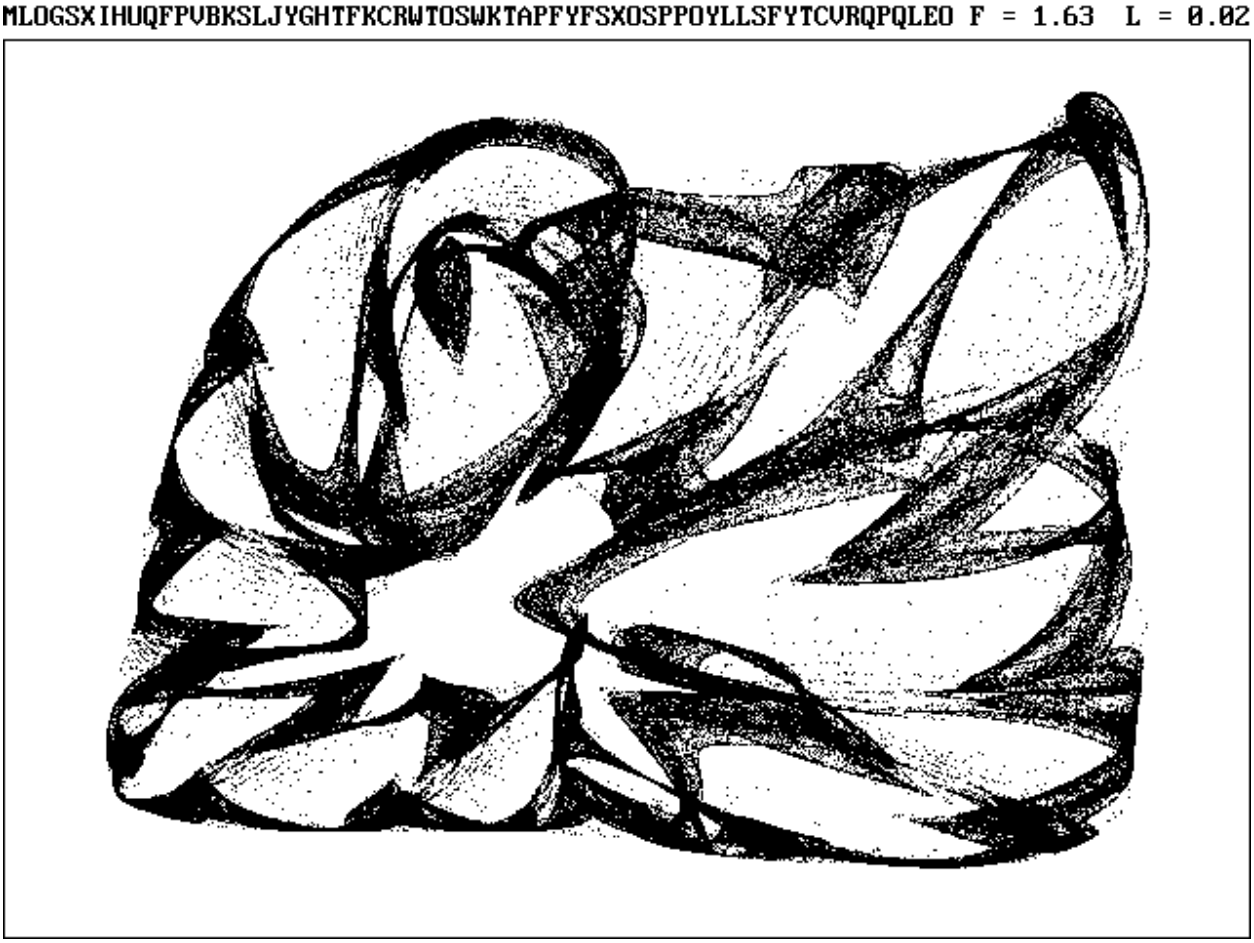


Figure 5-6. Projection of four-dimensional quadratic map

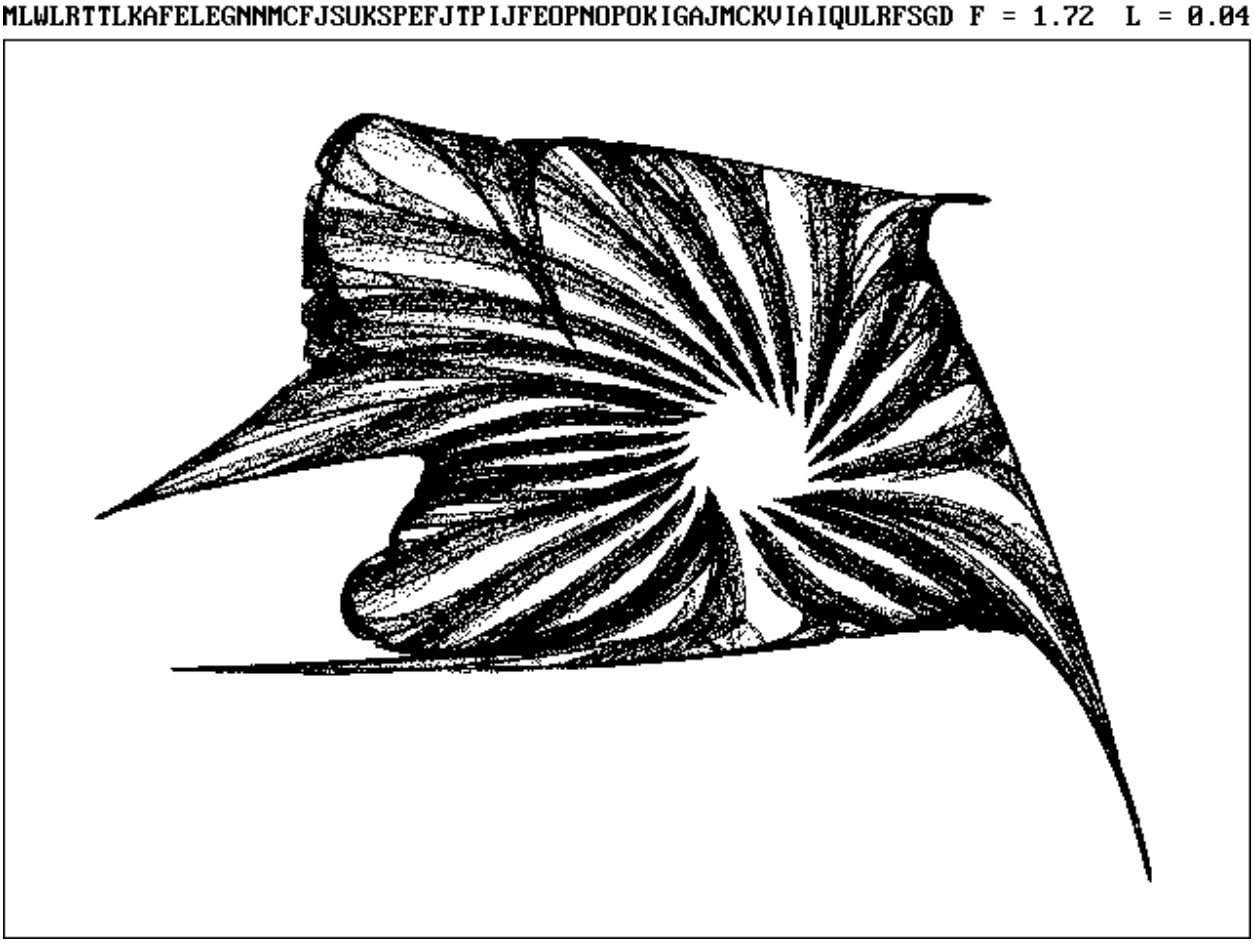


Figure 5-7. Projection of four-dimensional quadratic map

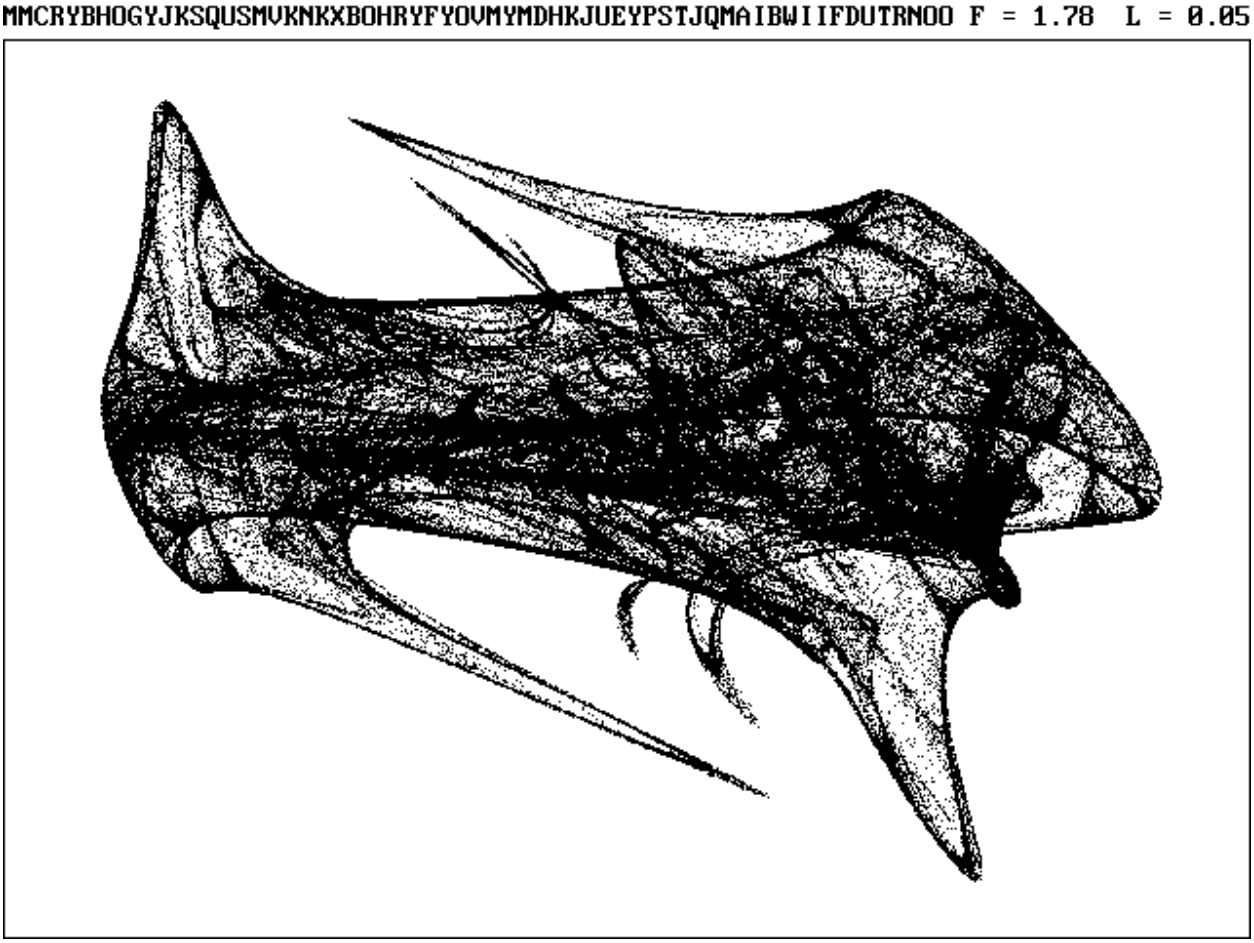


Figure 5-8. Projection of four-dimensional quadratic map

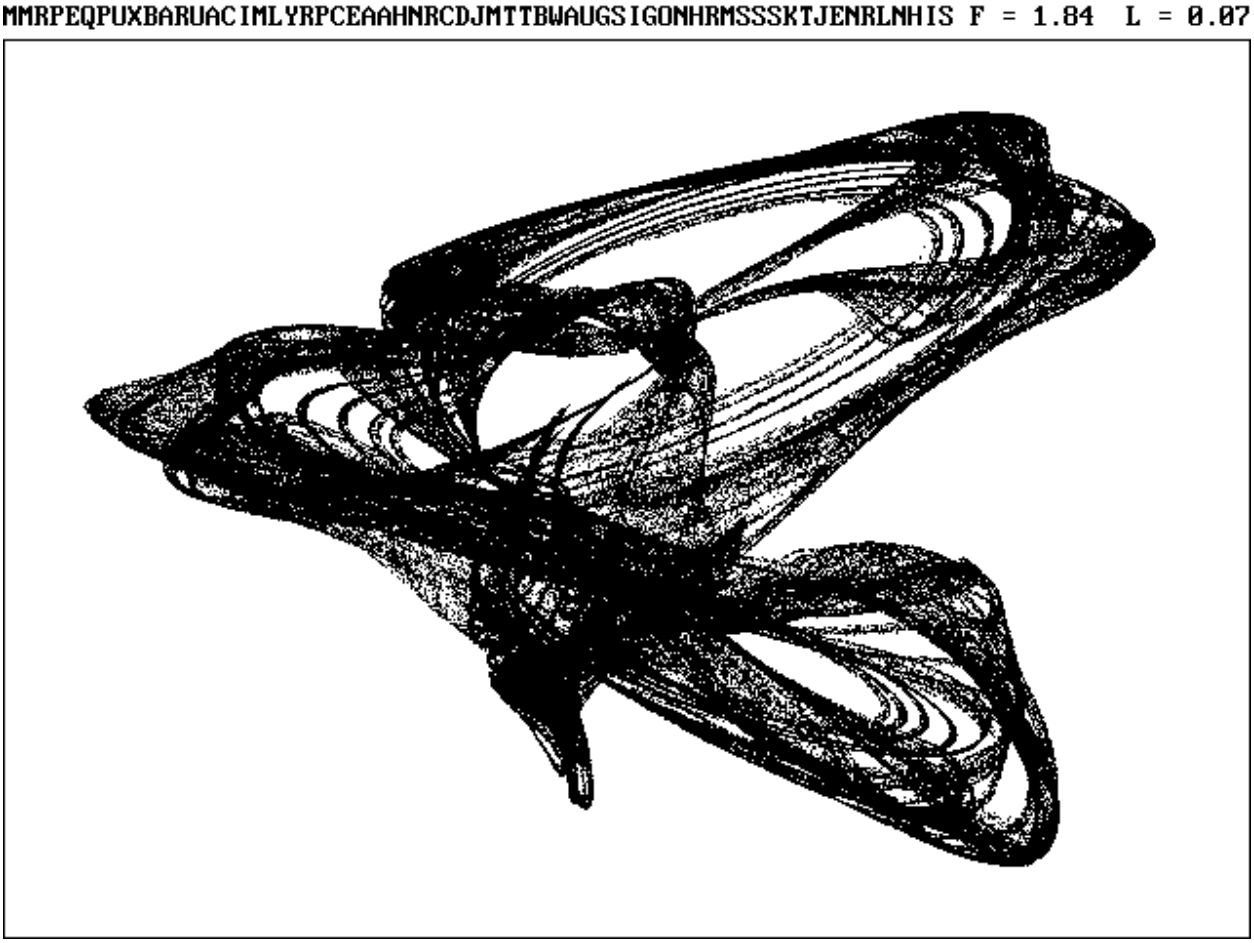


Figure 5-9. Projection of four-dimensional quadratic map

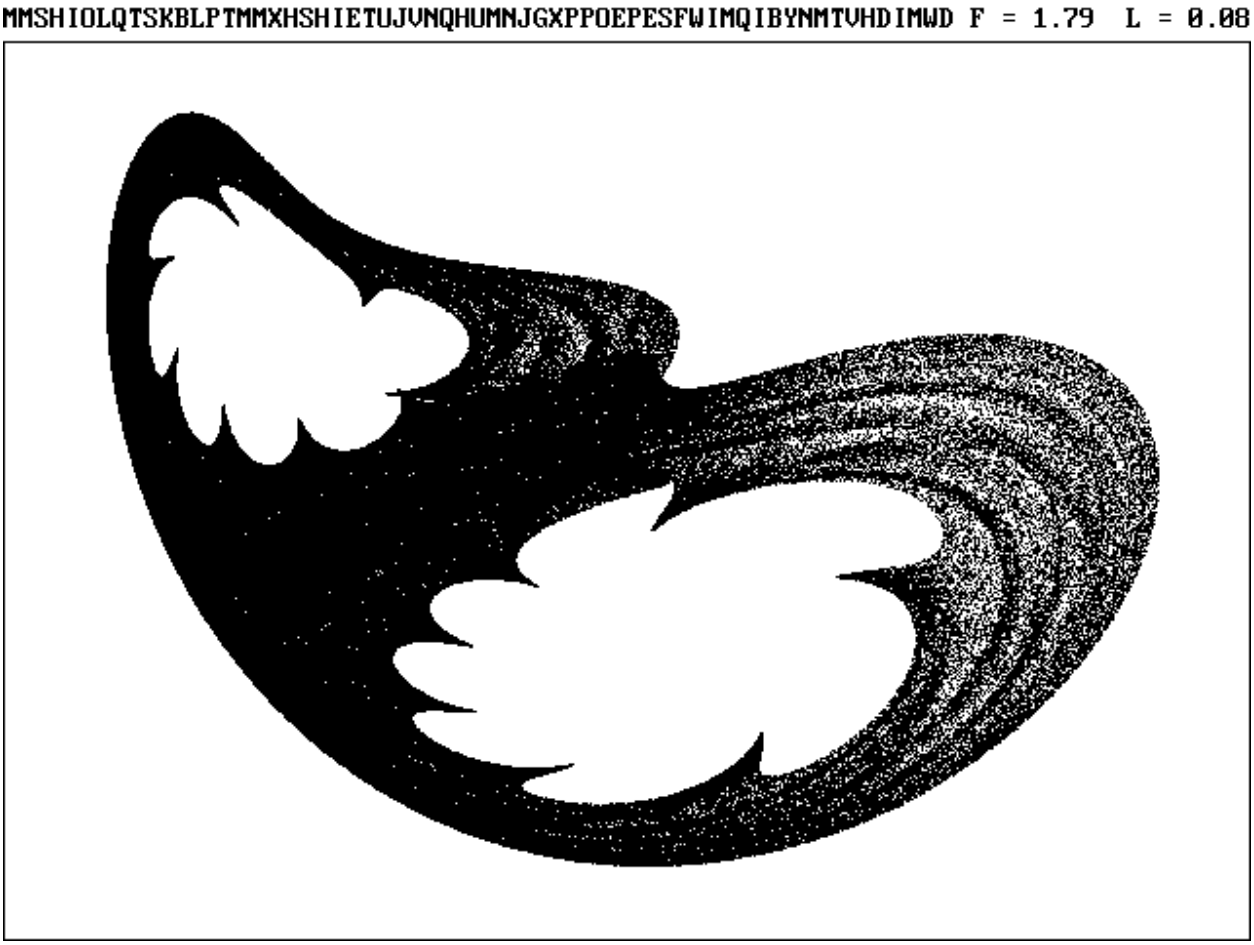


Figure 5-10. Projection of four-dimensional quadratic map

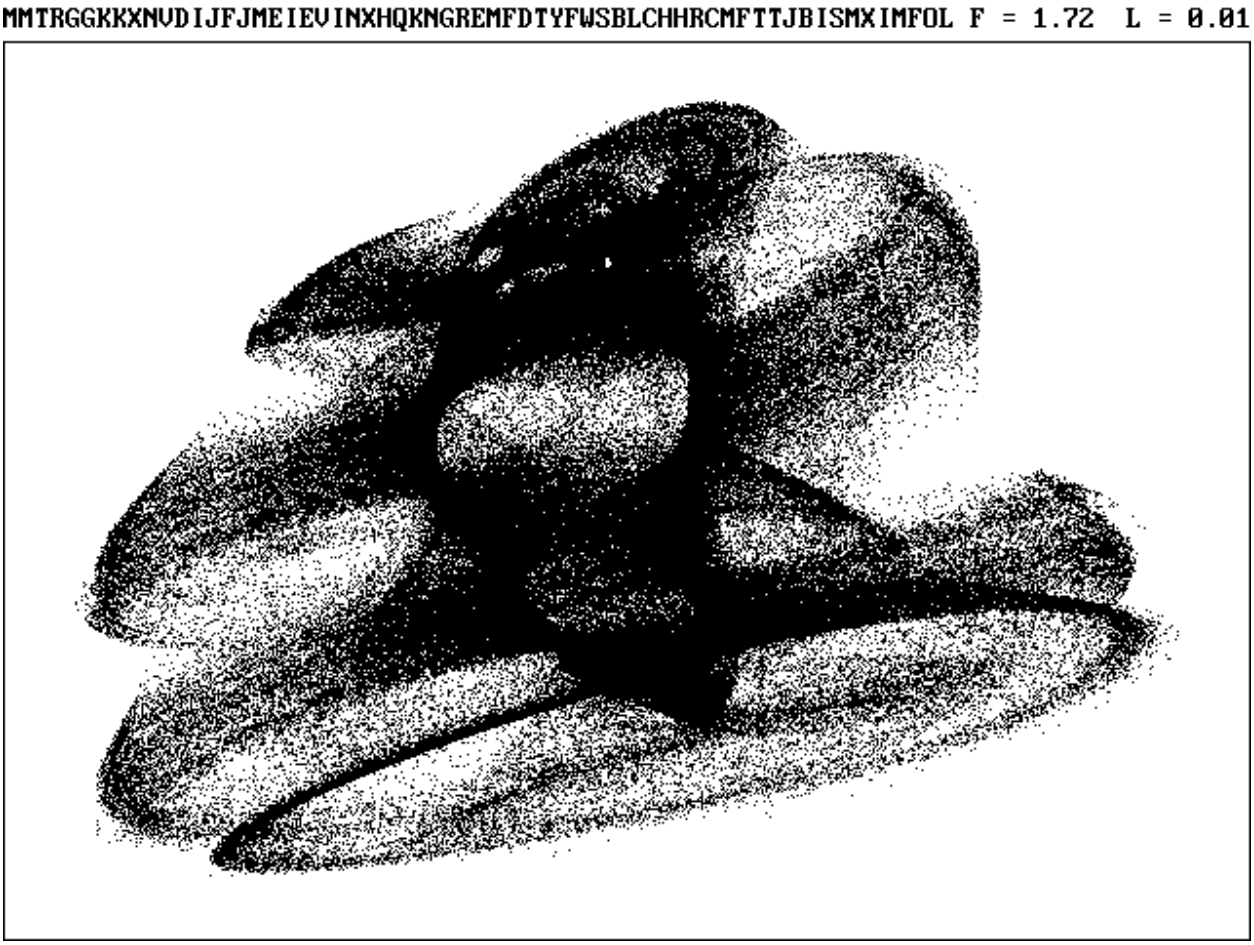


Figure 5-11. Projection of four-dimensional quadratic map

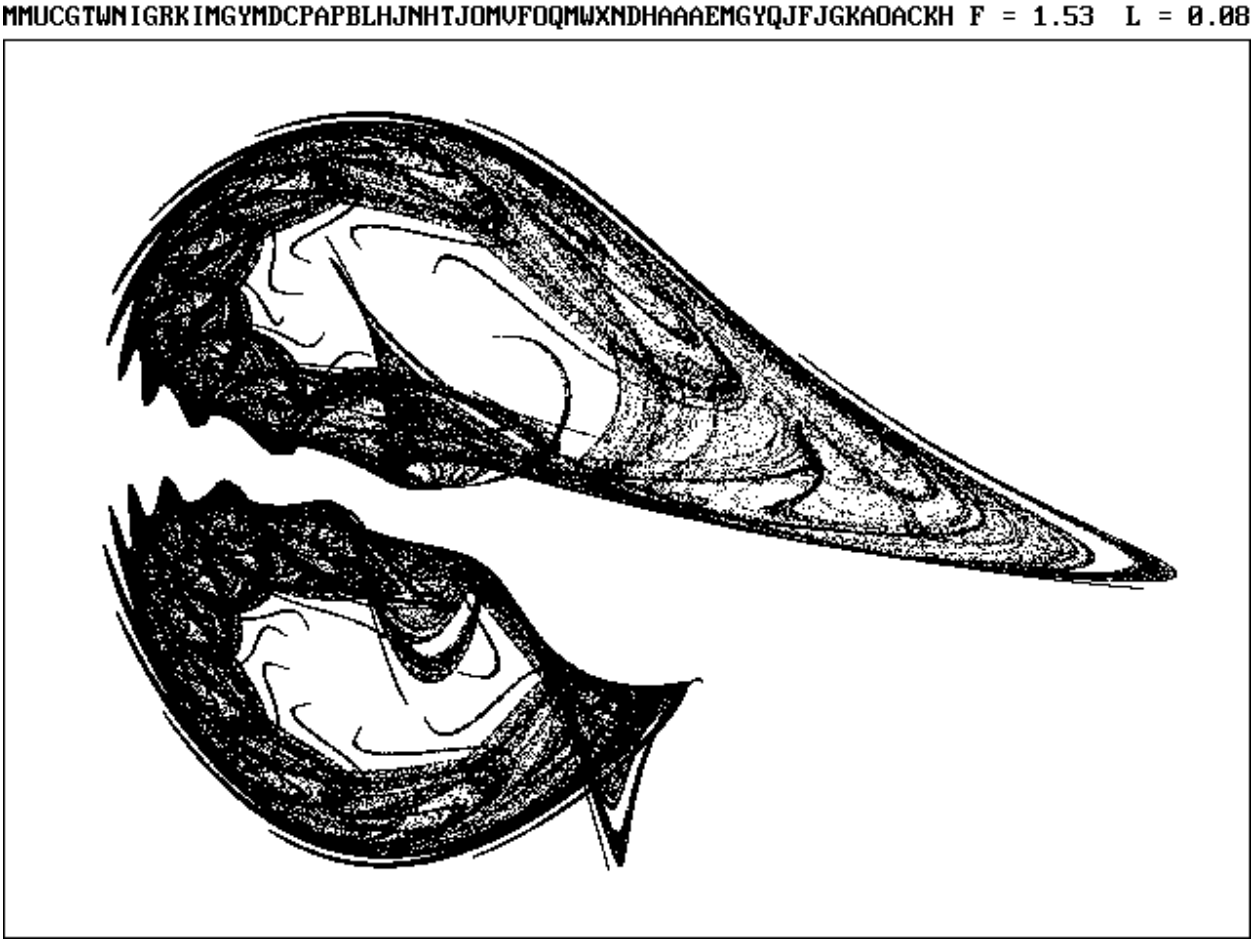


Figure 5-12. Projection of four-dimensional quadratic map

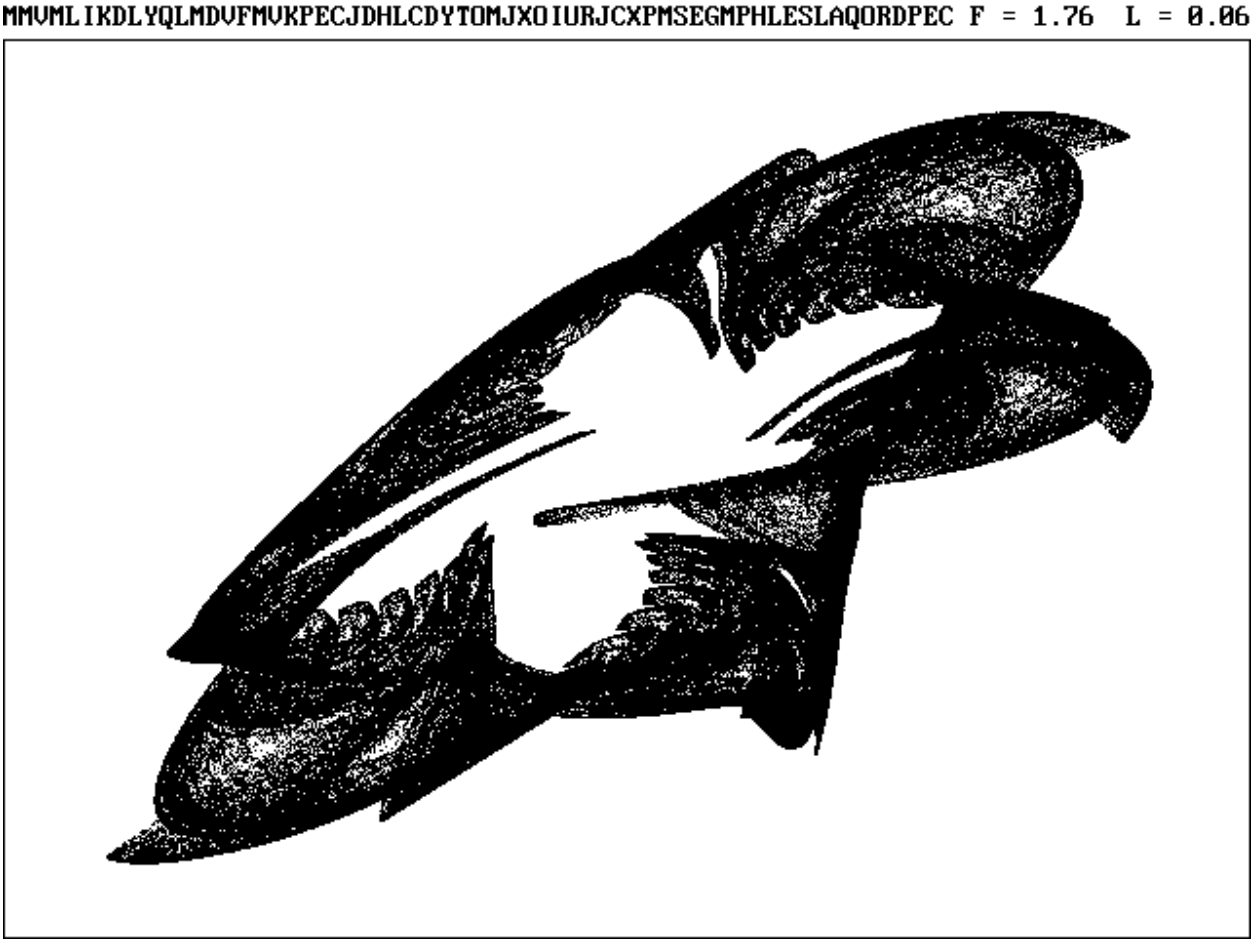


Figure 5-13. Projection of four-dimensional quadratic map

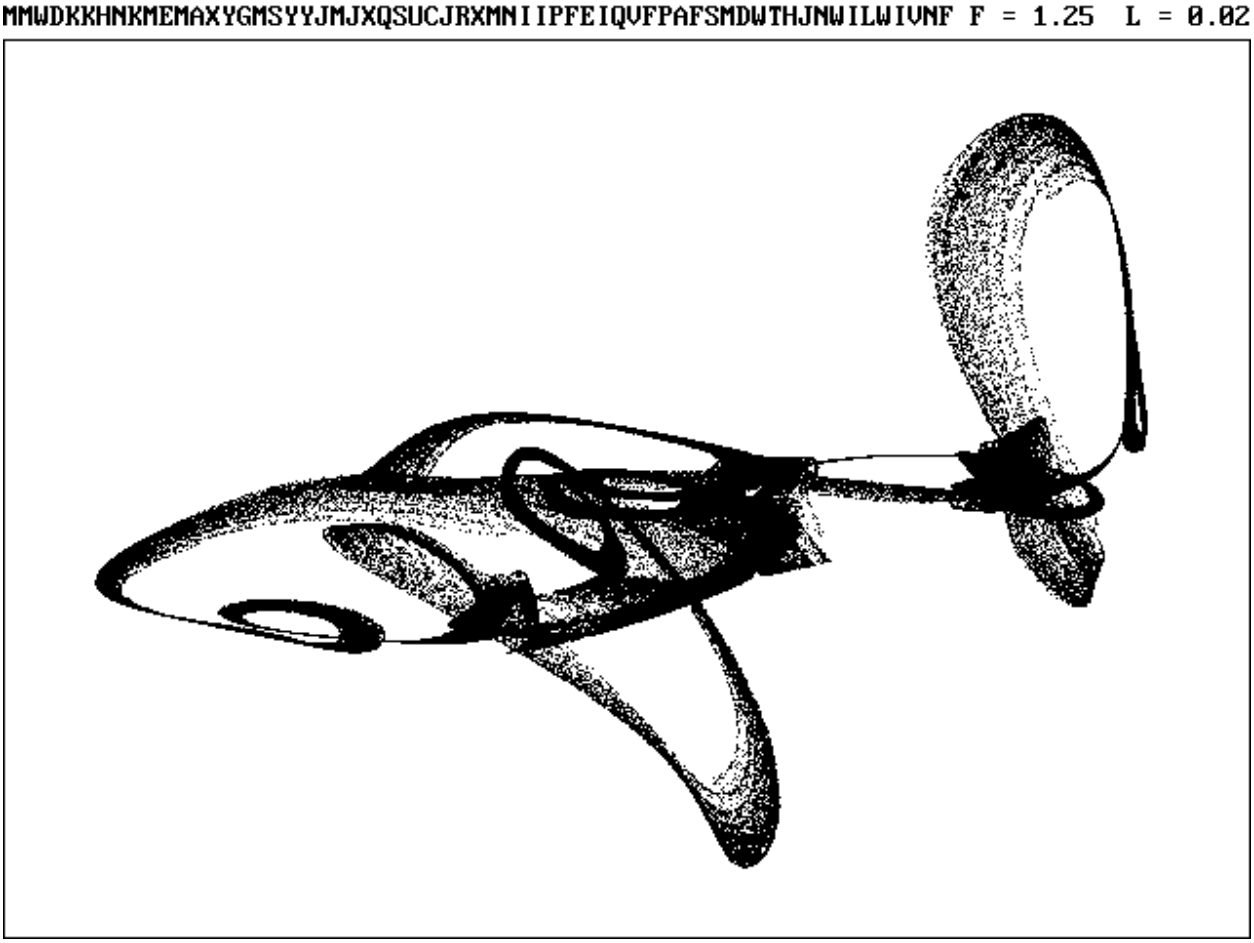


Figure 5-14. Projection of four-dimensional quadratic map

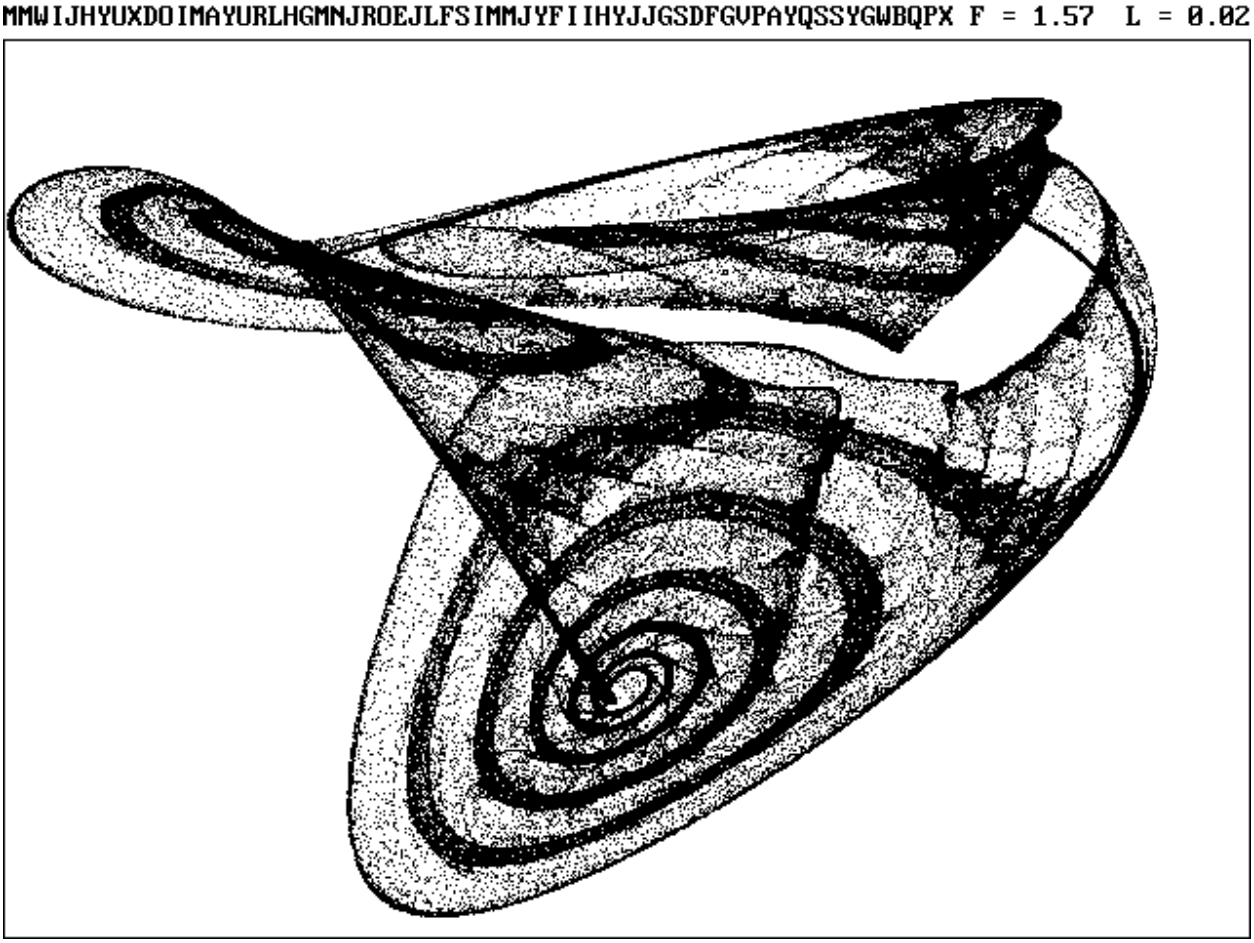


Figure 5-15. Projection of four-dimensional quadratic map

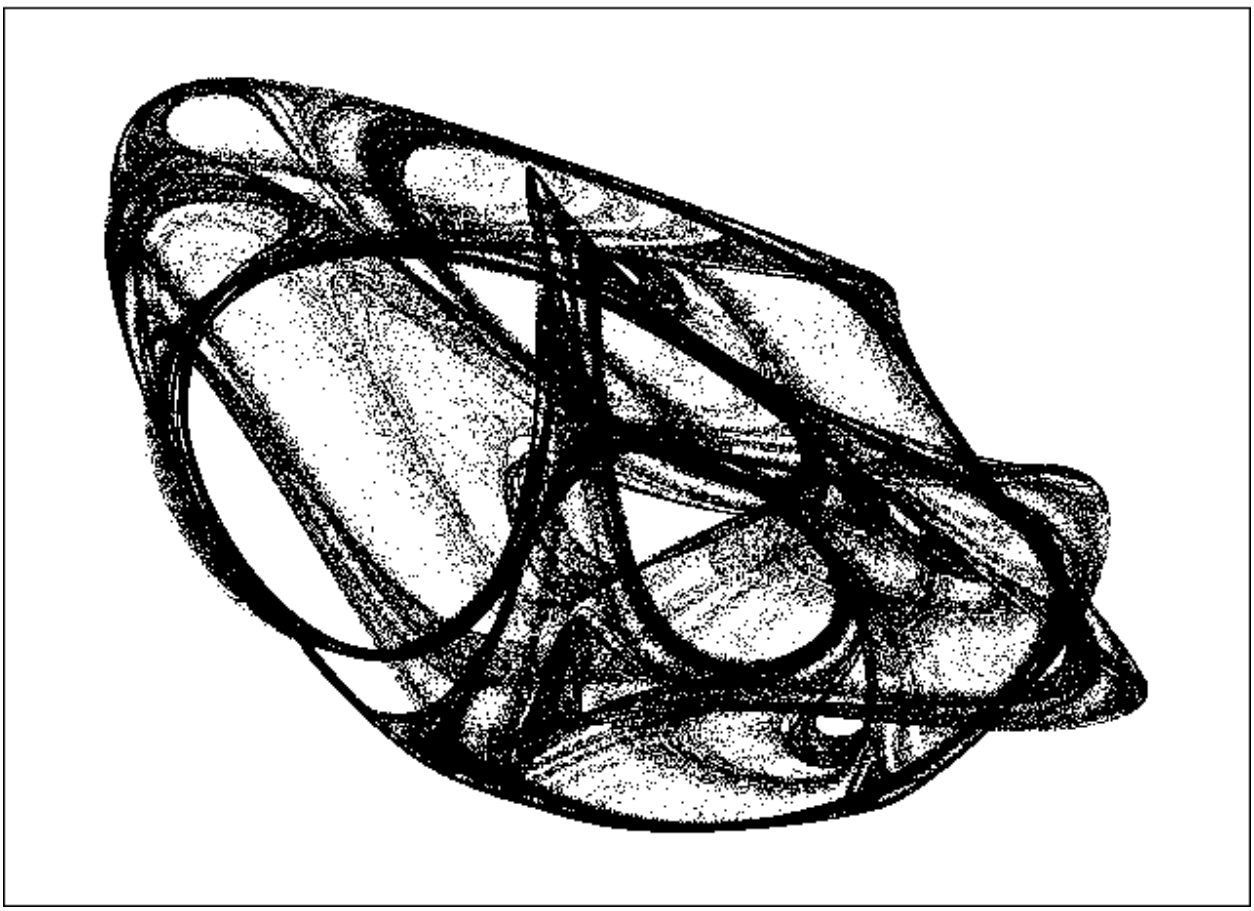


Figure 5-16. Projection of four-dimensional quadratic map

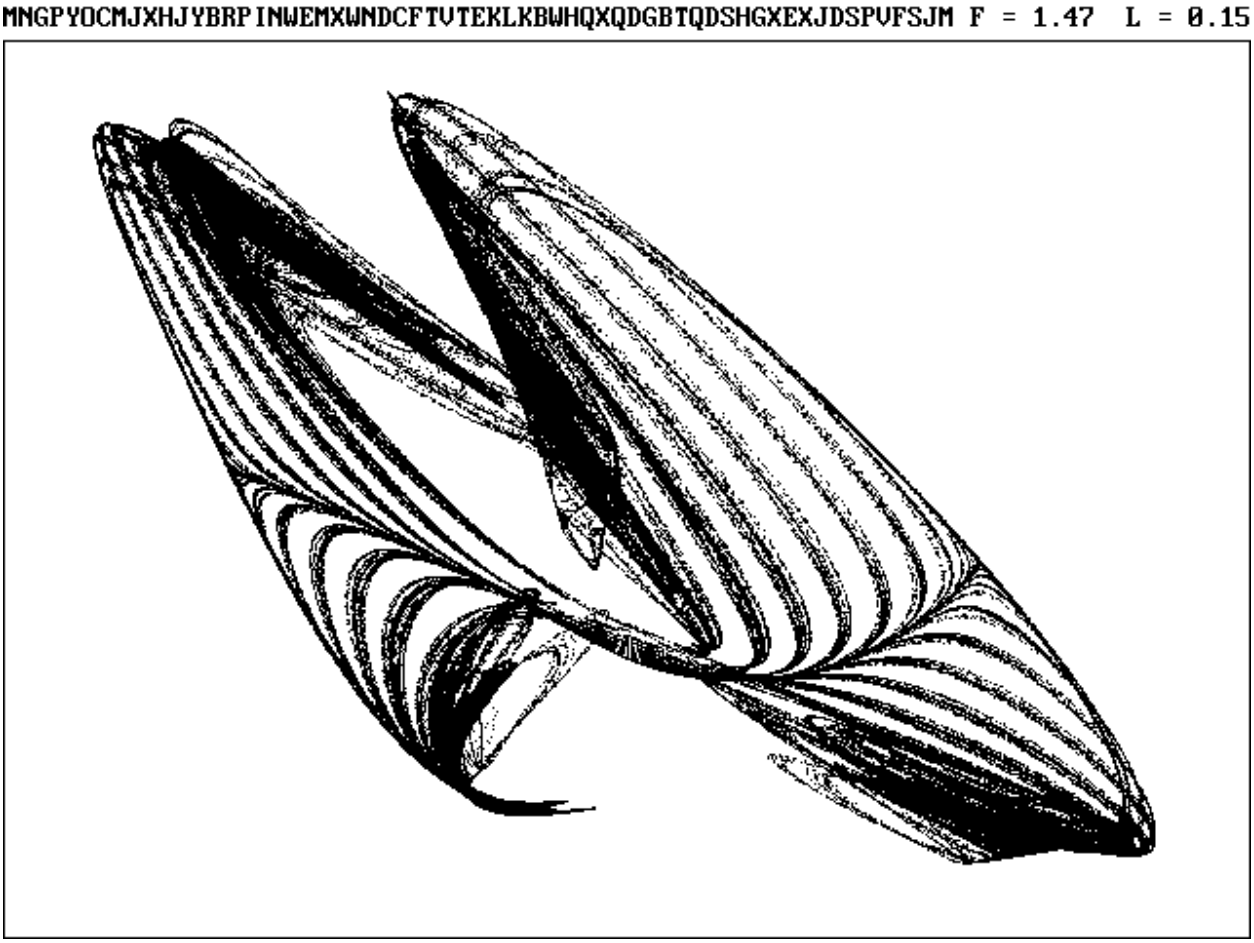


Figure 5-17. Projection of four-dimensional quadratic map

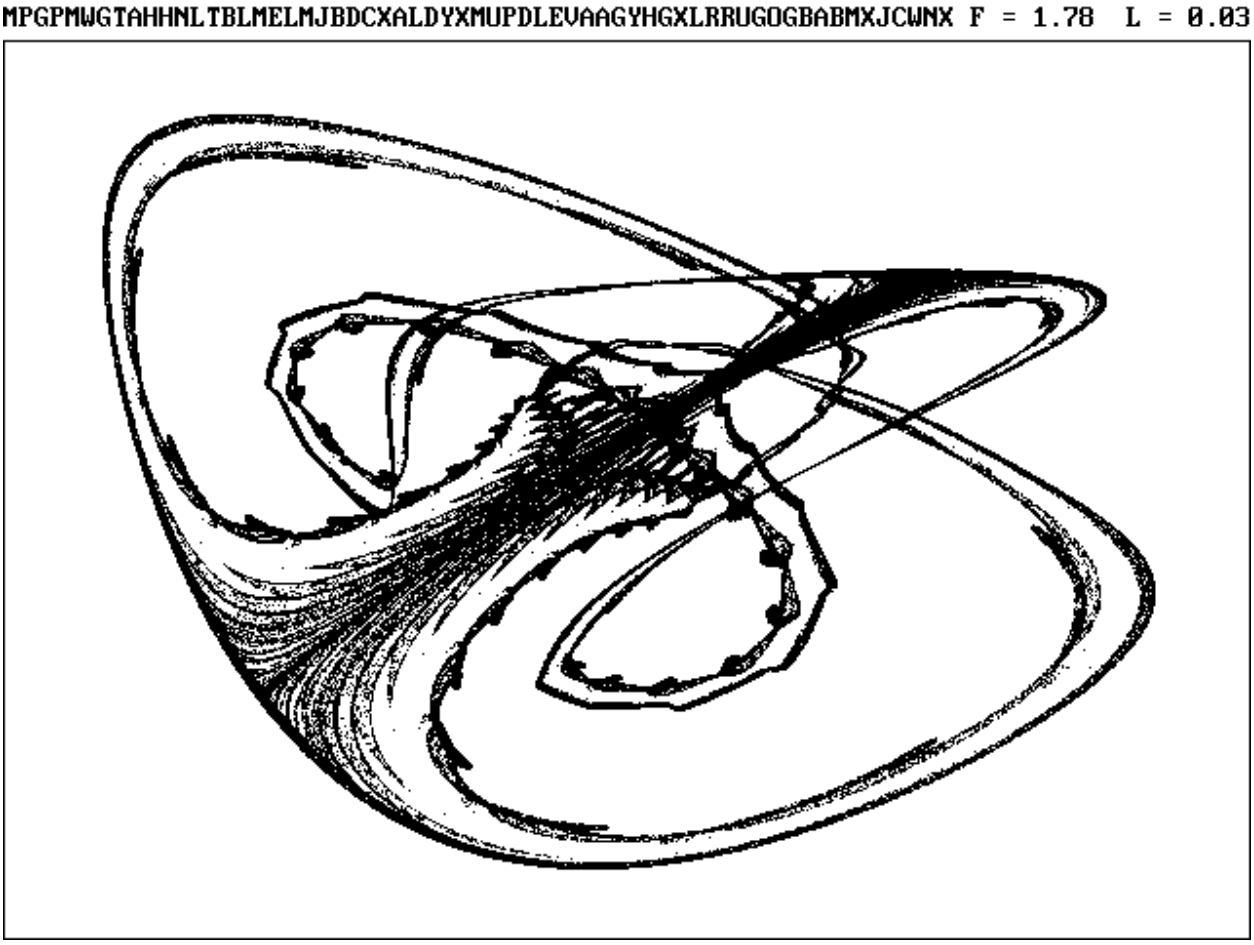


Figure 5-18. Projection of four-dimensional cubic map

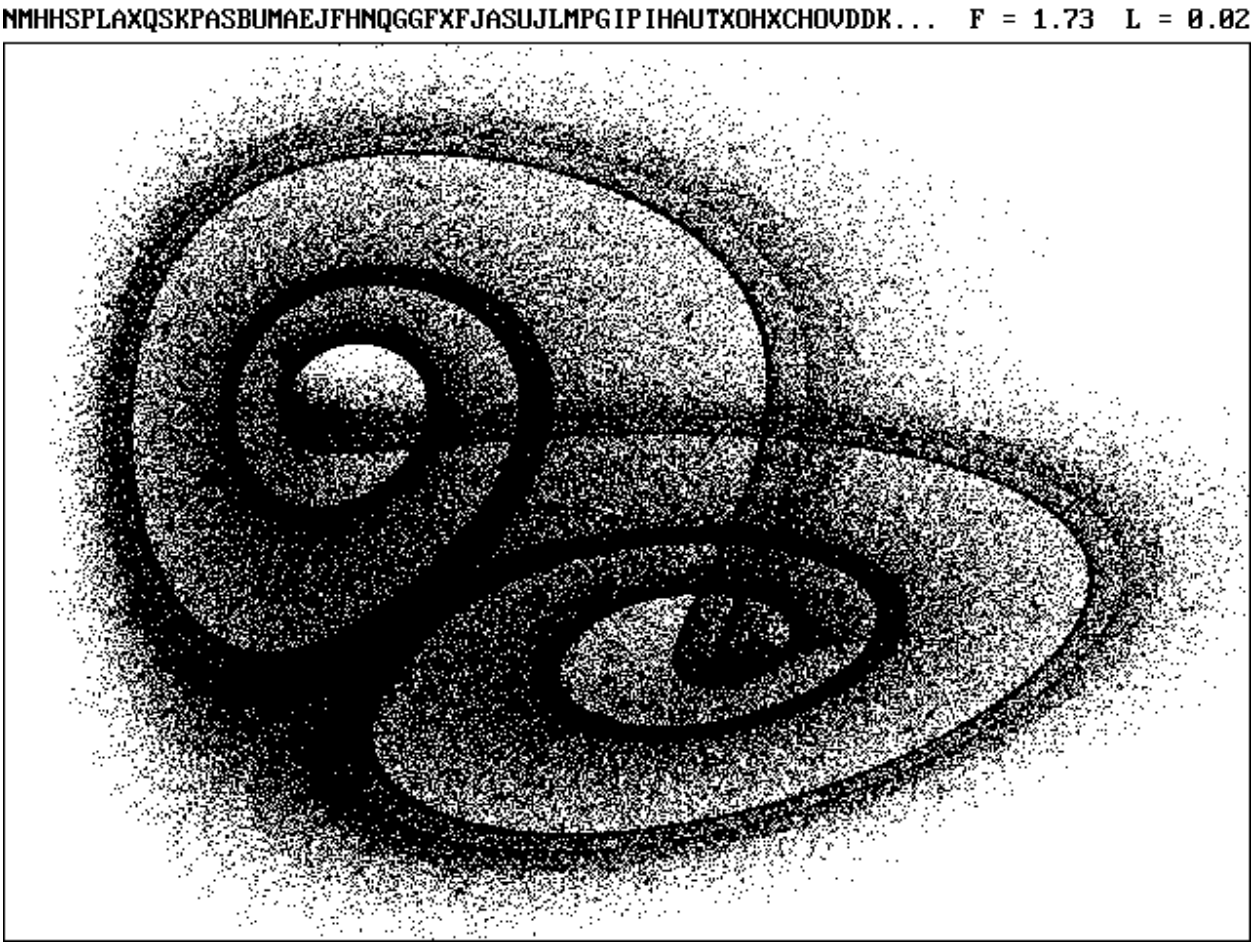


Figure 5-19. Projection of four-dimensional cubic map

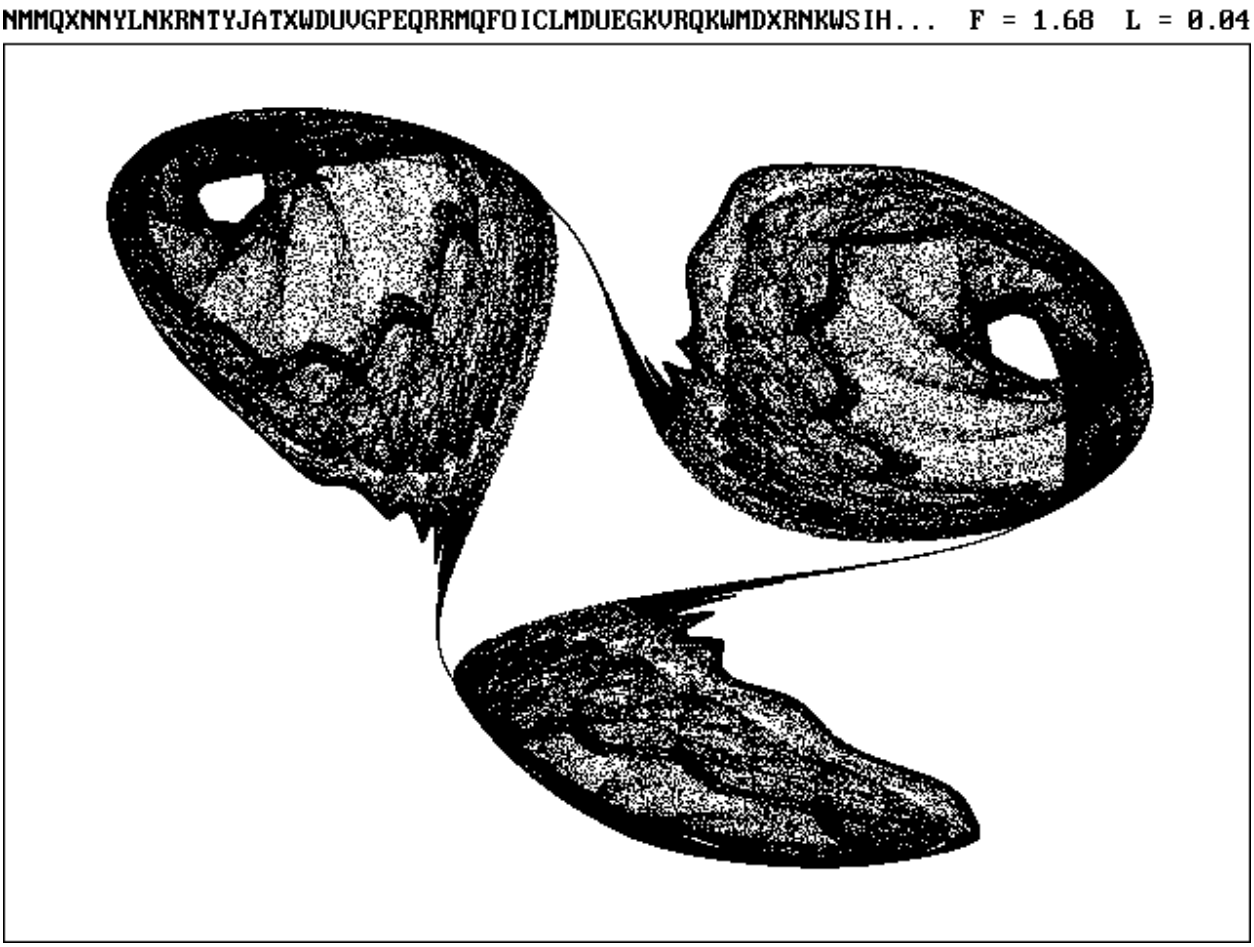
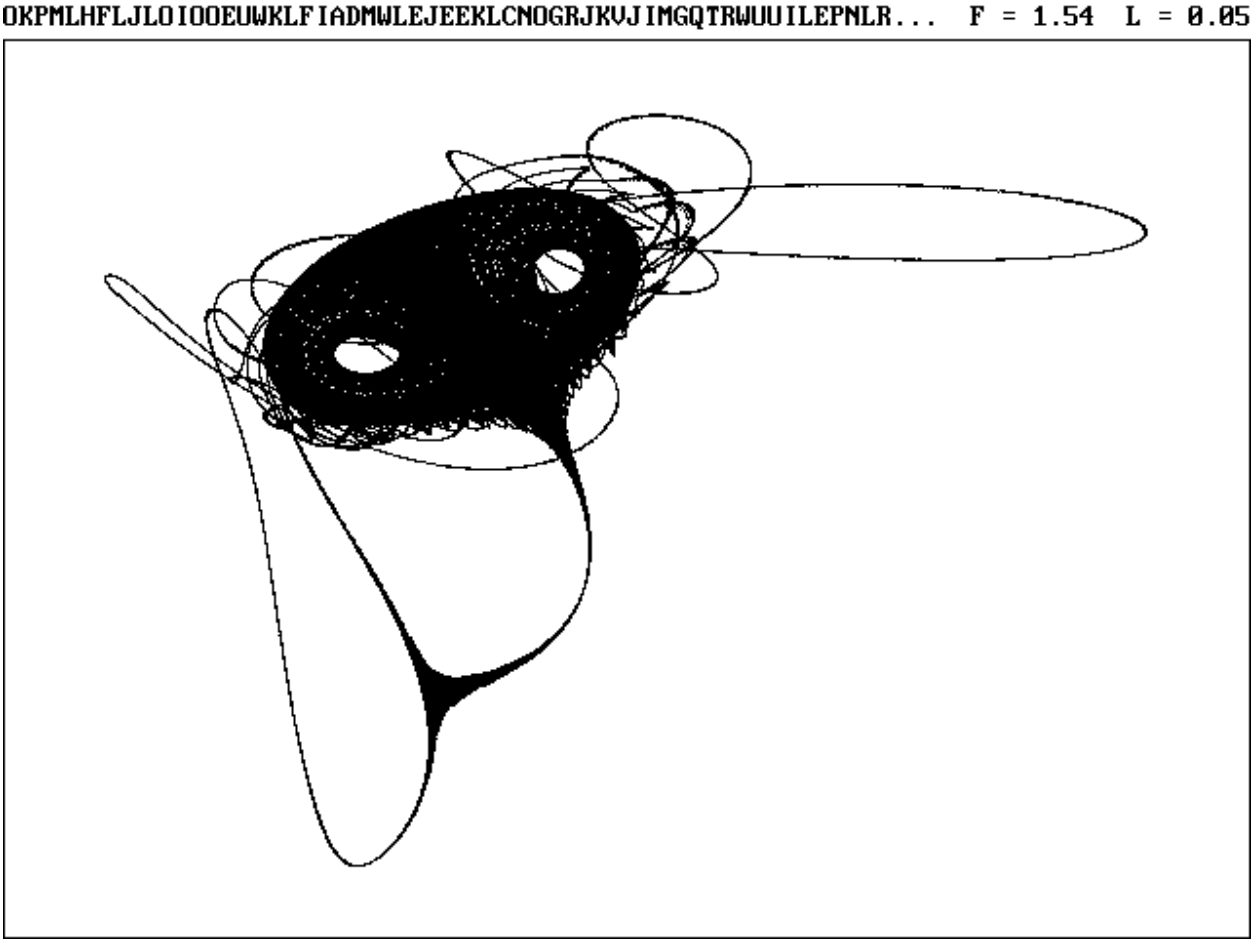


Figure 5-20. Projection of four-dimensional quartic map



5.3 Other Display Techniques

Projecting two of the four dimensions onto the remaining two is akin to buying a Ferrari to make trips to the grocery store. Much of our effort is wasted. We need to use the techniques developed in the last chapter to display three dimensions and devise additional methods to display simultaneously the fourth dimension.

Since we have several methods for displaying three dimensions, we should be able to use some of them in combination to visualize all four dimensions. Table 5-1 summarizes the display techniques we have used and indicates the number of dimensions that can be visualized with various combinations of them. In the table, a dash indicates that the combination is not possible, and a question mark indicates that the combination is possible but leads to contradictory visual information.

Table 5-1. Combinations of display techniques and the number of dimensions that can be visualized with each

	Third Dimension						
Fourth Dimension	Project	Shadow	Bands	Color	Anaglyph	Stereo	Slices
Project	**2D**	**3D**	**3D**	**3D**	**3D**	**3D**	**3D**
Shadow	3D	-	4D	4D	?	?	4D
Bands	**3D**	**4D**	?	**4D**	**4D**	**4D**	**4D**
Color	**3D**	**4D**	**4D**	-	-	**4D**	**4D**
Anaglyph	3D	?	4D	-	-	?	4D
Stereo	3D	?	4D	4D	?	-	4D
Slices	3D	4D	4D	4D	4D	4D	-

In Table 5-1, the entries in **boldface** are the ones we will implement in the program. They were chosen because of their visual effectiveness, ease of programming, and lack of redundancy with other combinations. Cases below and to the left of the diagonal duplicate those above and to the right. The changes needed in the program to produce such four-dimensional displays are shown in **PROG19**.

PROG19. Changes required in PROG18 to display the fourth dimension

```
1000 REM FOUR-D MAP SEARCH (With 4-D Display Modes)

1040 PREV% = 5                  'Plot versus fifth previous iterate

1120 TRD% = 1                   'Display third dimension as shadow

1130 FTH% = 2                   'Display fourth dimension as colors



3630 IF Q$ = "" OR INSTR("ADHIPRSX", Q$) = 0 THEN GOSUB 4200

3720 IF Q$ = "H" THEN FTH% = (FTH% + 1) MOD 3: T% = 3: IF N > 999 THEN N = 999:
GOSUB 5600



4330    PRINT TAB(27); "H: Fourth dimension is ";

4340        IF FTH% = 0 THEN PRINT "projection"

4350        IF FTH% = 1 THEN PRINT "bands     "

4360        IF FTH% = 2 THEN PRINT "colors    "



5010 C4% = WH%

5020 IF D% < 4 THEN GOTO 5050

5030    IF FTH% = 1 THEN IF INT(30 * (W - WMIN) / (WMAX - WMIN)) MOD 2 THEN GOTO
5330

5040    IF FTH% = 2 THEN C4% = 1 + INT(NC% * (W - WMIN) / (WMAX - WMIN) + NC%)
MOD NC%

5050 IF D% < 3 THEN PSET (XP, YP): GOTO 5330    'Skip 3-D stuff

5060 IF TRD% = 0 THEN PSET (XP, YP), C4%

5080    IF D% > 3 AND FTH% = 2 THEN PSET (XP, YP), C4%: GOTO 5110

5130 IF TRD% <> 2 THEN GOTO 5160
```

```
5140    IF D% > 3 AND FTH% = 2 AND (INT(15 * (Z - ZMIN) / (ZMAX - ZMIN) + 2) MOD
2) = 1 THEN PSET (XP, YP), C4%

5150   IF D% < 4 OR FTH% <> 2 THEN C% = COLR%(INT(60 * (Z - ZMIN) / (ZMAX - ZMIN)
+ 4) MOD 4): PSET (XP, YP), C%

5260    XRT = XA + (XP + XZ * (Z - ZA) - XL) / HSF: PSET (XRT, YP), C4%

5270    XLT = XA + (XP - XZ * (Z - ZA) - XH) / HSF: PSET (XLT, YP), C4%

5320    PSET (XP, YP), C4%



5630 IF TRD% = 3 OR (D% > 3 AND FTH% = 2 AND TRD% <> 1) THEN FOR I% = 0 TO NC%:
COLR%(I%) = I% + 1: NEXT I%
```

In presenting sample displays from **PROG19**, we ignore those that convey only three-dimensional information and concentrate on the new combinations that permit full four-dimensional displays. They fall into two groups—those that require the use of color and those that do not. Examples of the three 4-D monochrome combinations are shown in Figures 5-21 through 5-44, and examples of the six color combinations are shown in Plates 17 through 22.

Figure 5-21. Four-dimensional quadratic map with shadow bands

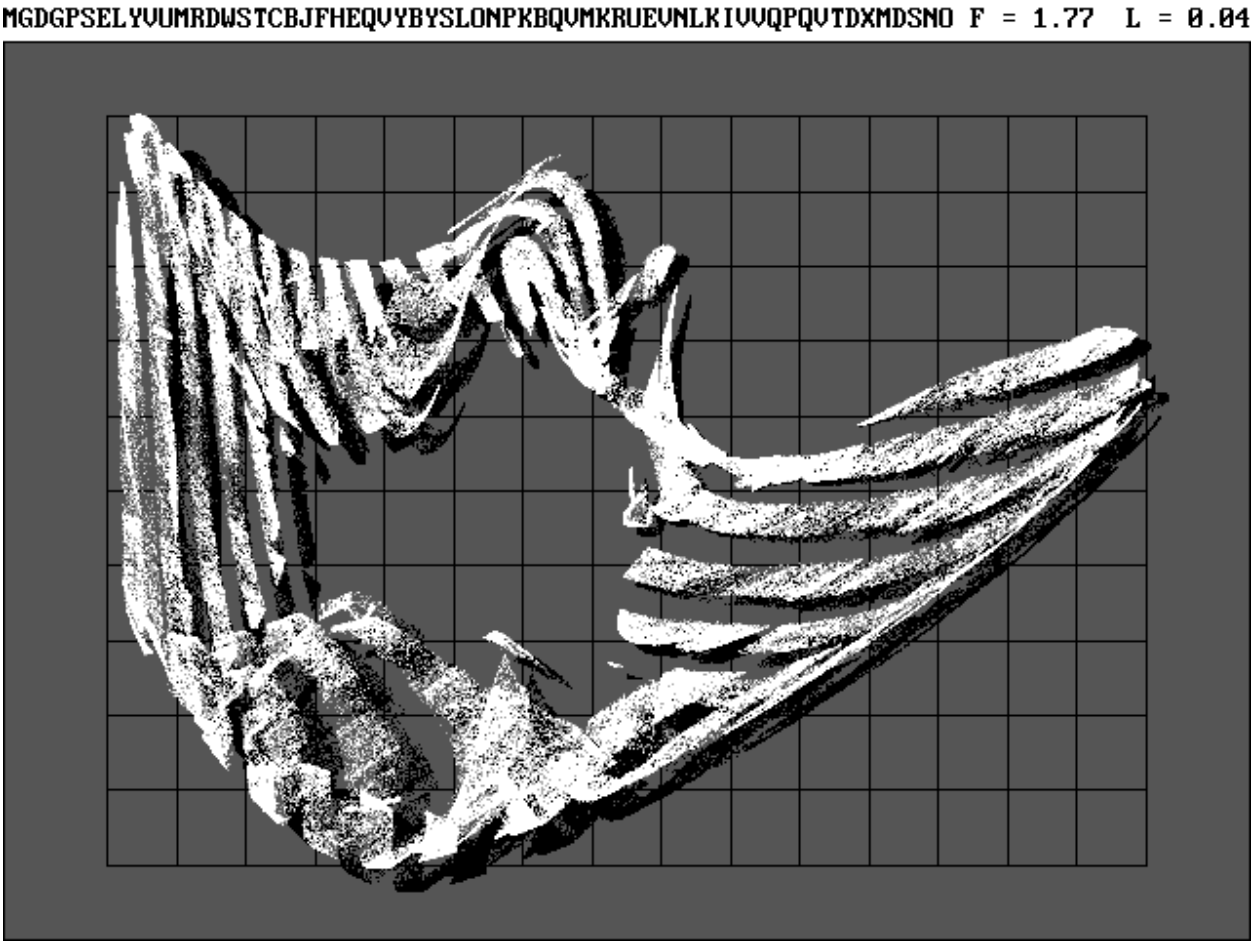


Figure 5-22. Four-dimensional quadratic map with shadow bands

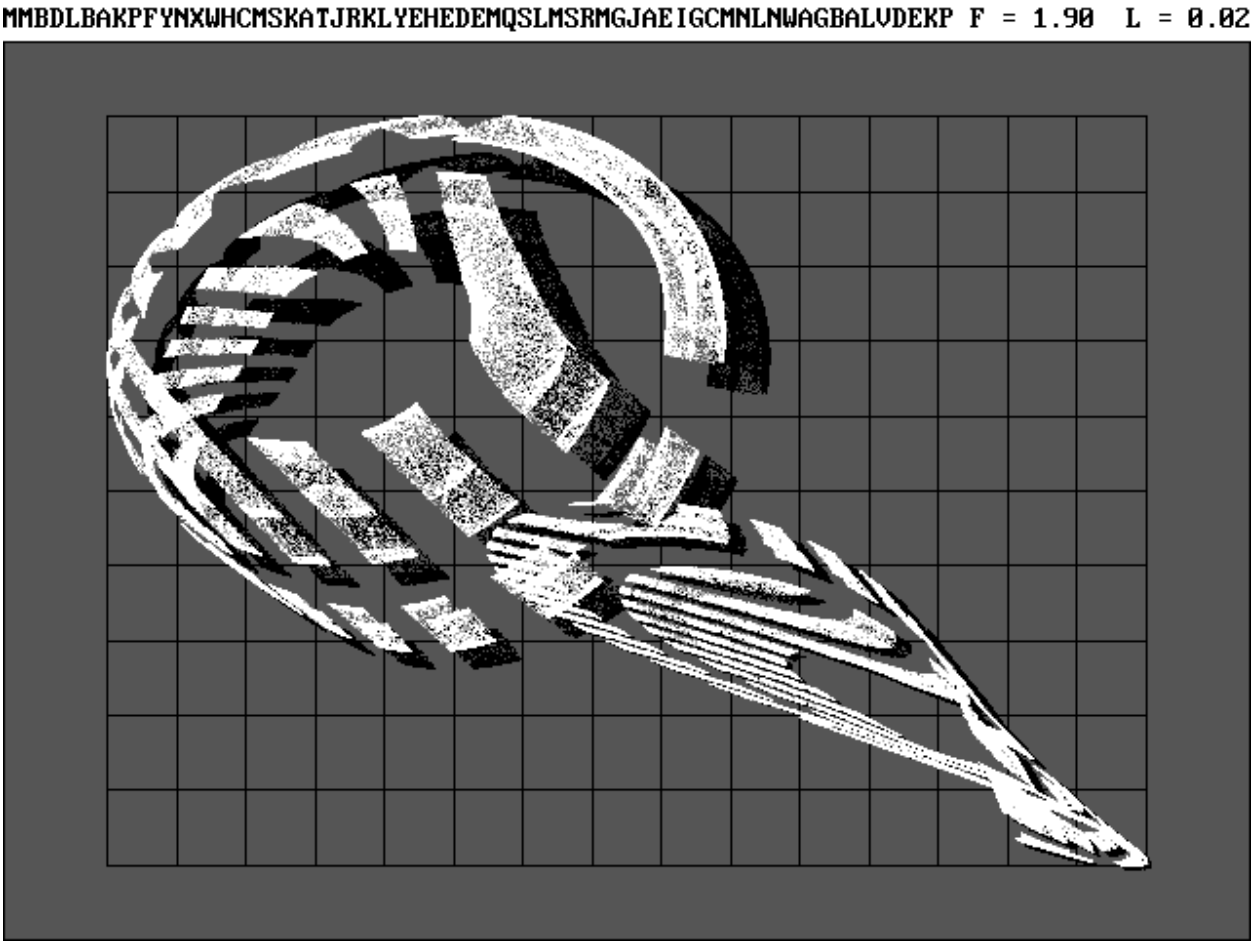


Figure 5-23. Four-dimensional quadratic map with shadow bands

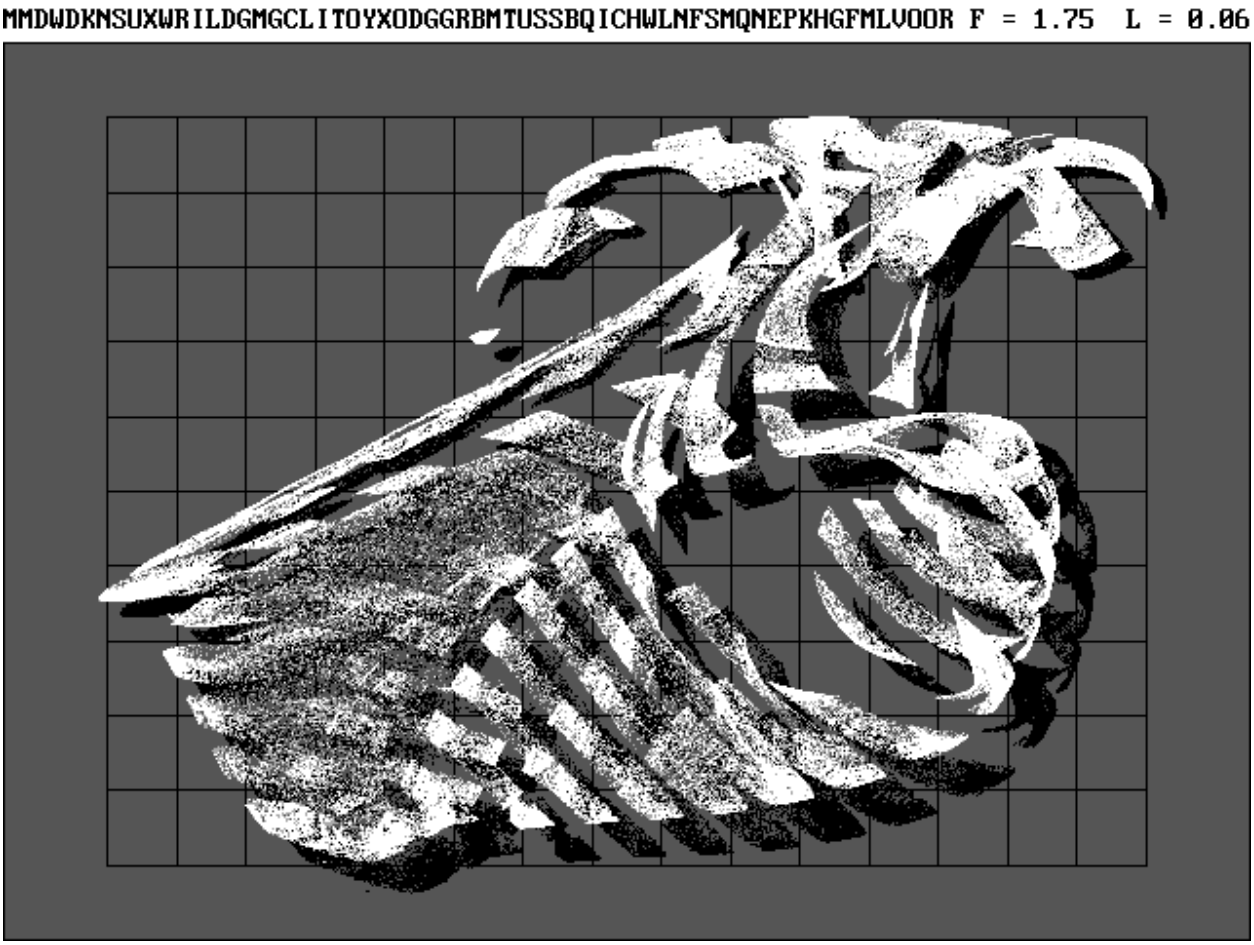


Figure 5-24. Four-dimensional quadratic map with shadow bands

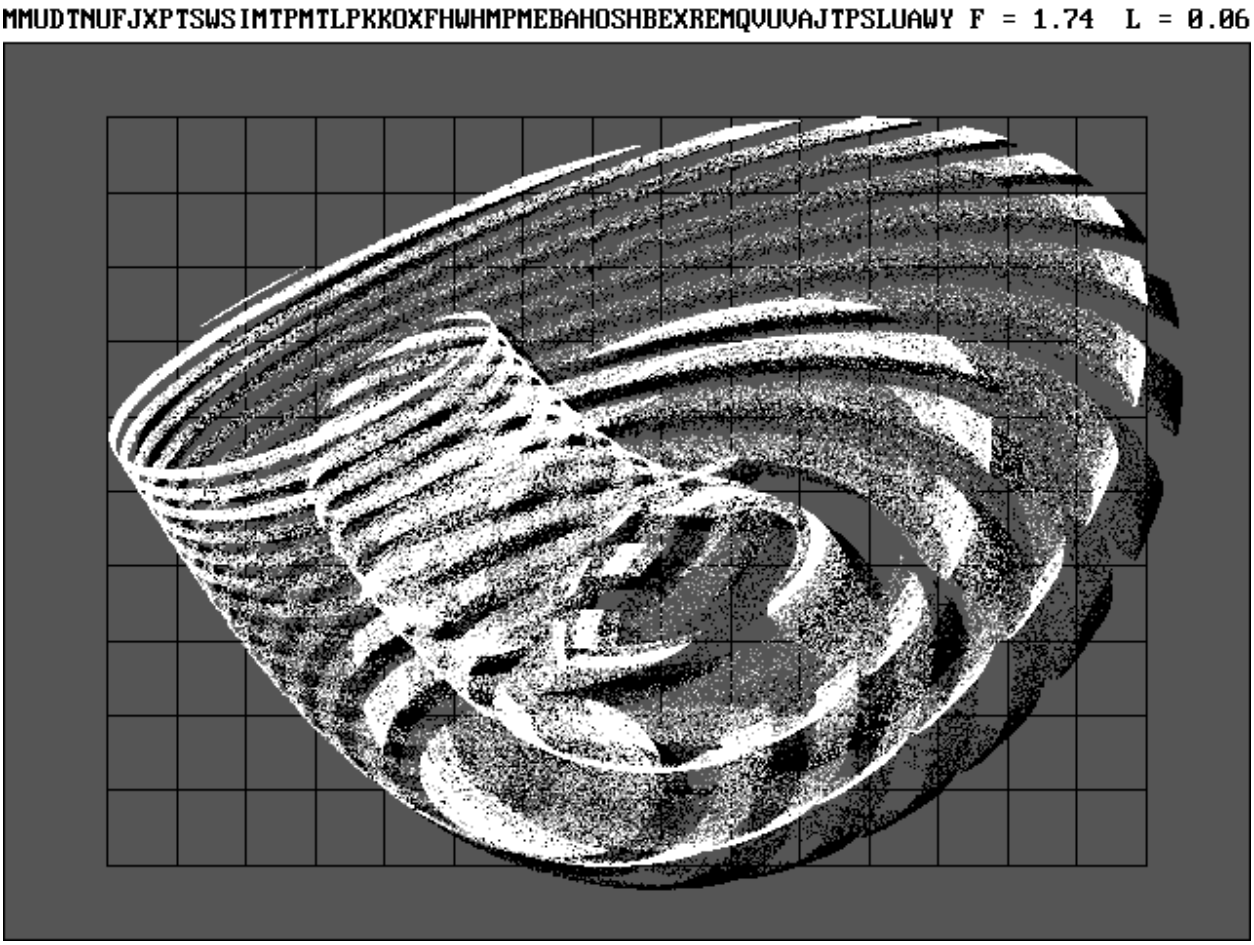


Figure 5-25. Four-dimensional quadratic map with shadow bands

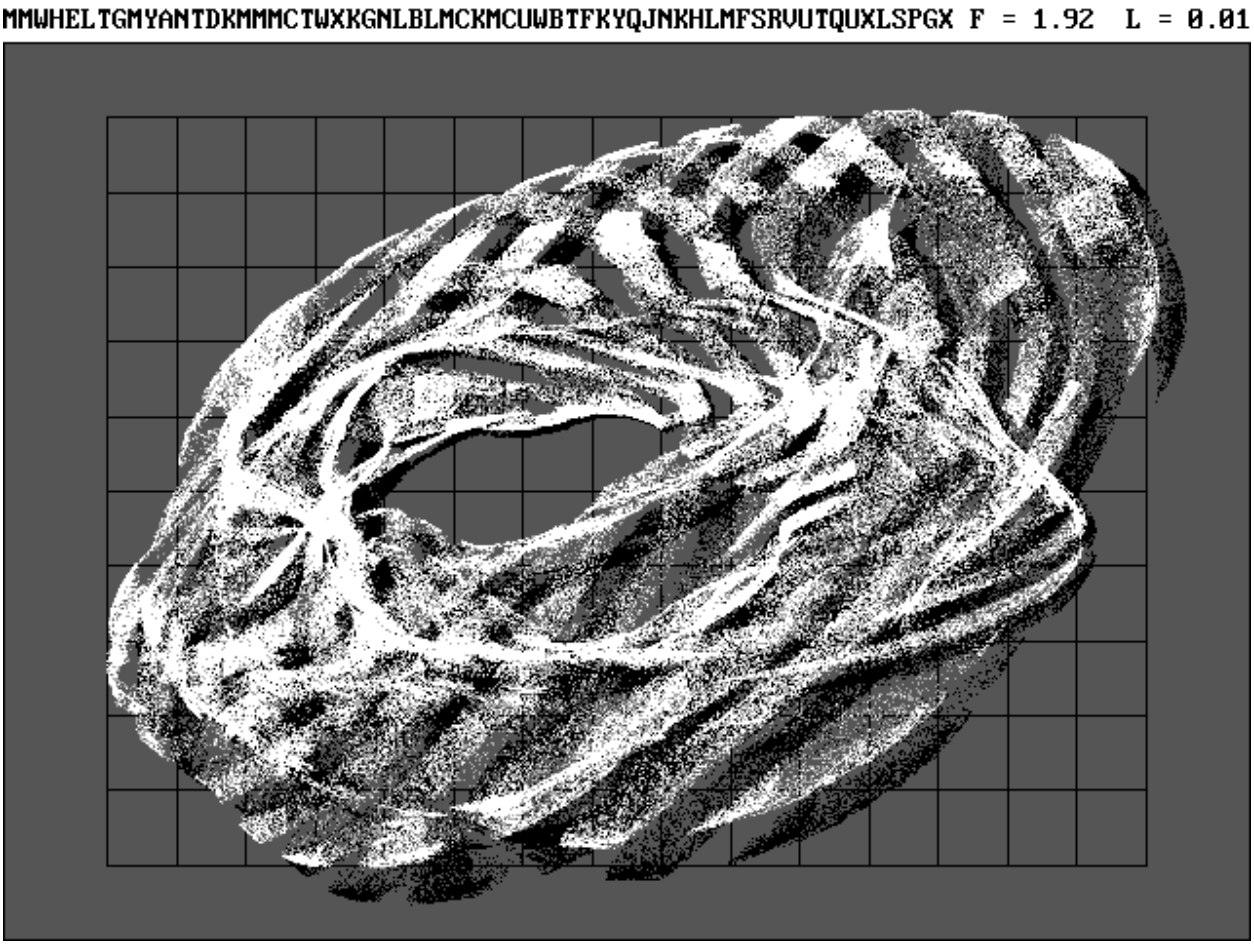


Figure 5-26. Four-dimensional quadratic map with shadow bands

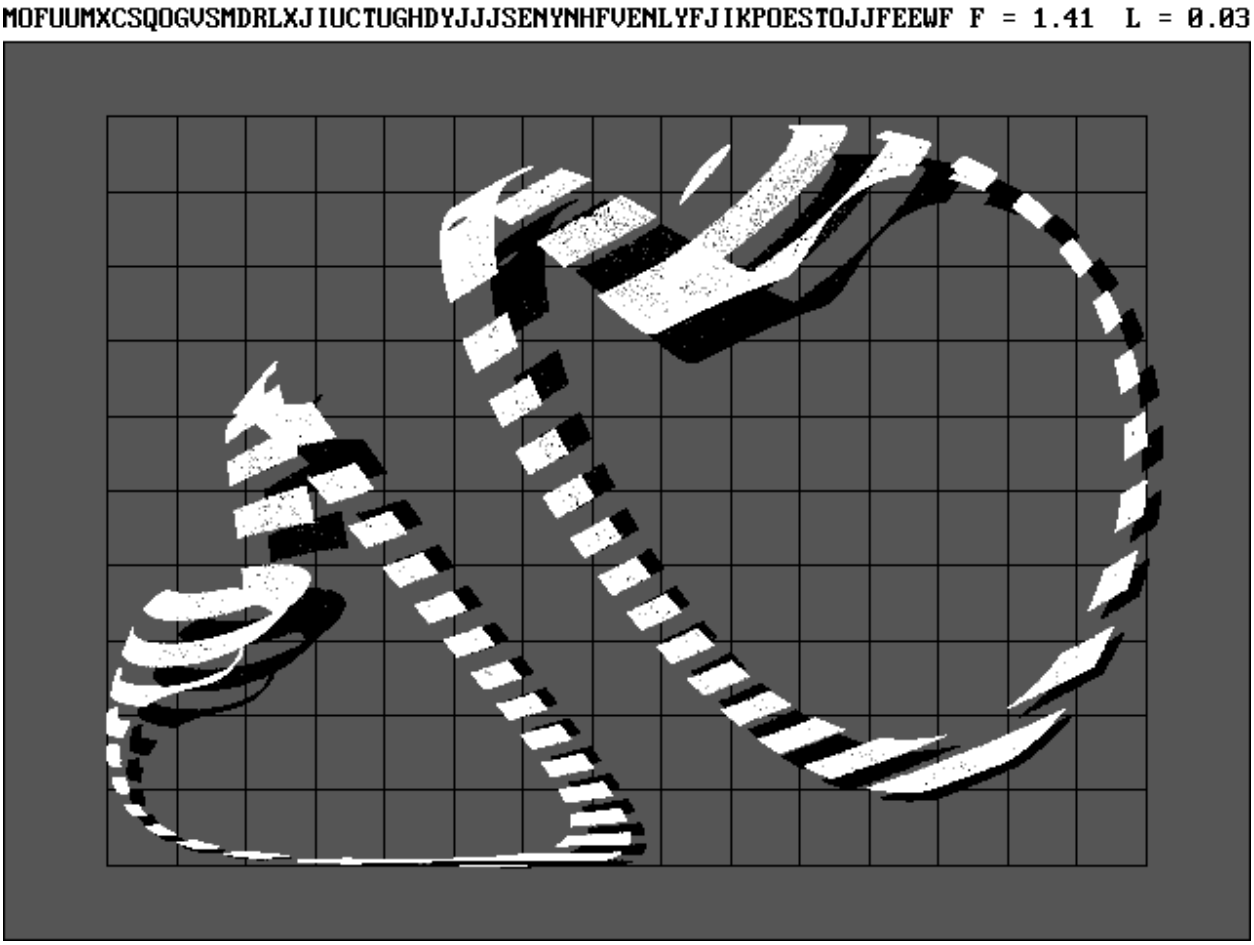


Figure 5-27. Four-dimensional quadratic map with shadow bands

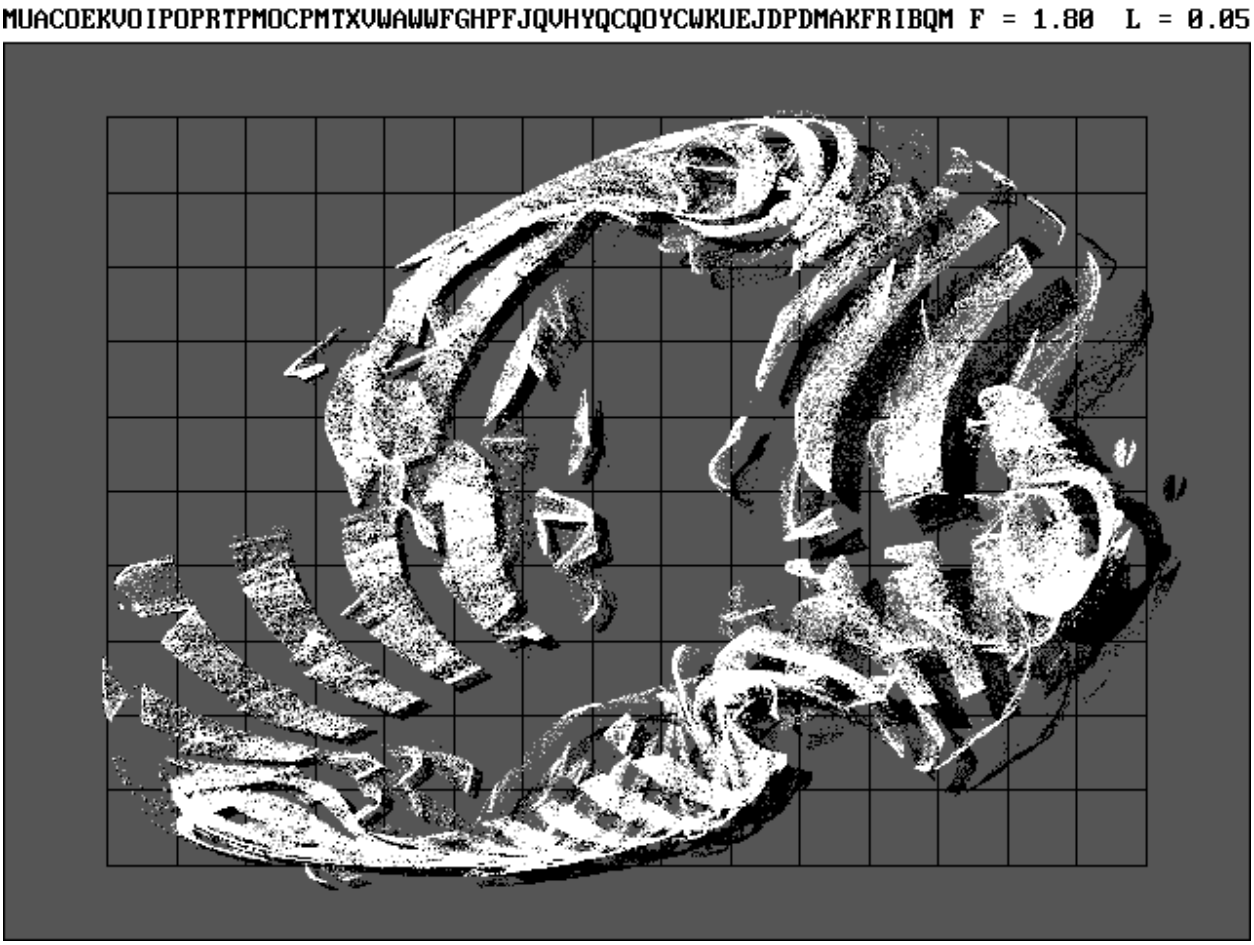


Figure 5-28. Four-dimensional cubic map with shadow bands

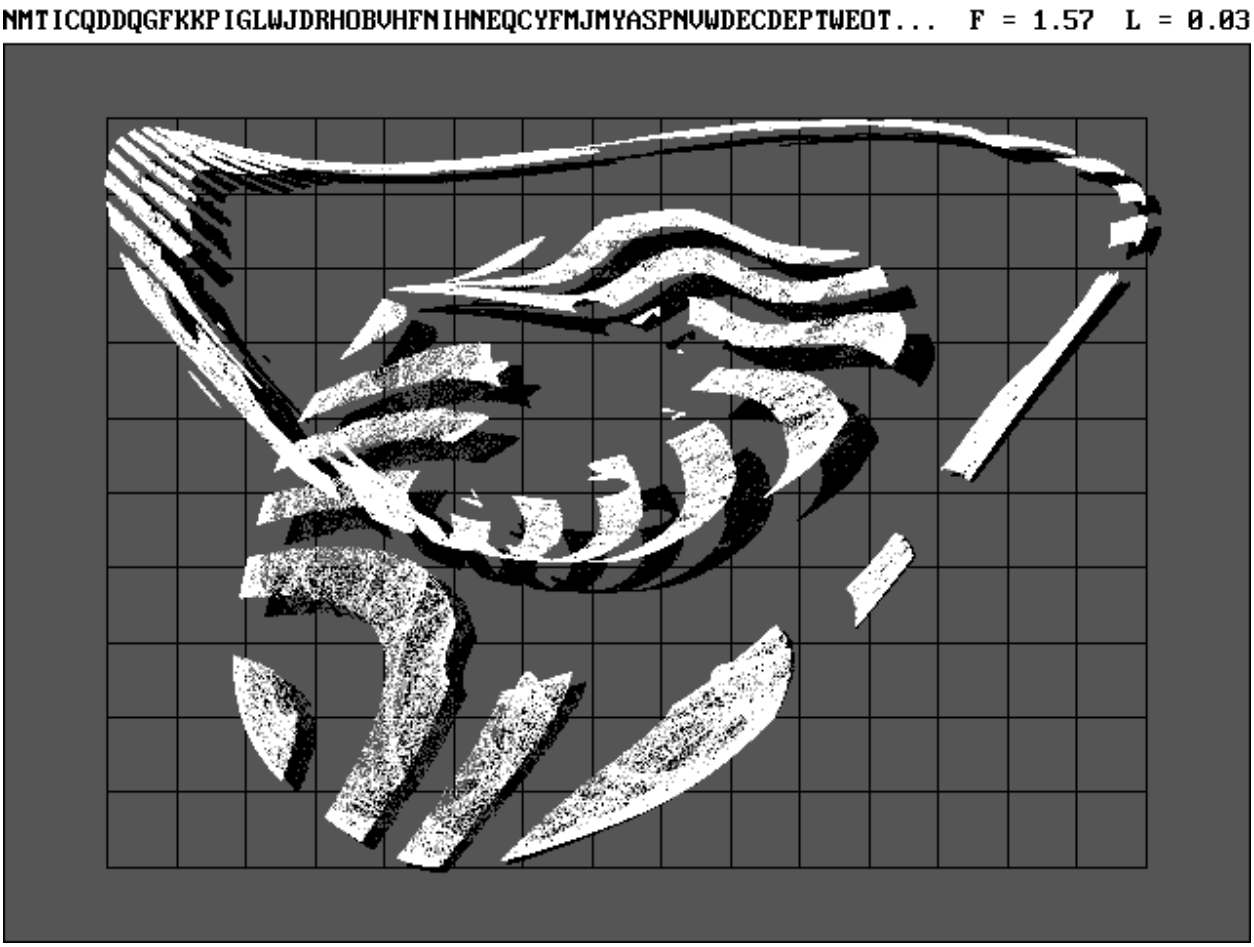


Figure 5-29. Four-dimensional quadratic map with stereo bands

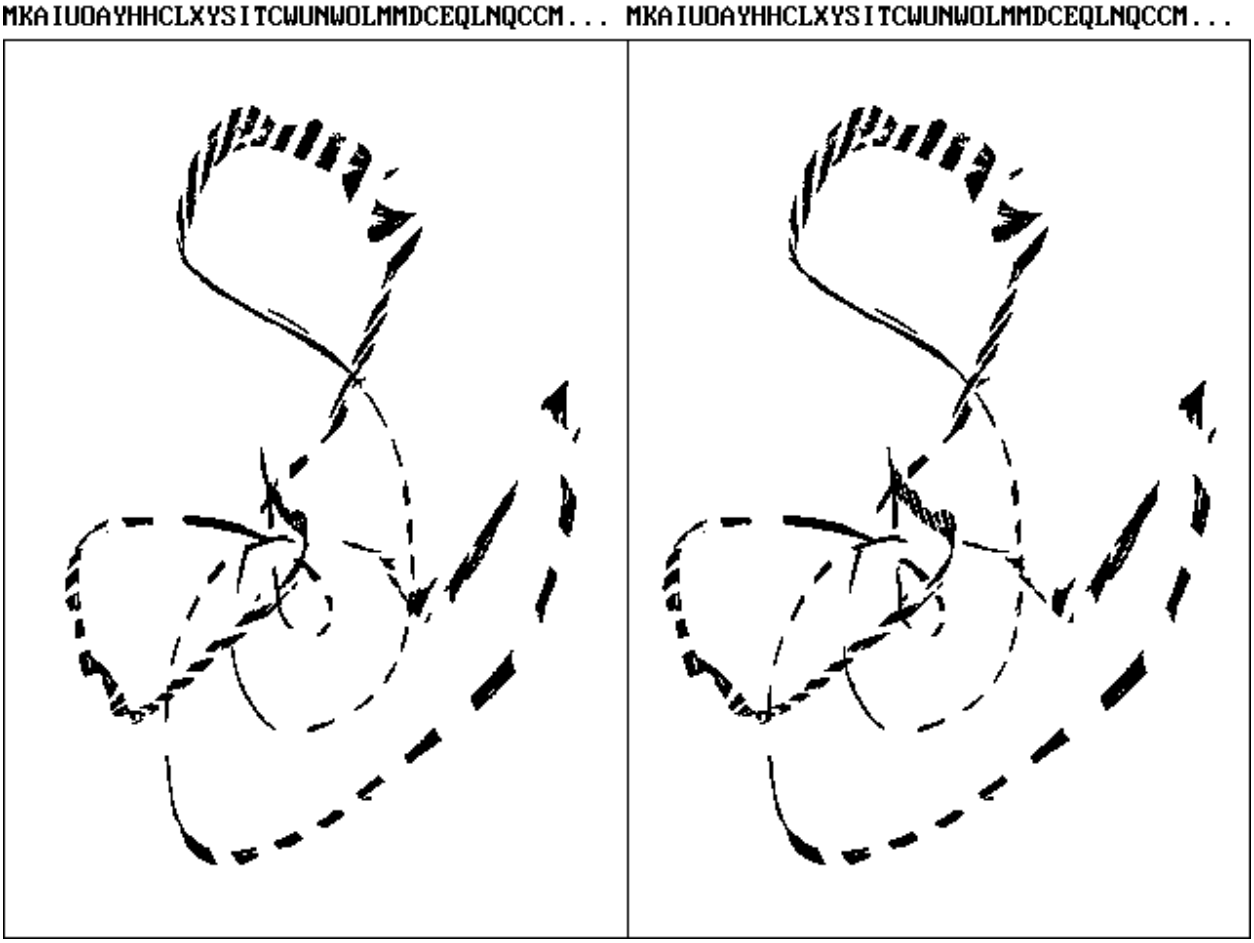


Figure 5-30. Four-dimensional quadratic map with stereo bands

MKWWFTQHWFFBKCFHMUSVFFFANCROGSSPPGOY... MKWWFTQHWFFBKCFHMUSVFFFANCROGSSPPGOY...

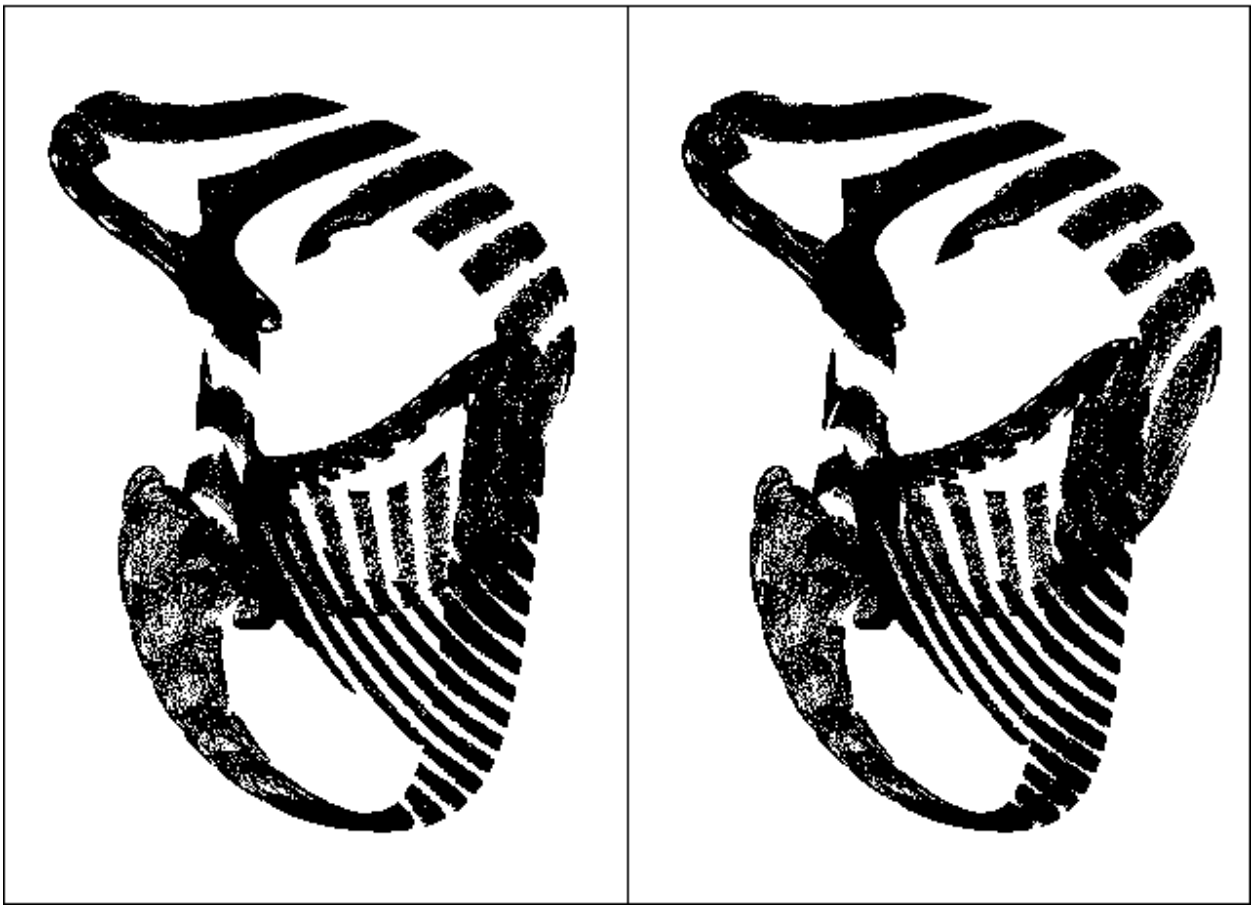


Figure 5-31. Four-dimensional quadratic map with stereo bands

MNGFOFMKCYIFGJSSOVQKUYKUHTITJWEMUHNP... MNGFOFMKCYIFGJSSOVQKUYKUHTITJWEMUHNP...

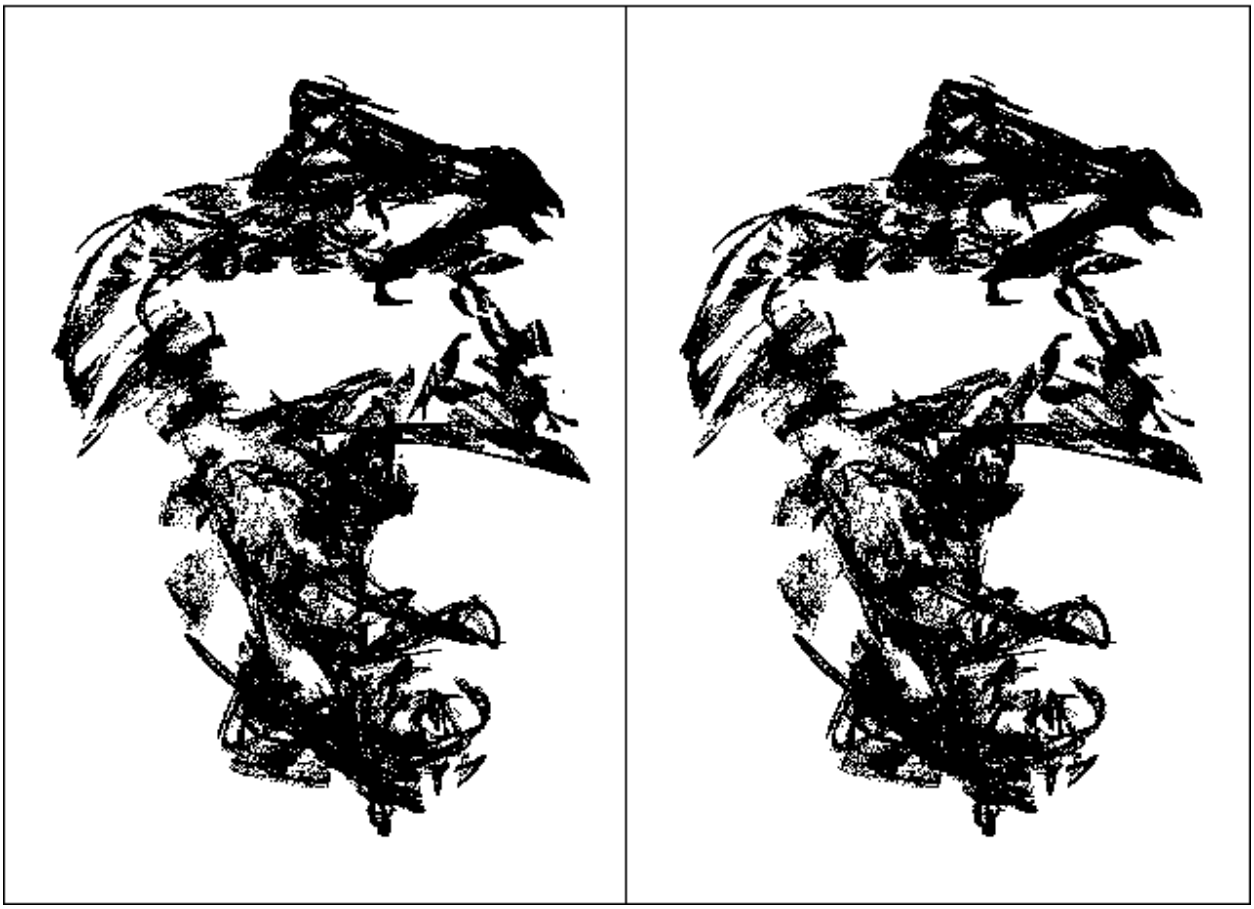


Figure 5-32. Four-dimensional cubic map with stereo bands

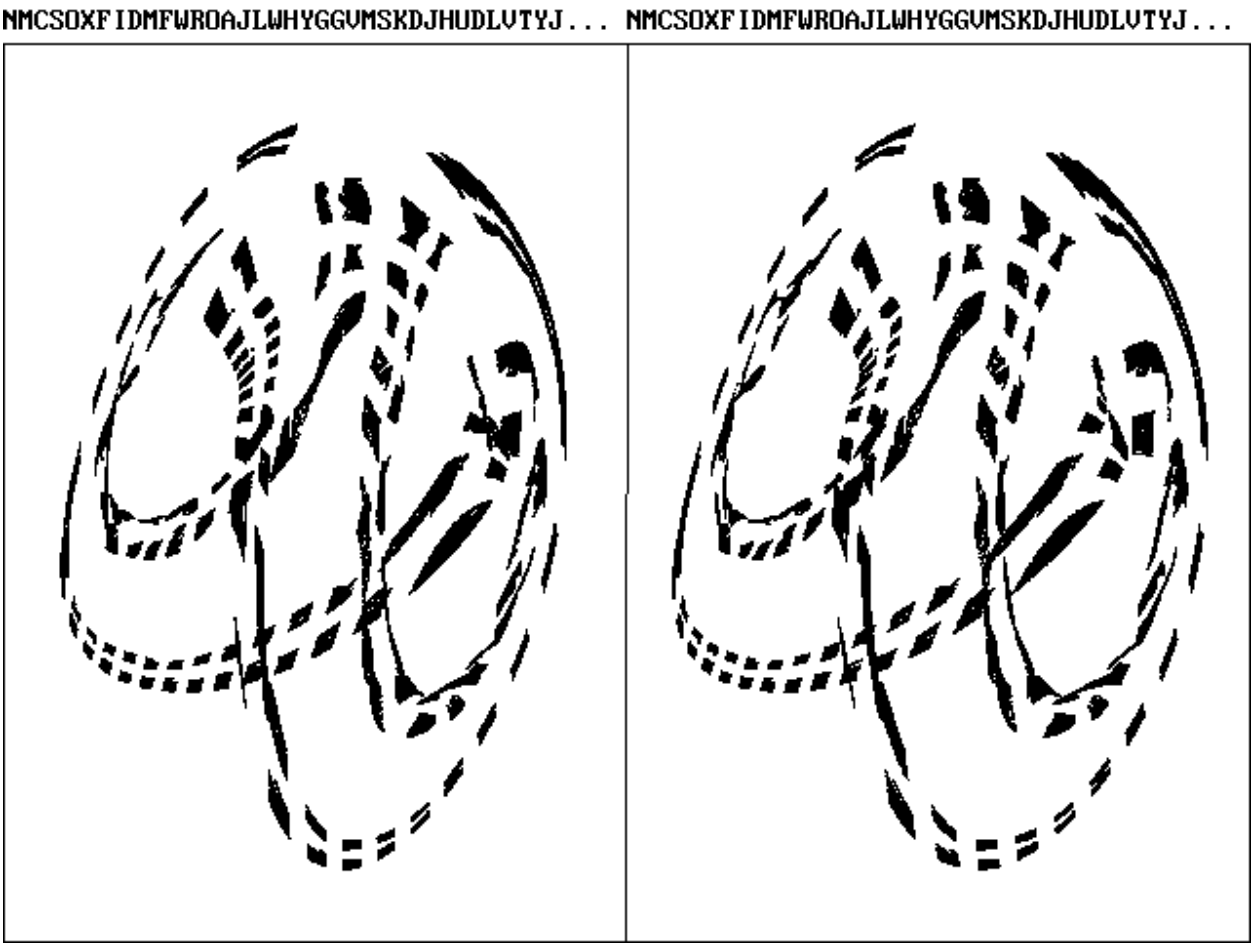


Figure 5-33. Four-dimensional cubic map with stereo bands

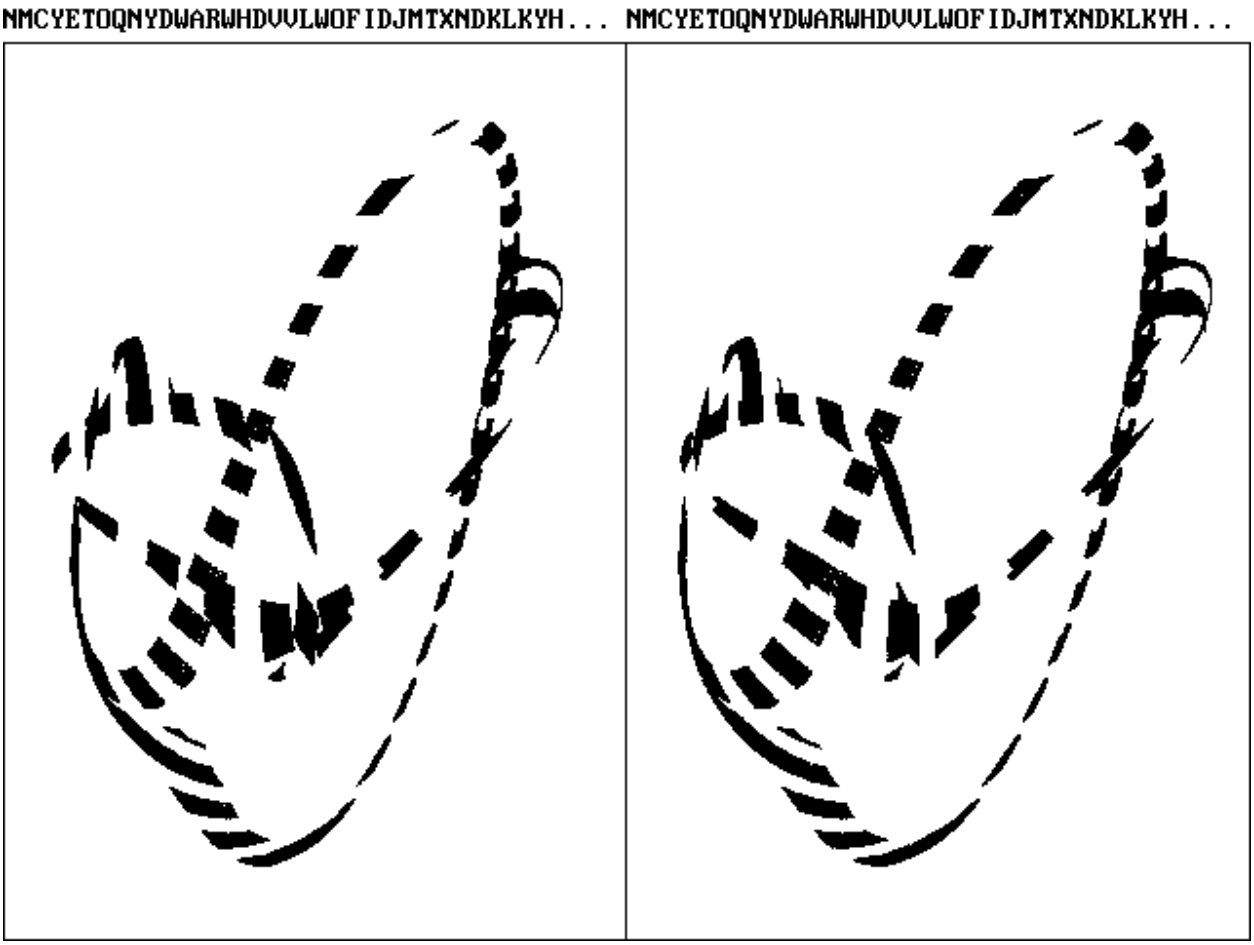


Figure 5-34. Four-dimensional cubic map with stereo bands

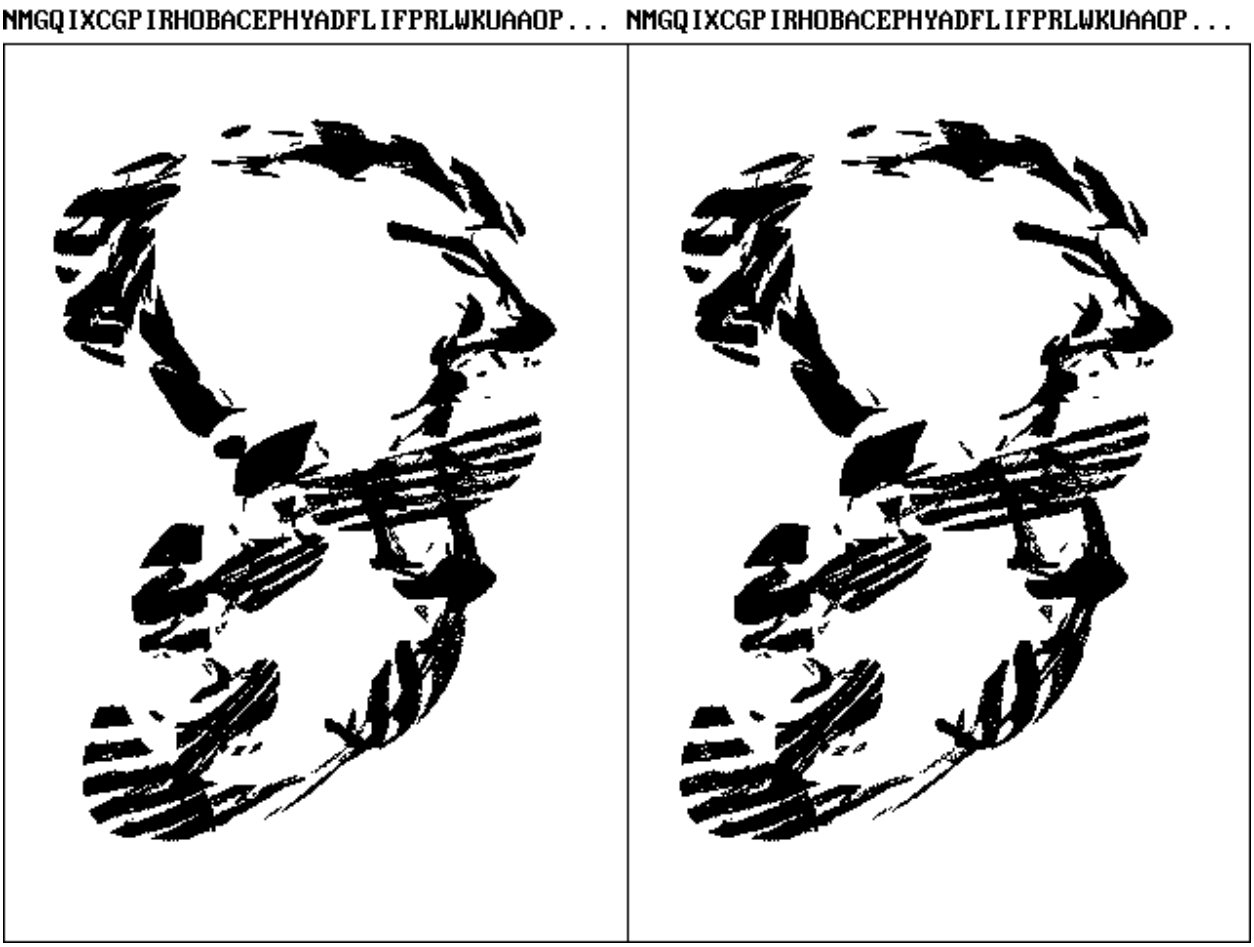


Figure 5-35. Four-dimensional quartic map with stereo bands

OMOQRUJSLQIHLLAVGPORHXCLMMXSOOMGQWBE... OMOQRUJSLQIHLLAVGPORHXCLMMXSOOMGQWBE...

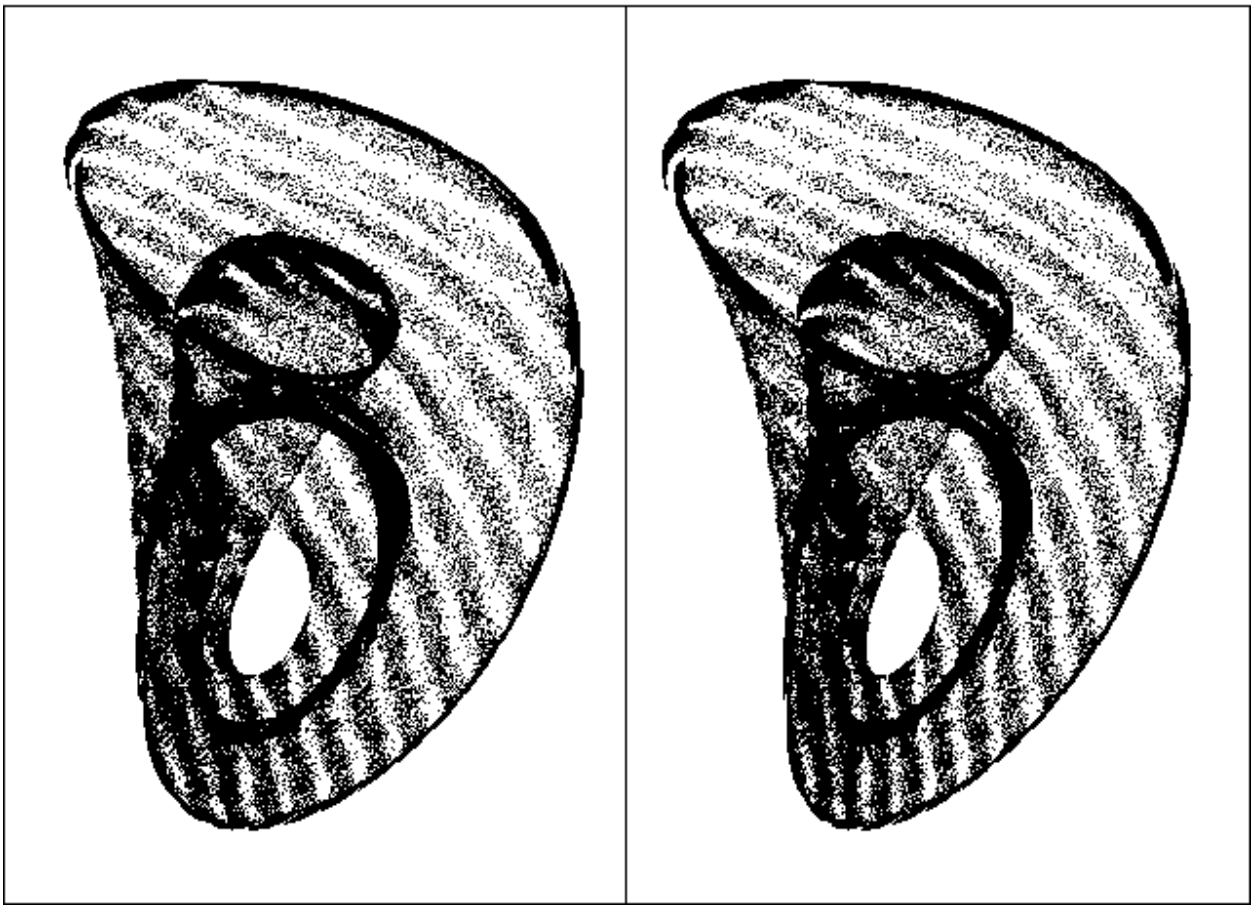


Figure 5-36. Four-dimensional quartic map with stereo bands

OMXLIBUDPLKNSYDVPOOVLCAIPMVSKGCOECHS... OMXLIBUDPLKNSYDVPOOVLCAIPMVSKGCOECHS...

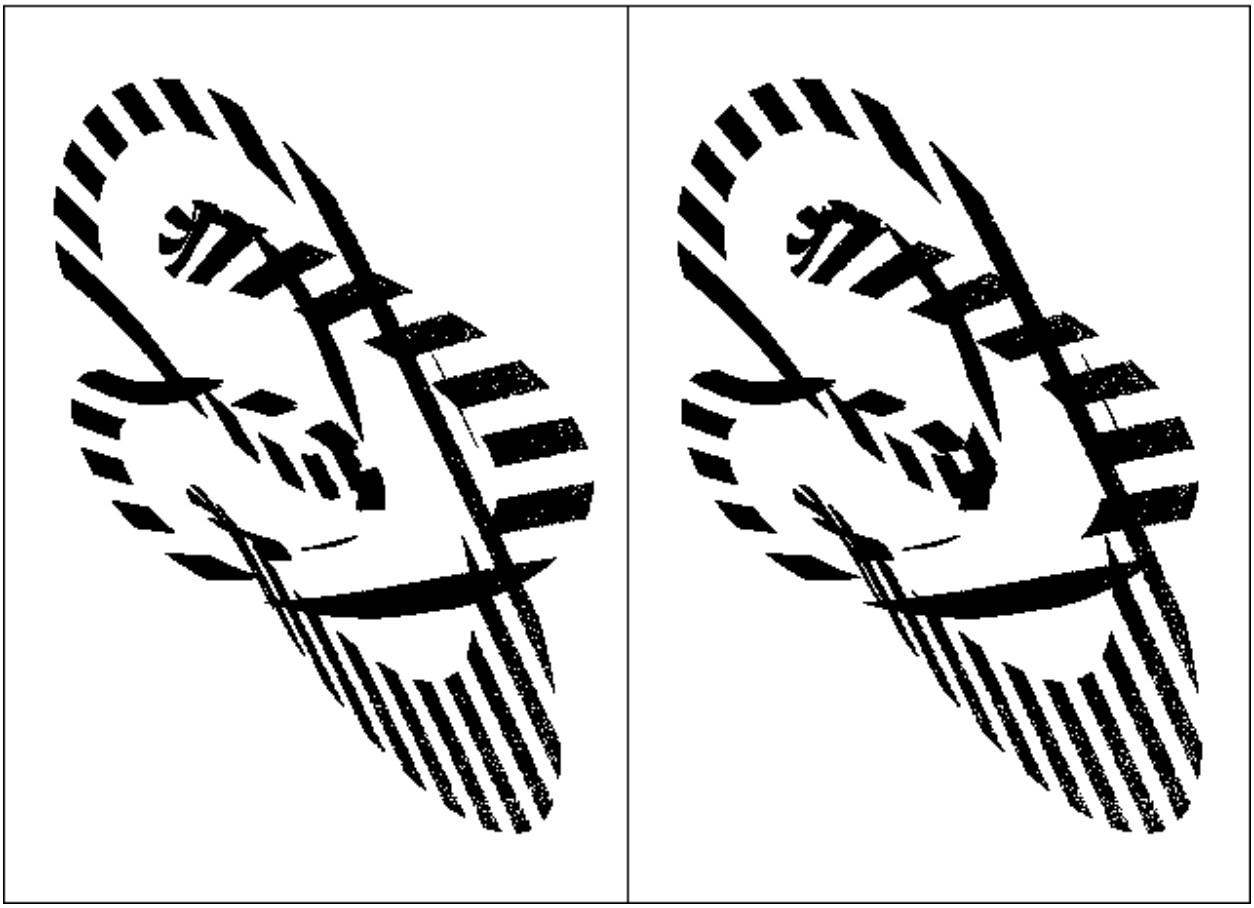


Figure 5-37. Four-dimensional quadratic map with sliced bands

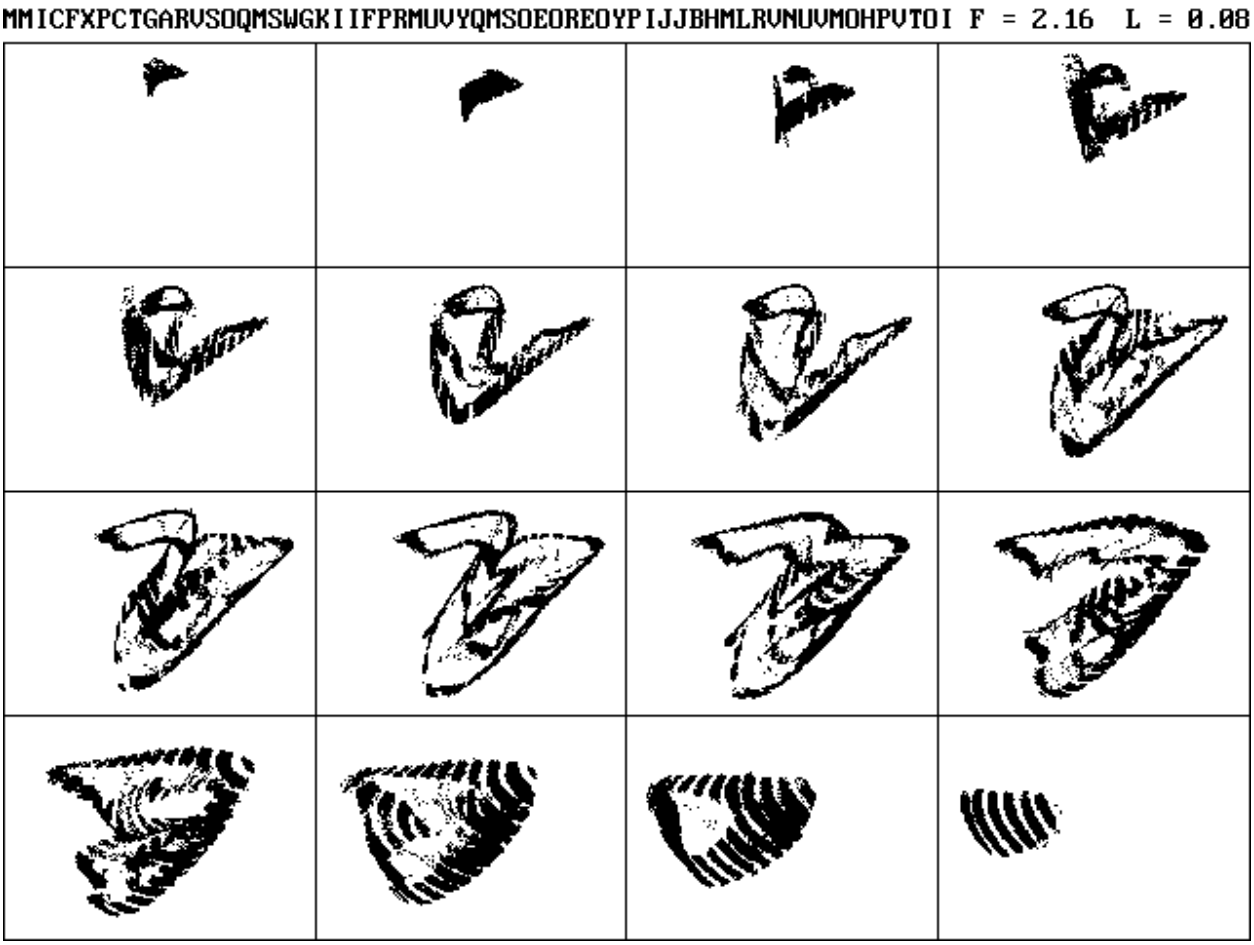


Figure 5-38. Four-dimensional quadratic map with sliced bands

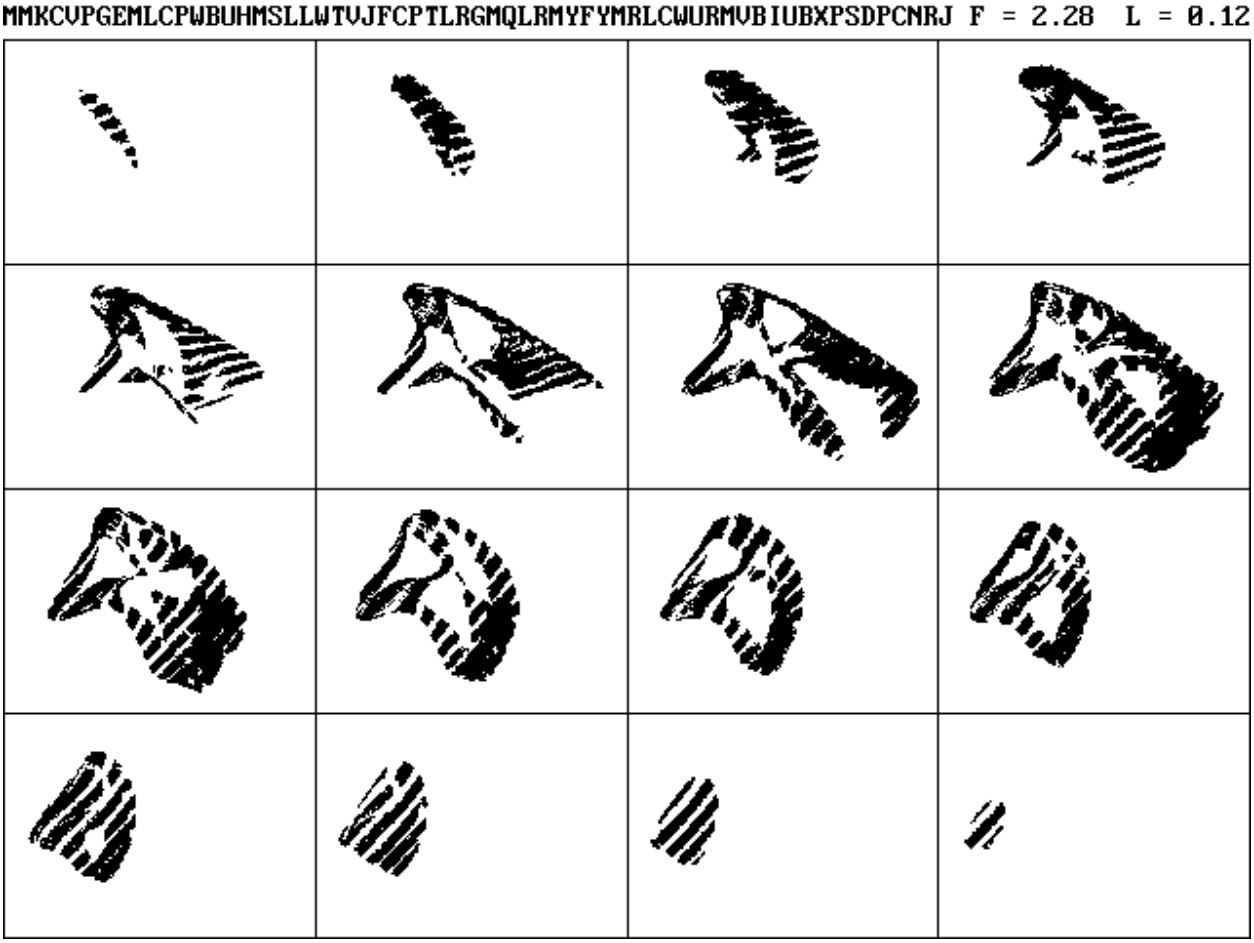


Figure 5-39. Four-dimensional quadratic map with sliced bands

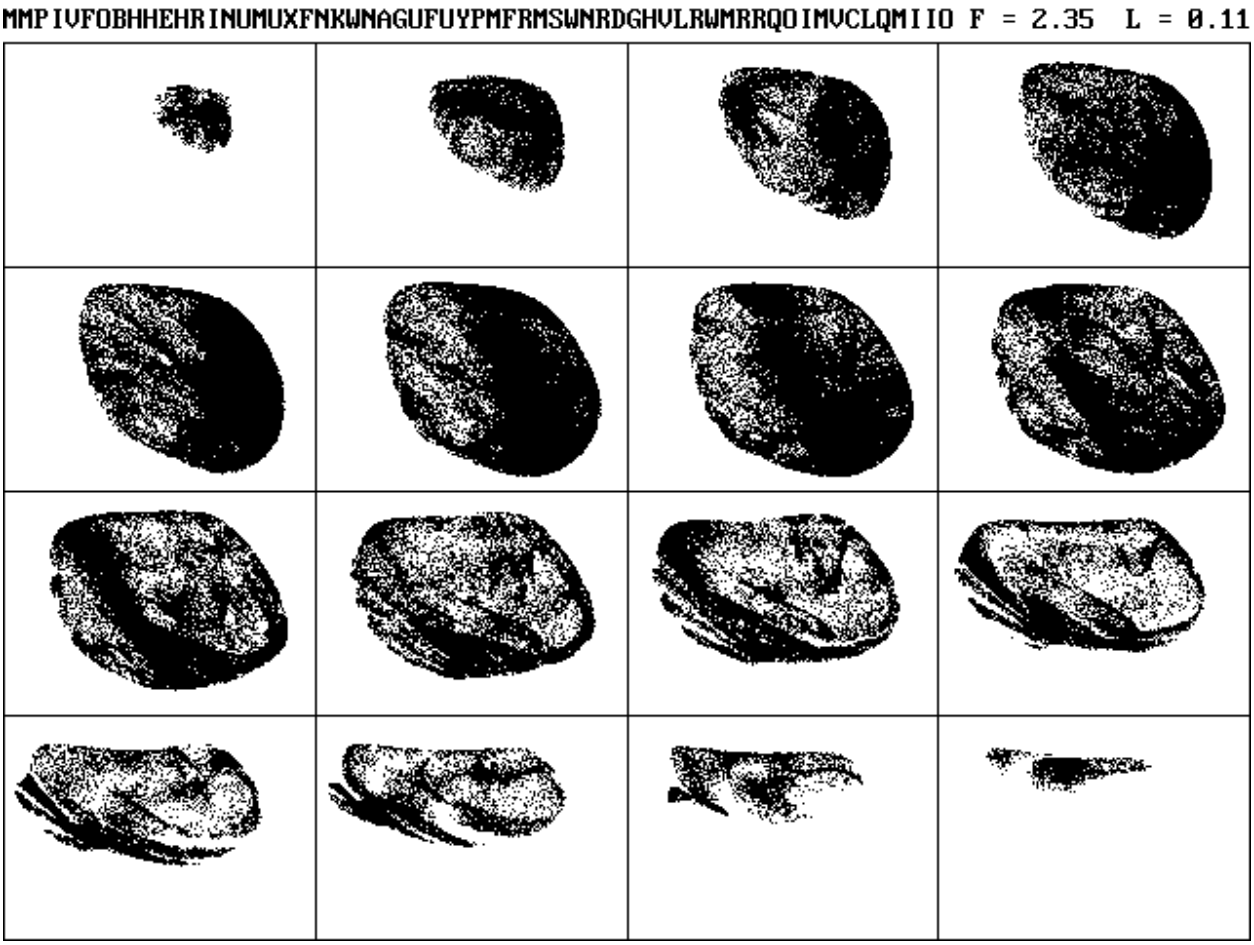


Figure 5-40. Four-dimensional quadratic map with sliced bands

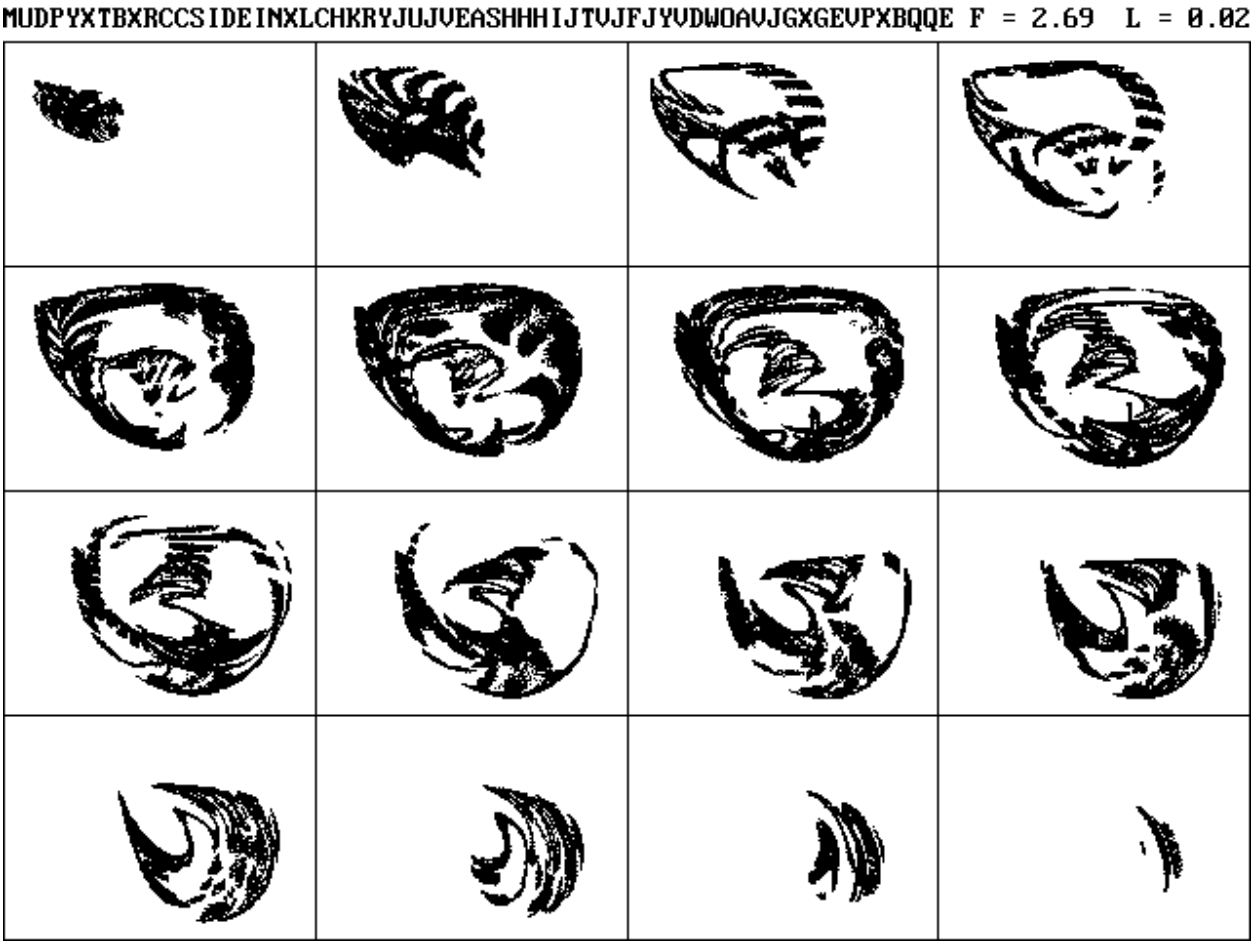


Figure 5-41. Four-dimensional cubic map with sliced bands

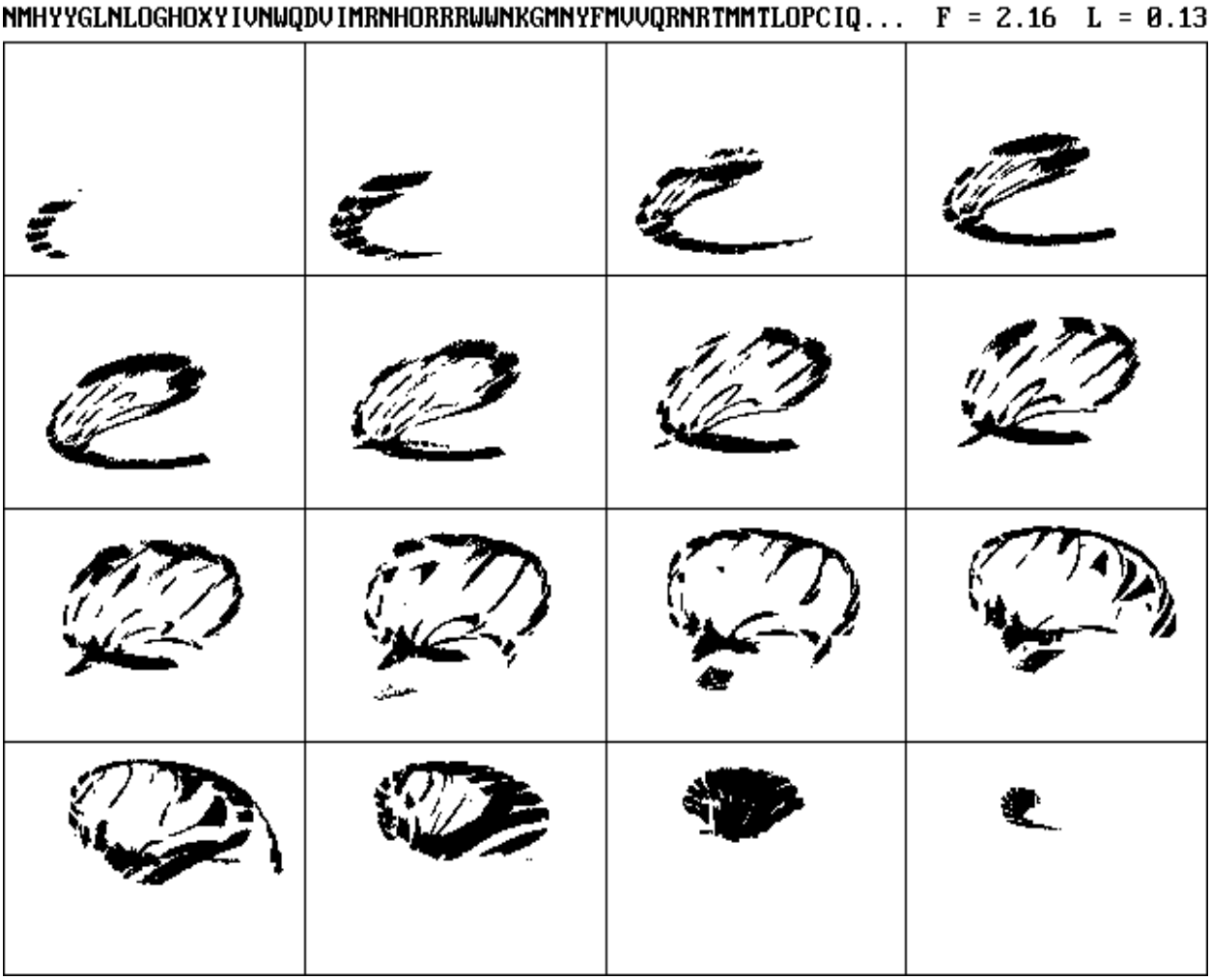


Figure 5-42. Four-dimensional quartic map with sliced bands

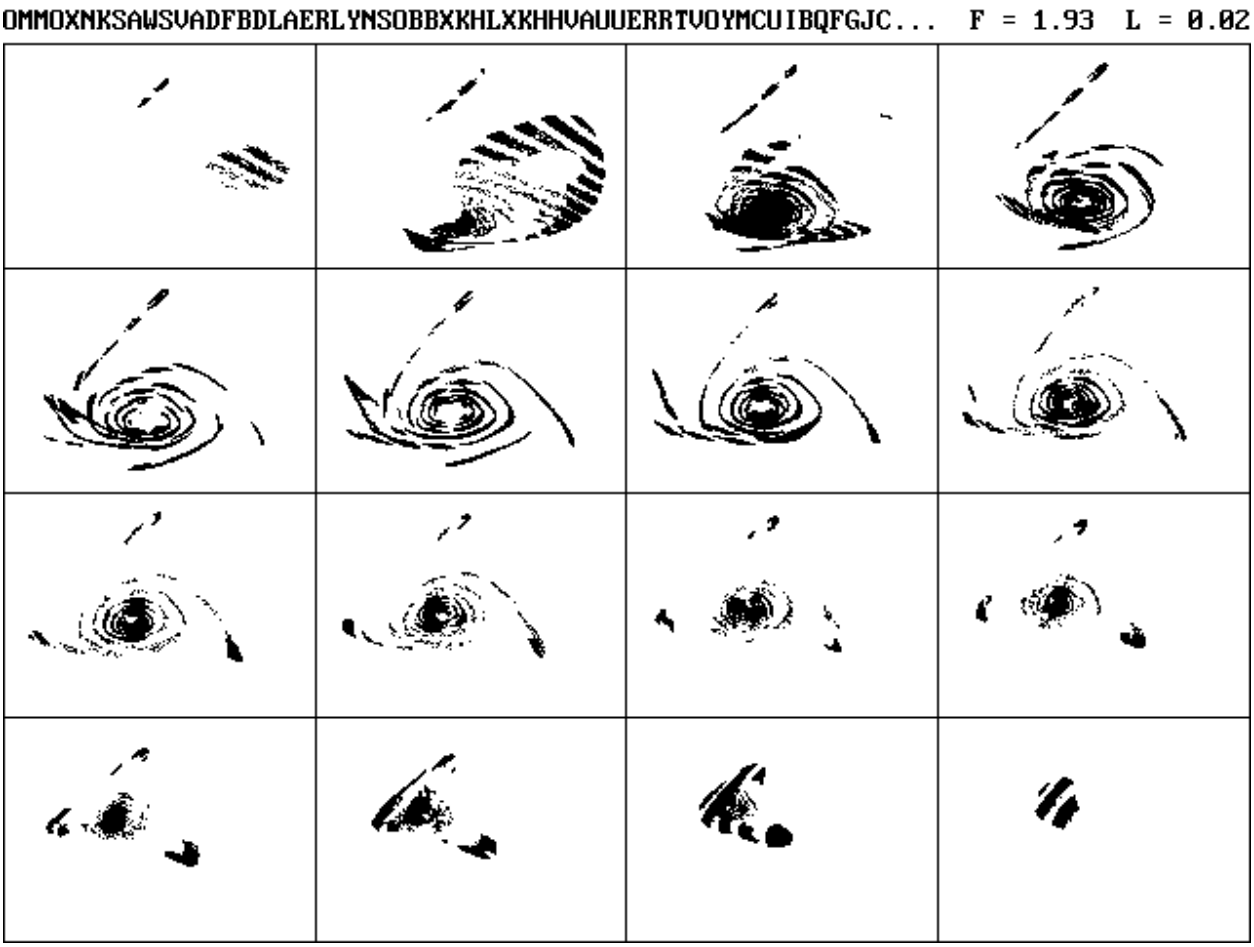


Figure 5-43. Four-dimensional quartic map with sliced bands

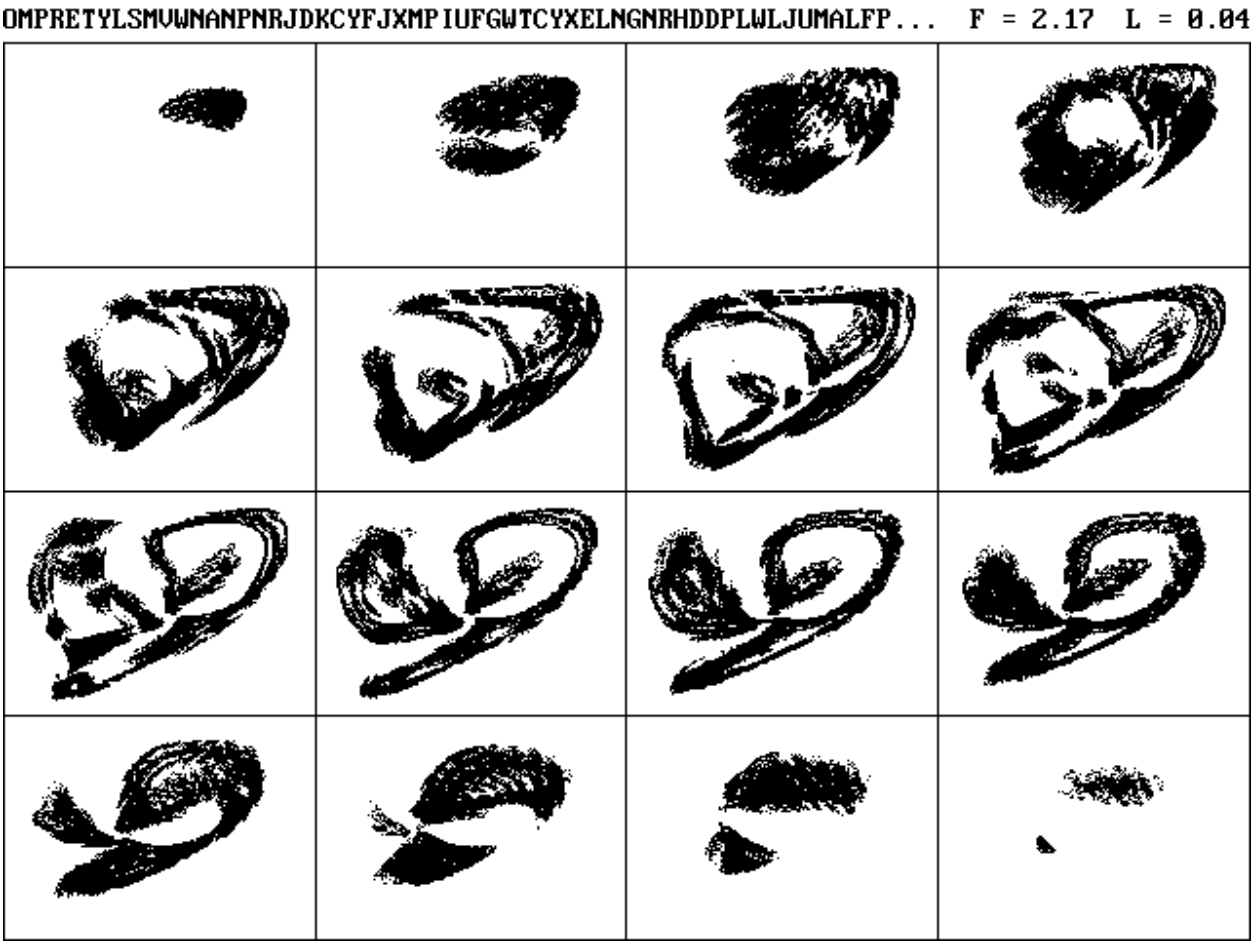
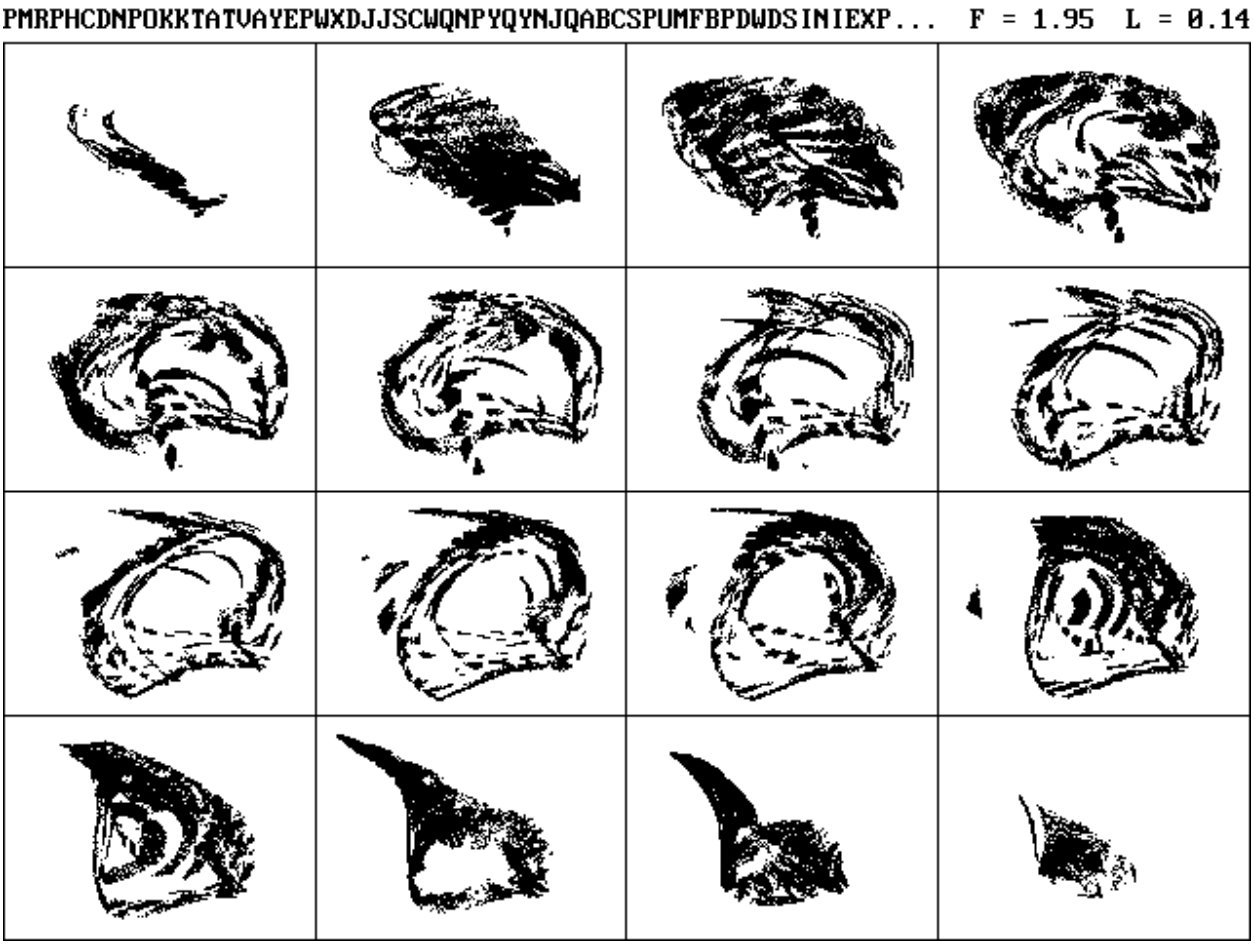


Figure 5-44. Four-dimensional quintic map with sliced bands



You might be interested in the challenge of producing attractors embedded in dimensions higher than four. In five dimensions, you need to define a new variable, say *V*, and modify the program as was done for four dimensions in **PROG18**. The program has been written to make it relatively easy to extend it to five or even higher dimensions. Be forewarned that the calculation will be very slow. You will almost certainly want to set the coefficients of the constant terms to zero and probably restrict your search to quadratic maps. The number of fifth-dimension polynomial coefficients for order *O* is $(O + 1)(O + 2)(O + 3)(O + 4)(O + 5) / 24$. With $O = 5$, the number is 1260.

The simplest display technique is to project the fifth dimension onto the other four. This is what the program does automatically if you don't do anything special. Several combinations of techniques, which we have already developed, are capable of displaying five dimensions. You might try combining shadows, bands,

and color, for example. Table 5-2 lists the seven possible combinations of five-dimensional display techniques that don't lead to visual contradictions.

Table 5-2. Combinations of display techniques that can be used in five dimensions

Shadow	Bands	Color
Shadow	Bands	Slices
Shadow	Color	Slices
Bands	Color	Stereo
Bands	Anaglyph	Slices
Bands	Stereo	Slices
Color	Stereo	Slices

For a heroic exercise in programming, visualization, and patience, you can try to extend the calculation to six dimensions. A six-dimensional, fifth-order system of polynomials has 2772 coefficients. There are only two appropriate combinations of display techniques suitable for six dimensions: shadow-bands-color-slices and bands-color-stereo-slices. If you decide to try seven dimensions, you must invent a new display technique.

5.4 Writing on the Wall

Since four-dimensional attractors have the greatest complexity and variety of all the cases described in this book, they offer the greatest potential as display art. For such purposes, you will probably want to print them on a large sheet of paper. With an appropriate printer or plotter, any of the visualization techniques previously described can be used to produce such large prints.

An alternate technique that has proved very successful is an extension of the character-based method described in Section 4.5. In this technique, the third dimension is coded as an ASCII character with a density related to the *Z* value, and the fourth dimension is coded in color. Color pen and pencil plotters and ink-jet plotters, as well as more expensive but high-quality electrostatic and thermal plotters, normally used for engineering and architectural drawings, can print text on sheets up to 36 inches wide. Ink-jet plotters are growing in popularity over the more

traditional pen plotters because they are faster and quieter and don't require special paper. They can also print gray scales. With care, you can piece together smaller segments printed by more conventional means.

When the attractors are reduced to sequences of text, resolutions of 640 by 480 (VGA) or 800 by 600 (Super VGA) produce large figures whose individual characters can be read when examined closely but that blend into continuous contours when viewed from a distance. Artists often use this technique in which the viewer is provided with a different visual experience on different scales. You should use the largest and boldest characters available to maximize the contrast, provided they remain readable. There should be little or no space between rows and columns of characters. With a pen plotter, the pen size can be chosen for the best compromise of contrast and readability. A pen that makes a line width of 0.35 mm (fine) is a reasonable choice.

Inks are available in only a limited number of colors, and pen plotters are usually capable of accommodating only a small number of pens. The pens can be sequenced to place compatible colors next to one another. With eight pens and commonly available inks, a good sequence is magenta, red, orange (or yellow), brown, black, green, turquoise, and blue. The closest color sequence for viewing on the computer screen from Table 4-1 is 13, 12, 4 (or 14), 6, 8, 2, 3, and 9, with a white (15) background. With upwards of 20 characters producing different color intensities, the limitation of eight colors of ink is not a serious one. With eight colors and ASCII codes from 32 to 255, you can have 28 different intensities for each color. The inks can be mixed to produce different shades of the colors. Pencils are less expensive and don't clog or dry out as pens often do, but pencil plots have a tendency to smudge. Ink, of course, also smudges until it is thoroughly dry. Plotters are relatively slow, and attractors produced by this method typically require a few hours to a full day to produce.

Paper commonly used for engineering drawings comes in at least five standard sizes—A (8 1/2 by 11 inches), B (12 by 18 inches), C (18 by 24 inches), D (24 by 36 inches), and E (36 by 48 inches). English sizes and architectural sizes are slightly different, and thus a sheet may vary somewhat from these dimensions. Also, 36-inch-wide paper is available on long rolls.

Common paper types are tracing *bond*, which is the most economical, *vellum*, which is smooth and translucent, and *polyester film*, which is highly translucent, dimensionally stable, and relatively expensive. The translucent papers offer the interesting possibility of backing the print with a monochrome or color copy of itself to enhance the contrast or to produce a shadow effect if the two are displaced slightly. Other interesting effects can be achieved by backing one

translucent attractor with a print of another or by back-lighting the print. Some papers stretch slightly and thus have a tendency to wrinkle. Paper with significant acid content should be avoided because it turns yellow and becomes brittle with age.

Some of the most artistic examples of strange attractors have been produced by these techniques, but they cannot be adequately illustrated in this book. No computer program is offered, since it is so dependent on your hardware. You will want to experiment to find the technique that works best for you and that makes the most effective use of your printer or plotter.

5.5 Murals and Movies

The technique of making large-scale attractors for display can be carried to its logical extreme by making a mural. Special techniques using some type of stencil are required to transform the computer output to paint on the wall. Silk screen is useful for transferring the image to fabrics. Fractal tee-shirts employing this technique have recently become popular.

To produce a mural, you need to start with a large number of plots, each showing a small section of the attractor. A property of fractals is that they have detail on all scales, and thus a large mural should look interesting when viewed either from a distance or close up.

You might also photograph the computer screen or a high-quality print and produce slides that can be projected onto a large surface or screen with a slide projector. Equipment is available commercially for producing slides directly from digital computer output. A sequence of such slides makes a very compelling presentation or visual accompaniment to a lecture or musical production.

The color slices shown in Plate 22 suggest the possibility of making color movies by extending the technique to a very large number of slices and using each one as a frame of a movie. The effect is to cause the attractor to emerge at a point in an empty field and to grow slowly, bending and wiggling until fully developed, and then to disappear slowly into a different point. If the technology for doing this is not available to you, try printing a large number of attractor slices on small cards and fanning through them to produce a semblance of animation. This technique, using the attractors described in Section 7.6, was used to produce the animation in the upper-right corner of the odd pages of this book.

If the idea of making strange-attractor movies appeals to you, another technique is to take one of your favorite attractors and slowly change one or more of the coefficients in successive frames of the movie. A good way to start is to multiply all the coefficients by a factor that varies from slightly less than 1.0 to slightly greater than 1.0. You must determine the range over which the coefficients can be changed without the solutions becoming unbounded or nonchaotic. The ends of this range then become the beginning and end of the movie.

Sometimes the attractor slowly and continuously alters its shape. The changes can involve bifurcations, such as the period-doubling sequence in the logistic equation described in Chapter 1. Such bifurcations are called *subtle*. At other times, the attractor and its basin abruptly disappear at a critical value of the control parameter. Such *discontinuous bifurcations* are called *catastrophes*.

If the control parameter is changed in the opposite direction, the result may be different from simply running the movie backward. This is an example of *hysteresis*, which is a form of memory in a dynamical system. It serves to limit the occurrence of catastrophes. The thermostat that controls your heat probably uses hysteresis to keep the furnace from cycling on and off too frequently. Catastrophic bifurcations usually exhibit hysteresis, whereas subtle bifurcations do not.

These four-dimensional maps are also well suited for color holographic display or for experimentation with virtual reality, in which the view is controlled by the motion of your head and hands to give the sensation of moving through the object. The technology is complicated, but the results are visually and mentally stimulating.

5.6 Search and Destroy

If you have worked carefully through the text, your program has created a disk file **SA.DIC** containing the codes of all the attractors generated since you ran the **PROG11** program. We now develop the capability to examine these attractors and save the interesting ones in a file **FAVORITE.DIC**, while discarding the others. This feature allows you to run the program overnight and collect attractors for rapid viewing the next day. This capability is especially useful if you have a slow computer. The required program changes are shown in **PROG20**.

PROG20. Changes required in PROG19 to evaluate the attractors in SA.DIC and save the best of

them in FAVORITE.DIC

```
1000 REM FOUR-D MAP SEARCH (With Search and Destroy)


1380 IF QM% <> 2 THEN GOTO 1420
1390    NE = 0: CLOSE
1400    OPEN "SA.DIC" FOR APPEND AS #1: CLOSE
1410    OPEN "SA.DIC" FOR INPUT AS #1


2420 IF QM% = 2 THEN GOTO 2490  'Speed up evaluation mode


2610 IF QM% <> 2 THEN GOTO 2640 'Not in evaluate mode
2620    IF EOF(1) THEN QM% = 0: GOSUB 6000: GOTO 2640
2630    IF EOF(1) = 0 THEN LINE INPUT #1, CODE$: GOSUB 4700: GOSUB 5600


3340 IF QM% <> 2 THEN GOTO 3400         'Not in evaluate mode
3350    LOCATE 1, 1: PRINT "<Space Bar>: Discard       <Enter>: Save";
3370    LOCATE 1, 49: PRINT "<Esc>: Exit";
3380    LOCATE 1, 69: PRINT CINT((LOF(1) - 128 * LOC(1)) / 1024); "K left";
3390    GOTO 3430


3620 IF QM% = 2 THEN GOSUB 5800                 'Process evaluation command
3630 IF INSTR("ADEHIPRSX", Q$) = 0 THEN GOSUB 4200
3710 IF Q$ = "E" THEN T% = 1: QM% = 2
```

```
4220 WHILE Q$ = "" OR INSTR("AEIX", Q$) = 0
4320    PRINT TAB(27); "E: Evaluate attractors"

5800 REM Process evaluation command
5810 IF Q$ = " " THEN T% = 2: NE = NE + 1: CLS
5820 IF Q$ = CHR$(13) THEN T% = 2: NE = NE + 1: CLS : GOSUB 5900
5830 IF Q$ = CHR$(27) THEN CLS : GOSUB 6000: Q$ = " ": QM% = 0: GOTO 5850
5840 IF Q$ <> CHR$(27) AND INSTR("HPRS", Q$) = 0 THEN Q$ = ""
5850 RETURN

5900 REM Save favorite attractors to disk file FAVORITE.DIC
5910 OPEN "FAVORITE.DIC" FOR APPEND AS #2
5920 PRINT #2, CODE$
5930 CLOSE #2
5940 RETURN

6000 REM Update SA.DIC file
6010 LOCATE 11, 9: PRINT "Evaluation complete"
6020 LOCATE 12, 8: PRINT NE; "cases evaluated"
6030 OPEN "SATEMP.DIC" FOR OUTPUT AS #2
6040 IF QM% = 2 THEN PRINT #2, CODE$
6050 WHILE NOT EOF(1): LINE INPUT #1, CODE$: PRINT #2, CODE$: WEND
```

```
6060 CLOSE

6070 KILL "SA.DIC"

6080 NAME "SATEMP.DIC" AS "SA.DIC"

6090 RETURN
```

The program uses the **E** key to enter the evaluation mode. When in this mode, the attractors in **SA.DIC** are displayed one by one. Each case remains on the screen and continues to iterate until you press the spacebar, which deletes it, the **Enter** key, which saves it in the file **FAVORITE.DIC**, the **Esc** key, which exits the evaluation mode, or, in rare cases, until the solution becomes unbounded, whereupon it is deleted. While an attractor is being displayed, you can press the **H**, **R**, **P**, and **S** keys to change the way it is displayed without returning to the menu screen. The upper-right corner of the screen shows the number of kilobytes left to be evaluated in the **SA.DIC** file. When in the evaluation mode, the program bypasses the calculation of the fractal dimension and Lyapunov exponent so that each case is displayed more quickly.

As you begin to accumulate a collection of favorite attractors, you will probably want to go back and find your favorites of the favorites. You merely need to rename the **FAVORITE.DIC** file to **SA.DIC** and evaluate them a second time. The attractors exhibited in this book were selected by this method after looking at about 100,000 cases. Since the **FAVORITE.DIC** file is in ordinary ASCII text, you can share your favorites with a friend who may have a different computer or operating system. You can easily e-mail the file to someone or upload it to a computer bulletin board or mainframe computer. Remember, however, that the programs in this book are copyrighted and are for your personal use. It is a violation of the copyright to share the programs with anyone else. You can now begin your own private collection of strange attractors artwork!

Chapter 6

Fields and Flows

In this chapter, we consider equations whose iterates move gradually rather than abruptly from one place to another. Such equations are called differential equations, and they are the basis for most dynamical systems that describe natural processes. The programming is a simple extension of what we have done before, but the calculation requires more computing time. The attractors produced by differential equations consist of continuous lines whose weavings and waverings describe the trajectory and yield objects of considerable beauty.

6.1 Beam Me *U*p Scotty!

Successive iterates of the maps in the previous chapters are usually at widely different positions on the attractors. The points dance around like fleas jumping on the back of a dog, eventually, but gradually, visiting every allowed location. Most processes in nature don't occur that way but progress slowly and continuously from some initial condition through a succession of nearby intermediate states to the final condition.

If you take a trip across the country, your trajectory through three-dimensional space (or even in four-dimensional space-time) is a continuous one-dimensional curve. Only in science fiction is Captain Kirk able to dematerialize at one position and rematerialize somewhere else, without occupying a succession of intermediate positions. Most substances in nature obey a *continuity equation*, which guarantees that if their quantity decreases at some position, the decrease must be accompanied by a flow of the substance away from the position. Note that this is a stronger condition than a *conservation law*, which requires only that the total quantity of the substance remains the same.

There is a relation between flows and maps. Imagine a fly trapped in a room and moving in a complicated, random manner. Its trajectory is a one-dimensional curve that eventually fills the entire room. However, if you observe the fly with a strobe lamp that flashes periodically, the trajectory is a succession of dots, with each dot separated from the previous dot by a significant distance. The dots also eventually fill the entire three-dimensional region, but it takes longer for this to occur.

However, if the fly's motion is chaotic rather than random, neither the curve

nor the dots fill the room; rather, they lie on a strange attractor that occupies a negligible portion of the room. The attractor consisting of all the possible dots often has a lower dimension than the attractor consisting of all the possible curves. Thus a map can be thought of as a crude description of a flow, in which the intervening details of the motion are ignored.

It's easy to think of an object such as a fly or a human, imbued with intelligence, however limited, moving by free will along a complicated trajectory. However, inanimate objects, such as astronomical bodies or sub-microscopic, electrically charged particles, can also execute complicated motions. They do so because they move through a space filled with gravitational or electromagnetic fields.

It is important to recognize that a field has no objective reality other than to describe mathematically the force on an object moving through it. When something is dropped, it falls toward Earth. It is a deeply philosophical question, not answered very well by science, how the object knows to move toward Earth rather than in some other direction. We say that it is acted upon by the gravitational field of the Earth, but this description, however useful for calculating the motion, begs the issue. Ultimately, the laws of physics describe very accurately how things move, but not very well why.

The equations that describe flows are of a different type than those that describe maps. They are called *differential equations*, and they involve the rate of change of a quantity. We will consider only *ordinary differential equations* (*ODEs*), as distinguished from the *partial differential equations* (*PDEs*) used to describe the behavior of complicated objects like fluids that have intrinsically infinite-dimensional state spaces. Dynamical systems described by ODEs involve only the time rate of change of the position of a point in state space, whereas with PDEs, the variables are quantities like density, temperature, and electric field that change in space as well as time. A wave is an example of a dynamical system described by a PDE.

Consider an object moving in the *X* direction. Its speed is the rate of change of its position, and we will denote this quantity by *X'* (pronounced "*X* prime"). It is the distance the object moves in a brief interval of time divided by the time interval. If you know some calculus, you recognize this as the *time-derivative* of *X*, usually denoted by *dX/dt*. The rate of change of position is what the speedometer on your car, or the police radar, reads. The rate of change of the speed is the *acceleration*. More properly, we should call these quantities the time rate of change, since quantities can also change in space. For example, the spatial rate of change in altitude of a road is called its *grade*.

An object moving in three-dimensional space has a constantly changing value not only of *X* but also of *Y* and *Z*. Furthermore, *X'*, *Y'*, and *Z'* usually depend on position (*X*, *Y*, and *Z*). For example, a particle moving clockwise in a circle about the origin in the *XY* plane is described by the following pair of differential equations:

$$X' = Y$$

$$Y' = -X \qquad \text{(Equation 6A)}$$

Such a set of equations describes, at least approximately, the motion of the earth around the sun. This type of regular motion is not chaotic, and it does not lead to visually interesting strange attractors.

Some differential equations can be solved easily using calculus. For example, Equation 6A has the solution

$$X = A \sin(t + \_)$$

$$Y = A \cos(t + \_) \qquad \text{(Equation 6B)}$$

which specifies the *X* and *Y* positions at any time *t*. The quantities *A* and φ are constants that are determined from the initial conditions (the values of *X* and *Y* at *t* = 0). If you are interested only in the shape of the trajectory, and not in where the object is along it at any particular time, you can eliminate the *t* in Equations 6B to get a relation between *X* and *Y*,

$$X^2 + Y^2 = A^2 \qquad \text{(Equation 6C)}$$

which is the equation for a circle of radius A centered on the origin (*X* = *Y* = 0).

Equation 6A also arises in a different context. Imagine an object moving back and forth in the *X* direction, perhaps attached to a spring that alternately stretches and compresses. Since *Y* is equal to *X'*, we can associate *Y* with the velocity in the *X* direction. The *XY* plane then becomes the two-dimensional phase space for this one-dimensional motion, and the trajectory in this plane is the phase-space trajectory. A circular phase-space trajectory is a characteristic of a one-dimensional, *simple harmonic oscillator*, such as a mass on a spring. Usually the phase-space trajectory is an ellipse, just as the orbit of the earth around the sun is an ellipse, but we can always measure *Y* in appropriate units, or adjust the scale of the graph, to change the ellipse into a circle.

With this interpretation, the first part of Equation 6A defines the velocity (*Y*) as

the rate of change of position (X'). If you remember your physics, the second part of Equation 6A is Newton's second law ($F = ma$), in which the force F obeys *Hooke's law* for springs ($F = -kX$), and the acceleration a is the rate of change of velocity (Y'). It is interesting that the same set of differential equations with a change in the meaning of the variables can describe the motion of an object traveling in a circle or an object oscillating on the end of a spring. Equation 6A describes many other phenomena in nature, such as the oscillations in an electrical circuit containing a capacitor and inductor.

A two-dimensional system of differential equations such as Equation 6A cannot exhibit chaos, according to the *Poincaré-Bendixon* theorem, because the trajectory cannot cross itself. The most complicated bounded behavior is thus a simple closed loop, corresponding to periodic motion. The reason the trajectory cannot cross itself is that every point in the XY plane has associated with it a unique direction of flow, so the trajectory must approach and leave every point in a single particular direction. If the orbit were to return to a point previously visited, it would thereafter repeat what it did before. In two dimensions, the orbit can do only one of three things: spiral into a fixed point, approach a stable limit cycle, or spiral off to infinity.

Trajectories may appear to cross if they come very close to a fixed point that is stable in one direction and unstable in another (called a *saddle point* or *X point* because of its shape). Such a trajectory is called a *separatrix* because it separates regions with different flows. Trajectories approaching the fixed point on one side of the separatrix veer off to the right, and those approaching from the other side veer off to the left. Such a separatrix exists upstream (and downstream) of an island in a river where two sticks placed side by side in the water end up going around opposite sides of the island. The island seems at first to attract the sticks and then to repel them at right angles as they approach it.

In three dimensions, we have the possibility of an orbit wrapping around in a complicated manner, like a ball of string, never intersecting itself, but producing a never-ending tangle. By contrast, maps can be chaotic in one or two dimensions because the points jump from place to place with little danger of intersecting another point. Captain Kirk need not be concerned about a collision while being transported from one point to another. He only needs to worry about landing on top of a diabolical Romulan at his destination!

6.2 Professor Lorenz and Dr. Rössler

Although differential equations have been the mathematical basis for most descriptions of nature for hundreds of years, almost no one suspected that the trajectories resulting from their solution could be a chaotic strange attractor. The history of the discovery of such solutions is interesting and bears retelling.

In the early 1960s, Edward Lorenz, a meteorologist at the Massachusetts Institute of Technology, was developing models of atmospheric convection to be solved by a primitive computer that required about one second per iteration. His models involved a large number of differential equations and produced solutions that varied with time in a complicated manner, not unlike the variation of the weather over long intervals of time. On one occasion, he happened to restart one of his computer runs using numbers rounded to three digits rather than the six significant figures used by the computer.

For some time, the solutions followed one another, but after a while they began to depart, and eventually they bore no relation to one another. He had discovered the sensitivity to initial conditions that is perhaps the most salient feature of chaos. He began simplifying his equations in an attempt to determine the minimum conditions necessary for this bizarre behavior. The result is the now famous *Lorenz equations*, which represent the first example of a strange attractor arising from differential equations,

$$X' = \_(Y - X)$$

$$Y' = -XZ + rX - Y$$

$$Z' = XY - bZ \qquad \text{(Equation 6D)}$$

where σ, r, and b are constants that Lorenz took to be $\sigma = 10$, $r = 28$, and $b = 8/3$. Lorenz published his findings in 1963 in the *Journal of the Atmospheric Sciences*, where they went largely unnoticed for the next decade. The title of his paper, "Deterministic Nonperiodic Flow," is an apt description of what we now call chaos.

Although the Lorenz equations were distilled from a model of atmospheric convection, the trajectory in *XYZ* space does not represent air currents in any literal way. Instead, *X* corresponds to the size of the convective motion, *Y* is proportional to the temperature difference between the ascending and descending fluids, and *Z* is proportional to the deviation of the vertical temperature profile from a linear function. Nevertheless, the behavior is reminiscent of a fluid with turbulent convection.

Since the Lorenz equations were proposed, several phenomena have been found that are at least approximately modeled by them. Perhaps the simplest example is the thermosiphon. Imagine a continuous tube, like a bicycle tube, filled with a liquid and mounted vertically. If the bottom of the tube is heated and the top cooled, a convection ensues, with the warm fluid rising and the cold fluid falling. The convection is equally likely to start in either direction. After it starts, the circulation continues in that direction a few times around the loop and then abruptly reverses.

In the 1970s other examples of chaotic differential equations began to be discovered. An important contribution was made in 1976 by Otto Rössler, a nonpracticing medical doctor in Germany. Rössler was interested in chaos in chemistry and theoretical biology, and he set about to find a system of equations even simpler than those of Lorenz that exhibited chaotic behavior. What he came up with are the now famous *Rössler equations*:

$$X' = -(Y + Z)$$

$$Y' = X + aY$$

$$Z' = b + Z(X - c) \qquad \text{(Equation 6E)}$$

where a, b, and c are constants that Rössler took to be a = 0.2, b = 0.2, and c = 5.7. The Rössler equations are sometimes described as the simplest known example of chaos arising from a system of ordinary differential equations. They contain a single nonlinearity (*ZX* in the third equation). Rössler's original paper is also interesting because it contains a stereoscopic view of his strange attractor as well as the Lorenz attractor.

Until very recently, the discovery of a new strange attractor was a cause to rush to publication. With the program in this book, you can produce them by the thousands! Even today researchers tend to focus on a few well-known examples such as the Lorenz and Rössler attractors. An entire book has been written on the Lorenz attractor alone. Think of the libraries that could be filled by books describing your attractors in similar detail!

The Lorenz and Rössler attractors are shown in Figures 6-1 and 6-2, respectively, albeit with slightly different values of the parameters than they used. These cases are known to have fractal dimensions slightly greater than 2.0. These examples are more important for their historical interest than for their visual appeal. If you have never seen these attractors in 3-D, be sure to return to these cases and view them with the various display techniques after the program has been appropriately modified, as described in the next section. The Lorenz attractor resembles

the wings of a butterfly, making it an appropriate emblem of chaos, since the sensitivity to initial conditions is most dramatically illustrated by the butterfly effect.

Figure 6-1. The Lorenz attractor

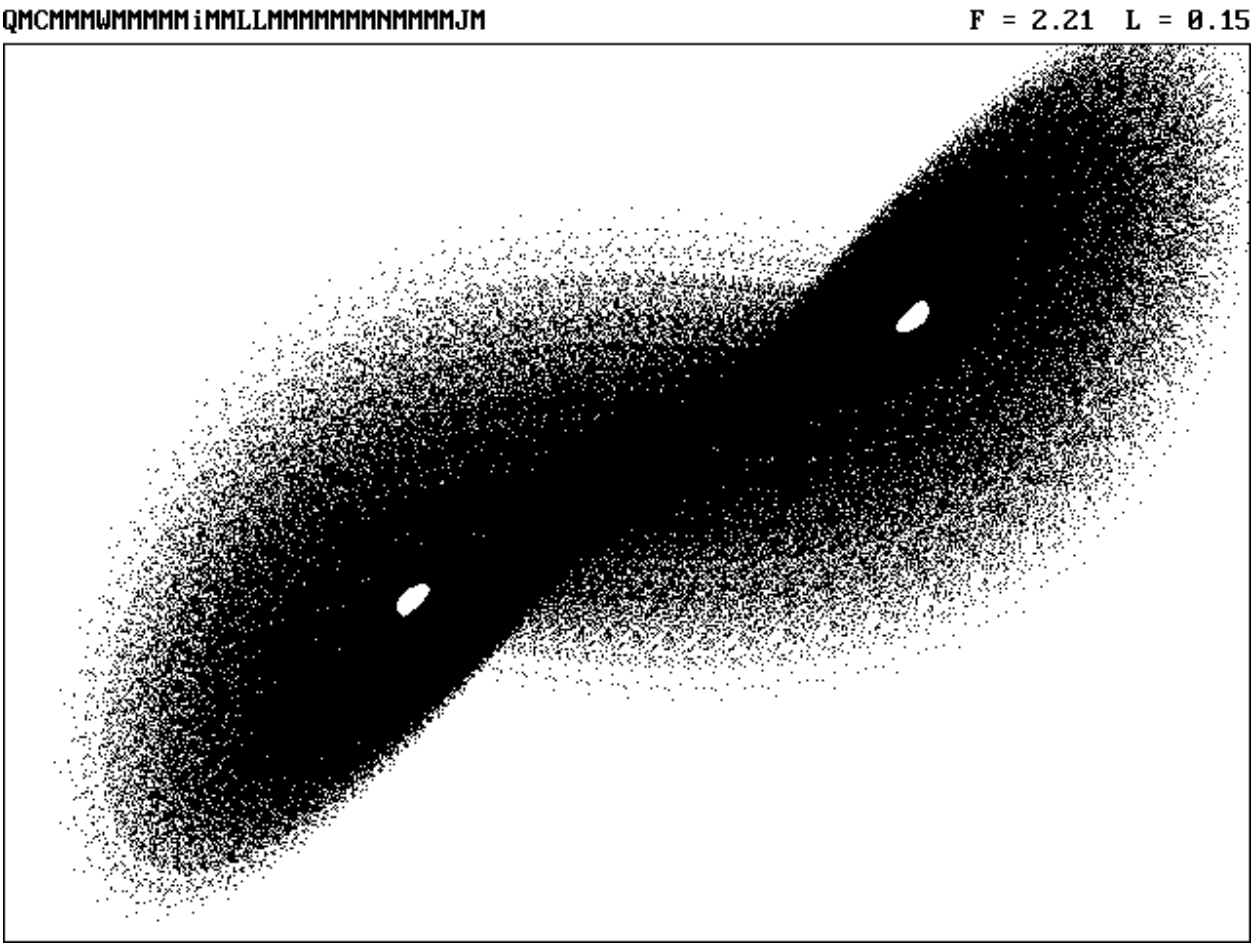
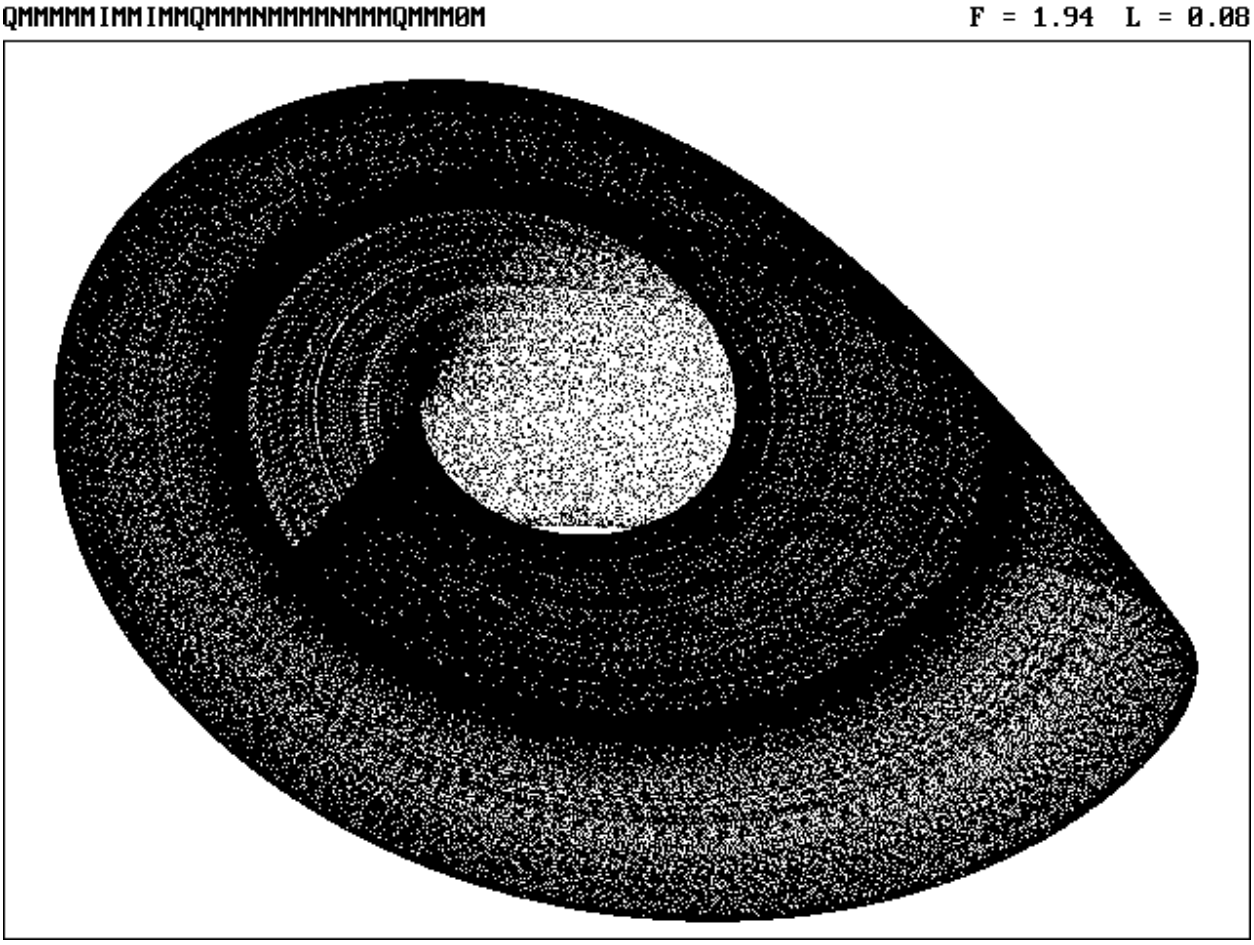


Figure 6-2. The Rössler attractor



6.3 Finite Differences

Some differential equations, such as Equation 6A, can be solved exactly in a straightforward manner using calculus. However, if a system of equations exhibits chaos, no such solution is possible. The reason is that no mathematical function analogous to the sine and cosine can describe a strange attractor the way those functions describe a circle. The equations must be solved by computer. We say that such solutions are numerical as opposed to analytical.

Unfortunately, digital computers, which are ideal for iterating maps, are inherently incapable of exactly solving differential equations. The equations require that the solution advance slowly and smoothly. That is, the successive iterates must differ by an infinitesimal amount, thus infinitely many iterations are required to make any progress. Special analog computers have been designed for the task, but they

are not common or simple to program.

Books have been written on methods for the approximate numerical solution of differential equations, and it is as much an art as a science. All the methods involve, in one form or another, a *finite-difference approximation* to the differential equation. Rather than taking infinitesimal steps, one advances in finite steps according to a prescription that attempts to minimize the inevitable errors. Fortunately, for our purpose, our solutions need not be highly accurate, so we can use a simple procedure.

Perhaps the easiest and most transparent method for finding approximate solutions to differential equations is the *Euler method*. When this procedure is applied to the simple example of Equation 6A, *X* and *Y* are advanced according to

$$X_{n+1} = X_n + \varepsilon Y_n$$

$$Y_{n+1} = Y_n - \varepsilon X_n \qquad \text{(Equation 6F)}$$

where ε is the time step that ideally should be negligibly small but in reality is made as large as possible to reduce the number of iterations required to advance the solution by a substantial distance along the trajectory. You see that the Euler method provides just another example of an iterated map in which successive iterates are near one another. It is perhaps the least accurate method for solving differential equations, and it is easily improved upon. However, for most of our purposes, the Euler method is adequate. Furthermore, it is simple to modify the program to solve differential equations by this method. The necessary changes are shown in **PROG21**.

PROG21. Changes required in PROG20 to solve differential equations by the Euler method

```
1000 REM ODE SEARCH

1070 D% = 3                    'Dimension of system

1080 EPS = .1                  'Step size for ODE

1090 ODE% = 1                  'System is ODE
```

```
1990 IF ODE% = 1 THEN XN(I%) = XY(I%) + EPS * XN(I%)


2660    CODE$ = CHR$(59 + 4 * D% + O% + 8 * ODE%)


3050    IF ODE% = 1 THEN L = L / EPS


3660 IF ODE% = 1 THEN D% = D% + 2
3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 6): T% = 1
3700 IF D% > 4 THEN ODE% = 1: D% = D% - 2 ELSE ODE% = 0


4300    PRINT TAB(27); "D: System is"; STR$(D%); "-D polynomial ";
4310    IF ODE% = 1 THEN PRINT "ODE" ELSE PRINT "map"


4730 IF D% > 4 THEN D% = D% - 2: ODE% = 1 ELSE ODE% = 0
```

In **PROG21**, the value of ε is 0.1, and the three-dimensional equations are polynomials up to fifth order, with coefficients chosen by analogy to the three-dimensional polynomial maps previously described. We don't consider differential equations in less than three dimensions because they cannot have chaotic solutions. The second-order through fifth-order equations are coded with the first letters Q, R, S, and T, respectively.

If ε is sufficiently small, its value should not affect whether a system is chaotic or the general appearance of the attractor, but it certainly changes the trajectory on the attractor. Just as a chaotic trajectory is sensitive to initial conditions, it also is sensitive to the approximations used to calculate it. Unfortunately, a value of ε = 0.1 is not sufficiently small, and many of the resulting attractors would disappear or change their appearance if ε were reduced. Conversely, other attractors would emerge for smaller values of e. Fortunately, for our purposes, the solutions need not be even qualitatively correct. Be forewarned that reducing e has unpredictable effects on the attractors and increases the computation time.

The Lyapunov exponent is calculated as with the corresponding maps, except that it is divided by e; thus its units are bits per second rather than bits per iteration, because each iteration advances the solution by e seconds. It is customary to express the Lyapunov exponent in this way for differential equations because the step size depends on the numerical approximation that is being used, whereas the divergence of the trajectories per unit time is an intrinsic property of the differential equations.

Sample attractors produced by three-dimensional ordinary differential equations projected onto the *XY* plane are shown in Figures 6-3 through 6-6.

Figure 6-3. Projection of three-dimensional quadratic ODE

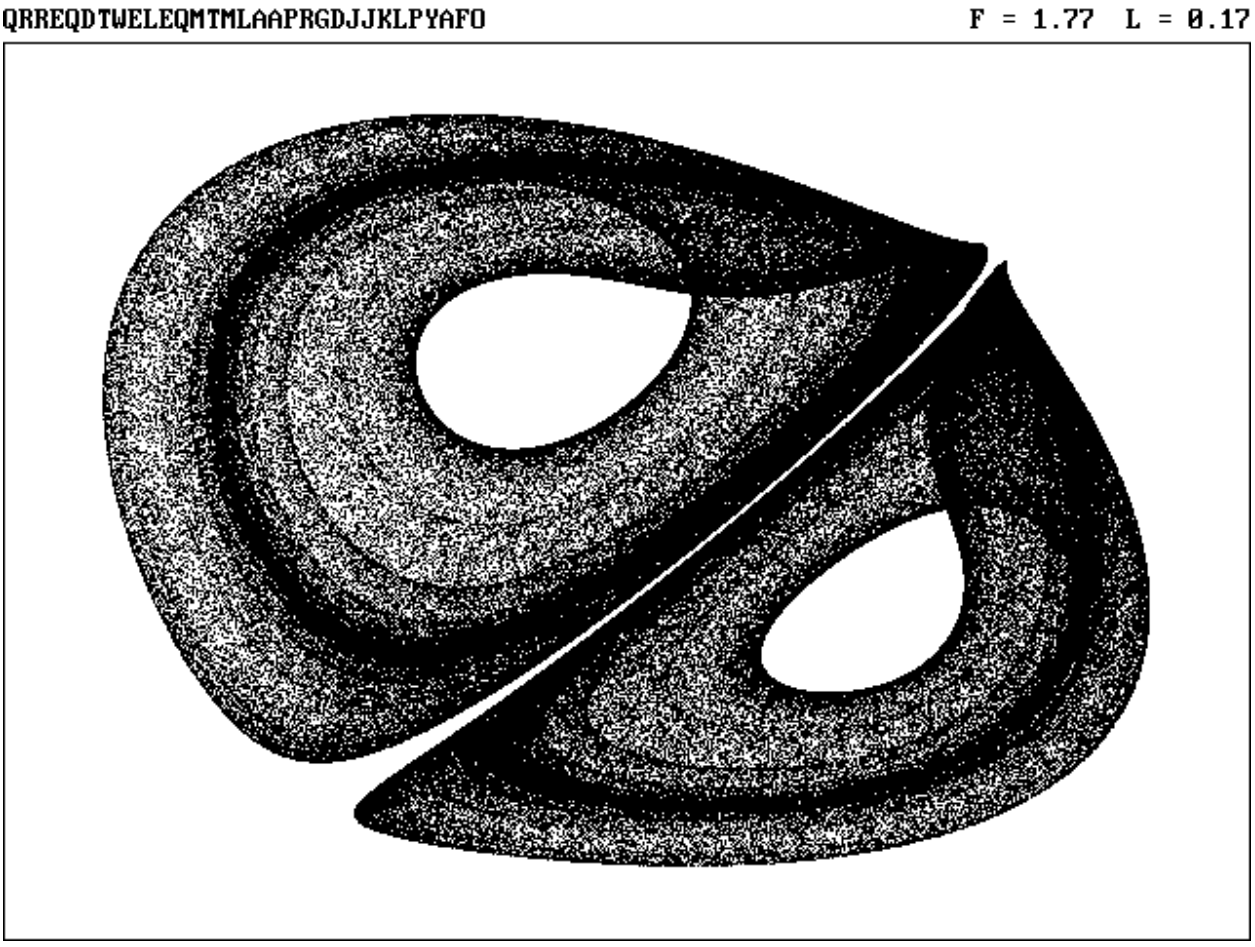


Figure 6-4. Projection of three-dimensional cubic ODE

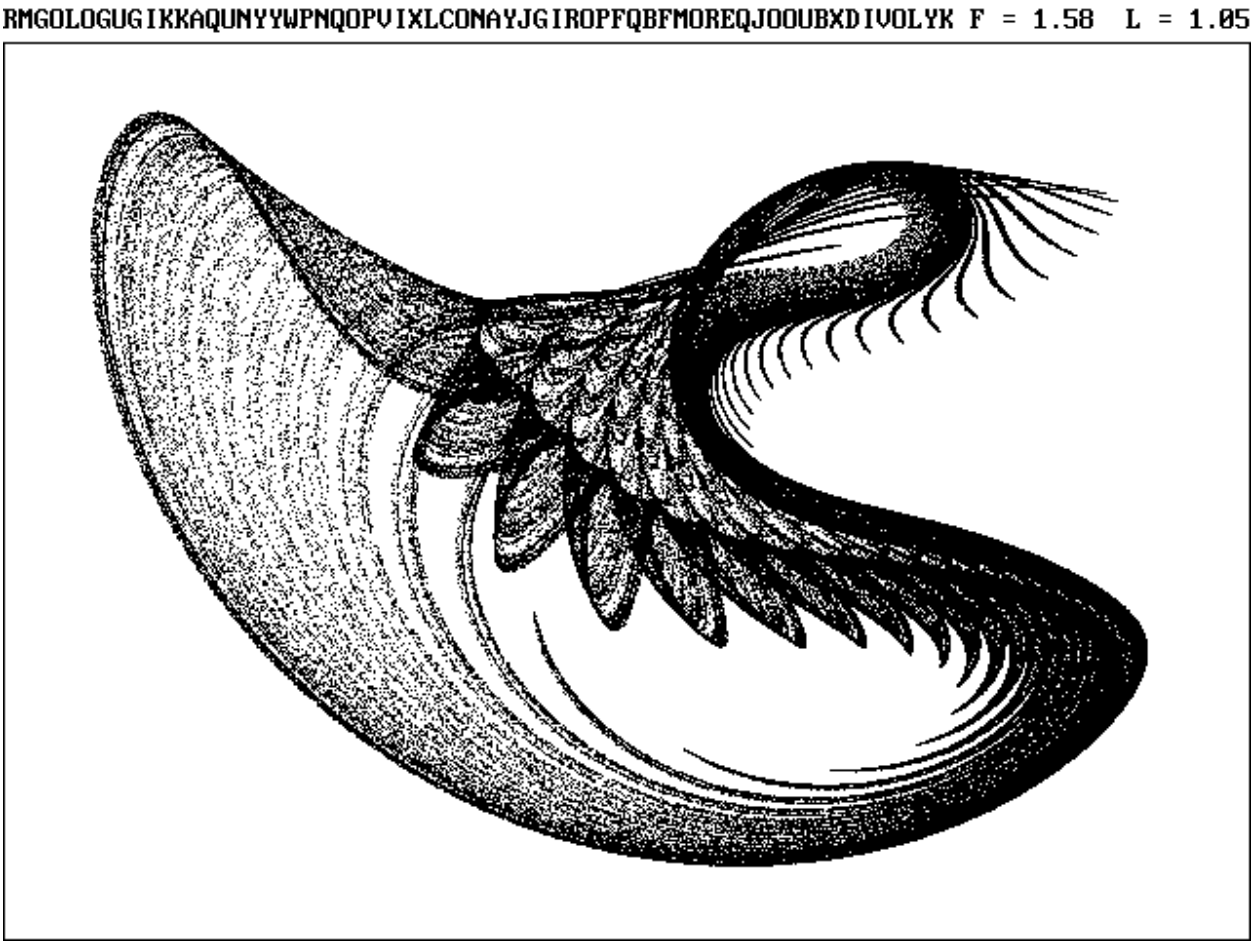


Figure 6-5. Projection of three-dimensional quartic ODE

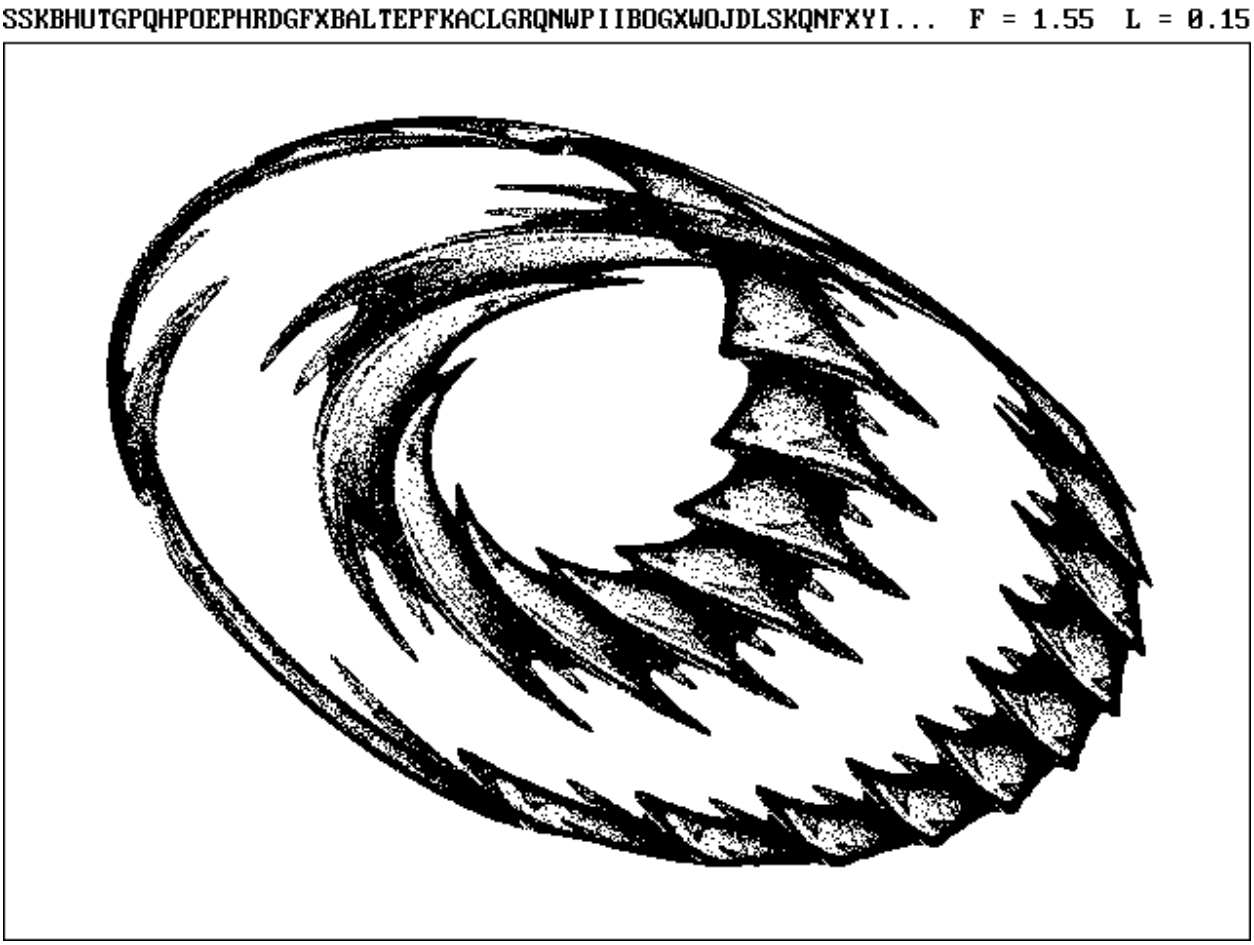
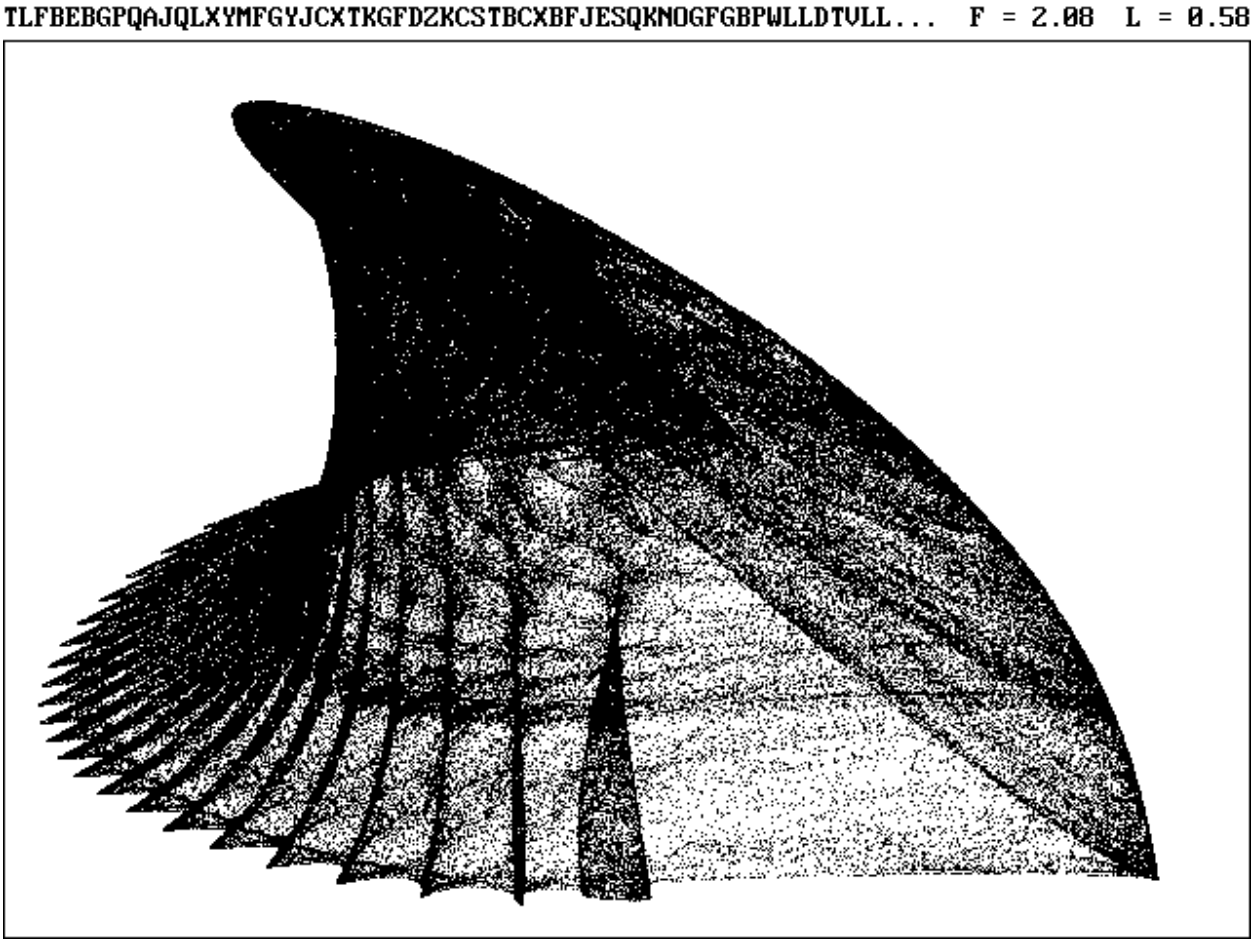


Figure 6-6. Projection of three-dimensional quintic ODE



These figures are like time-exposed photographs of the shadow on the wall of a fly moving chaotically in a room. However, because of the finite difference approximation used to solve the equation of motion, you must imagine that the fly is illuminated by a strobe lamp that flashes rapidly. The trajectory thus consists of a large number of closely spaced dots. The separation of the dots provides a measure of the accuracy of the solution. Although some of the cases produce apparently continuous trajectories, others more nearly resemble the maps of the previous chapters. You might prefer to alter the program so that the dots are connected by lines. This is most easily done by changing line 5060 to

```
5060 IFTRD% = Ø THEN IFODE% = 1 THEN

LINE-(XP, YP), C4% ELSEPSET(XP, YP), C4%
```

Another consequence of dealing with differential equations is that many

criterion for the number of iterations as we did for the maps, a significant number (perhaps 20%) of the attractors found in a random search are not chaotic, and a few are even unbounded. When you evaluate the attractors found by the search, you will recognize these cases by the way they eventually settle onto a simple closed loop that is visually indistinguishable from a limit cycle, spiral into a fixed point, or leave the screen. You will also notice a few cases that consist of isolated islands with no bridge connecting them, such as the one in Figure 6-3. You can be sure these are not true flows, because such discontinuities are impossible in the trajectories that arise from the solution of our differential equations.

There is no reason to limit the display of attractors arising from differential equations to projections onto a plane. All the display techniques developed in Chapter 4 for three-dimensional maps are also available here. Figures 6-7 through 6-22 and Plates 23 and 24 show a selection of such examples.

Figure 6-7. Three-dimensional quadratic ODE with shadows

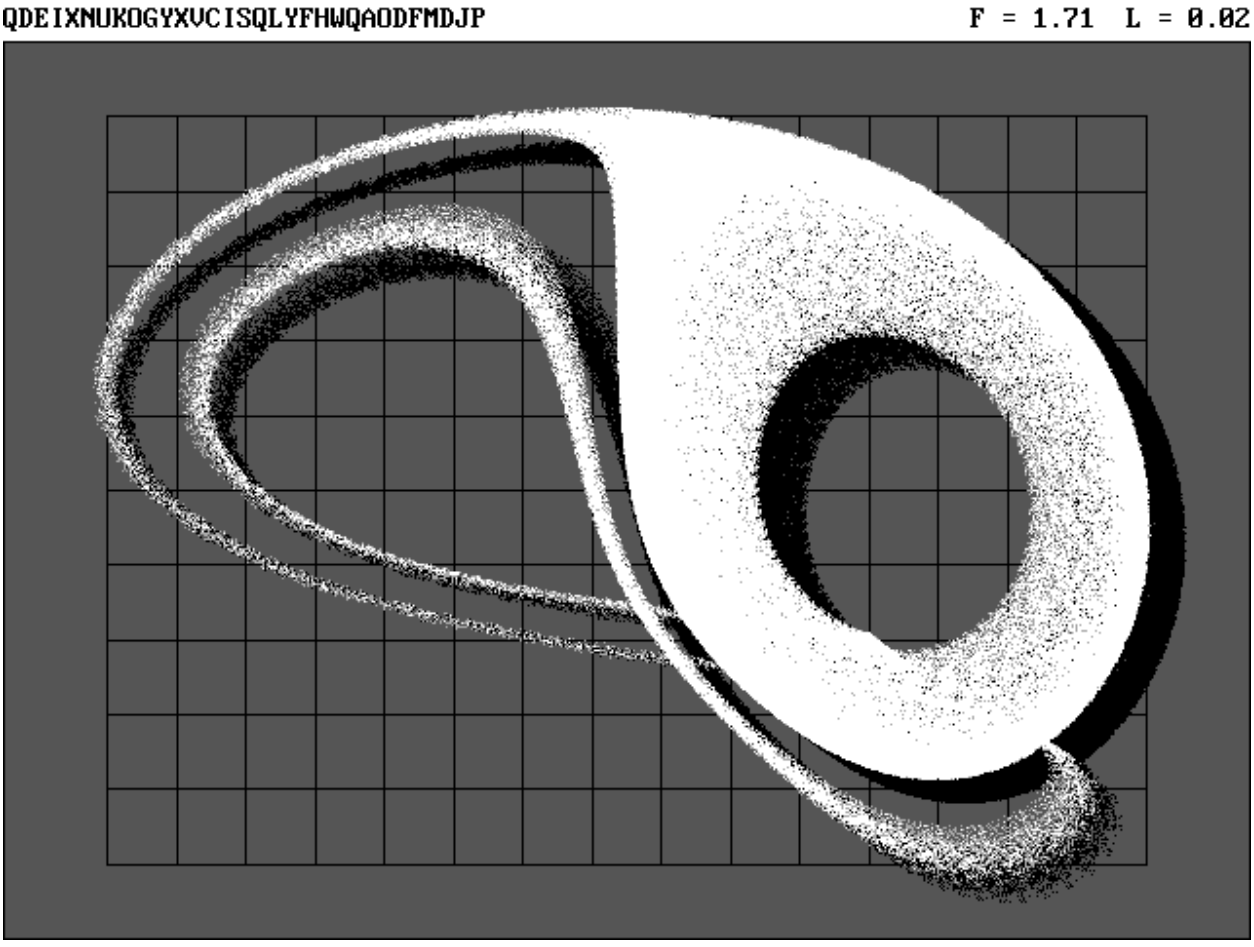


Figure 6-8. Three-dimensional cubic ODE with shadows

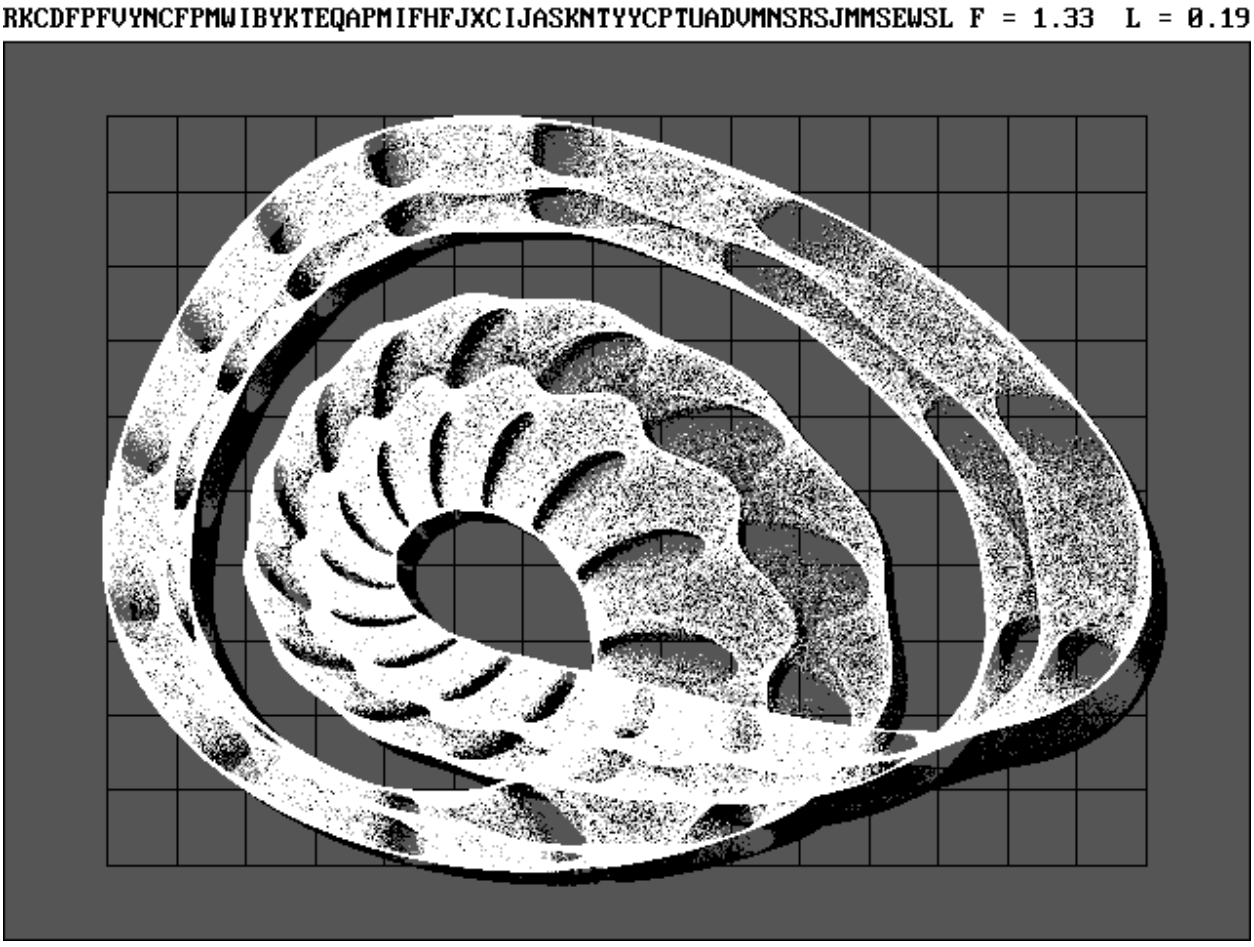


Figure 6-9. Three-dimensional quartic ODE with shadows

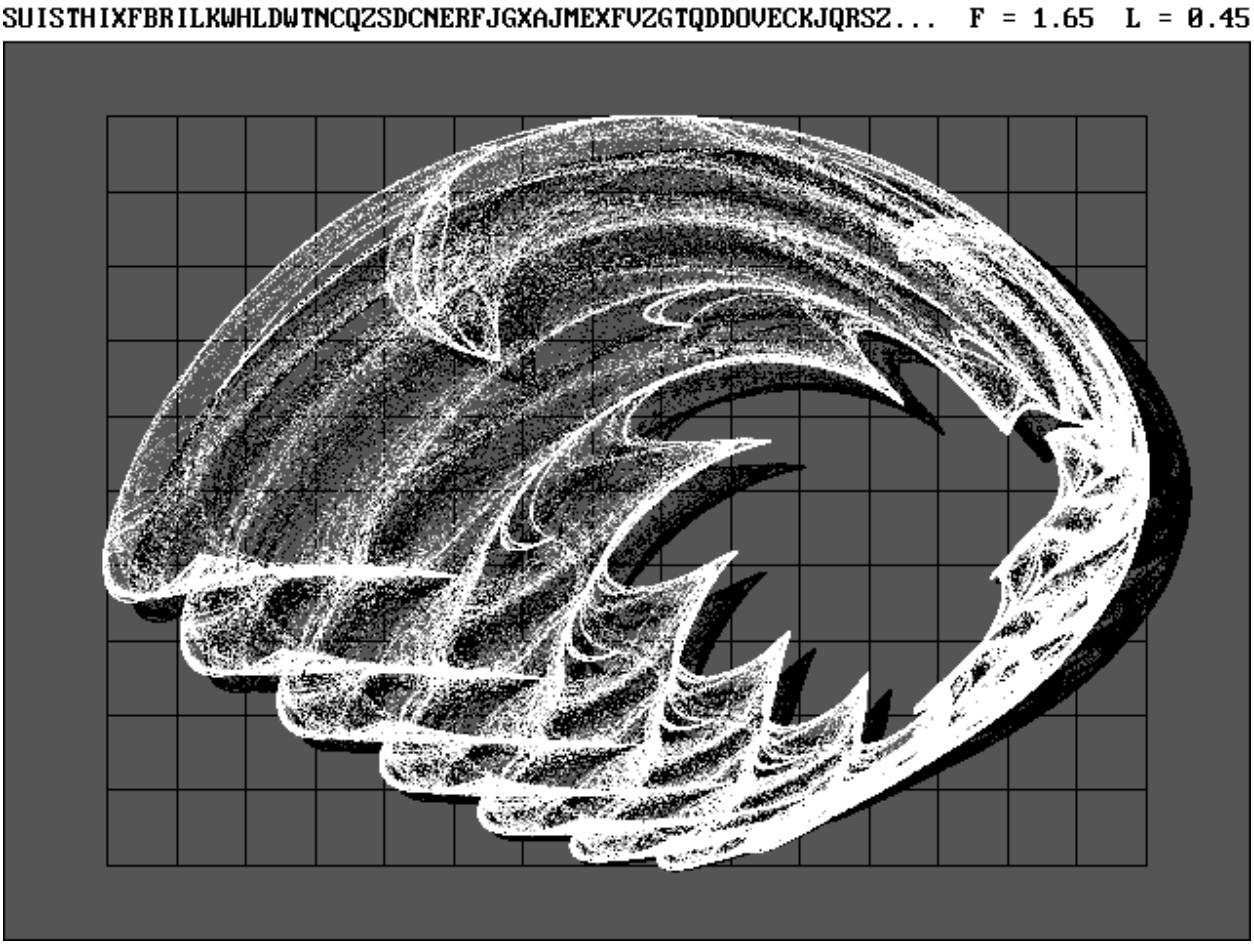


Figure 6-10. Three-dimensional quintic ODE with shadows

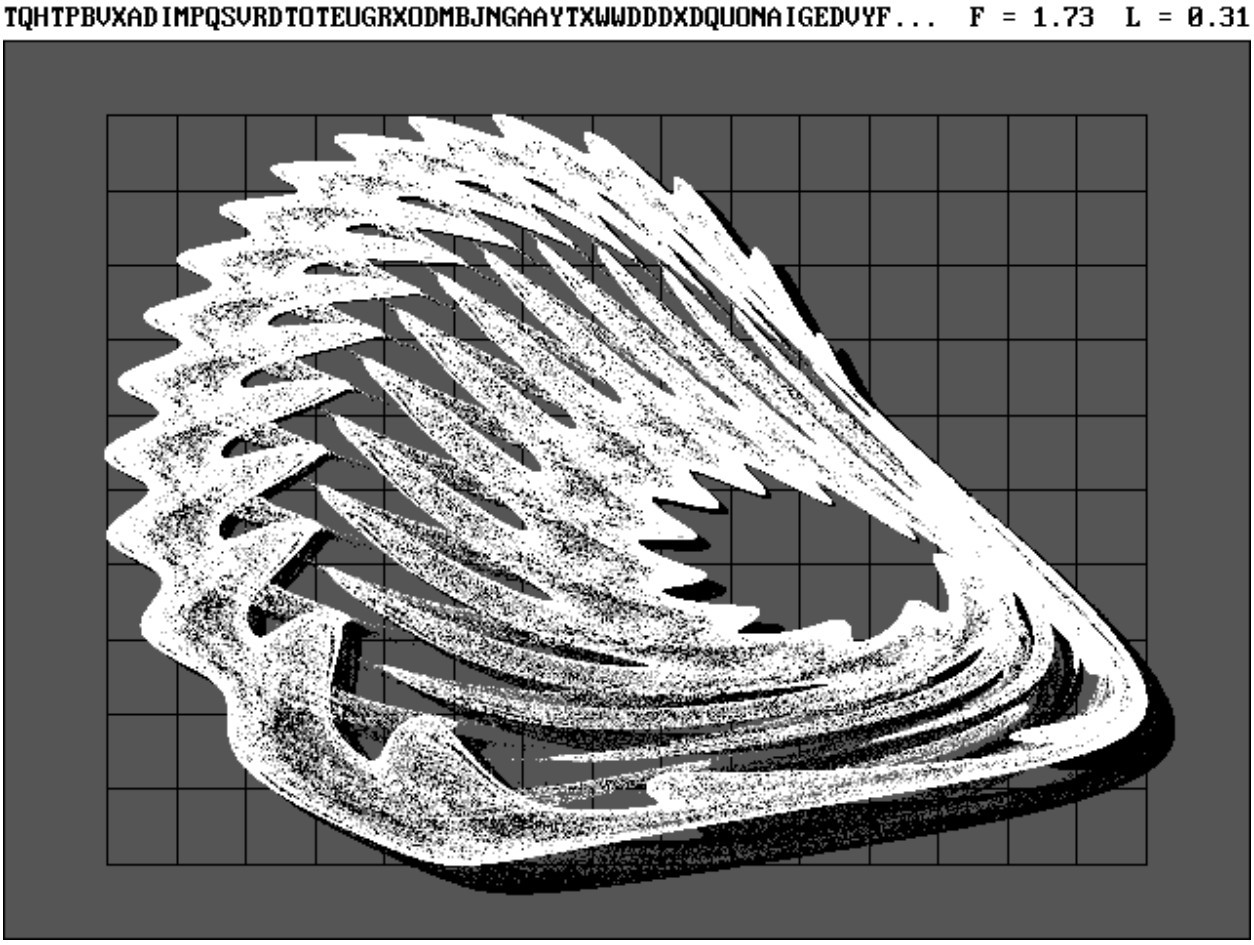


Figure 6-11. Three-dimensional quadratic ODE with contour bands

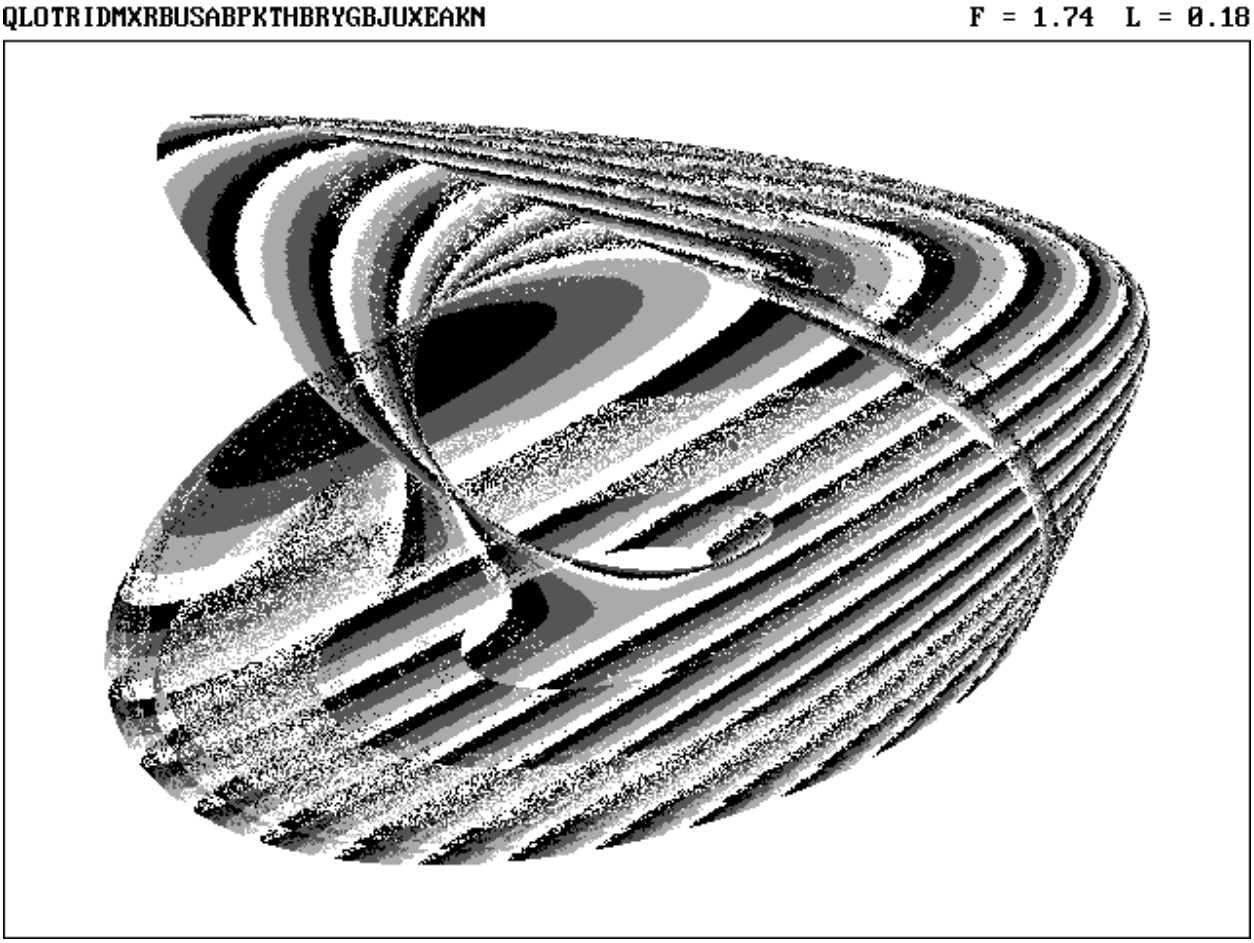


Figure 6-12. Three-dimensional cubic ODE with contour bands

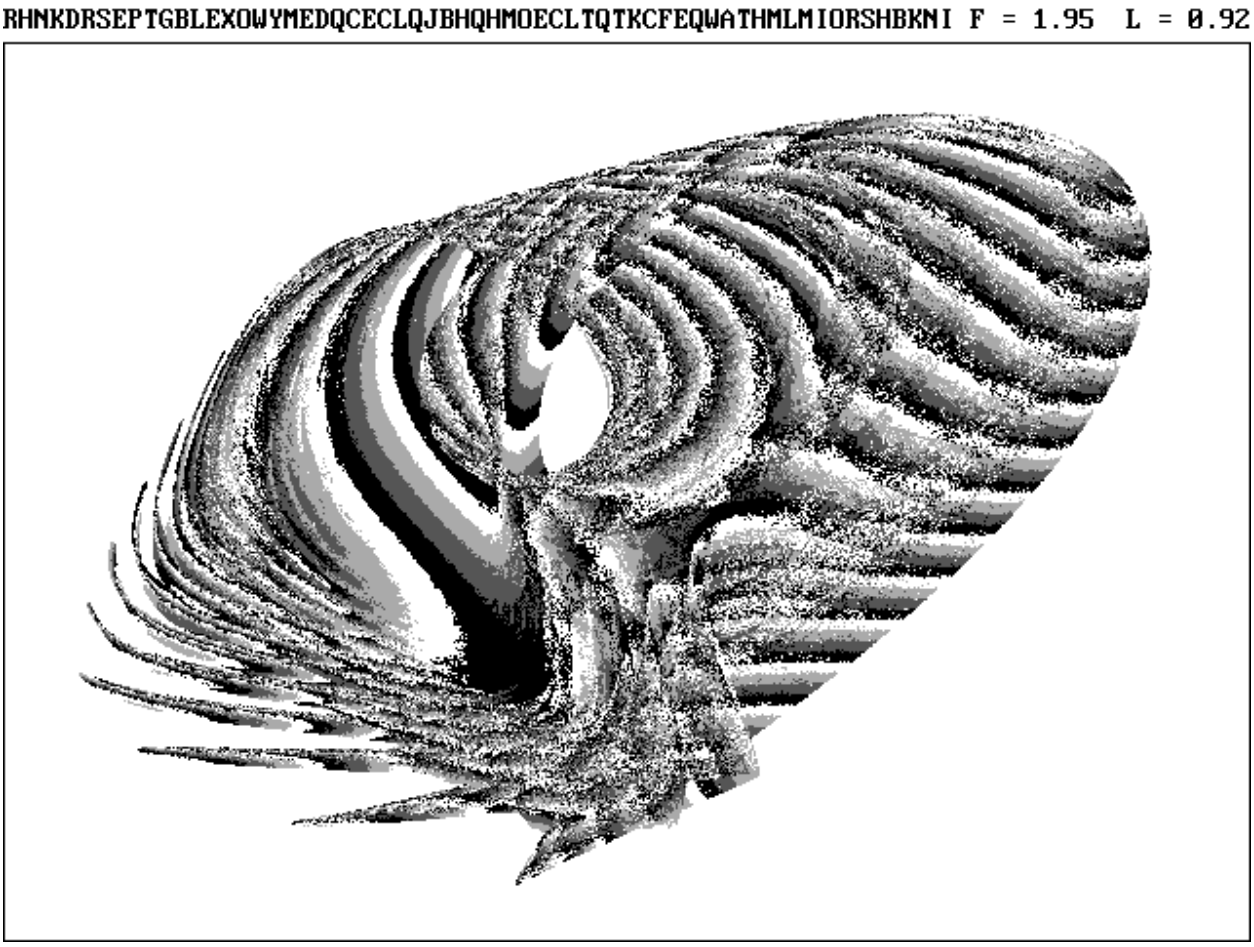


Figure 6-13. Three-dimensional quartic ODE with contour bands

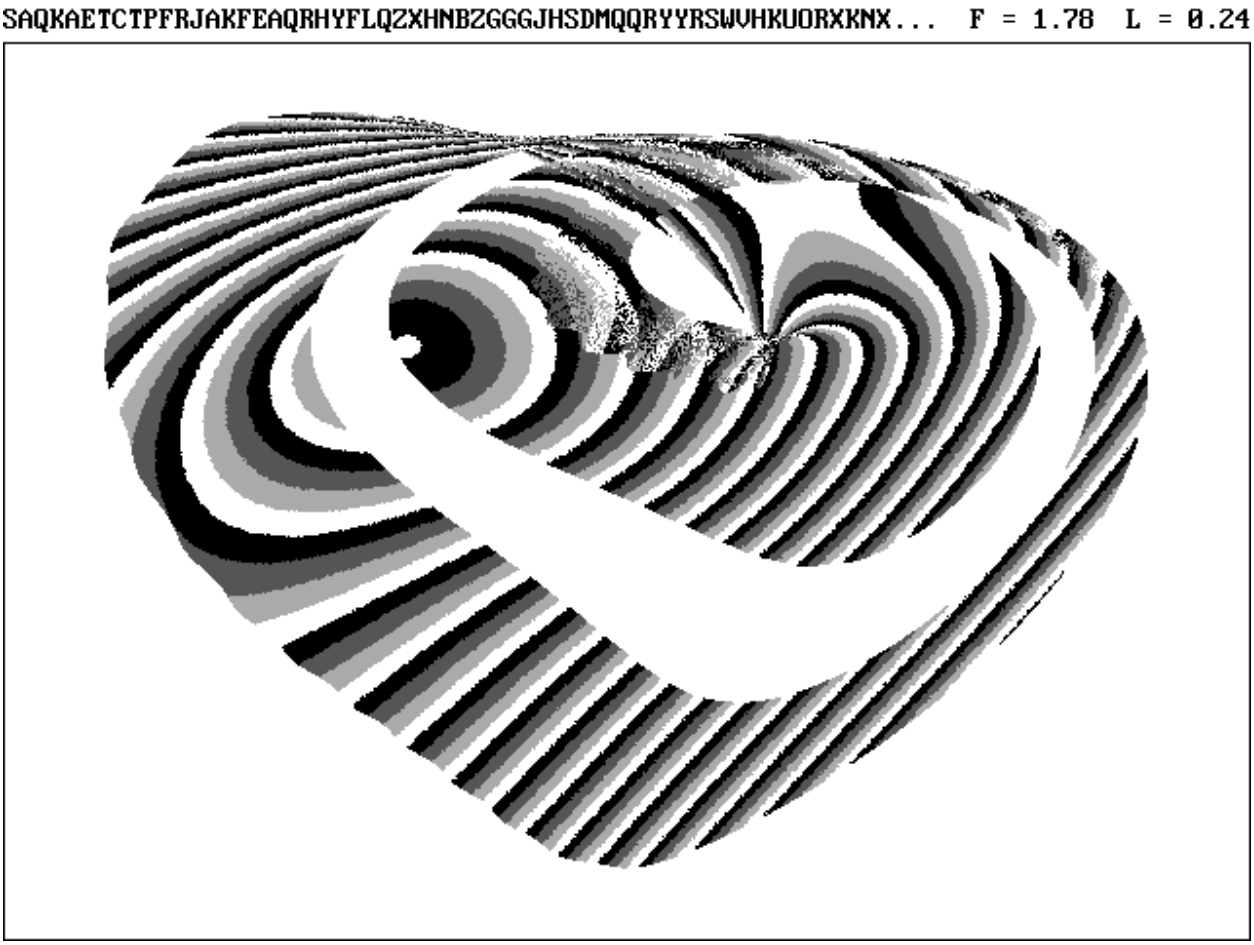


Figure 6-14. Three-dimensional quintic ODE with contour bands

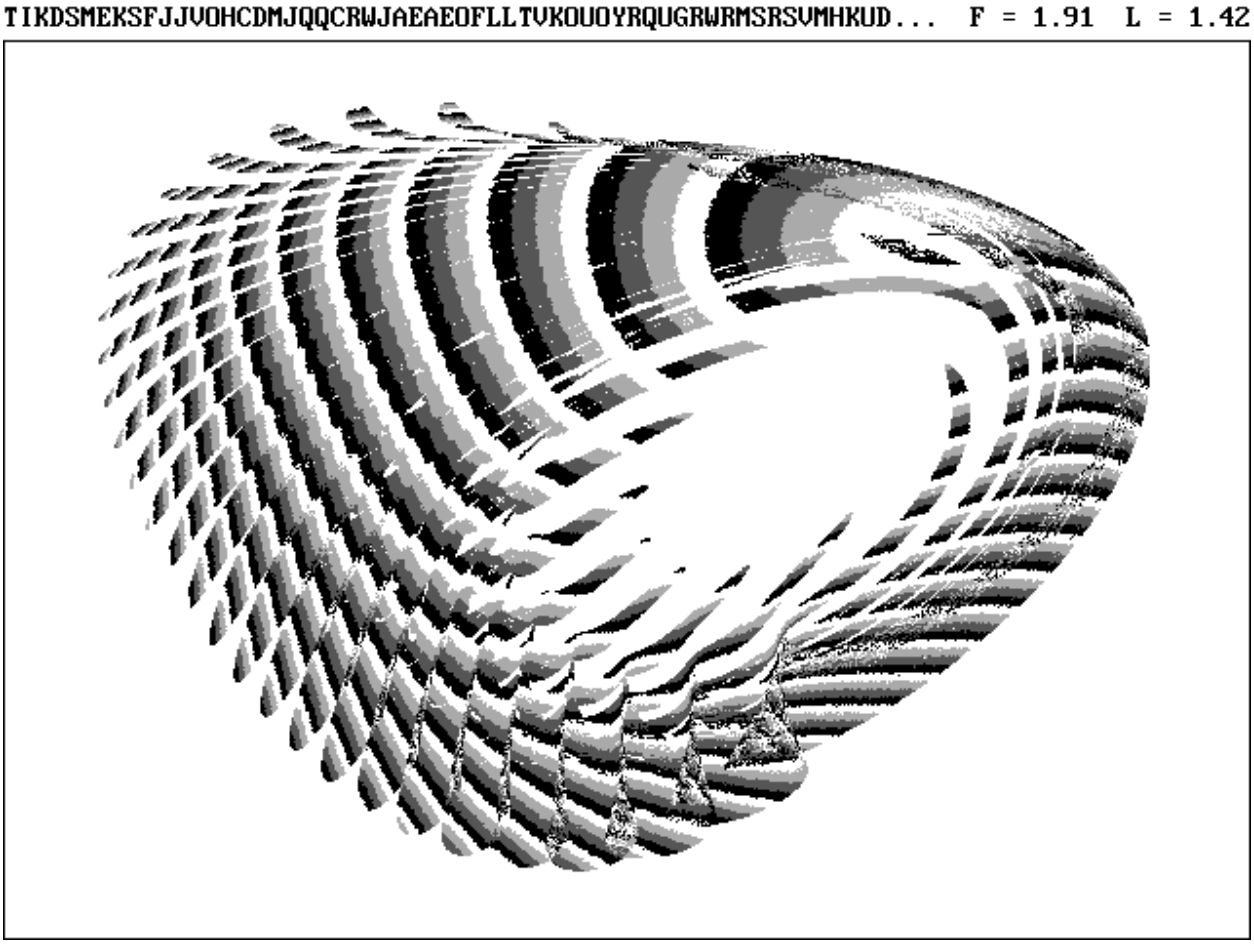


Figure 6-15. Stereo pair of three-dimensional quadratic ODE

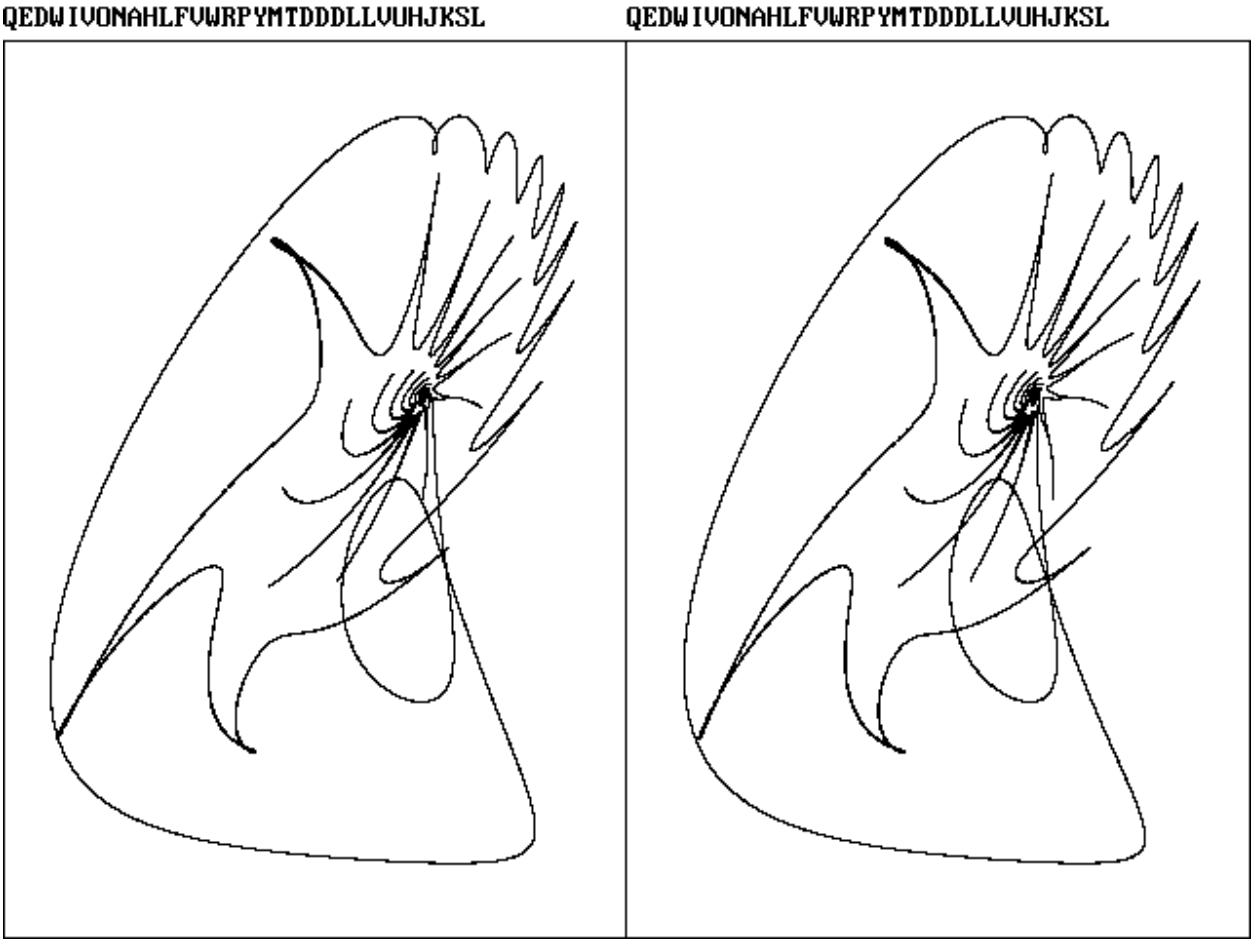


Figure 6-16. Stereo pair of three-dimensional cubic ODE

RFEJMURNKYSKVHFJXVFDMBKSBJFFUUIBTEFU... RFEJMURNKYSKVHFJXVFDMBKSBJFFUUIBTEFU...

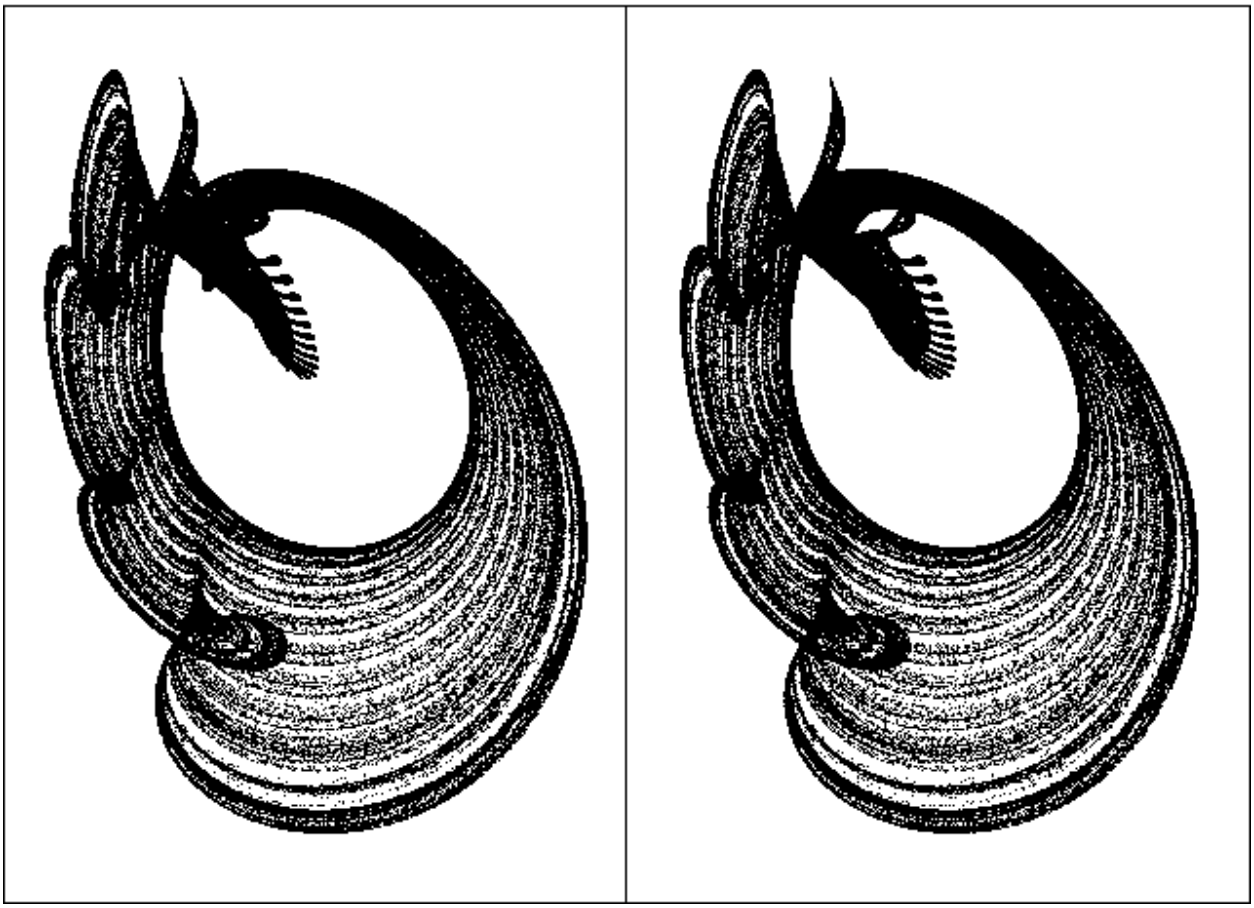


Figure 6-17. Stereo pair of three-dimensional quartic ODE

SHBNLCIMXEHDFWCTGGTHMISUIWQVKNOGMRKX... SHBNLCIMXEHDFWCTGGTHMISUIWQVKNOGMRKX...

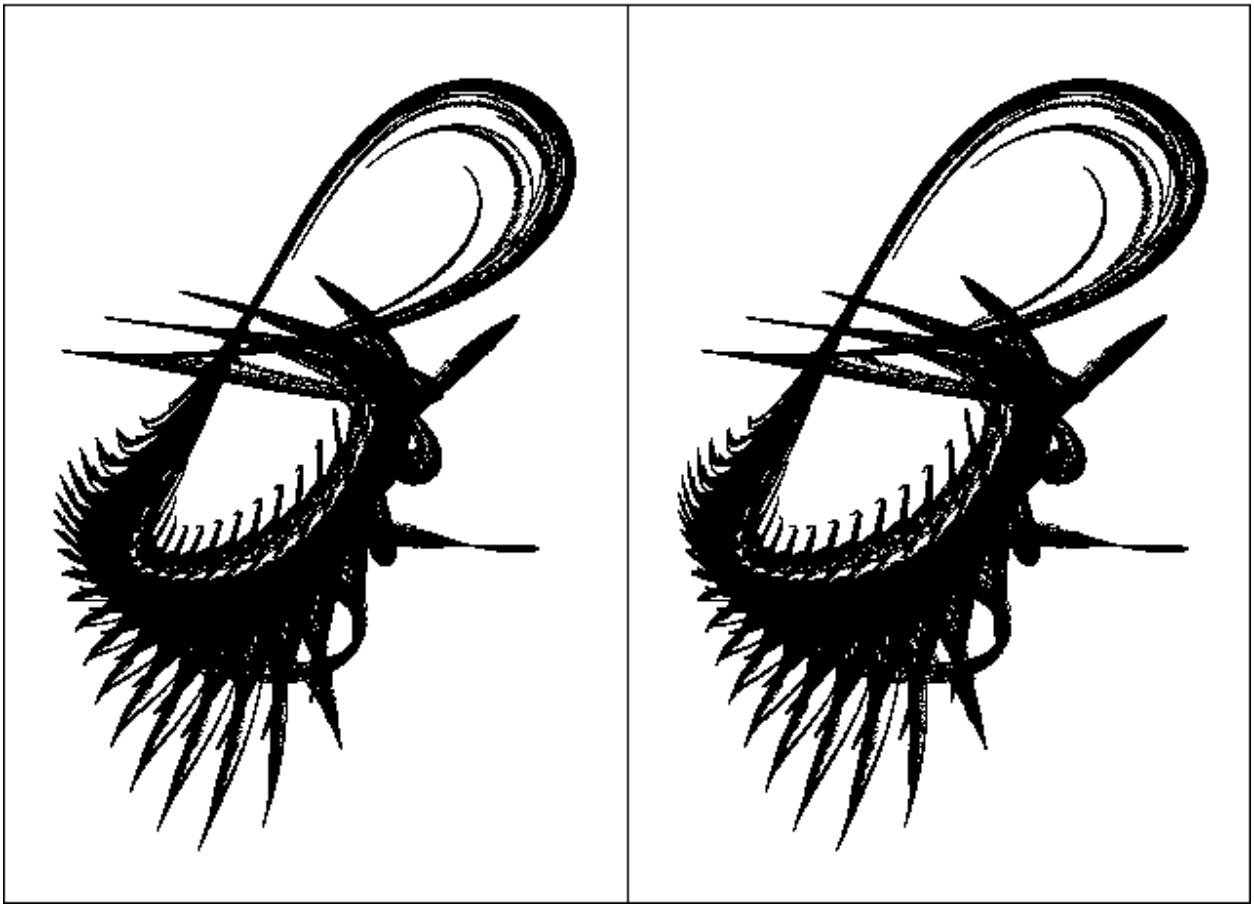


Figure 6-18. Stereo pair of three-dimensional quintic ODE

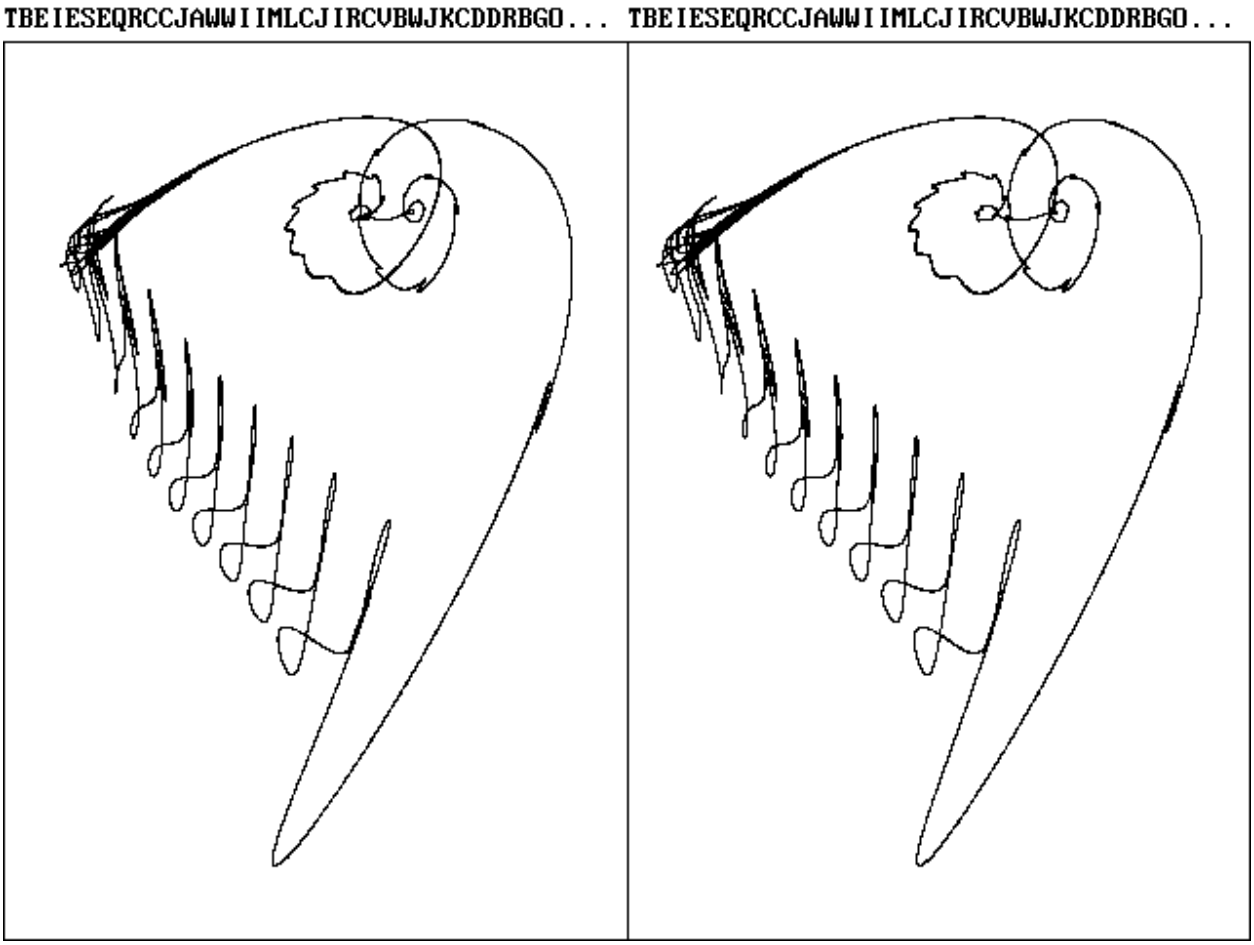


Figure 6-19. Slices of a three-dimensional quadratic ODE

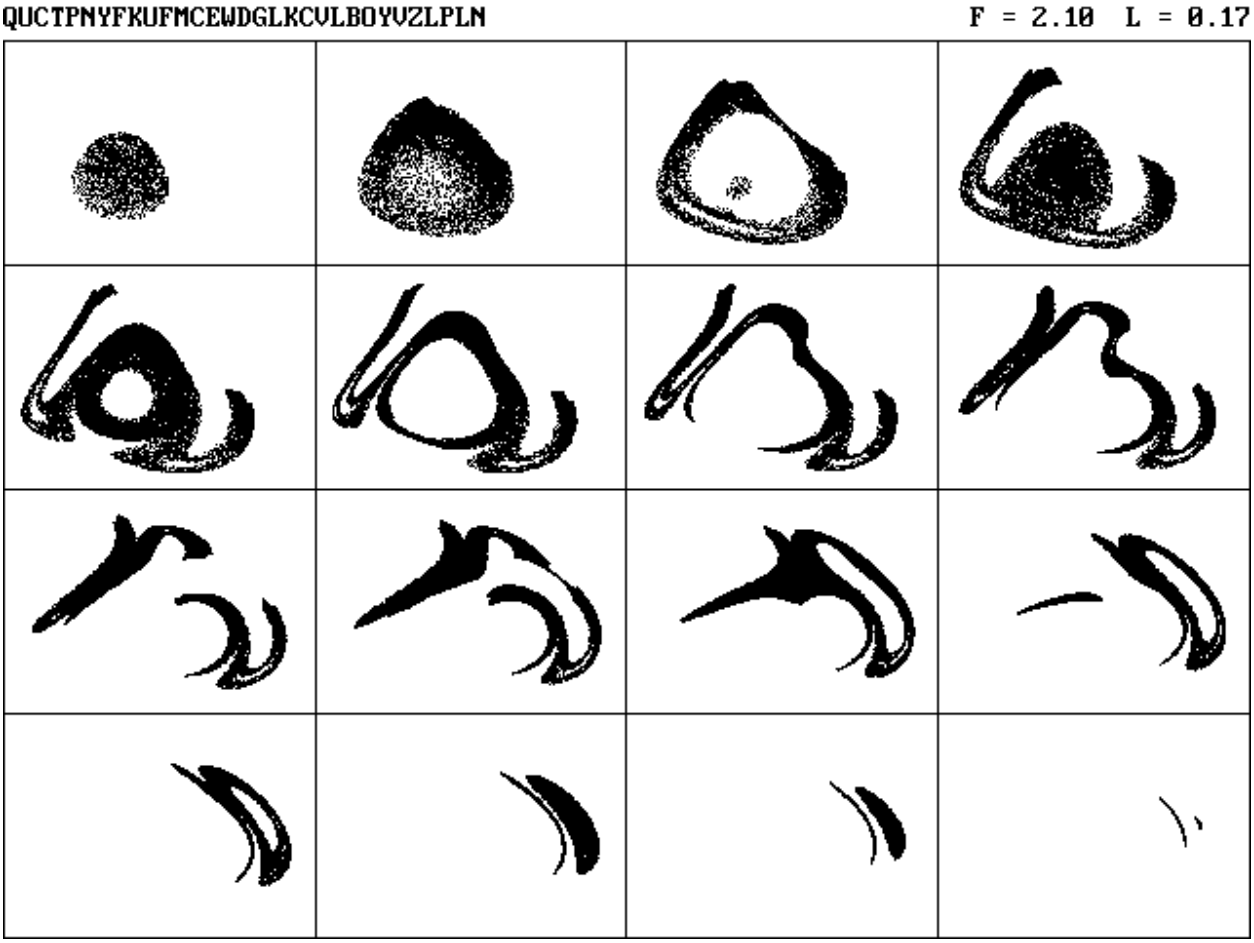


Figure 6-20. Slices of a three-dimensional cubic ODE

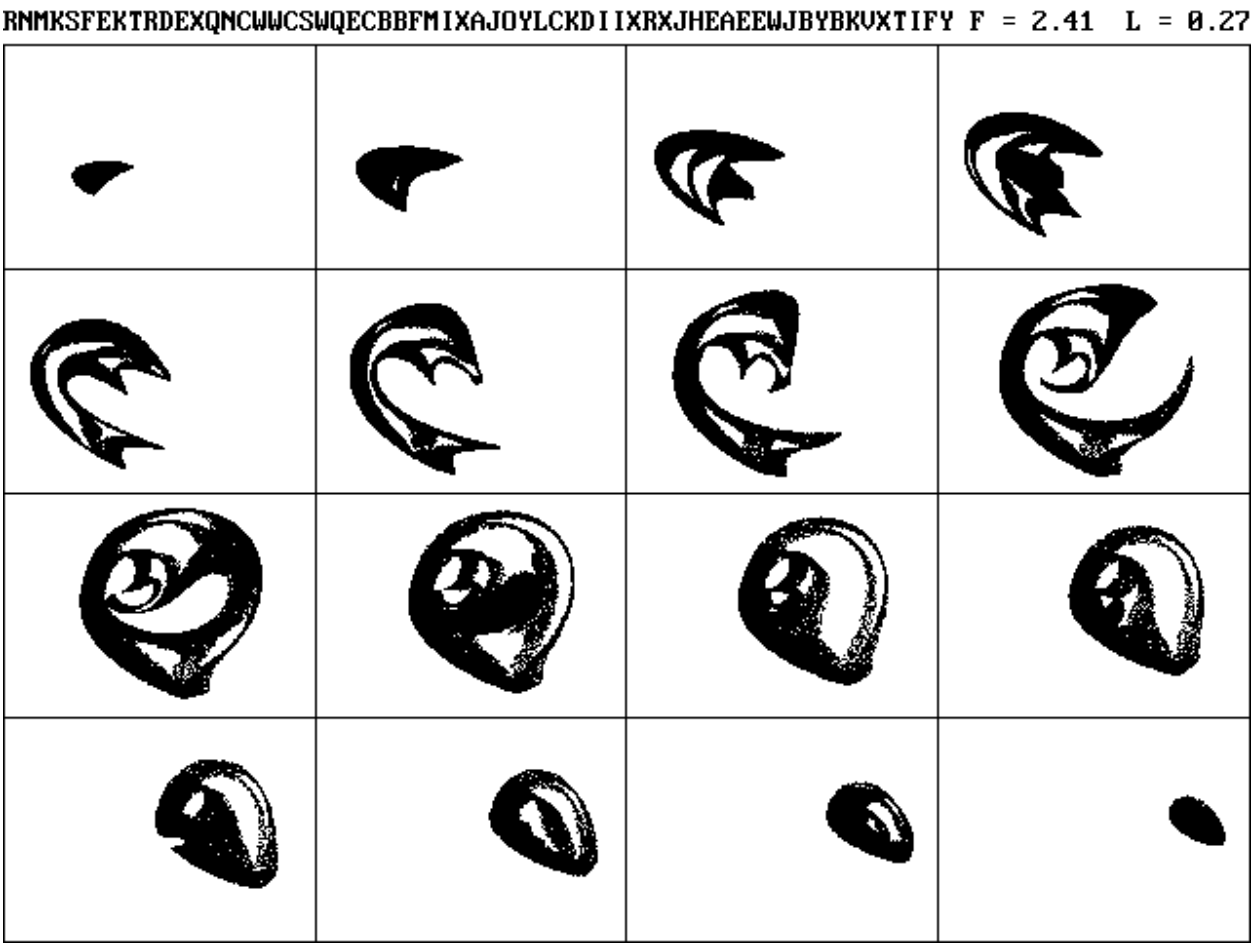


Figure 6-21. Slices of a three-dimensional quartic ODE

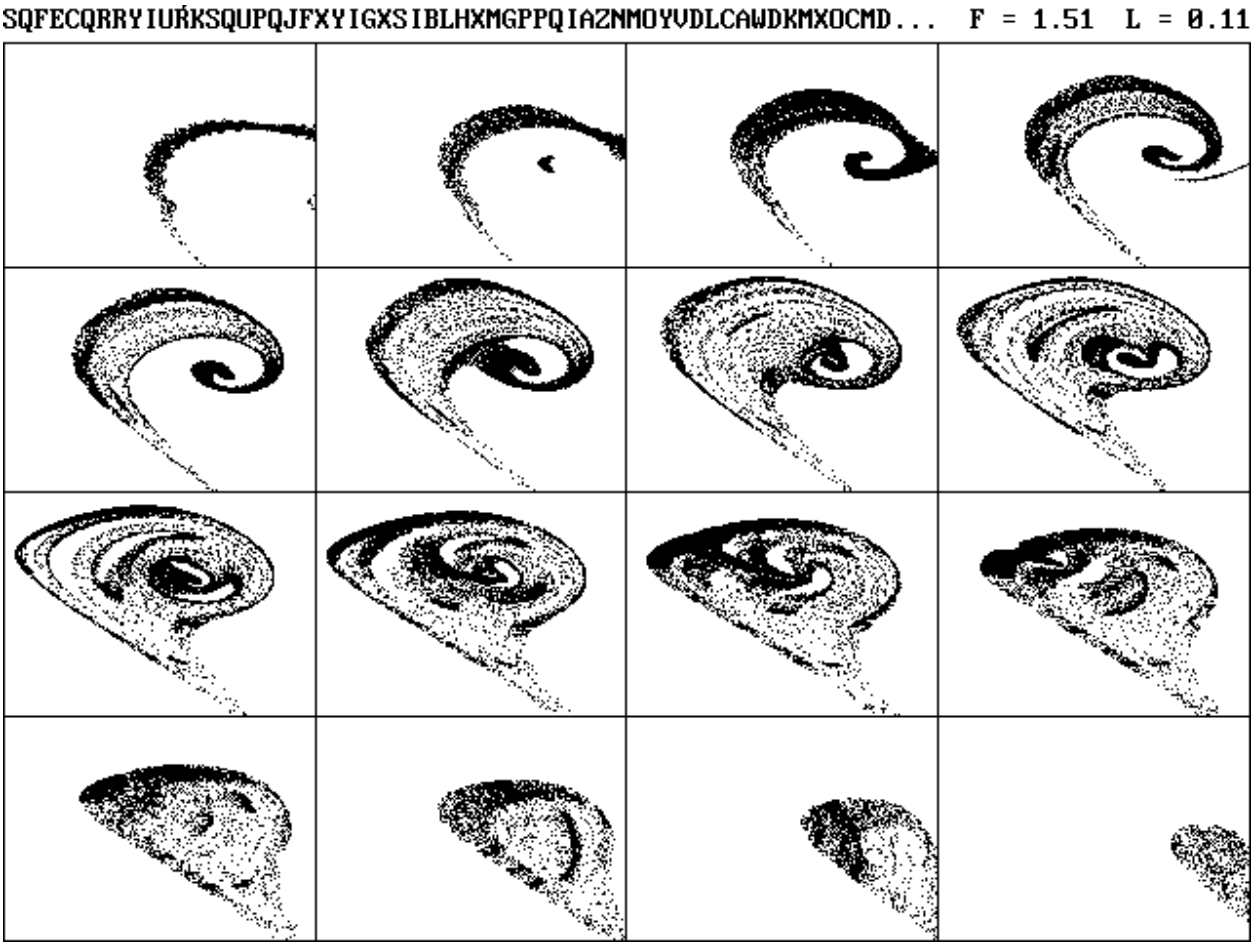
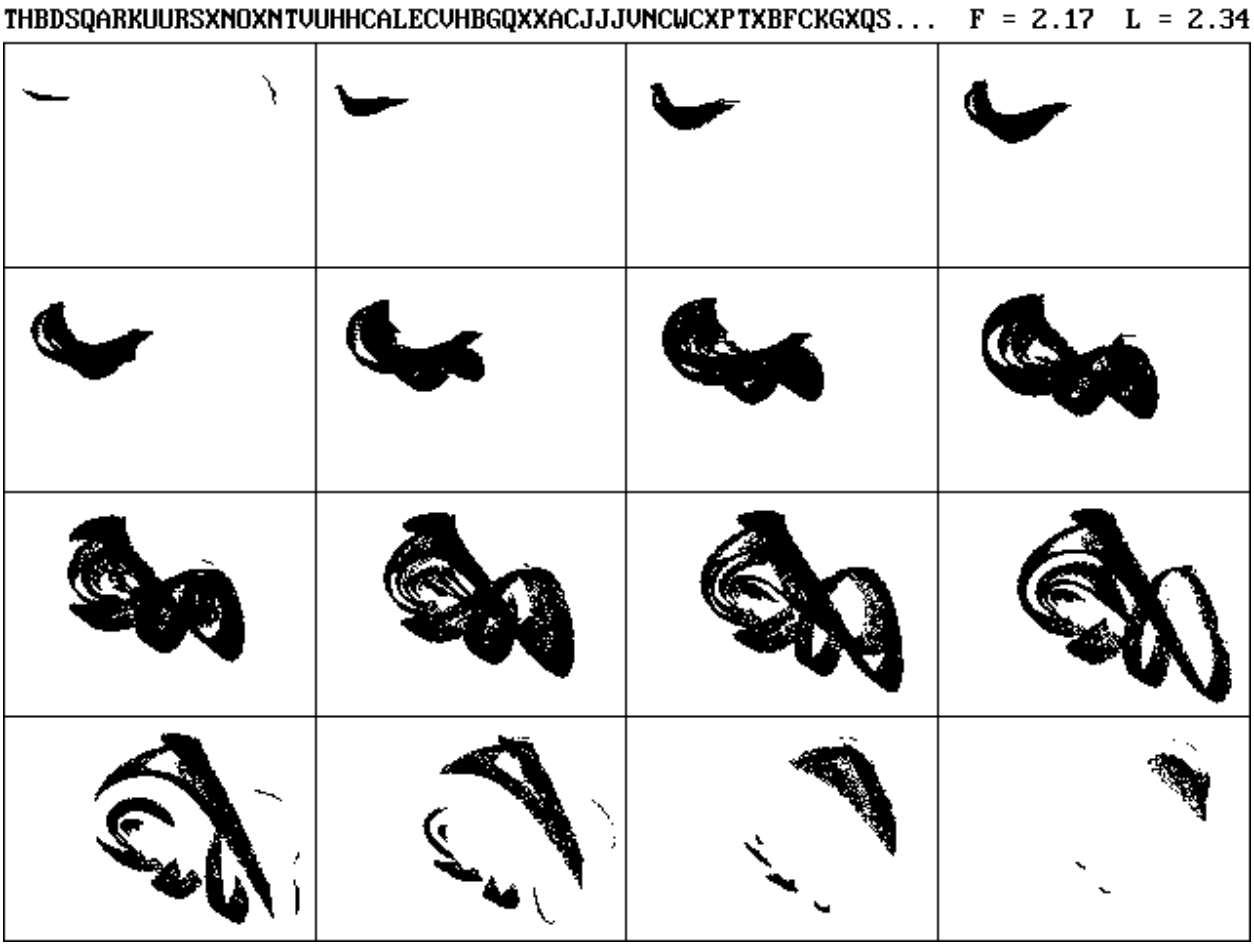


Figure 6-22. Slices of a three-dimensional quintic ODE



6.4 Flows in Four Dimensions

Flows, like maps, can be embedded in spaces of arbitrary dimension. Four-dimensional flows are hard to visualize but pose no difficulty for the computer to calculate. Buried in **PROG21** is the capability for calculating four-dimensional flows. You only need to press the **D** key to access the four-dimensional ODEs. All the techniques previously developed for displaying four-dimensional maps are available. The quadratic, cubic, quartic, and quintic equations are coded with the first letters U, V, W, and X, respectively. Figures 6-23 through 6-38 and Plates 25 through 30 show a selection of strange attractors arising from four-dimensional ordinary differential equations with polynomial terms.

Figure 6-23. Projection of a four-dimensional quadratic ODE

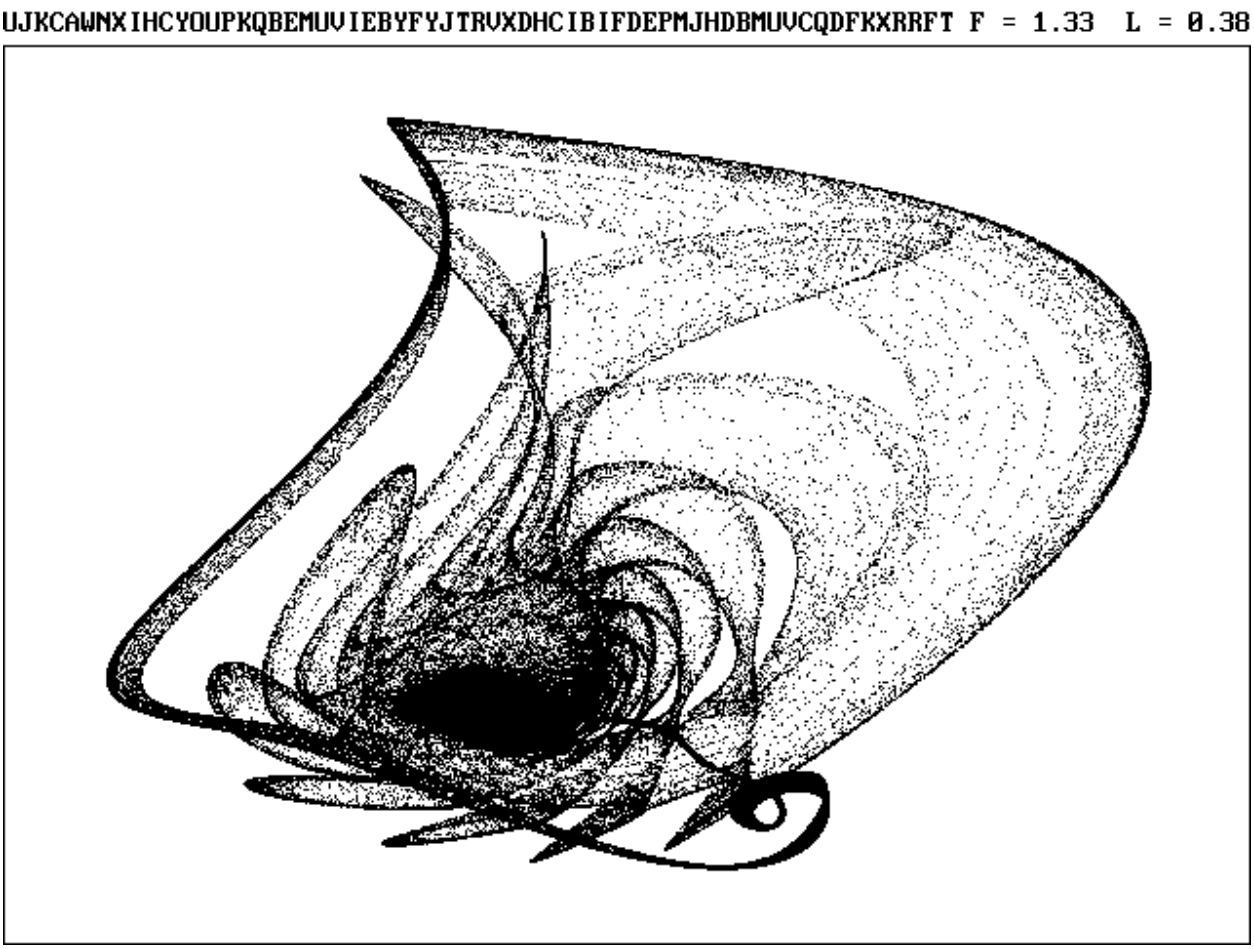


Figure 6-24. Projection of a four-dimensional cubic ODE

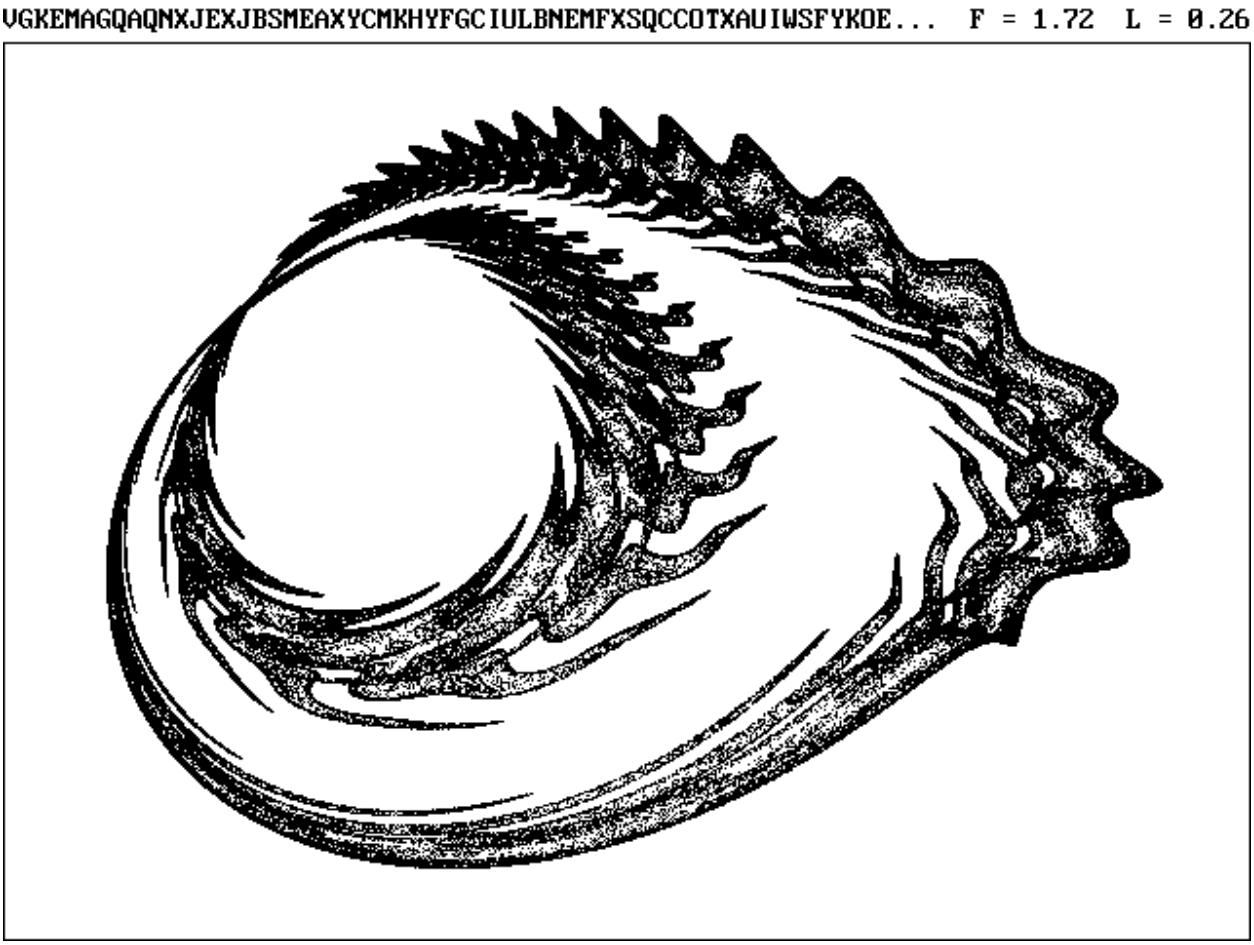


Figure 6-25. Projection of a four-dimensional quartic ODE

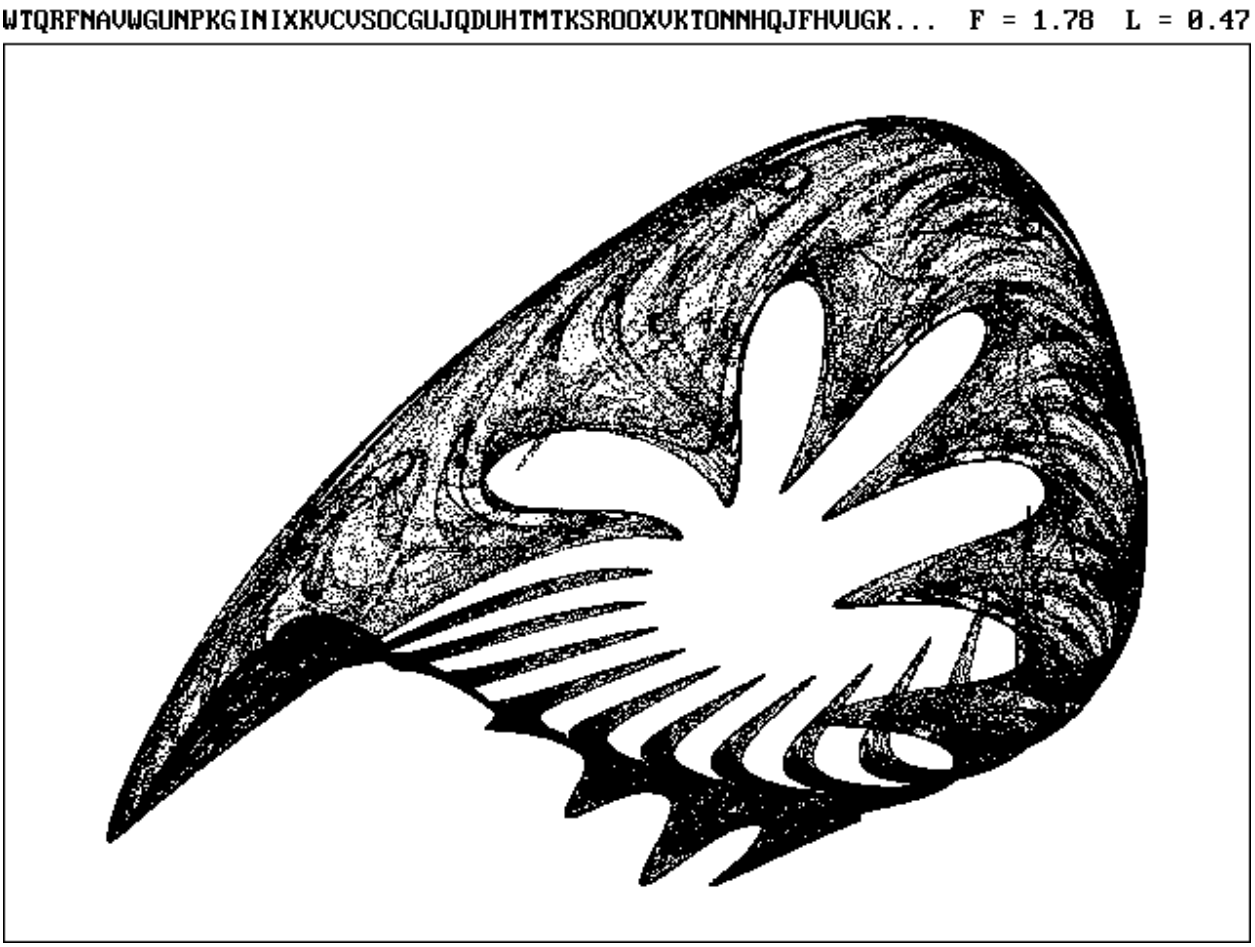


Figure 6-26. Projection of a four-dimensional quintic ODE

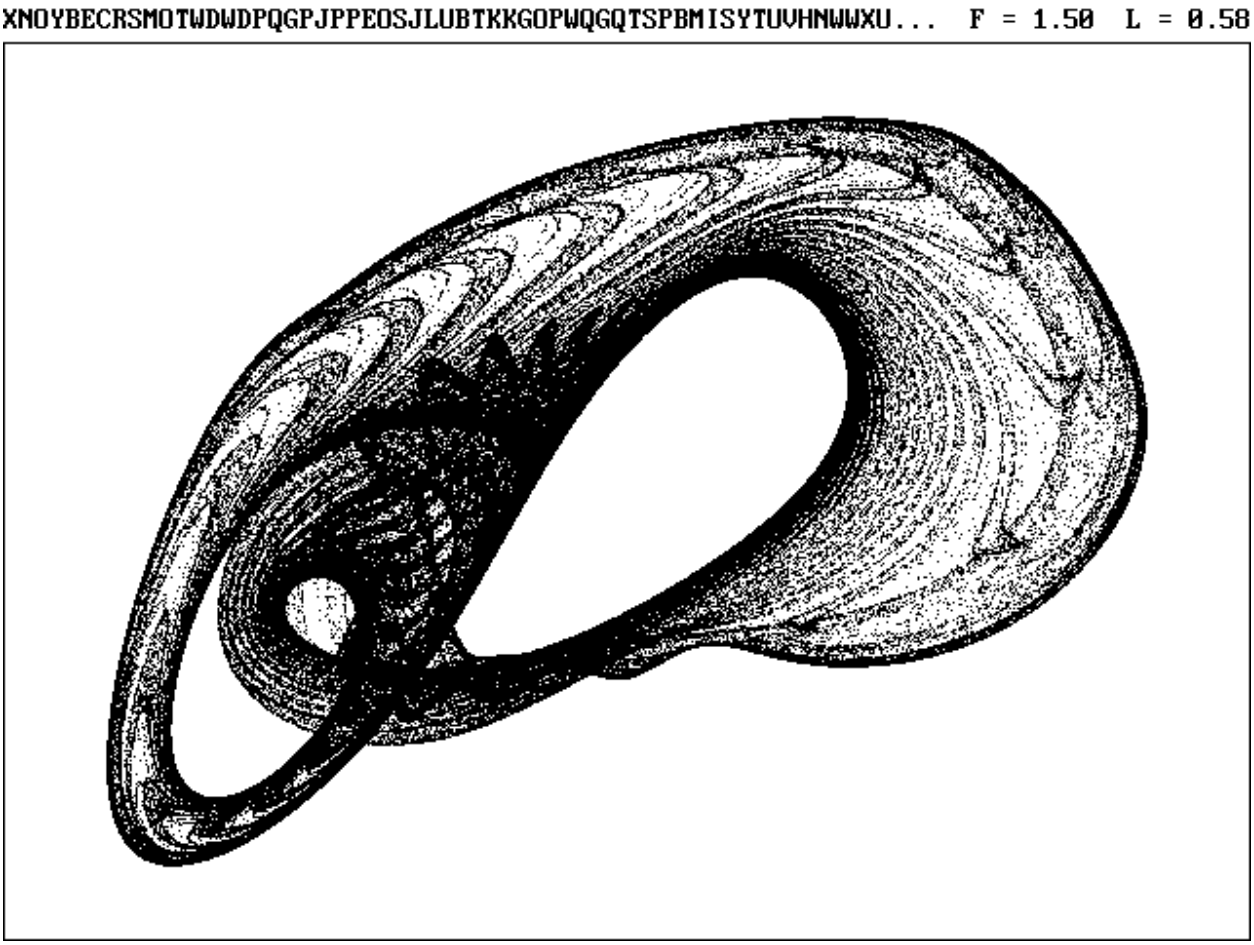


Figure 6-27. Four-dimensional quadratic ODE with shadow bands

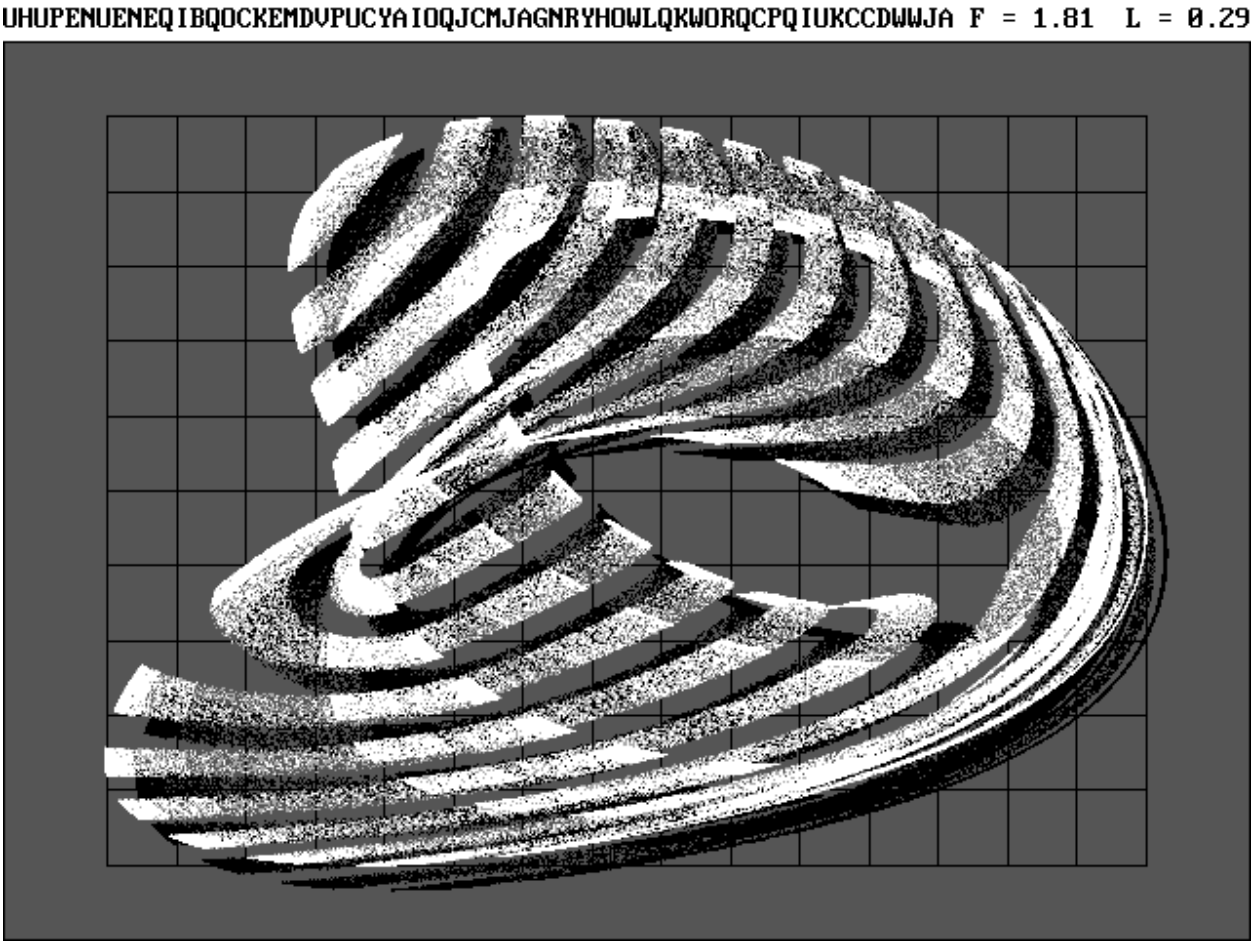


Figure 6-28. Four-dimensional cubic ODE with shadow bands

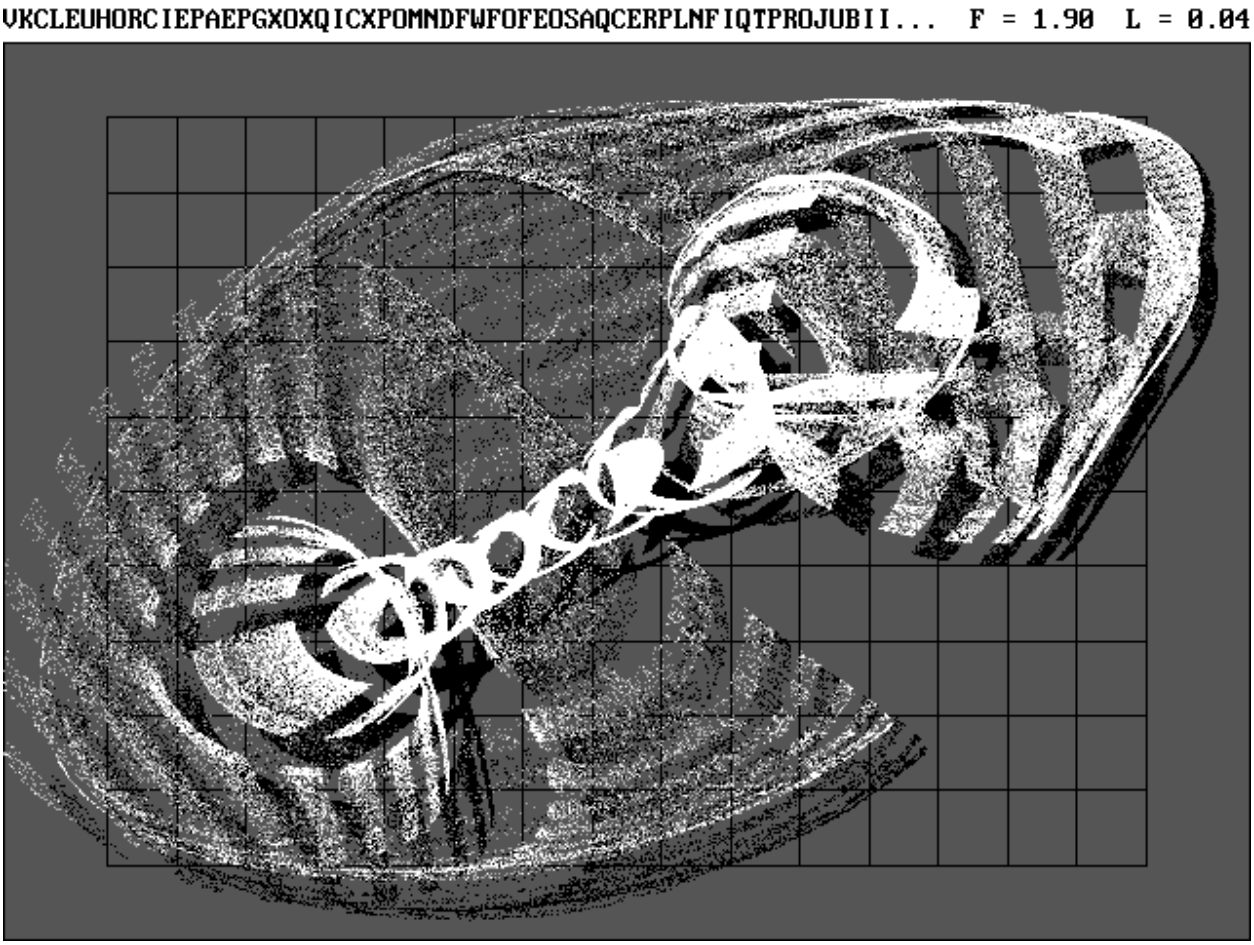


Figure 6-29. Four-dimensional quartic ODE with shadow bands

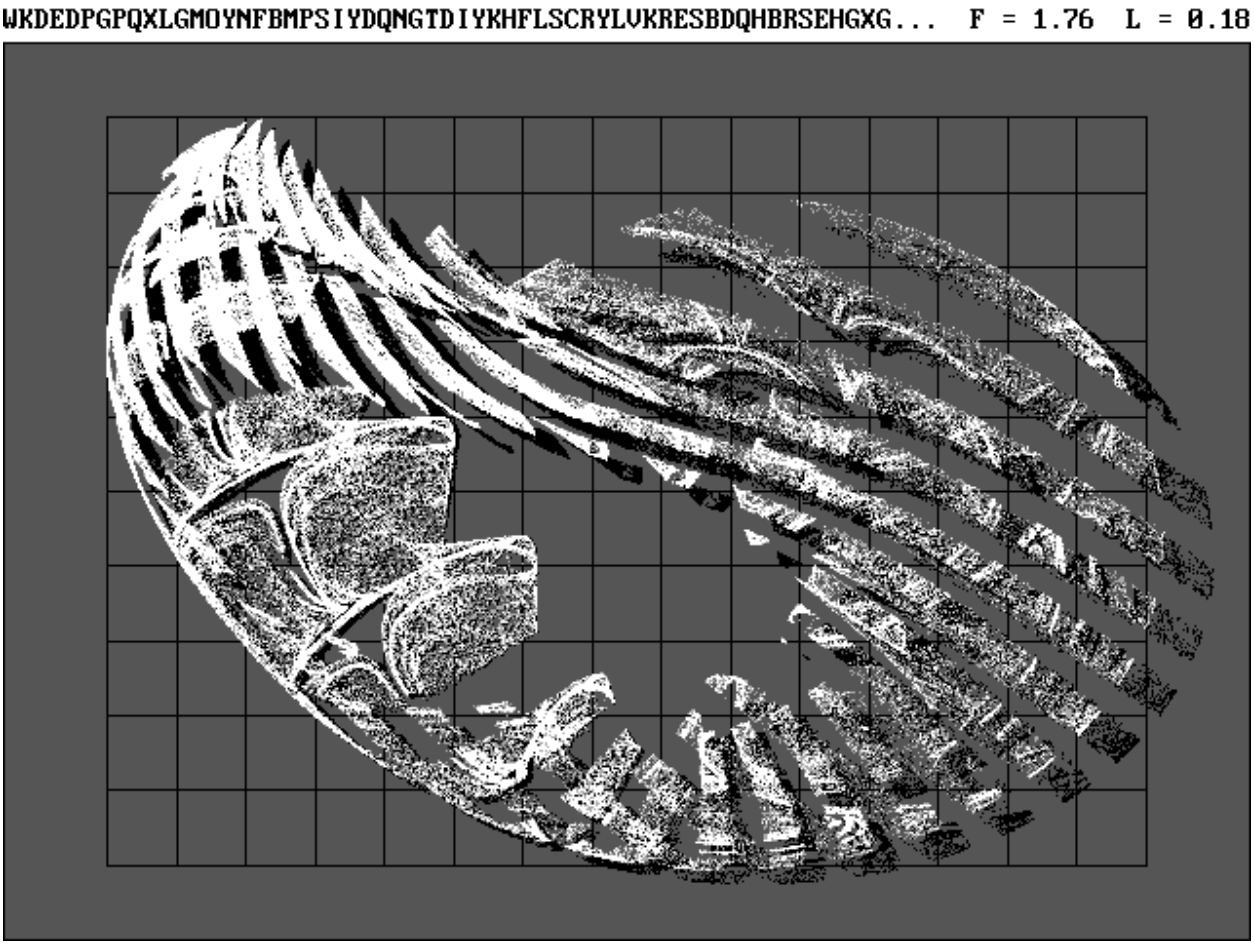


Figure 6-30. Four-dimensional quintic ODE with shadow bands

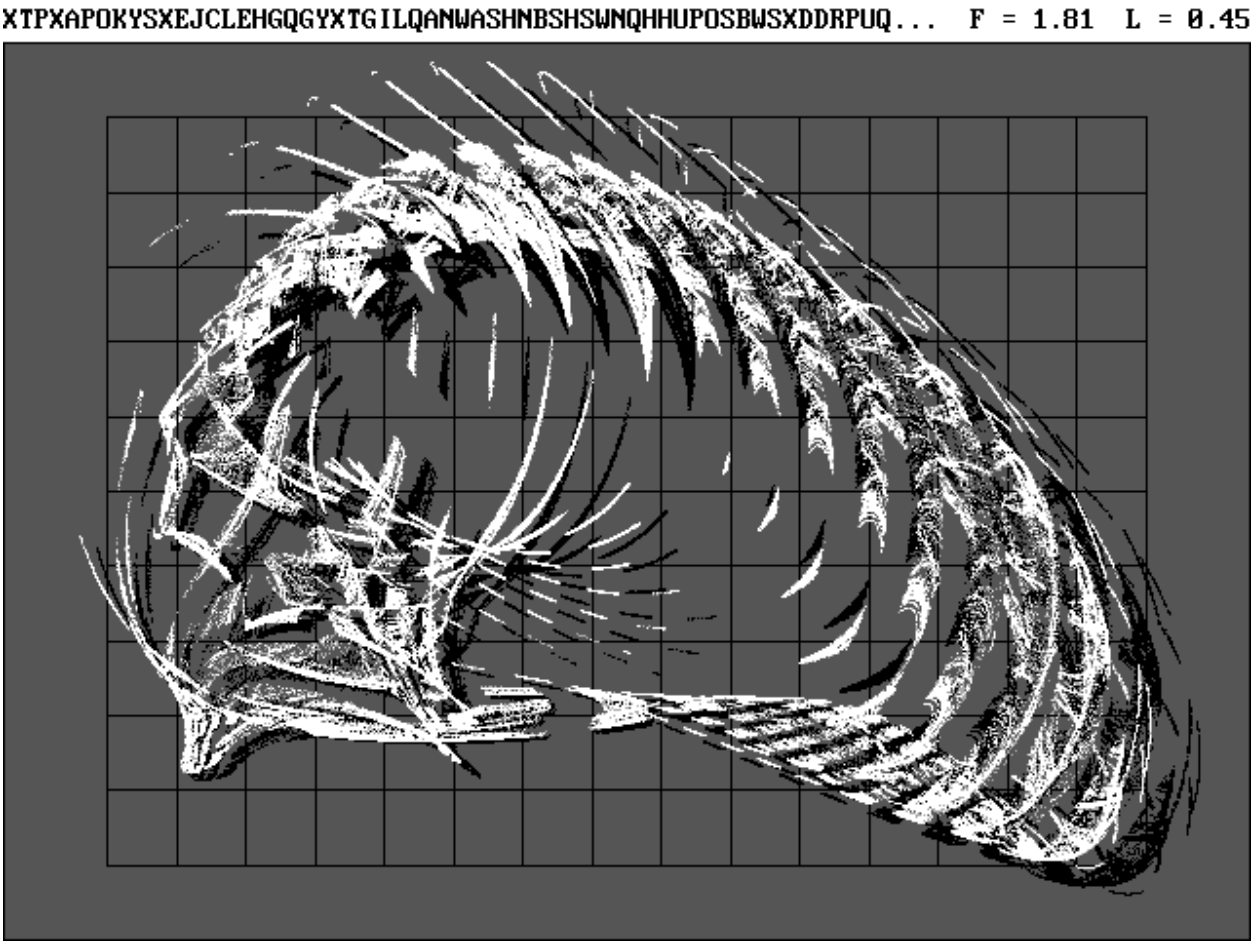


Figure 6-31. Four-dimensional quadratic ODE with stereo bands

UGLICIRJDBAEOQBFPHVDSPDIUVHNRQHCVYAE... UGLICIRJDBAEOQBFPHVDSPDIUVHNRQHCVYAE...

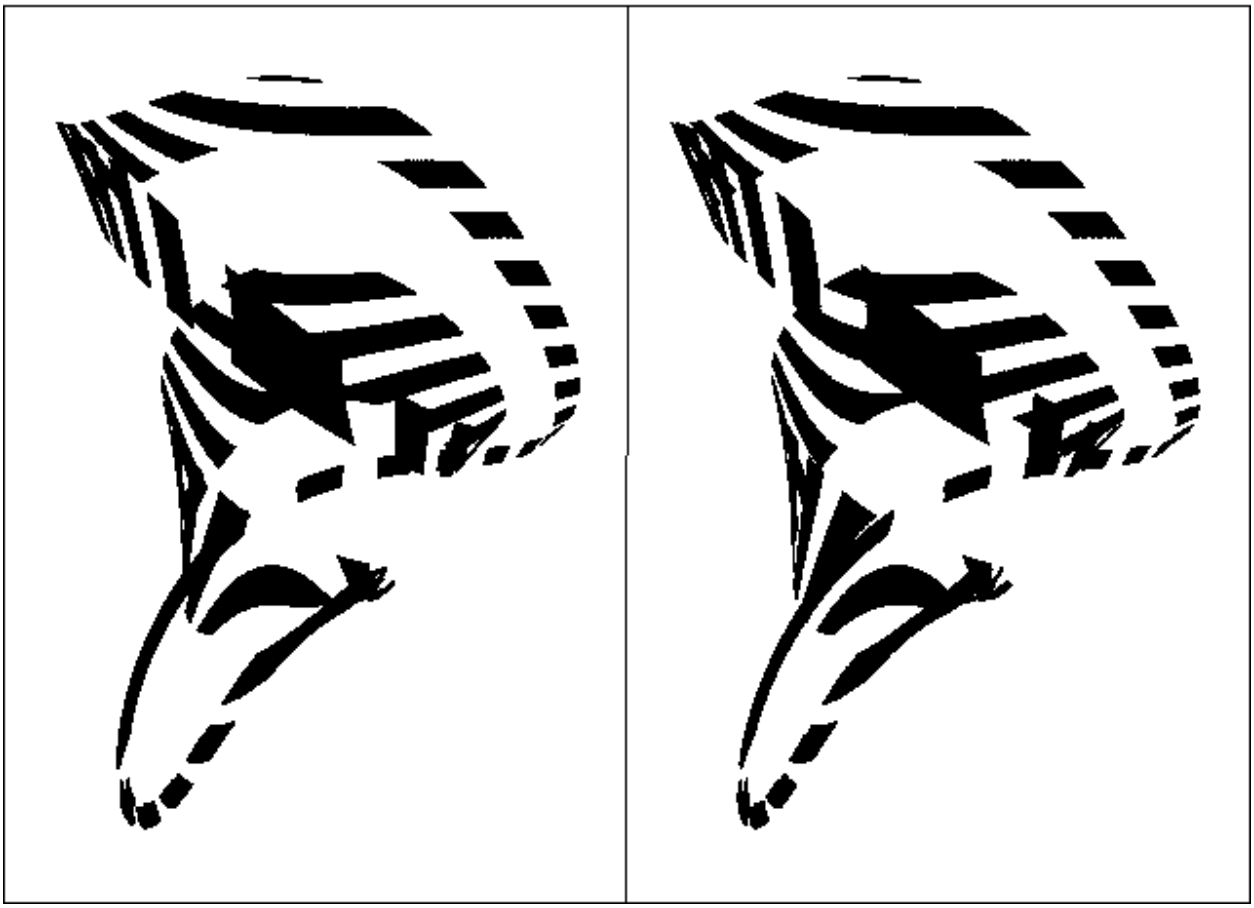


Figure 6-32. Four-dimensional cubic ODE with stereo bands

VHTQNWHDQRFGFJXRDQYRWNCRCQFPSBSNVPES... VHTQNWHDQRFGFJXRDQYRWNCRCQFPSBSNVPES...

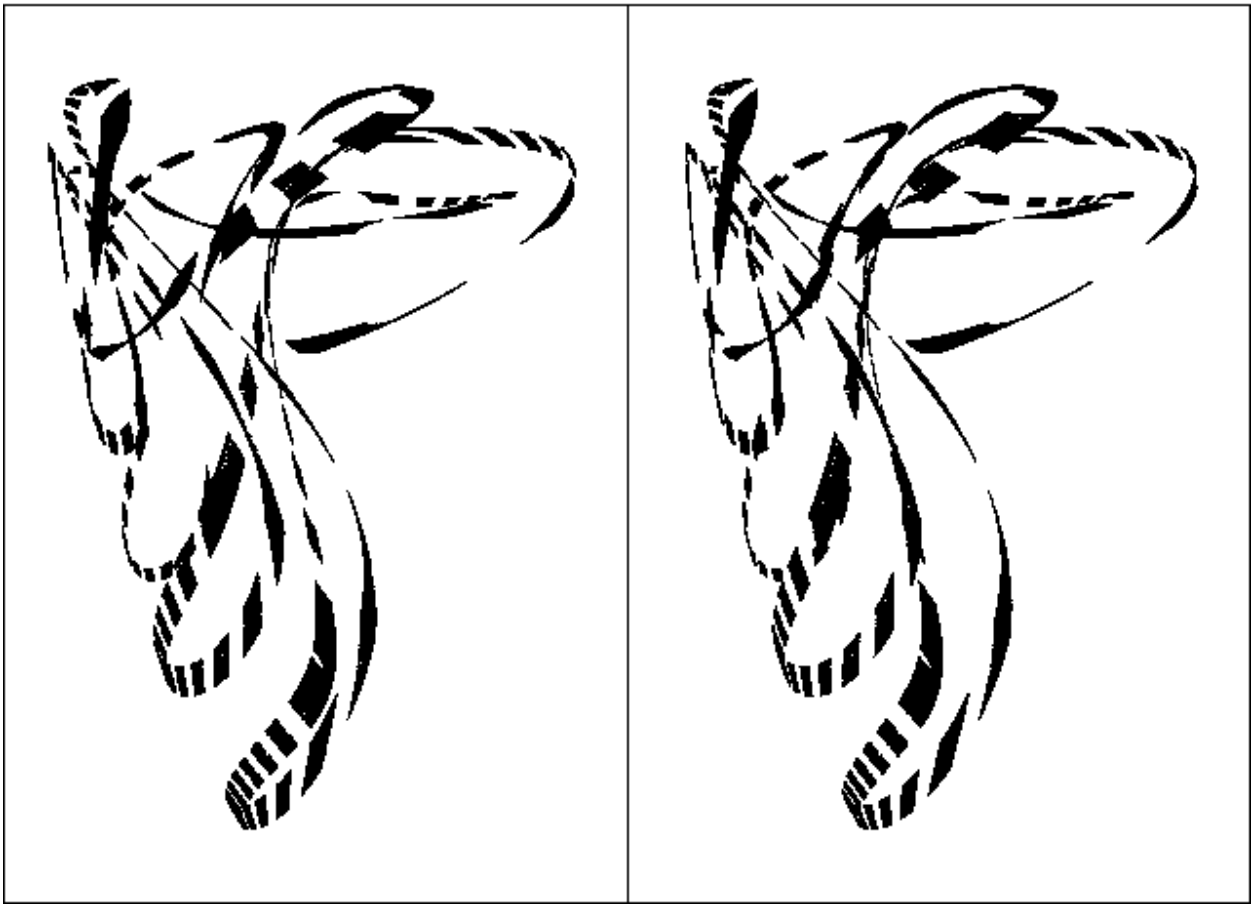


Figure 6-33. Four-dimensional quartic ODE with stereo bands

WJCUJYONOYWSFMVSWQXTDCQTPWHWTFDISTVU... WJCUJYONOYWSFMVSWQXTDCQTPWHWTFDISTVU...

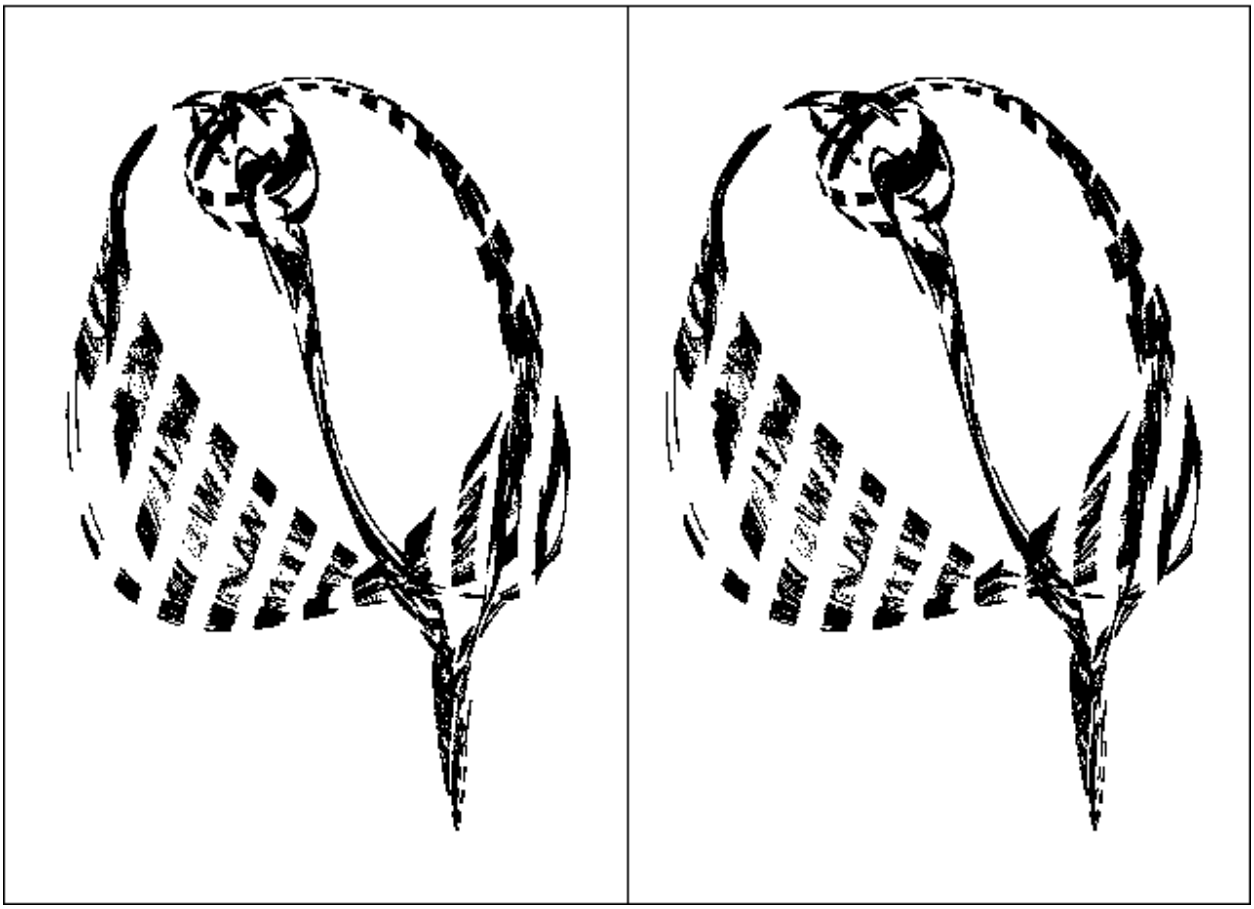


Figure 6-34. Four-dimensional quintic ODE with stereo bands

XJUSDQKDEJYCDRBCBQFKKINFBFCIHLYWDDGH... XJUSDQKDEJYCDRBCBQFKKINFBFCIHLYWDDGH...

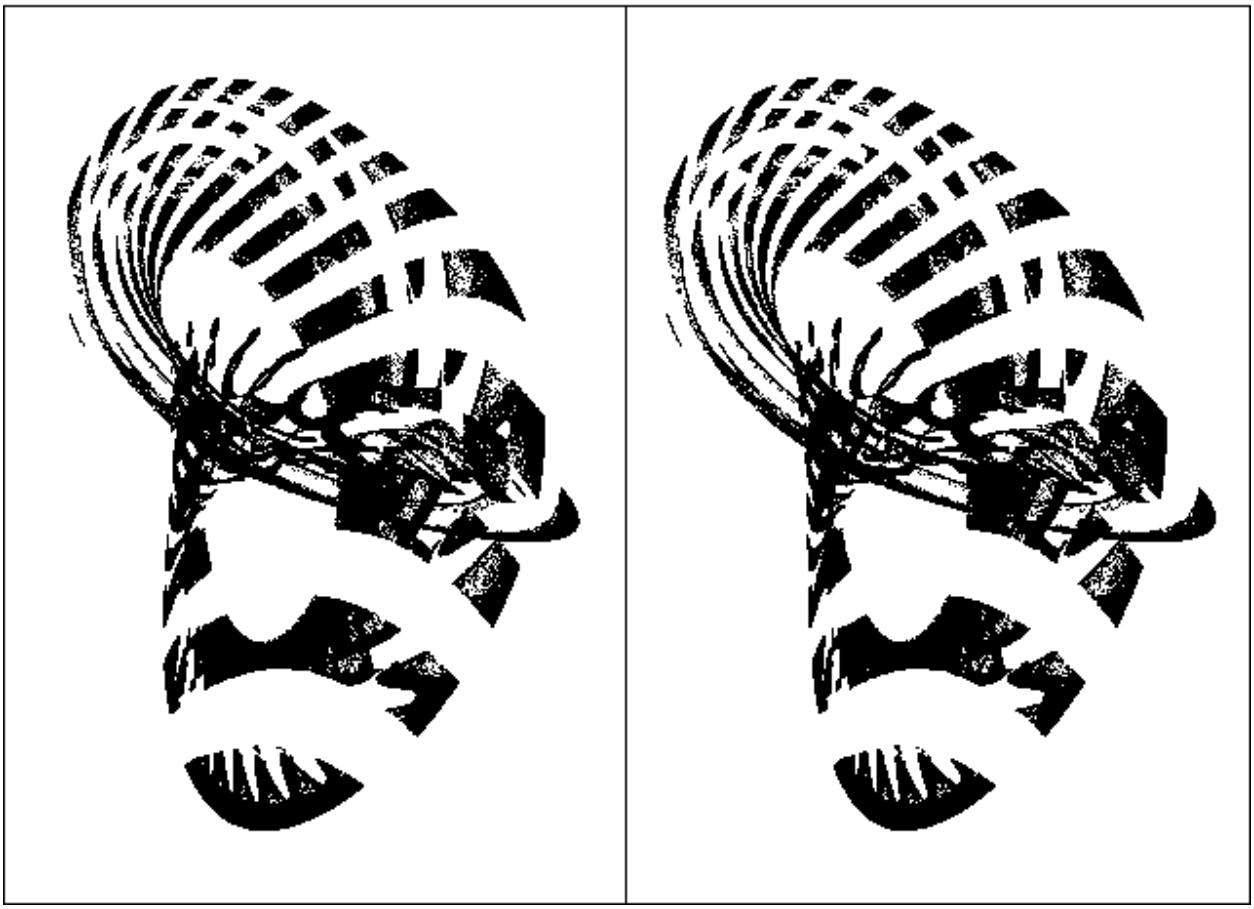


Figure 6-35. Four-dimensional quadratic ODE with sliced bands

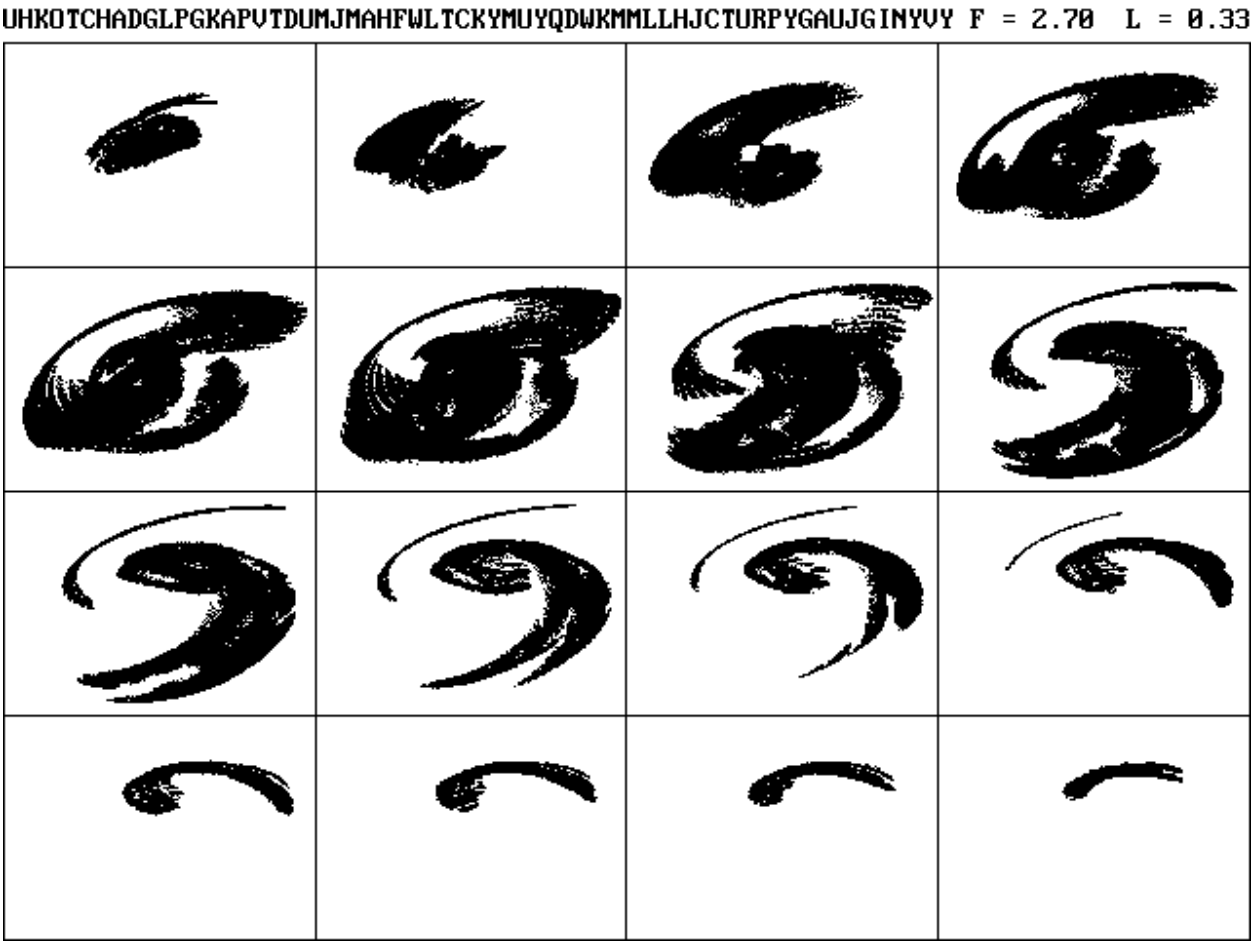


Figure 6-36. Four-dimensional cubic ODE with sliced bands

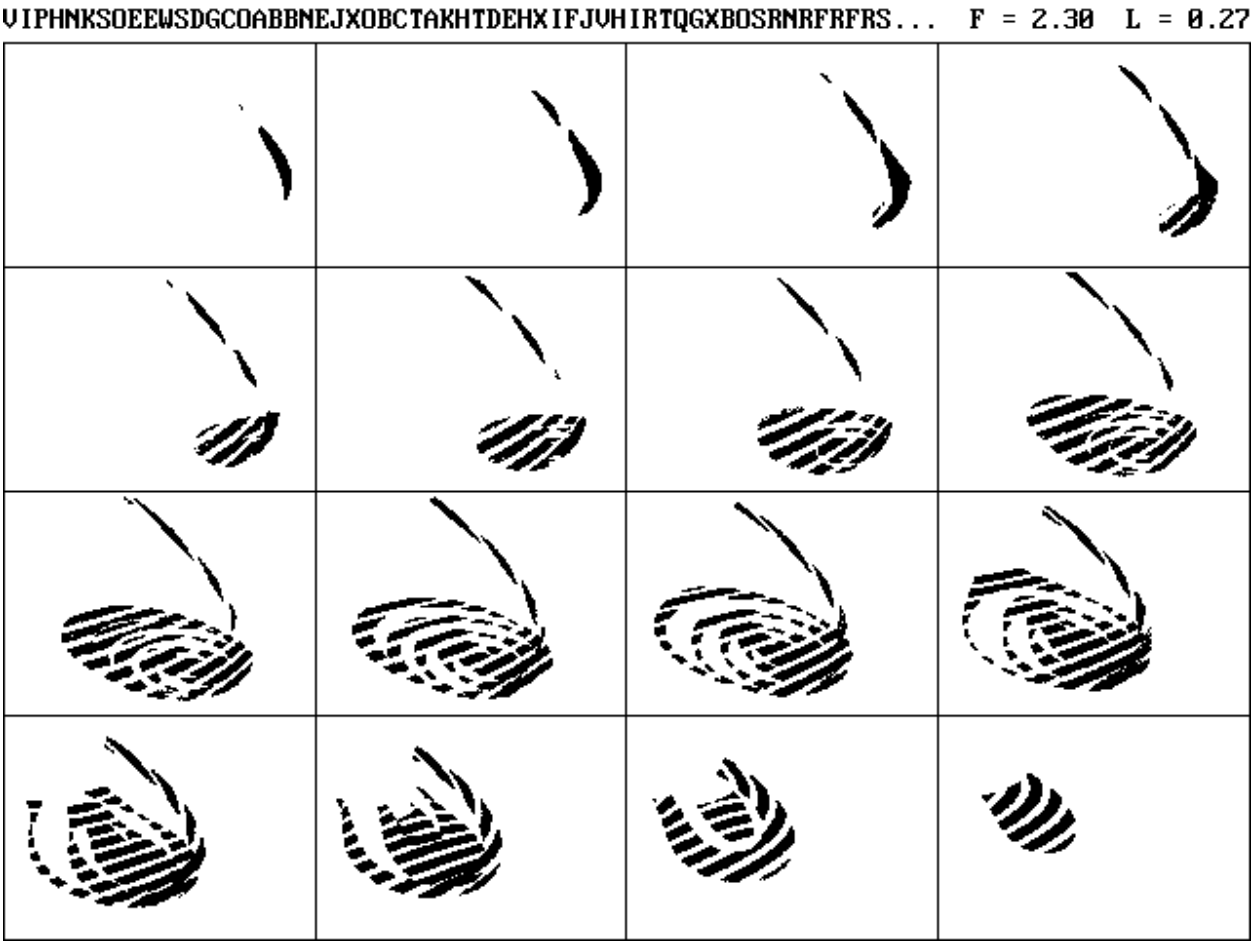


Figure 6-37. Four-dimensional quartic ODE with sliced bands

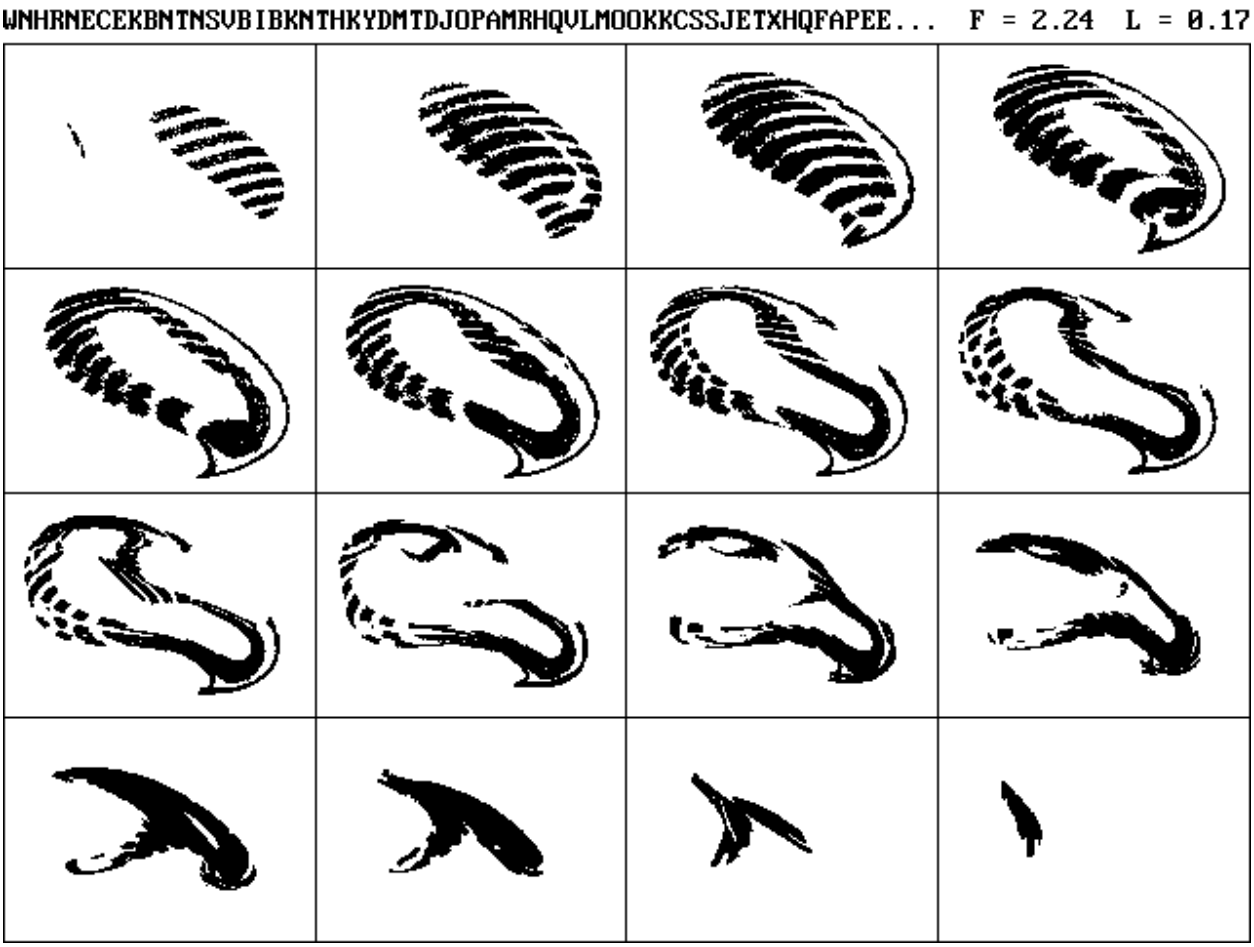
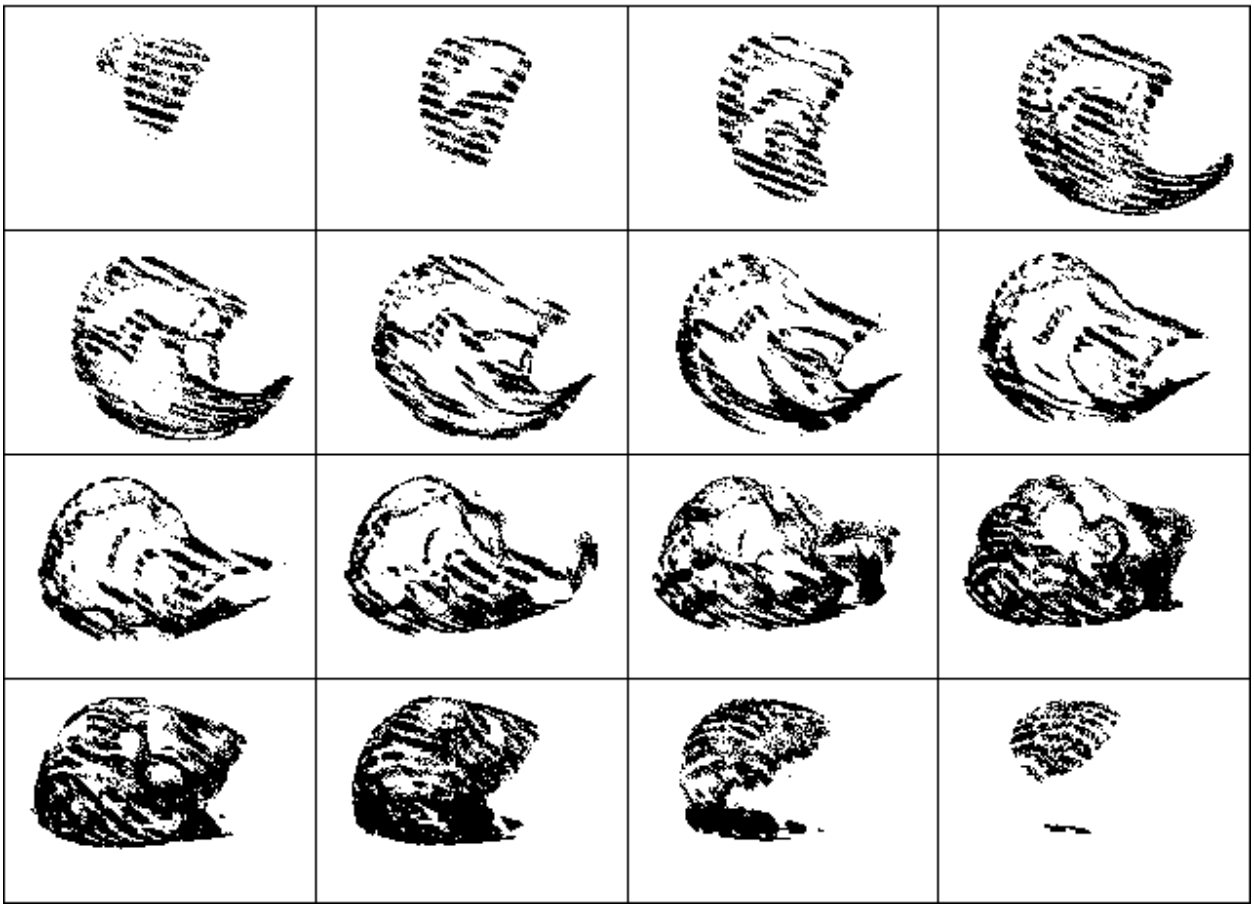


Figure 6-38. Four-dimensional quintic ODE with sliced bands

XPOOCSENDLUDHWQRTQDCQDHPWSTUYDLBKQKWIAKNPVLAFNNANJXBYNSYP... F = 1.37 L = 0.14



6.5 Strange Attractors that Aren't

In Section 3.8, we discussed chaotic orbits that don't approach an attractor ("Strange Attractors that Don't"). Here we consider nonchaotic orbits that approach attractors that aren't strange. These attractors are not fractals. They have dimensions that are integers such as 0, 1, 2, or 3. Some of them are beautiful, so they are worth displaying even if they are technically outside the scope of this book.

Such attractors can arise from maps as well as from differential equations. They don't require high embedding dimensions, although the dimension of the attractor always is at least one less than the dimension of the embedding space. Thus some of the examples are taken from equations described in earlier chapters.

The simplest nonchaotic attractor is a point attractor. Suppose we modified Equation 6A so the solution is not a circle but an inward spiral. One way to do this is as follows:

$$X' = Y - bX$$

$$Y' = -X - bY \qquad \text{(Equation 6G)}$$

You can think of the coefficient *b* as a measure of the friction that eventually brings the trajectory to rest at the origin ($X = Y = 0$) in phase space. If *b* is zero (frictionless), the orbit is a circle. Negative values of *b* (antifriction) cause the solution to spiral outward, approaching infinity. This case corresponds to a point repellor at the origin. An attractive fixed point is called a *sink* and a repelling fixed point is called a *source*. Some authors reserve the term *fixed point* for maps and prefer to call the stationary solutions of ODEs *critical points* or *equilibrium points*.

The two occurrences of *b* in Equation 6G need not have the same value or even the same sign. In such a case, the orbit moves in or out but not in a symmetrical manner. Many physical processes have $b = 0$ in one of the equations. If *b* is close to zero, it doesn't matter much in which equation it appears.

If *b* is zero in one of the equations above, small positive values of the other *b* cause the radius of the circle to decrease slowly, approaching what is called a *spiral-point* or *focal-point attractor*, or simply a *focus*. Larger positive values of *b* cause the radius to decrease more rapidly. With very large values of *b*, there is little circulation around the point, and the trajectory is more nearly radial toward what is called a *radial-point* or *nodal-point attractor*, or simply a *node*. The boundary between the two cases occurs at $b = 2$ and corresponds to critical damping in an oscillator. In either case, the resulting attractor is a point at the origin with a

dimension of zero. A code that produces a point attractor (with b = 1) is $QMLM^3NM^5LM^3LM^{14}$. For this case, the largest Lyapunov exponent is negative (L = - 0.1/*ln* 2 = - 0.14), and it produces a single dot on the screen.

More interesting cases can occur because the program assumes the trajectory is on the attractor after 1000 iterations. For trajectories that approach the attractor very slowly, there can be interesting behavior after the thousandth iteration and before the fixed point is reached. Such slowly attracting fixed points have negative Lyapunov exponents, at least one of which is very close to zero. The search can be expanded to include them as well as other nonchaotic attractors by changing the .005 in line 2480 of the program to -.005, for example. Then attractors that have Lyapunov exponents near zero but have not settled to a fixed point after *NMAX* iterations are included in the file **SA.DIC**. If you prefer, you can collect them in a separate file **TORUS.DIC** by changing line 4910 of **PROG21** to

```
4910 IF L > .005 THEN OPEN "SA.DIC" FOR APPEND AS #1 ELSE OPEN "TORUS.DIC" FOR
APPEND AS #1
```

Several such cases are shown in Figures 6-39 through 6-42. Figure 6-39 shows a spiral-point attractor. Figure 6-40 shows what appears to be a radial-point attractor with several different initial conditions; it is really a spiral-point attractor with successive iterates that move rapidly around the fixed point. This phenomenon is called *aliasing*, and it is most easily detected by connecting temporally successive points with continuous lines. Figures 6-41 and 6-42 show cases where the rate of circulation around the fixed point changes significantly as the fixed point is approached.

Figure 6-39. Trajectory approaching a spiral-point attractor

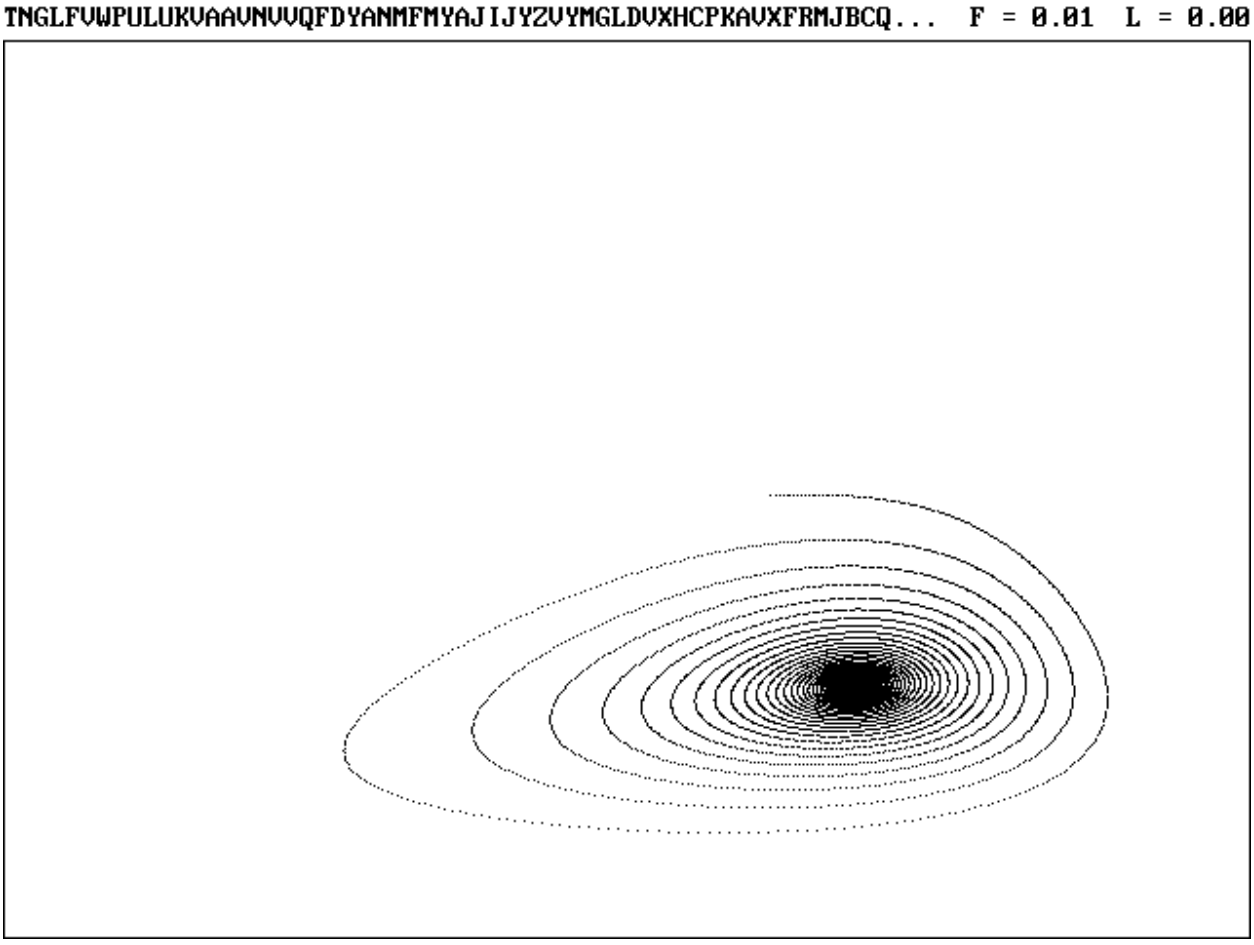


Figure 6-40. Trajectory emulating a radial-point attractor

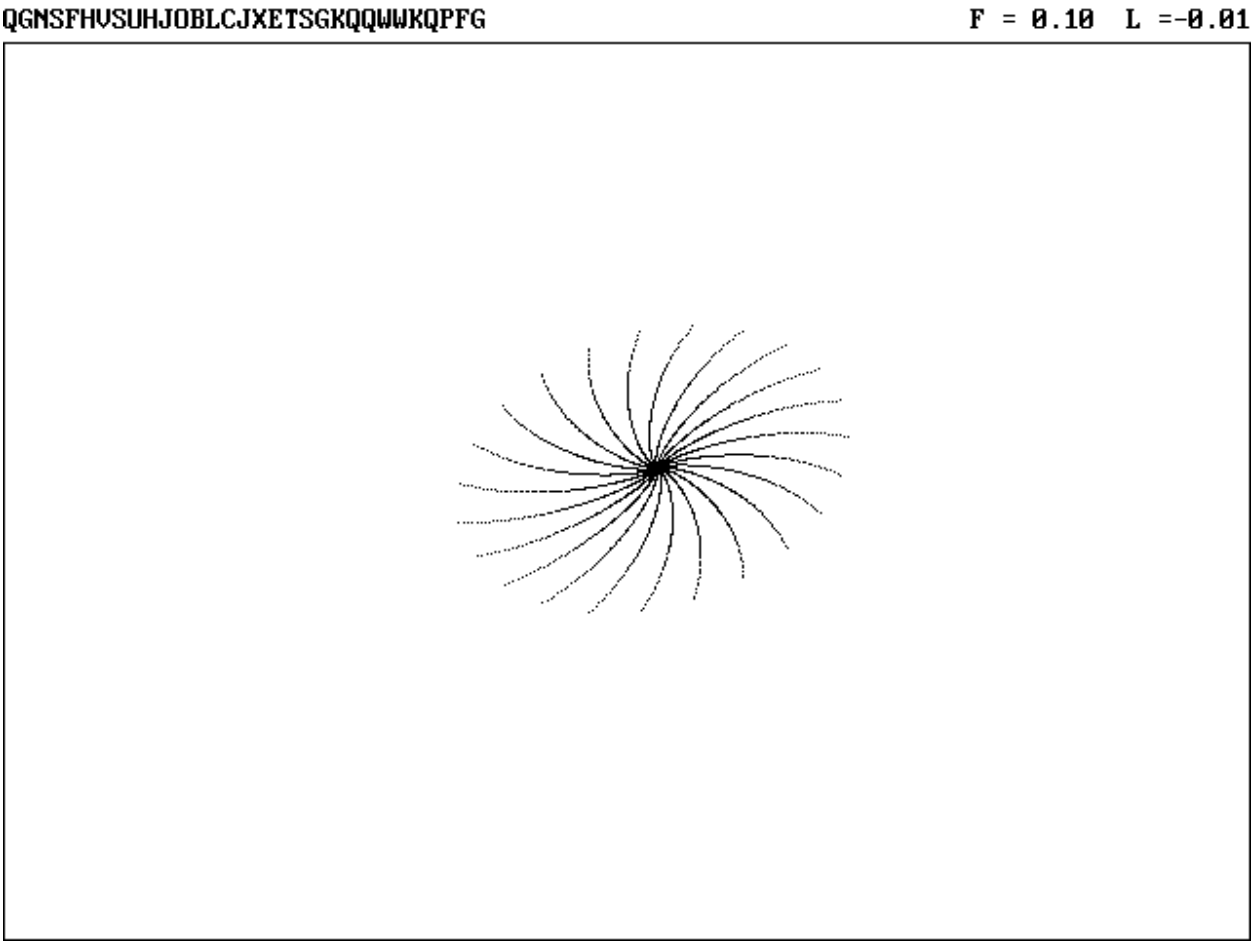


Figure 6-41. Trajectory approaching a point attractor

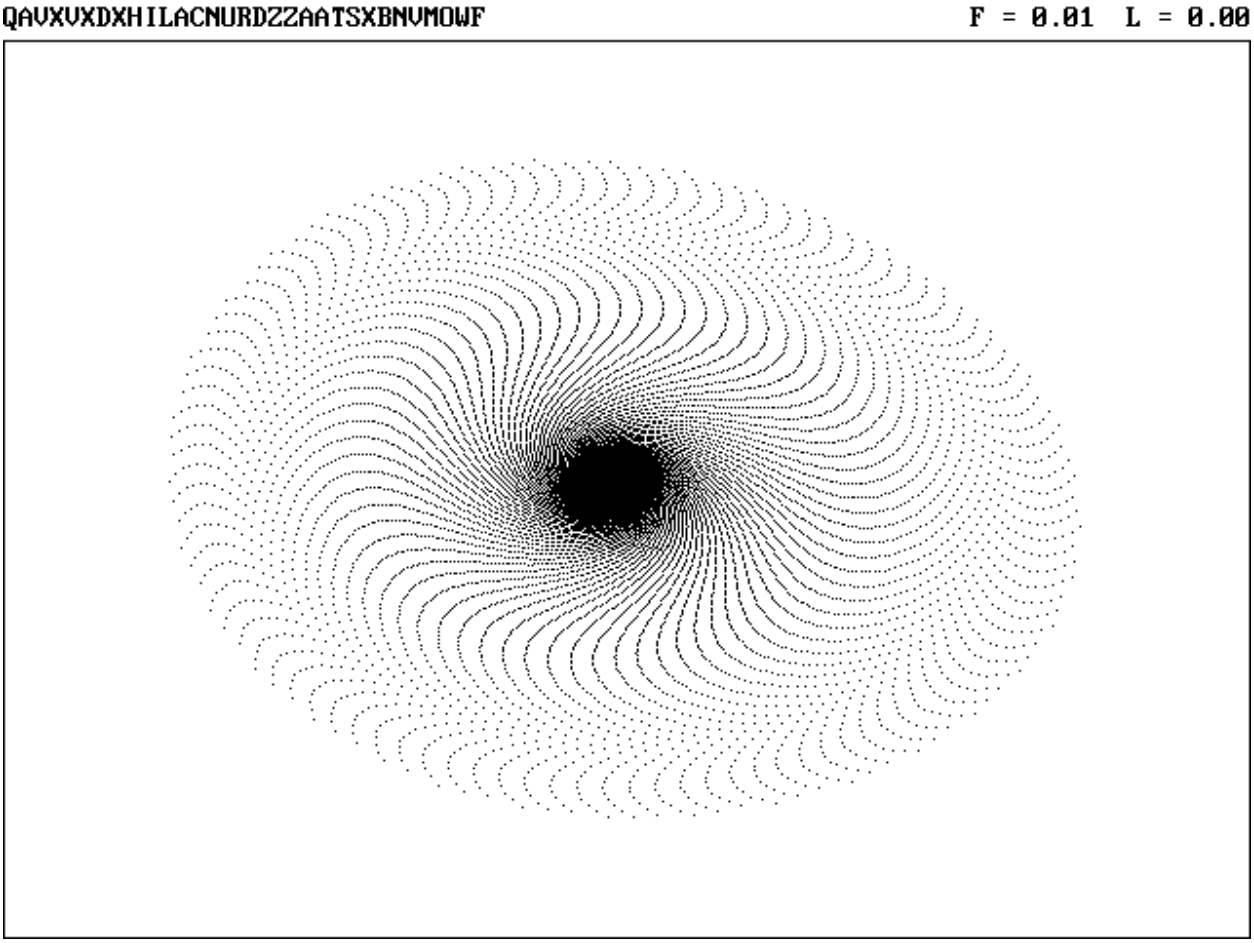
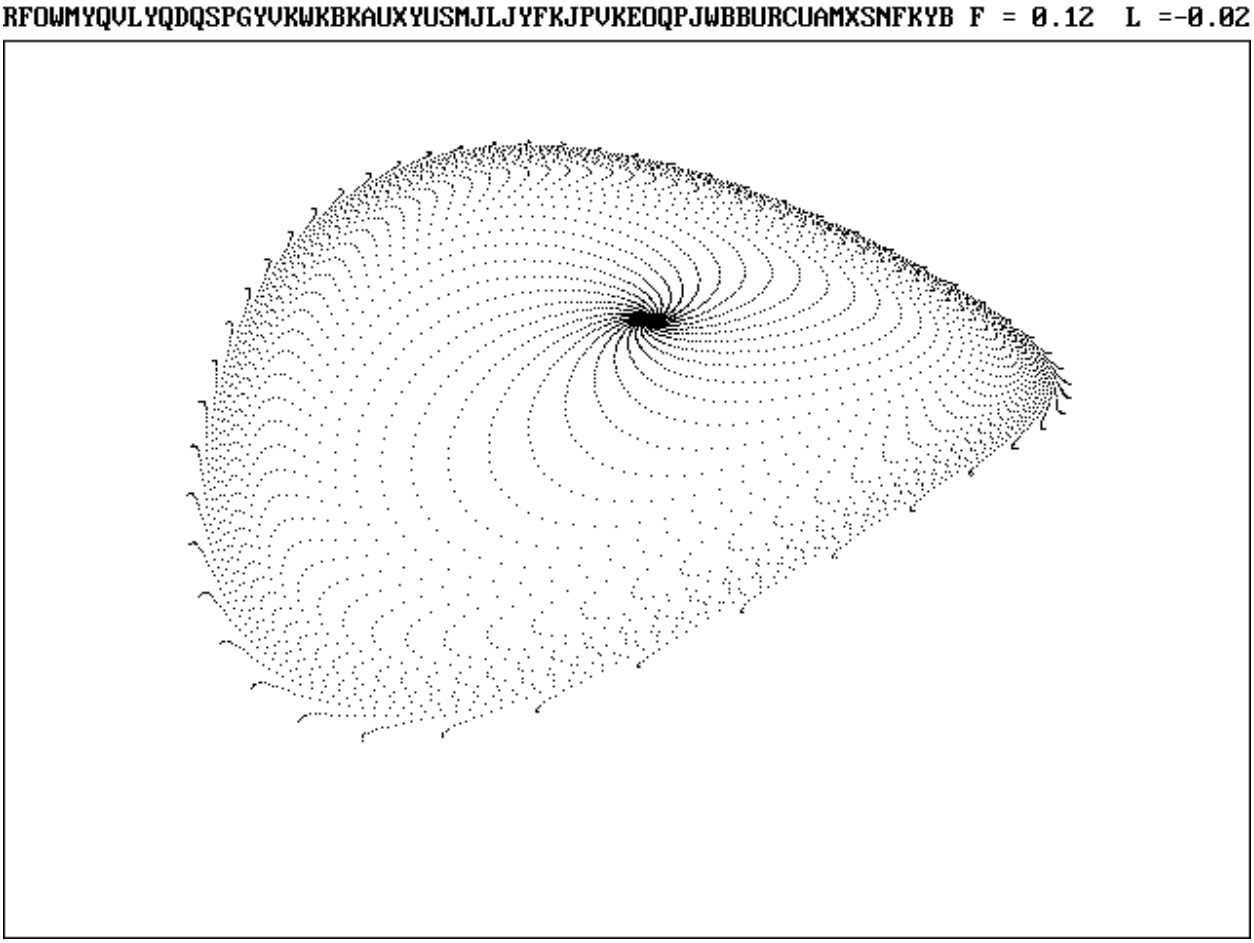


Figure 6-42. Trajectory approaching a point attractor



The point attractor is the simplest type of nonchaotic attractor. It has a dimension of zero. An attractor can also have a dimension of one, which is a line. Such attractors are limit cycles. They correspond to systems that settle into a periodic or cyclic behavior. Consequently, such attractors are also called cyclic attractors.

The simplest differential equations that produce a cyclic trajectory are the equations in Equation 6A. The resulting orbit is a circle in the *XY* plane. This case is not an attractor, however, because every initial condition produces a circular trajectory whose radius is the distance of the initial point from the origin. There is no unique circle to which nearby orbits are attracted, and there is no basin of attraction. Furthermore, if you attempt to display the trajectory using a code such as $QM^5NM^5LM^{18}$, you will find that the orbit is unbounded and spirals outward as if there were a point repellor at the origin. The reason is that our iteration scheme for

approximating the solution of the differential equations is not exact. The errors compound and eventually cause the orbit to be lost.

The simple undamped (frictionless) oscillator is said to be *structurally unstable* because an arbitrarily small perturbation to the equation (such as using the Euler finite difference approximation of Equation 6F) changes the structure of the solution from a closed loop to a never-ending spiral. Note the distinction between an unstable *equation*, in which a small modification of the equation causes a large change in the solution, and an unstable *solution*, in which a small variation of the initial condition away from the equilibrium value causes the solution to move ever farther from equilibrium.

To produce a true limit cycle that is structurally stable, we need a system of equations whose solutions spiral outward from initial conditions in the interior and spiral inward from initial conditions on the exterior of the attractor. A suitable set of such equations is the following:

$$X' = Y + (1 - X^2 - Y^2)X$$

$$Y' = -X + (1 - X^2 - Y^2)Y \qquad \text{(Equation 6H)}$$

The quantity $(1 - X^2 - Y^2)$ plays the role of $-b$ in Equation 6G. Whenever the trajectory lies inside the circle of radius one, it spirals outward, and whenever it lies outside the circle of radius one, it spirals inward. Thus the limit cycle is defined by the circle $X^2 + Y^2 = 1$. There is a point repellor at the origin, and the basin of attraction is the entire *XY* plane. A code that produces such a stable limit cycle, except with a smaller radius is $RMNMAM^3AM^3NM^9LM^2AM^6NMAM^{26}$.

A limit cycle may be either stable or unstable, just like a fixed point. With an unstable limit cycle, nearby orbits move progressively farther from the limit cycle. An unstable limit cycle can be identified in an invertable map or system of ordinary differential equations by running time backwards, in which case the limit cycle becomes stable and attracts rather than repels nearby orbits.

A slightly simpler version of Equation 6H that produces a stable limit cycle, although not a symmetrical one, is the following:

$$X' = Y$$

$$Y' = -X + (1 - X^2)Y \qquad \text{(Equation 6I)}$$

This system is called the *Van der Pol equation*, and it was first used to model vacuum-

tube oscillator circuits, but it has been used in other applications such as the modeling of pulsating stars called *Cephids*. A code for the Van der Pol equation is RM^{11}OM^{9}KM^{2}KM^{6}OM^{28}.

Such limit cycles are characterized by a dimension of one and a Lyapunov exponent of zero. The dimension as approximated by the program will usually not be exactly 1.0 for the reasons discussed in Section 3.4. The Lyapunov exponent is a much better criterion for identifying limit cycles. In a two-dimensional embedding space, as in the previous examples, there are two Lyapunov exponents. The smaller (more negative) of them is the rate at which trajectories with different initial conditions approach the attractor. The larger exponent (the one calculated by the program) is the rate at which two nearby points on the limit cycle separate. For a limit cycle produced by ODEs, this exponent must be zero because points along the trajectory are governed by the same rates of flow, except delayed in time.

In two dimensions, the limit cycles cannot cross, so the most complicated shapes are simple distorted loops. In three or more dimensions, they can wrap around in a complicated tangle like a ball of string, but without ends. Figures 6-43 through 6-50 show a collection of visually interesting limit cycles. They are plotted as stereo pairs so that you can see how the trajectories pass beneath one another.

Figure 6-43. Limit cycle from a three-dimensional quadratic map

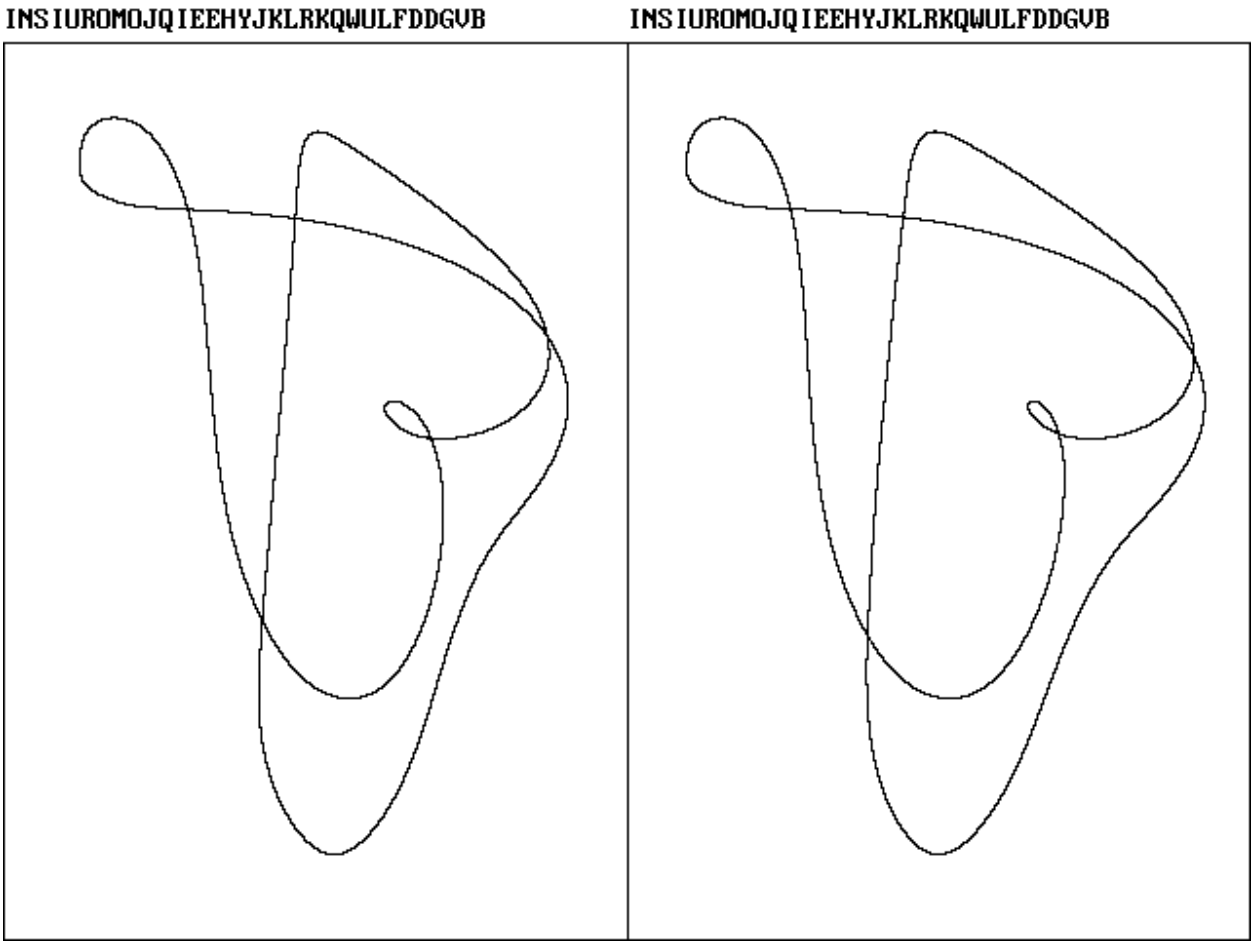


Figure 6-44. Limit cycle from a three-dimensional quadratic ODE

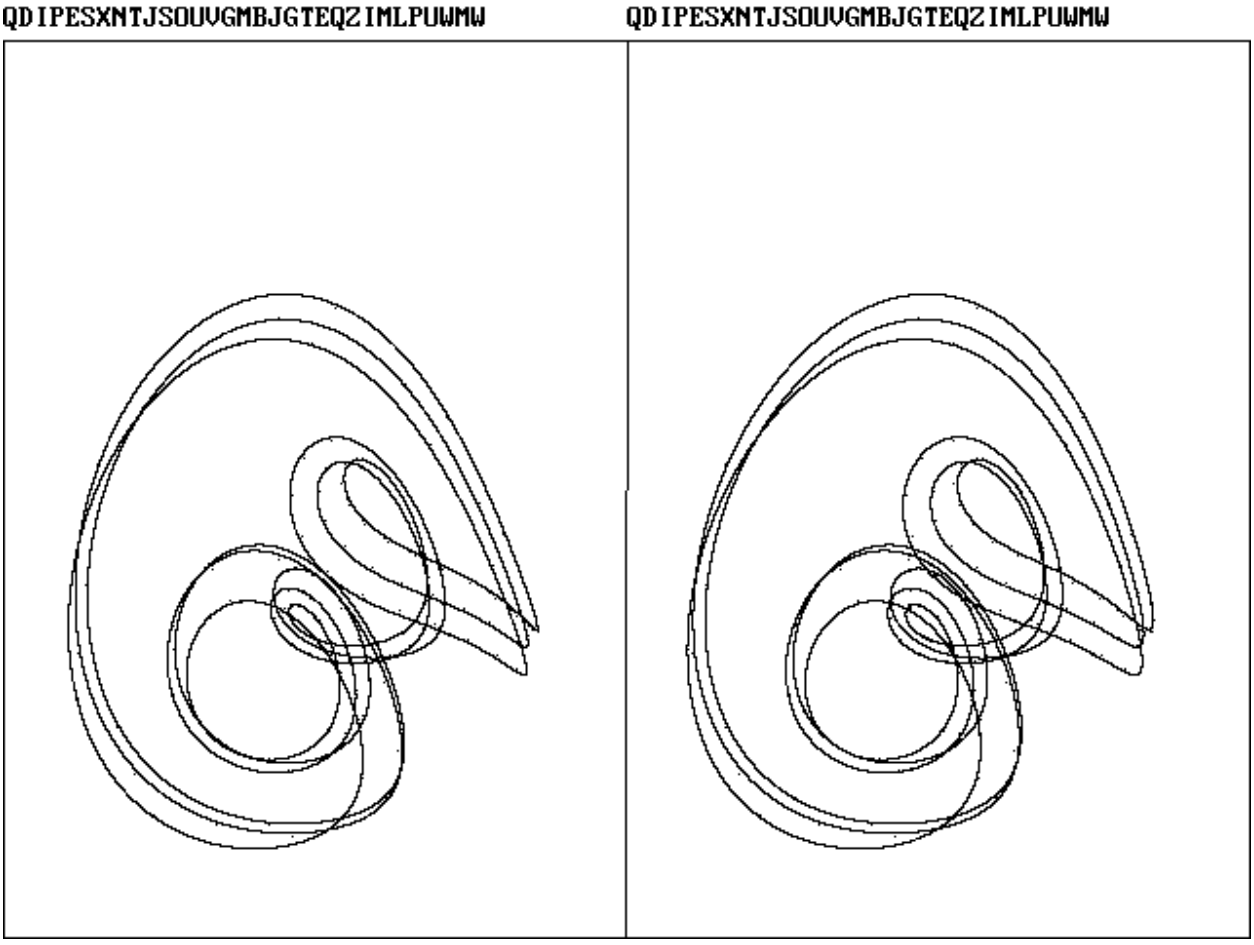


Figure 6-45. Limit cycle from three-dimensional quadratic ODE

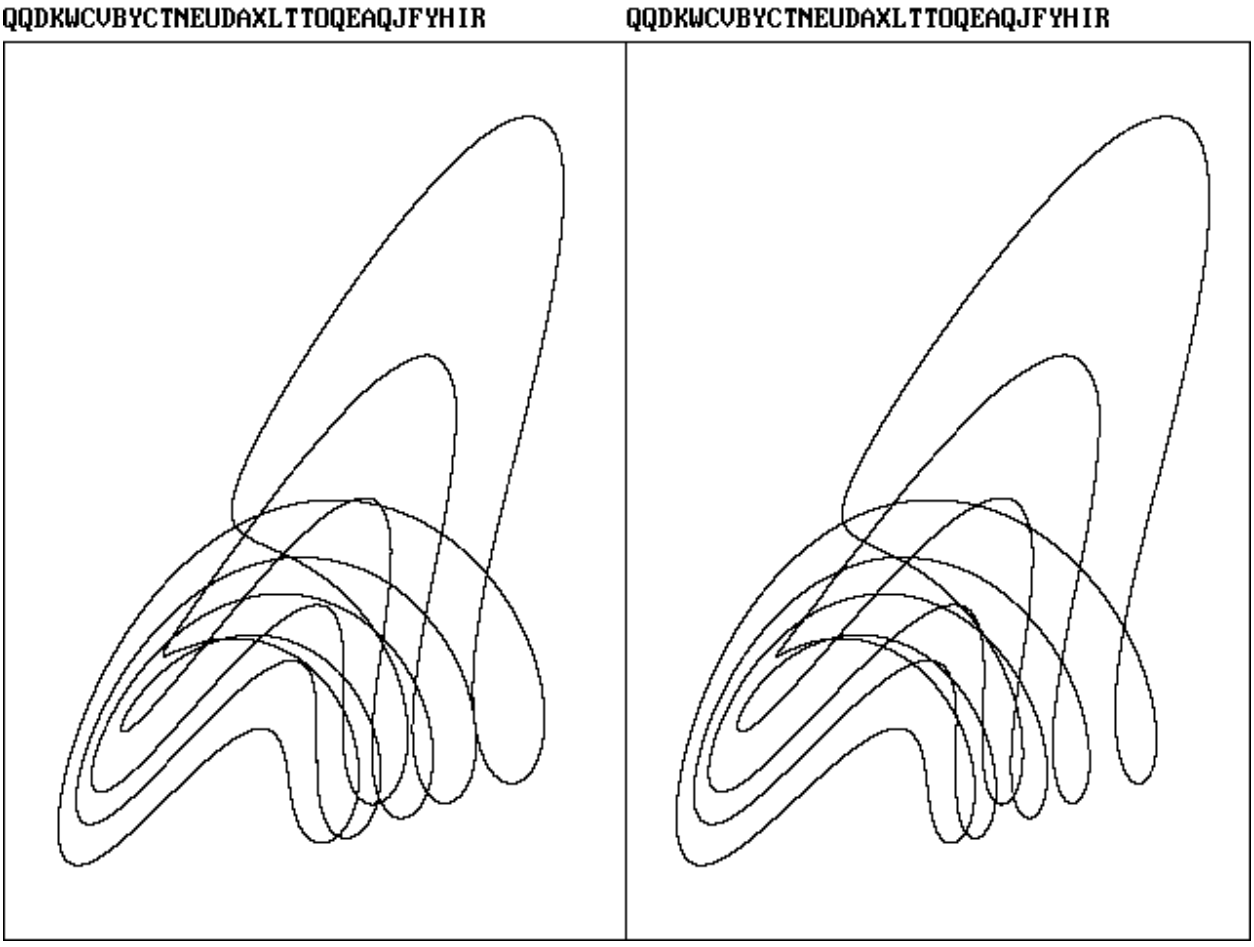


Figure 6-46. Limit cycle from a three-dimensional quadratic ODE

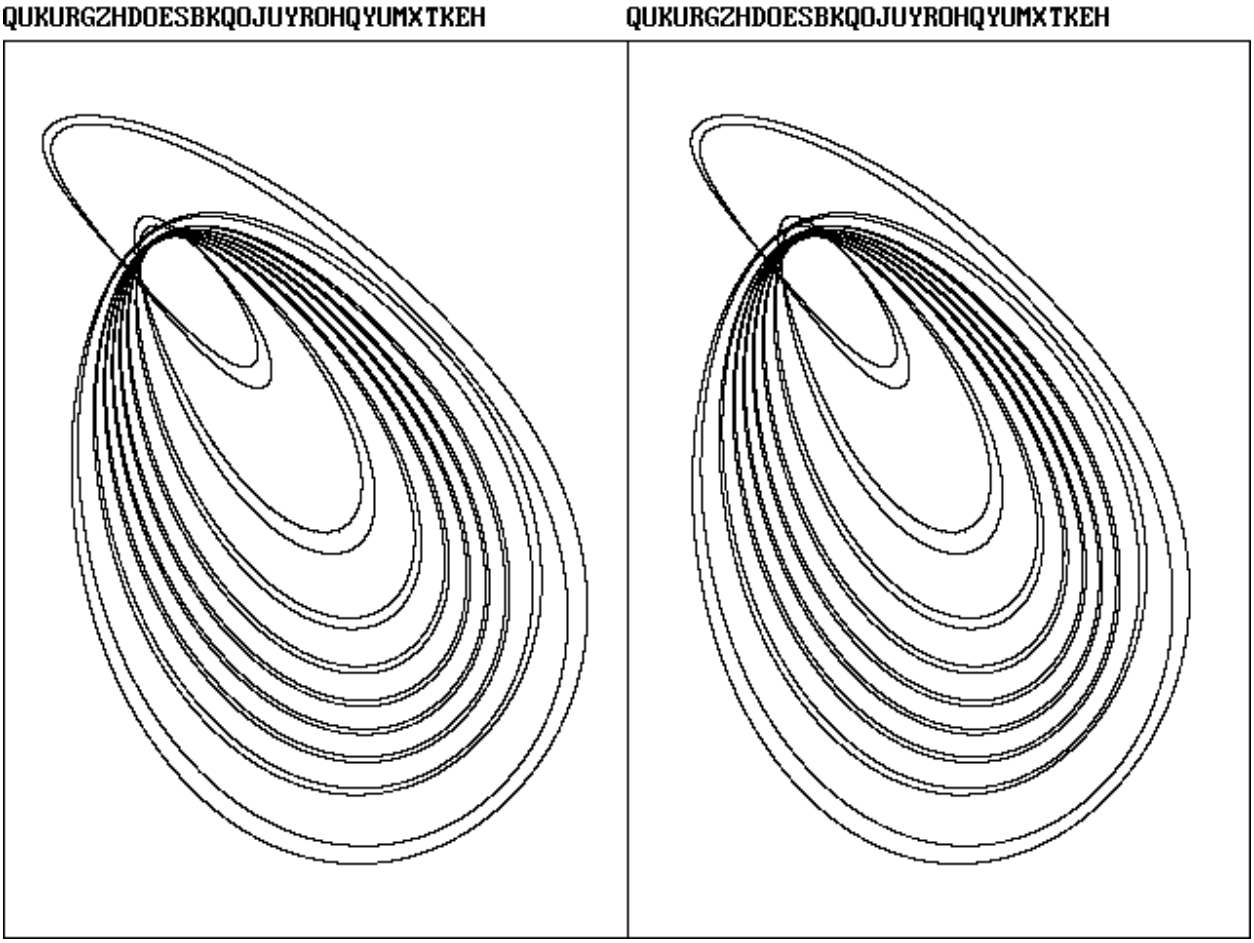


Figure 6-47. Limit cycle from a three-dimensional cubic ODE

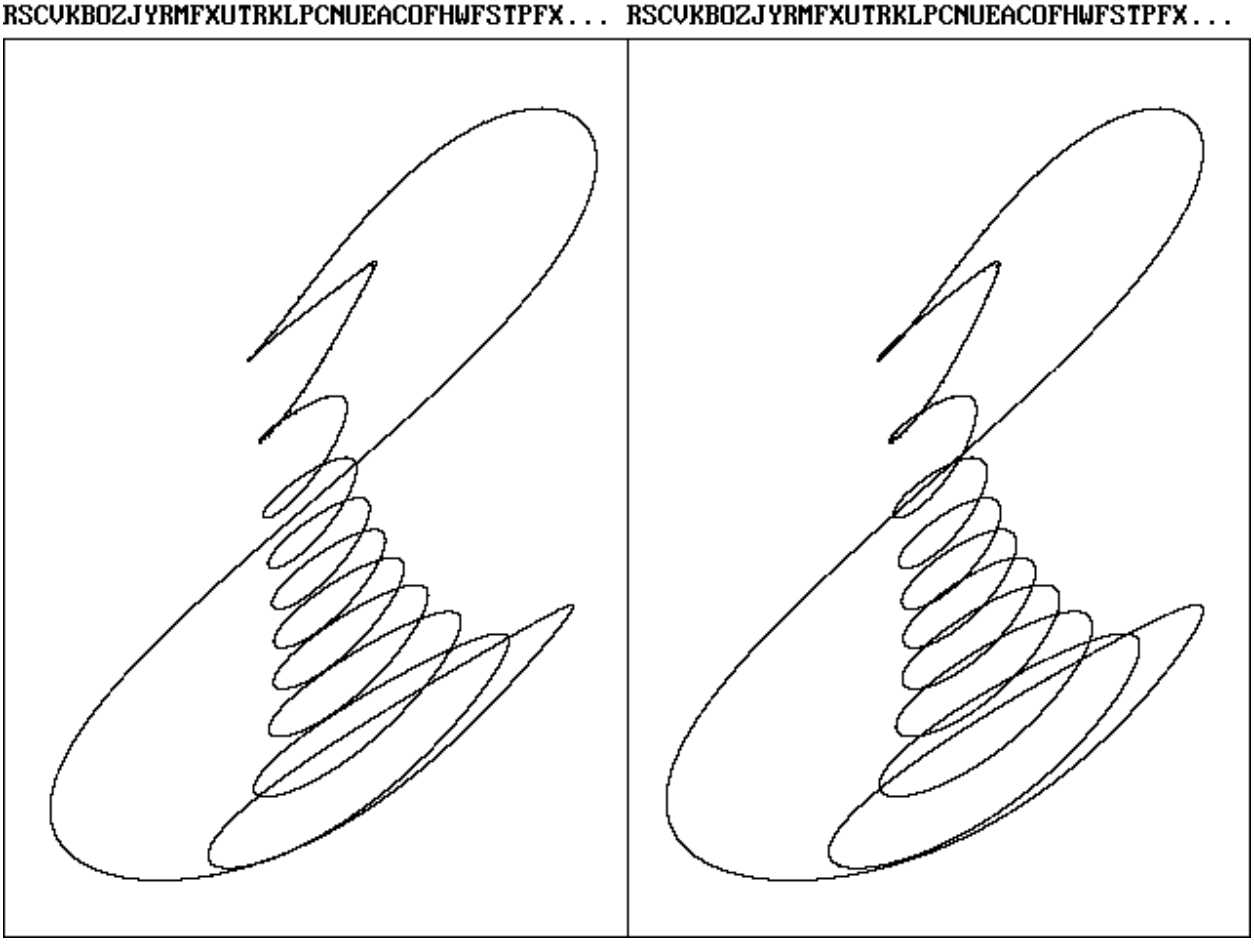


Figure 6-48. Limit cycle from a three-dimensional quintic ODE

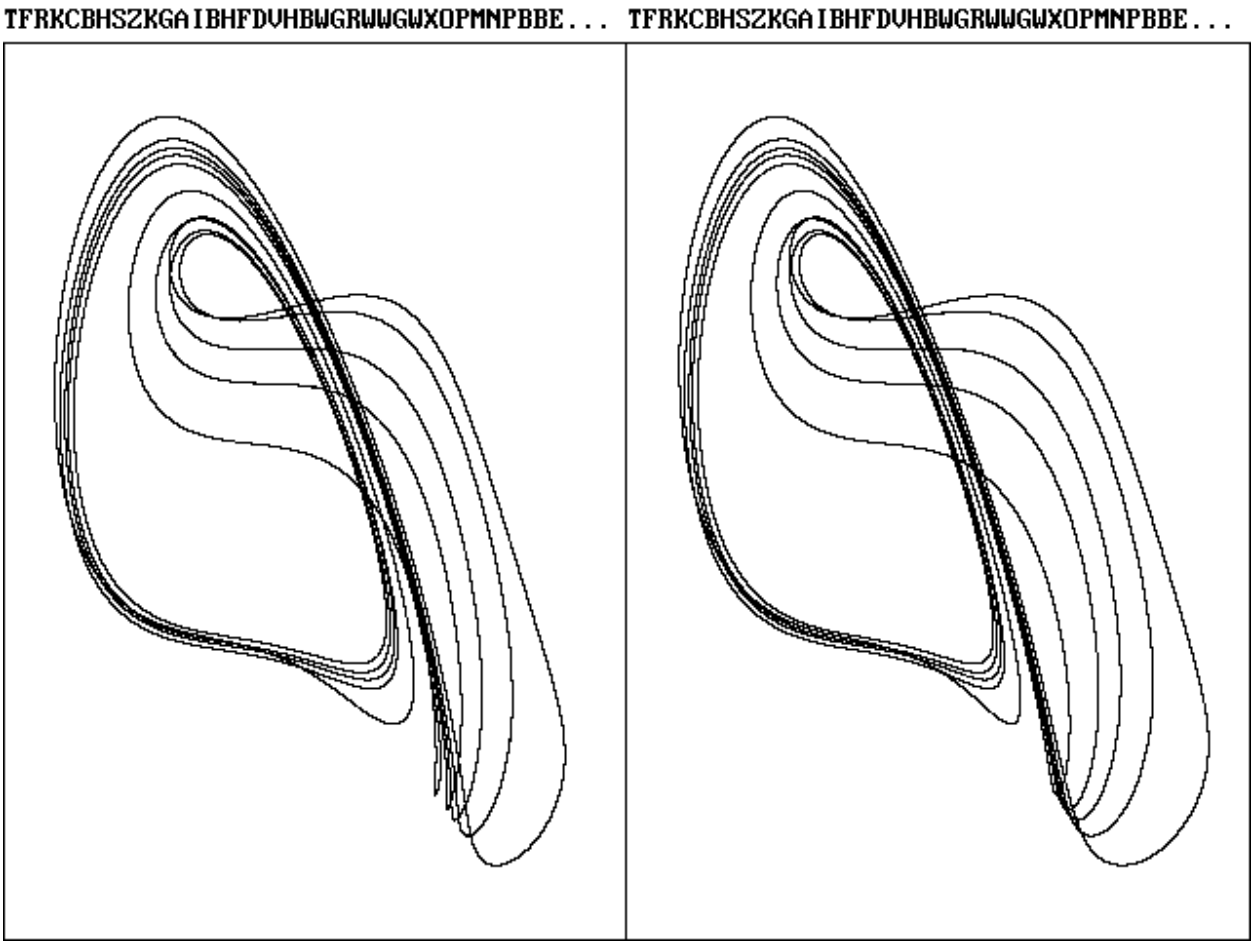


Figure 6-49. Limit cycle from a four-dimensional quadratic ODE

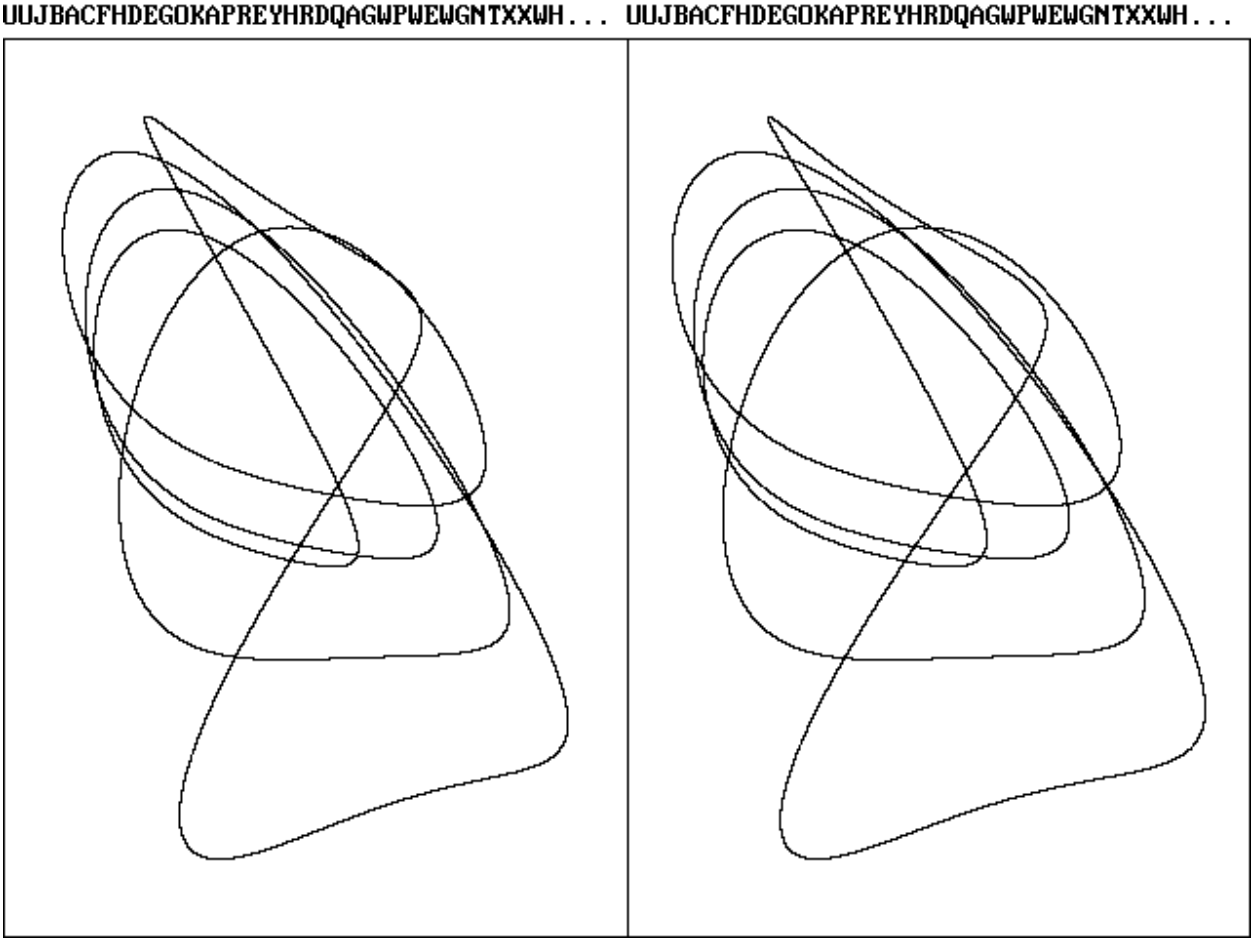
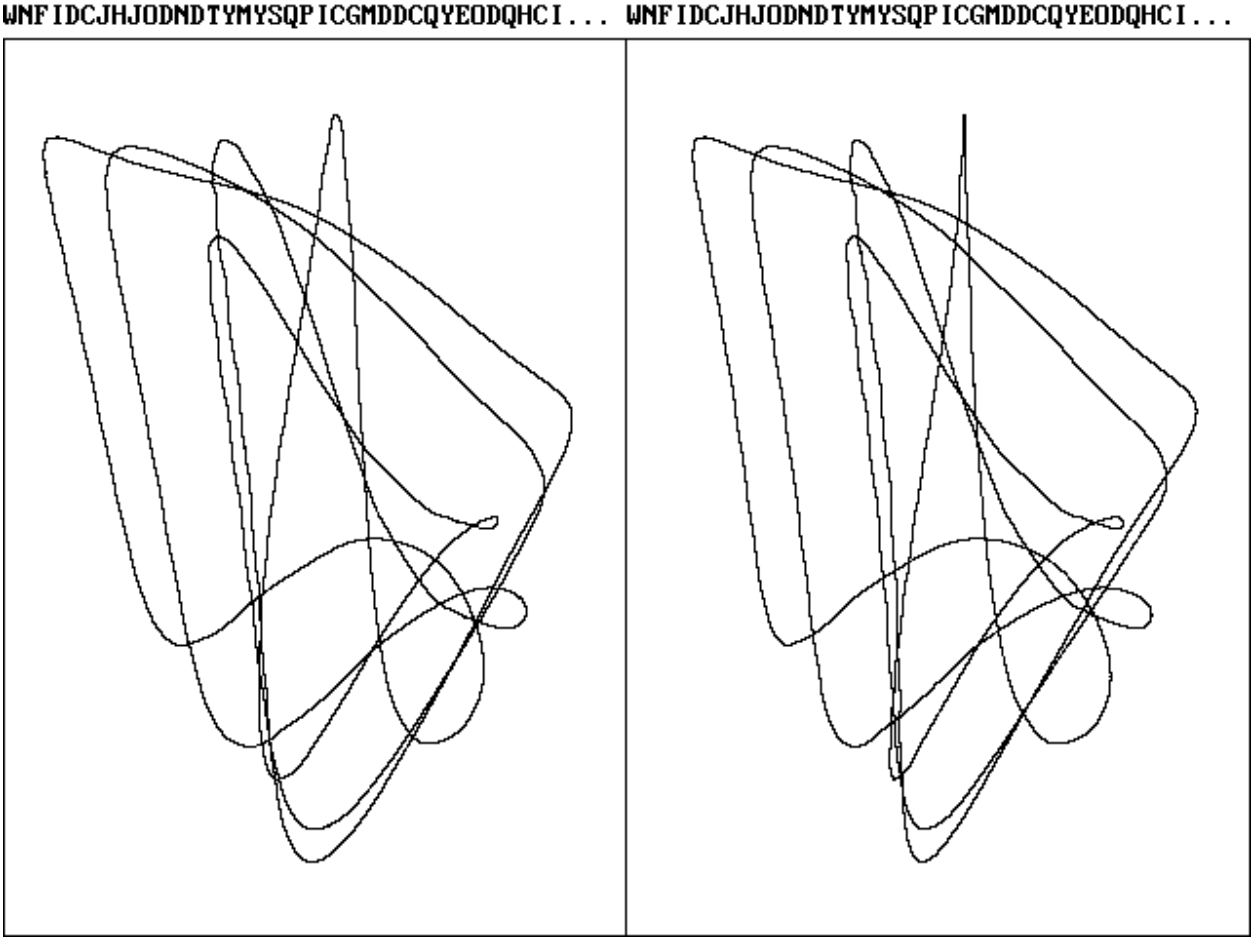


Figure 6-50. Limit cycle from a four-dimensional quartic ODE



As you examine the figures, note that some of the limit cycles, such as the one in Figure 6-43, form knots. You cannot straighten them out into circular loops. By contrast, Figure 6-47 is unknotted. This *knottedness* or *helicity* is an important topological property of an attractor. Some processes in nature tend to conserve helicity, just as mechanical energy is conserved in frictionless motion. Thus when some parameter of the system is changed, the limit cycle may change its size and shape but in such a way that it always links itself in the same way. An example is a magnetic field line in a turbulent conducting fluid such as a plasma of electrically charged particles.

For many of the limit cycles exhibited here, it is very hard to tell whether they are knotted. Even when they appear to be knotted, it is hard to tell whether two cases are knotted in the same way. Such patterns might provide a useful psychological test for one's spatial acuity because they require both depth perception

and a mental dexterity to visualize their shape when untangled as much as possible. See which of the limit cycles in the figures you think are knotted.

6.6 Doughnuts and Coffee Cups

Non-chaotic attractors can be points or lines. They can also be surfaces. Surfaces are two-dimensional manifolds. Perhaps the simplest set of equations whose solution is a trajectory that fills a surface is the following:

$$X' = Y$$

$$Y' = -X$$

$$Z' = aW$$

$$W' = -Z \qquad \text{(Equation 6J)}$$

You will recognize the first two equations as the same as Equation 6A that produces a circle in the *XY* plane. The second two equations produce an ellipse in the *ZW* plane. The two motions are *uncoupled* (*X* and *Y* don't depend on *Z* or *W*; *Z* and *W* don't depend on *X* or *Y*). The parameter *a* is the square of the angular frequency of the second motion. If the square root of *a* is a *rational number* (a ratio of two integers) the trajectory is a closed one-dimensional loop in the four-dimensional embedding space.

If the square root of *a* is irrational, the trajectory fills a two-dimensional toroidal surface (called a *2-torus*). The trajectory winds endlessly around the surface of a dougnut, never intersecting itself. In such a case we say the frequencies (the number of transits per second the long way around and the short way around) are *incommensurate* and that the trajectory is *quasi-periodic*. The sequence never repeats, but it is not chaotic. It is sometimes difficult to distinguish between quasi-periodic and chaotic behavior.

A useful tool for distinguishing between a quasi-periodic and a chaotic attractor is the *power spectrum* of the time series, which has sharp peaks at discrete frequencies for quasi-periodic trajectories but a broad (continuous) spectrum for chaotic trajectories. The power spectrum contains about half of the information required to reconstruct the trajectory; the frequency information is present, but the phase information is lost. Nevertheless, the power spectrum serves as a kind of fingerprint that is very useful in categorizing attractors.

The equation set in Equation 6J has the same problems as Equation 6A. They don't represent an attractor because nearly all initial conditions produce different tori. Furthermore, the tori produced in this way are structurally unstable, just like the circles of Equation 6A. These difficulties can be circumvented by using instead an extension of Equations 6H to produce two uncoupled limit cycles as follows:

$$X' = Y + (1 - X^2 - Y^2)X$$

$$Y' = -X + (1 - X^2 - Y^2)Y$$

$$Z = aW + (1 - Z^2 - W^2)Z$$

$$W = -Z + (1 - Z^2 - W^2)W \qquad \text{(Equation 6K)}$$

A value of a = 2 provides an acceptable irrational frequency ratio, because the square root of 2 cannot be represented as a ratio of two integers. The corresponding trajectory can be generated using the code VMNMAM^{4}AM^{7}NM^{19}LM^{2}AM^{11}NMAM^{42}NMAM^{2}AOM^{28}LM^{2}AM^{2}NMA. A rotated version of the 2-torus in which one loop is in the *XZ* plane and the other is in the *YW* plane is produced by the code VMNMAM^{8}AM^{13}NM^{24}NMAM^{6}AM^{6}OM^{3}LM^{3}AM^{2}0NMAM^{22}LM^{3}AM^{11}NMA.

Two uncoupled limit cycles lie on a torus that is attractive, but it is not technically an attractor; it is called an *invariant manifold*. For an object to be an attractor, it must not only attract nearby trajectories, but most trajectories on it must wander all over it, in which case we say the set is *transitive* and the orbits are *ergodic*. Ergodic orbits produce *mixing*, which means that an orbit starting from anywhere on the attractor eventually comes arbitrarily close to every other point on the attractor. Mixing ensures that an attractor cannot be split into two different attractors, although the attractor need not be connected. Figure 5-11 shows an attractor that is not connected. Thus not all attractive tori are attractors, just as not all attractors are tori.

Tori can be identified in the computer search by a Lyapunov exponent close to zero and a dimension well above one. It is easy to distinguish them visually from limit cycles, which also have Lyapunov exponents close to zero but resemble lines rather than surfaces. A selection of tori projected onto the *XY* plane is shown in Figures 6-51 through 6-60.

Figure 6-51. Torus from a three-dimensional cubic ODE

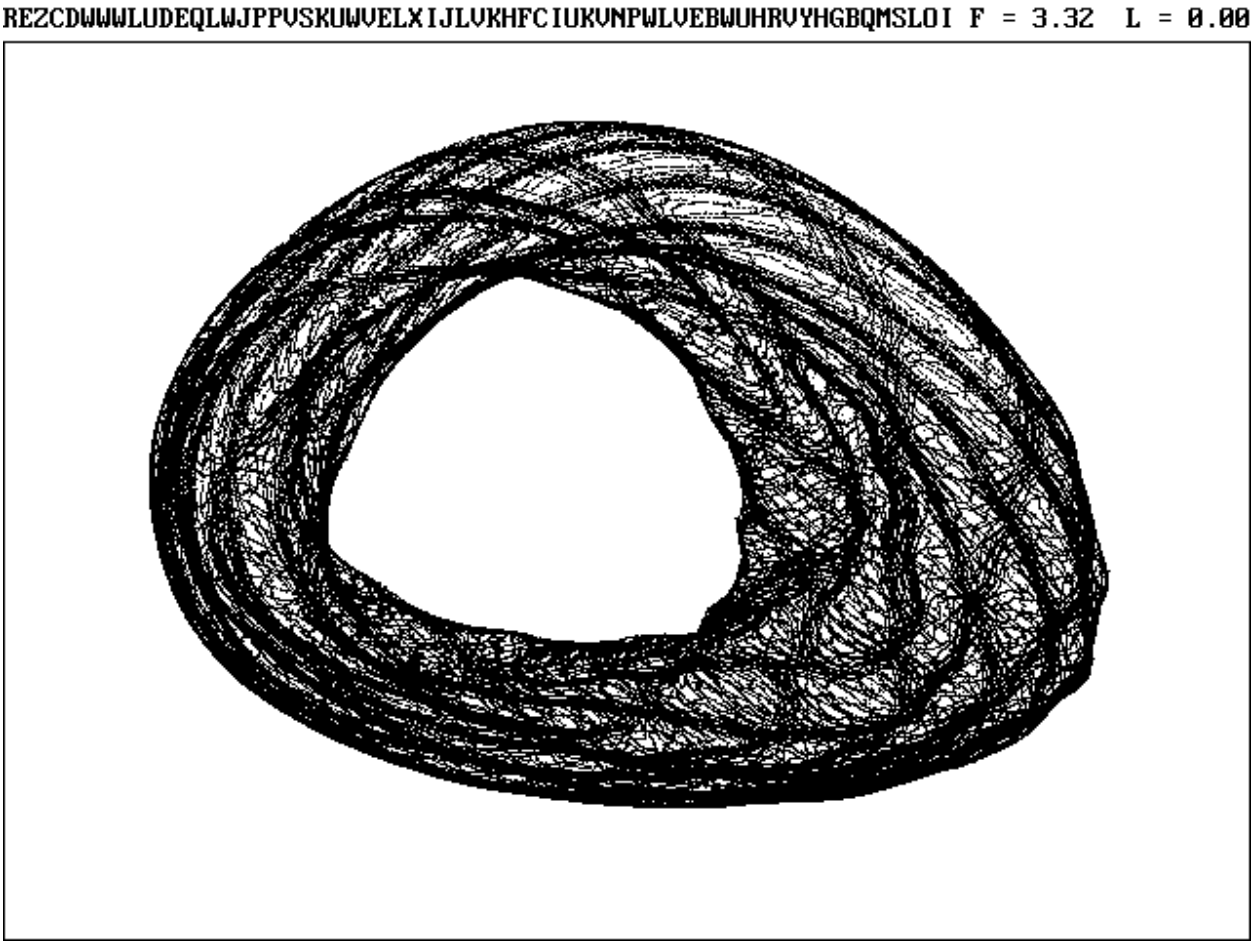


Figure 6-52. Torus from Three-Dimensional Cubic ODE

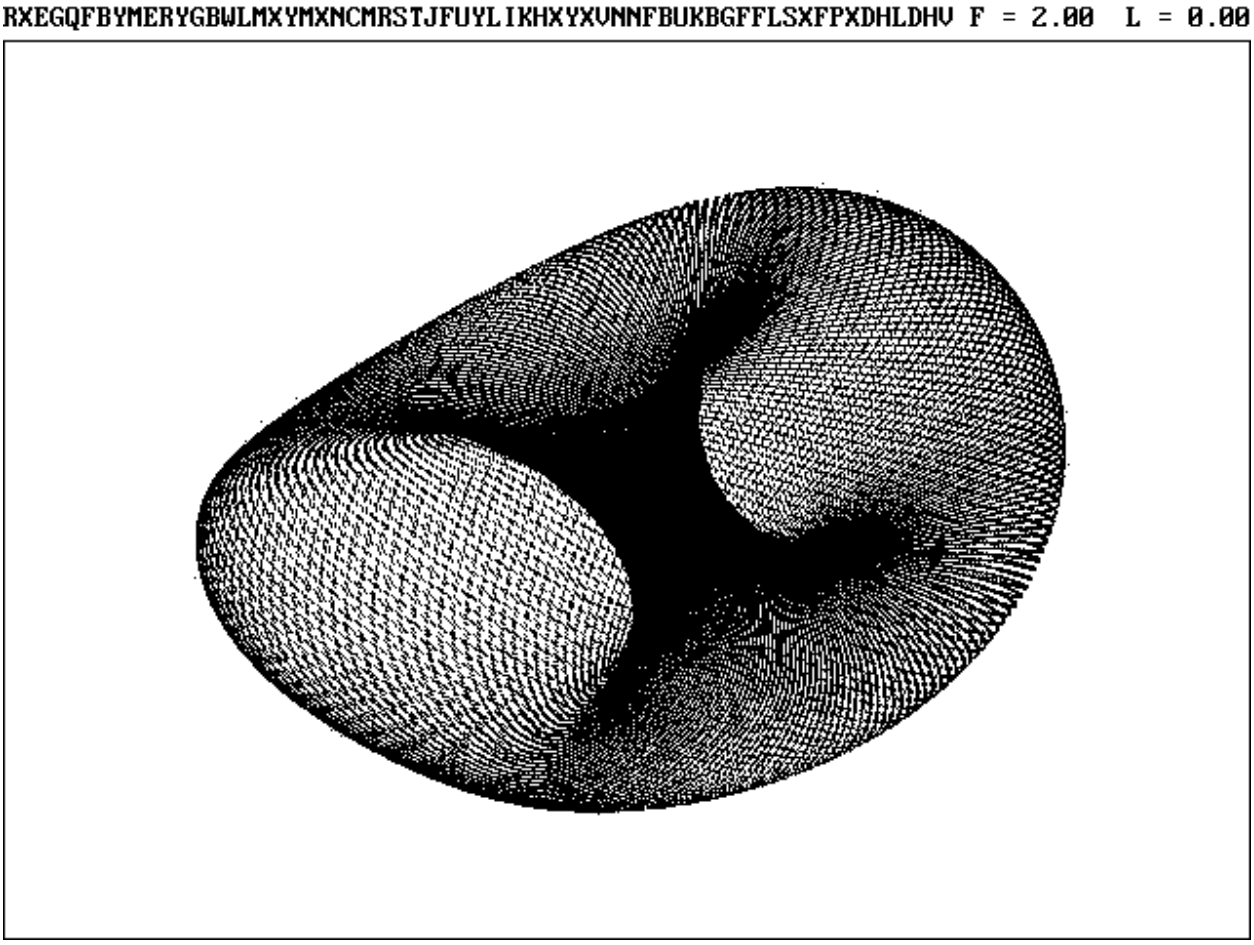


Figure 6-53. Torus from a four-dimensional quadratic ODE

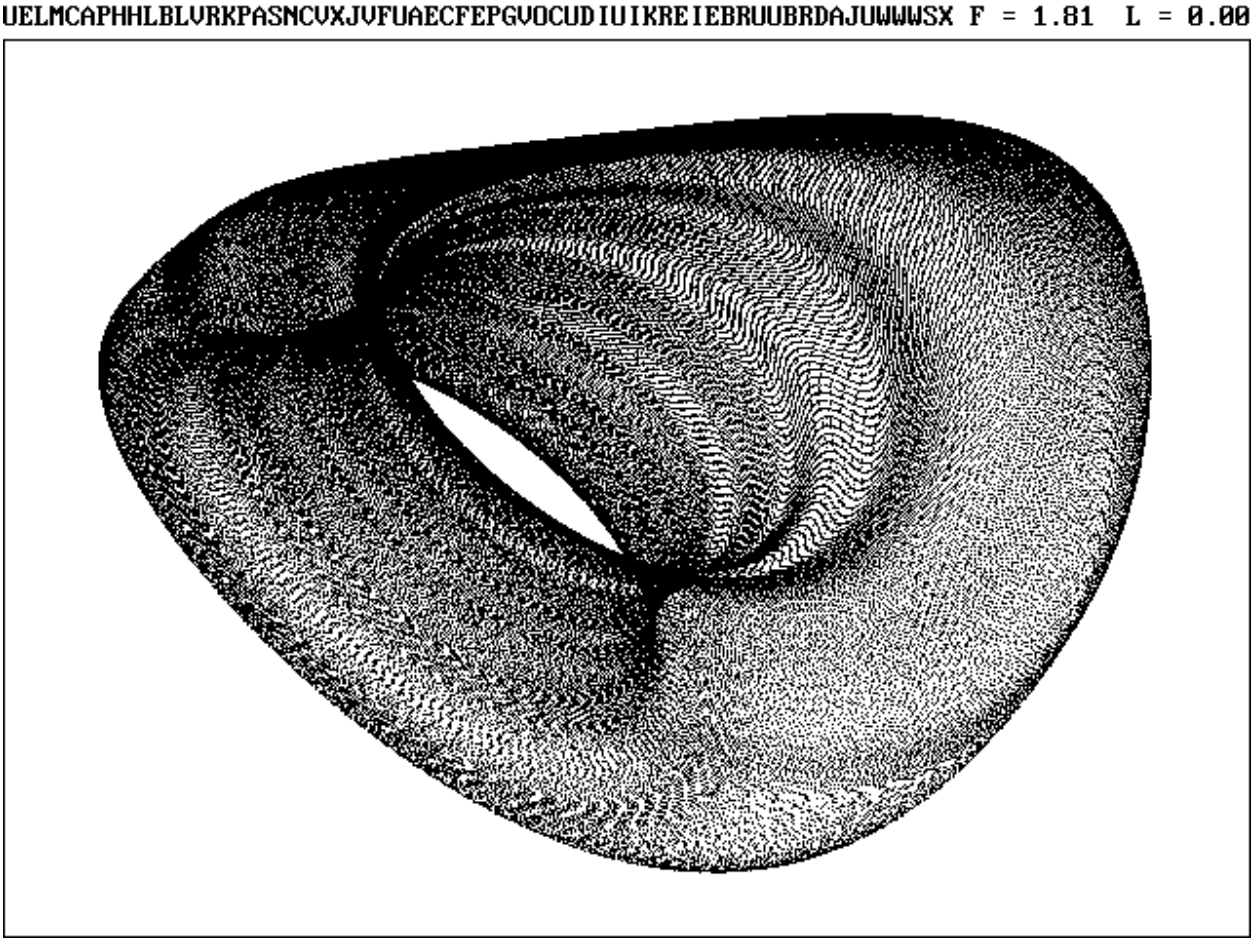


Figure 6-54. Torus from a four-dimensional quadratic ODE

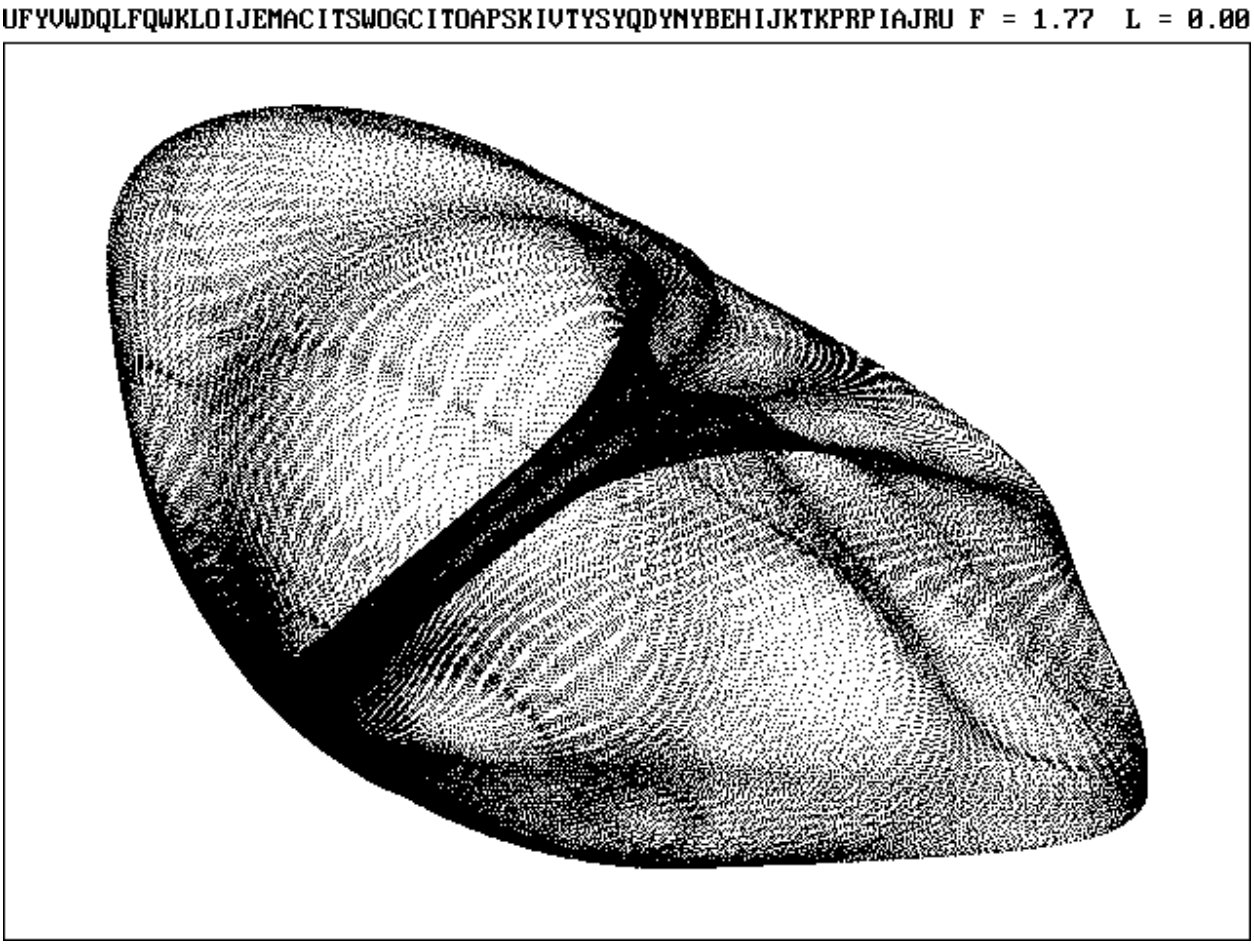


Figure 6-55. Torus from a four-dimensional quadratic ODE

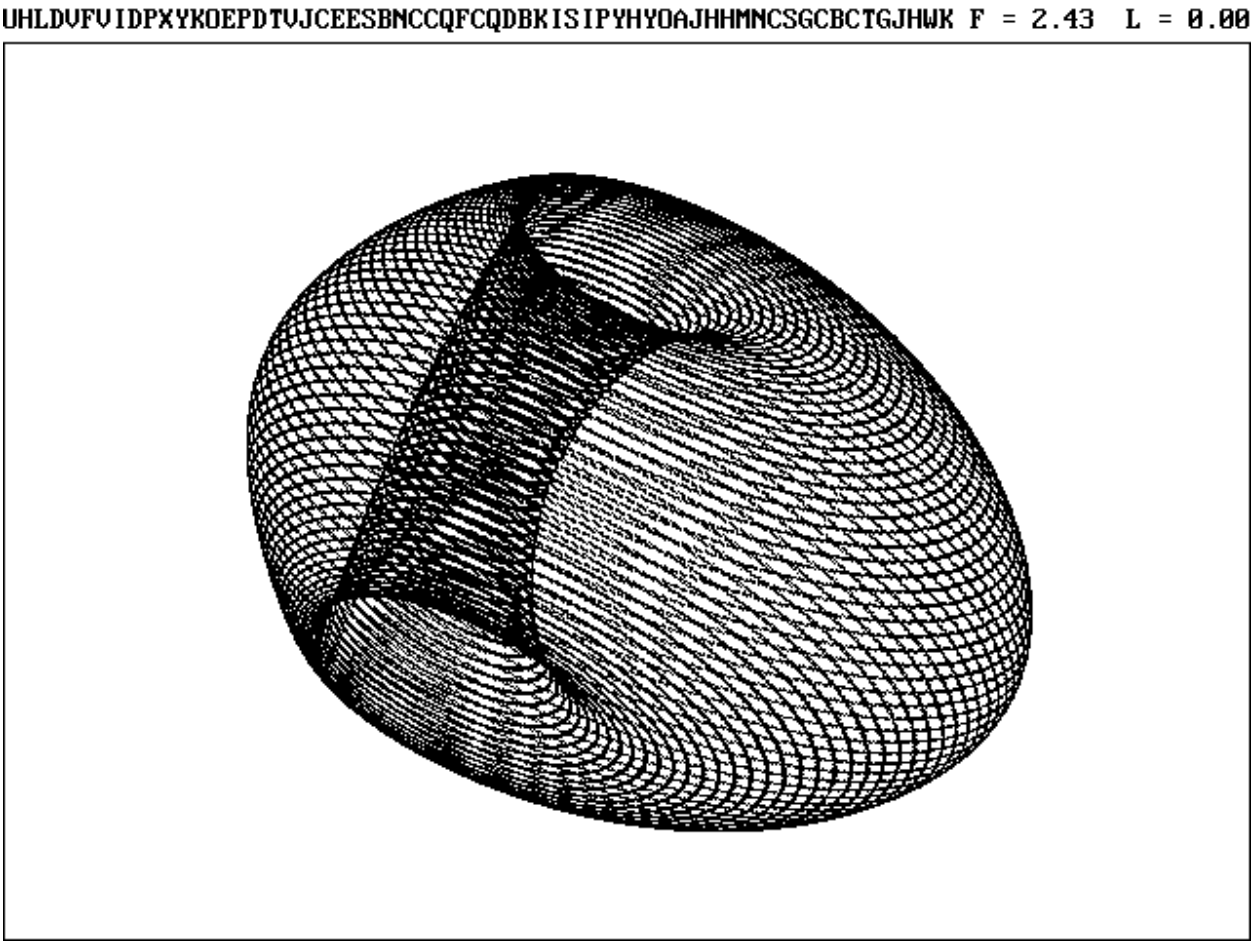


Figure 6-56. Torus from a four-dimensional quadratic ODE

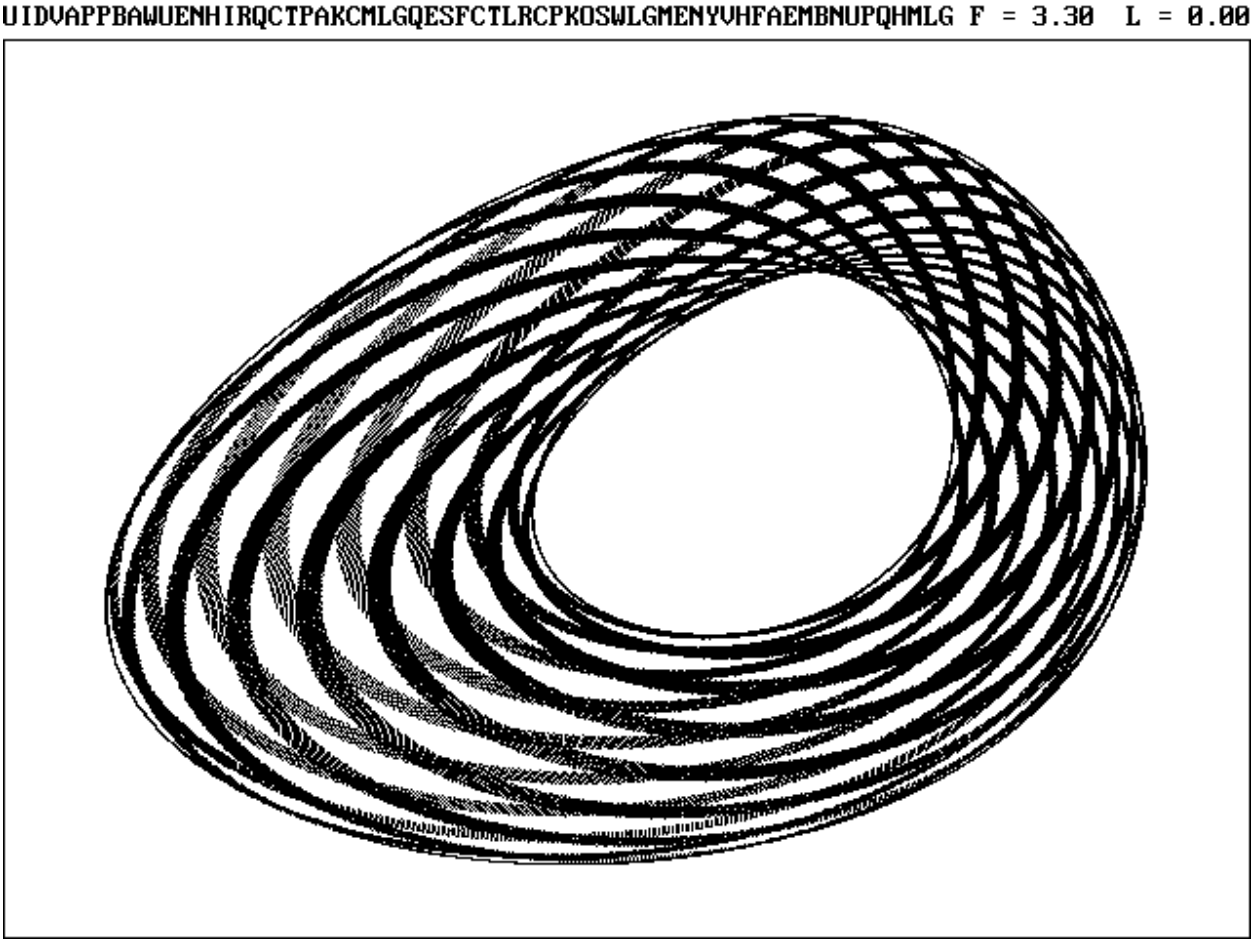


Figure 6-57. Torus from a four-dimensional quadratic ODE

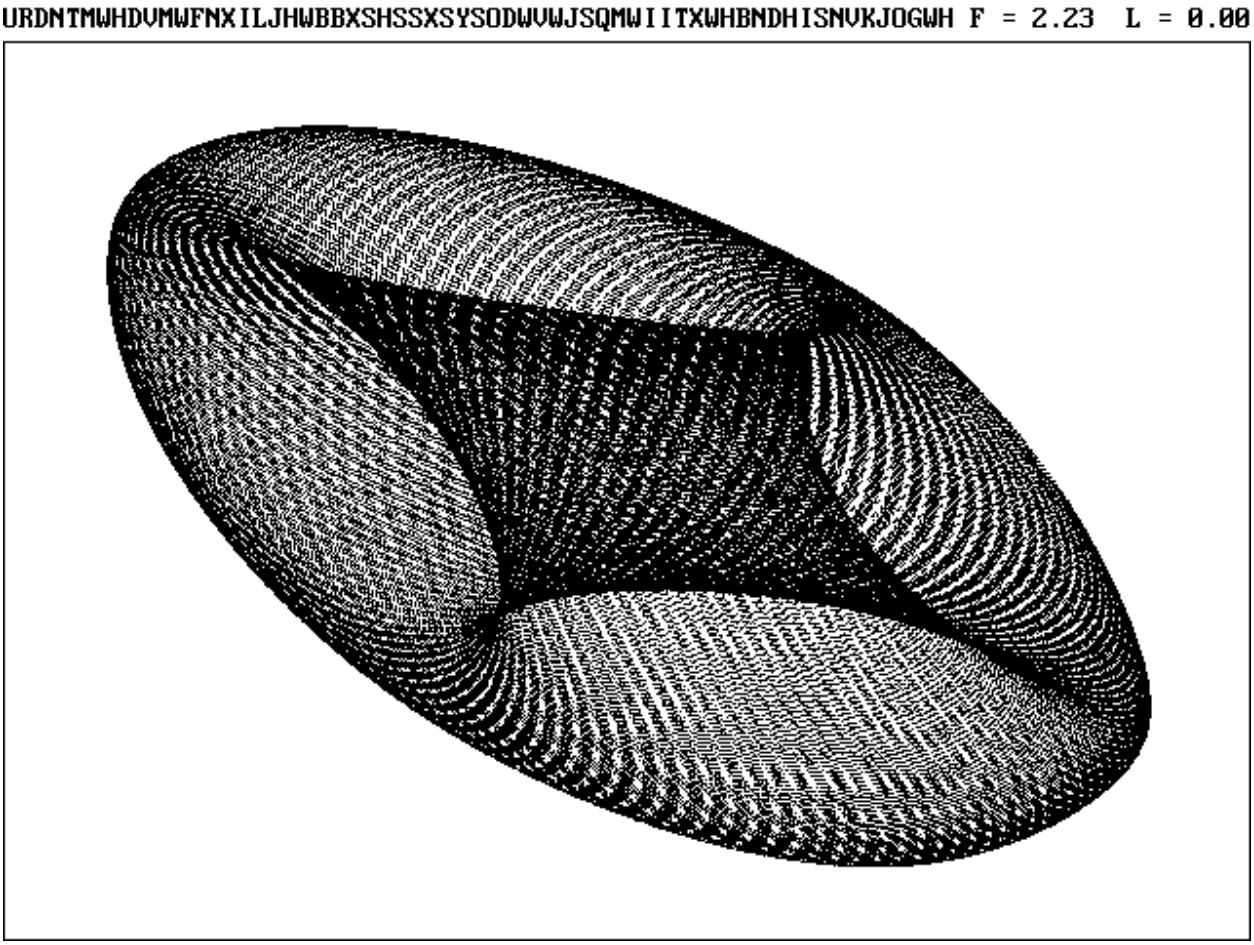


Figure 6-58. Torus from a four-dimensional quadratic ODE

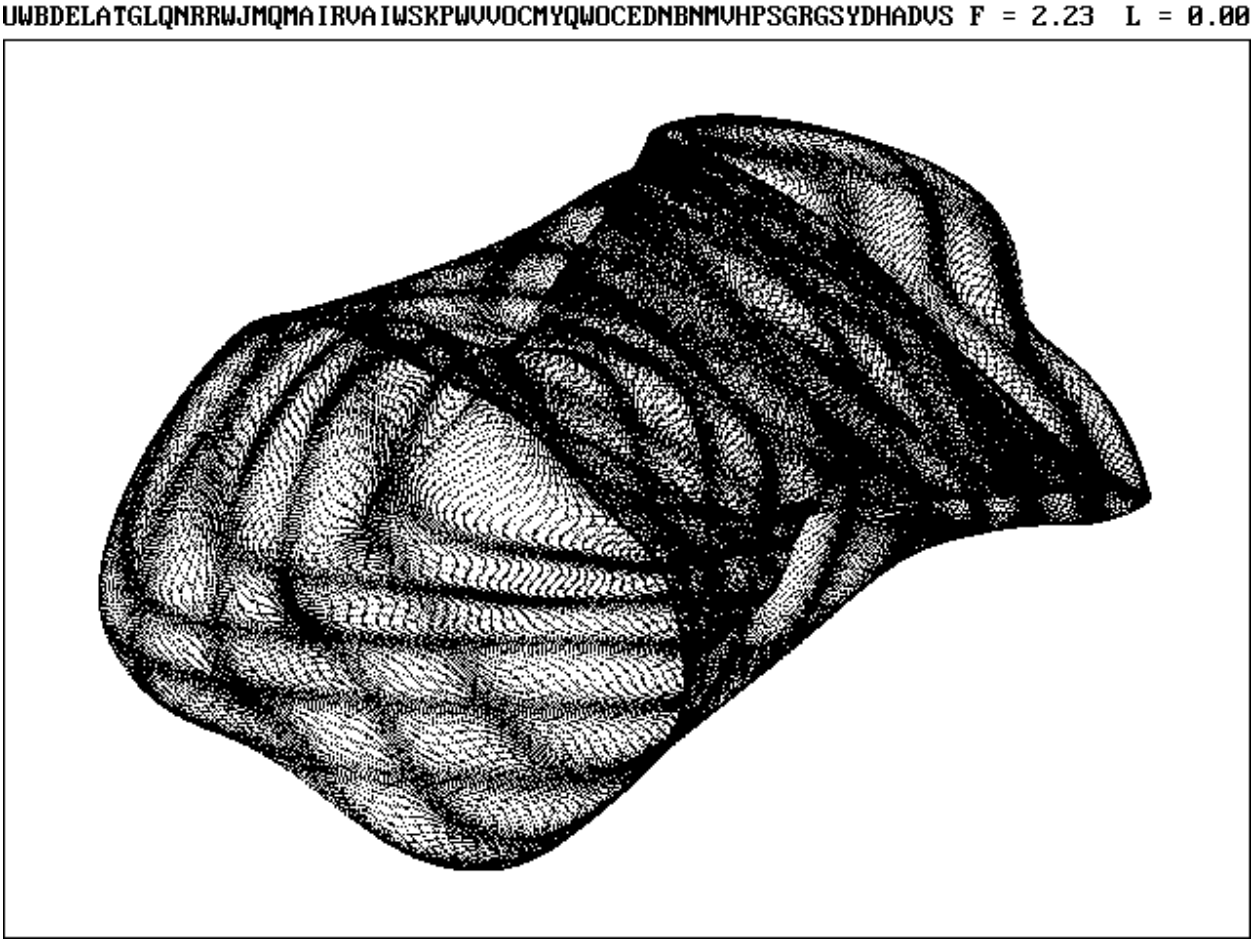


Figure 6-59. Torus from a four-dimensional cubic ODE

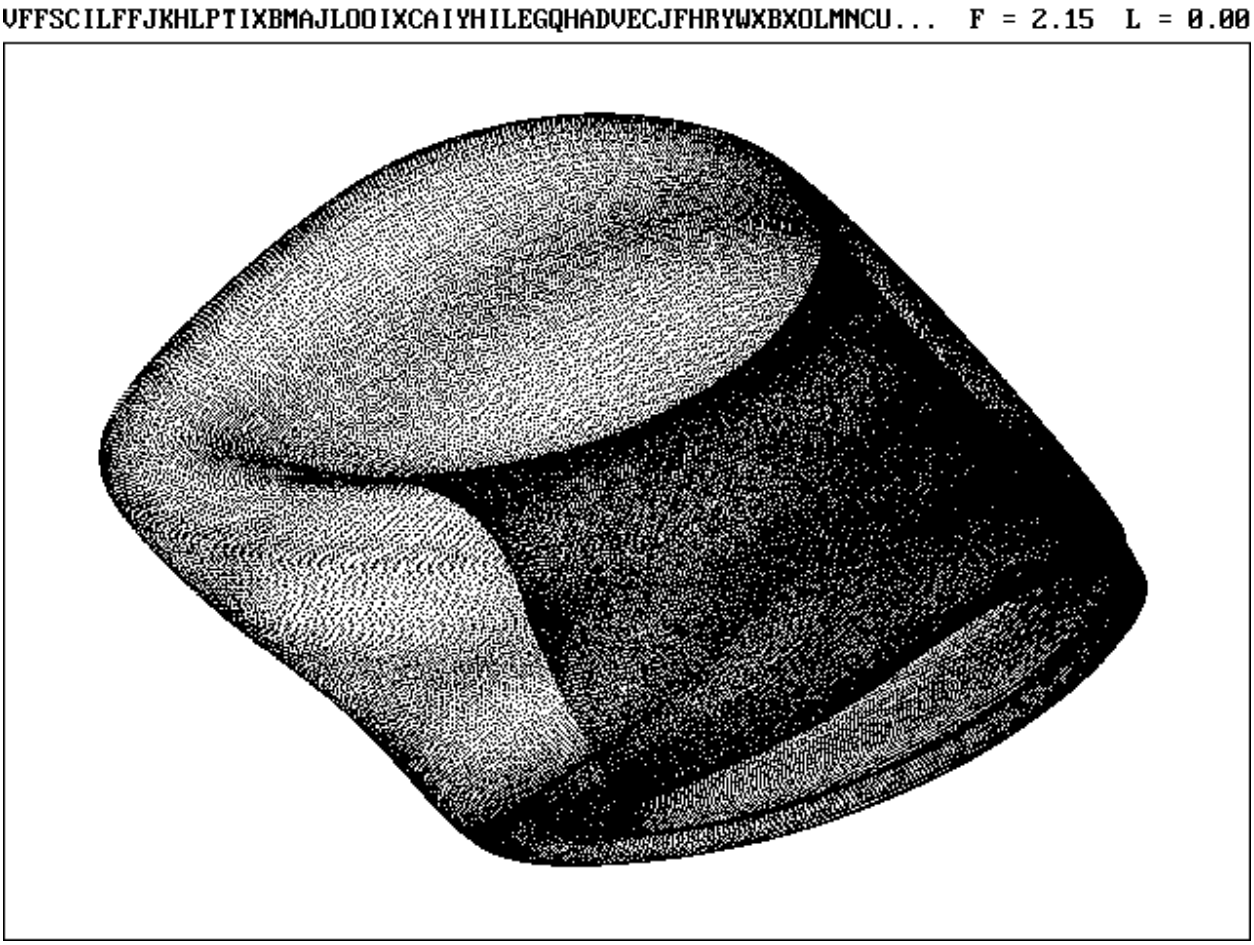
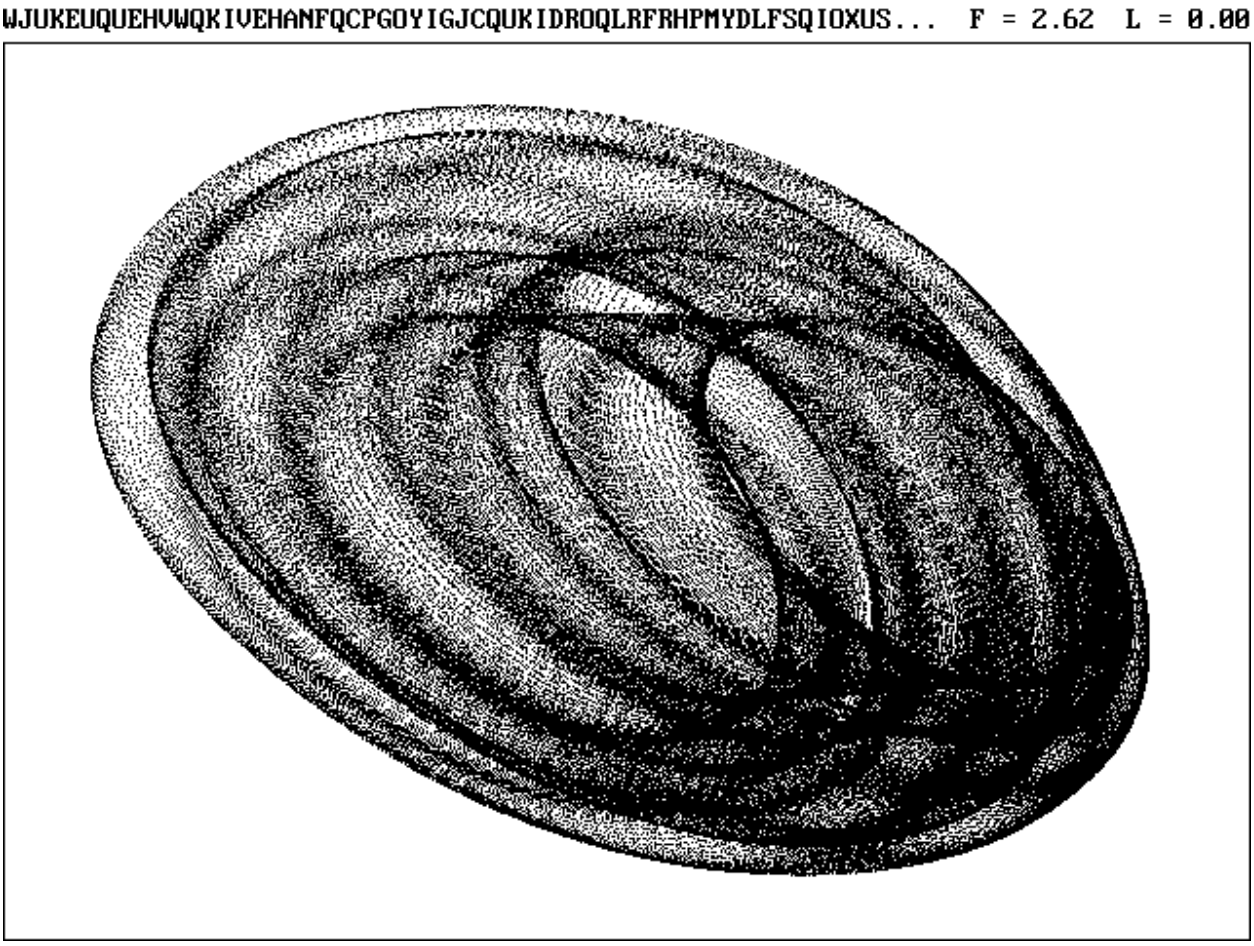


Figure 6-60. Torus from a four-dimensional quartic ODE



Most of the attractors shown in the figures look like tori in the sense that you can see or imagine the hole in the doughnut. However, it is important to understand that, just as not all limit cycles are circles, not all 2-tori look like doughnuts. They are topologically equivalent in the sense that there is a "rubber-sheet" deformation (called a *homeomorphism*) that maps them into a doughnut. A coffee cup, for example, is topologically equivalent to a doughnut as long as the handle is unbroken.

Those cases that are not obviously equivalent to a simple torus are distorted by the fact that they are viewed projected onto the *XY* plane or because they are rotated at an awkward angle. Also note that most of these tori are embedded in a four-dimensional space, so it is even more difficult to grasp their shape from a two-dimensional projection. You might want to display them using some of the advanced visualization techniques provided by the program.

It is possible, though difficult, to produce a *3-torus* in a four-dimensional embedding space. A 3-torus is a generalization of a 2-torus. It is hard to visualize. It involves three mutually incommensurate frequencies. It is characterized by a dimension of three and a largest Lyapunov exponent of zero. Some of the attractors in the figures seem to be 3-tori according to their calculated dimension. However, the calculation is not sufficiently precise to distinguish unambiguously between a 2-torus and a 3-torus. It is necessary to search embedding dimensions greater than four to have a good chance of finding 3-tori.

Dynamical systems whose trajectories lie on a 3-torus or other hypertori of even higher dimensions are difficult to observe in nature. The reason is that such attractors can be perturbed by an arbitrarily small change to the system that causes them to become strange attractors. According to *Peixoto's theorem* (which strictly applies only to compact, orientable manifolds), 2-tori tend to be structurally stable, while 3-tori and higher are structurally unstable. Thus it appears that complicated deterministic systems that exhibit nontrivial behavior are well represented by the strange attractors that constitute the subject of this book.

Chapter 7

Further Fascinating Functions

For a system of equations to exhibit chaos, the equations must contain at least one nonlinear term, that is, a term that is not simply proportional to one of the variables. In all the preceding examples, the nonlinearity involved simple polynomials. Such polynomials are capable of modeling an enormous variety of physical phenomena. Virtually all nonlinear functions can be approximated by polynomials with sufficiently many terms. However, by limiting the polynomials to fifth order, we have missed many interesting possibilities. In this chapter we examine a few of these possibilities and suggest others that you might want to explore on your own.

7.1 Steps and Tents

Perhaps the simplest nonlinear function is the *absolute value*, which is denoted by $|X|$ and programmed in BASIC with the command ABS(*X*). The absolute value of *X* is the magnitude of *X* without regard to its sign. For example, if *X* is -6 then $|X|$ is 6. It is a nonlinear function because a graph of $|X|$ versus *X* is a V-shaped curve with its notch at the origin rather than a straight line as would result if $|X|$ were proportional to *X*. By adding linear terms, the V can be rotated to resemble an L or a staircase step. Computers can evaluate ABS(*X*) very quickly, since they only need to discard the sign.

An example of a one-dimensional chaotic map that involves $|X|$ is the *tent map*, so called because its graph is an inverted V. A tent map that maps the interval -1 to 1 back onto itself is

$$X_{n+1} = 1 - 2|X_n| \qquad \text{(Equation 7A)}$$

The behavior of Equation 7A is very similar to the behavior of the logistic equation (Equation 1C) with $R = 4$, which maps the interval 0 to 1 back onto itself. A mapping that returns a set onto itself is called an *endomorphism*.

Since one-dimensional maps tend not to be very interesting visually, we can generalize Equation 7A to two dimensions as follows:

$$X_{n+1} = a_1 + a_2X_n + a_3Y_n + a_4|X_n| + a_5|Y_n|$$

$$Y_{n+1} = a_6 + a_7X_n + a_8Y_n + a_9|X_n| + a_{10}|Y_n| \qquad \text{(Equation 7B)}$$

This form is analogous to the general two-dimensional quadratic map in Equation 3B.

To make things a little more interesting, we can add two more dimensions (*Z* and *W*) to take advantage of the visualization techniques that we have previously developed. However, to keep things simple, we will demand that *X* and *Y* not depend on *Z* or *W*. They are just along for the ride, so to speak. The dynamical behavior is determined only by *X* and *Y*. We can choose any convenient equation for *Z* and for *W*. One possibility is to evaluate *Z* from *X* and *Y* according to

$$Z_{n+1} = X_n^2 + Y_n^2 \qquad \text{(Equation 7C)}$$

Thus *Z* is the square of the distance of the previous iterate from the origin. You might want to experiment with other forms.

For the fourth dimension (*W*), we will do something completely different. We will arrange for *W* to increase linearly with the iteration number. Thus *W* becomes the time coordinate in four-dimensional space-time.

The program modifications required to extend the computer search to such cases are shown in **PROG22**. Codes for this case begin with the letter Y.

PROG22. Changes required in PROG21 to search for special functions of the Y type

```
1000 REM SPECIAL FUNCTION SEARCH (Steps and Tents)

1090 ODE% = 2                    'System is special function Y


1710 IF ODE% > 1 THEN GOSUB 6200: GOTO 2020     'Special function


2670    IF ODE% > 1 THEN CODE$ = CHR$(87 + ODE%)


3650 IF ODE% > 1 THEN D% = ODE% + 5
```

```
3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 7): T% = 1

3690 IF D% > 6 THEN ODE% = D% - 5: D% = 4: GOTO 3710



4290    IF ODE% > 1 THEN PRINT TAB(27); "D: System is 4-D special map "; CHR$(87
+ ODE%); " ": GOTO 4320



4720 IF D% > 6 THEN ODE% = ASC(LEFT$(CODE$, 1)) - 87: D% = 4: GOSUB 6200: GOTO
4770



6200 REM Special function definitions

6210 ZNEW = X * X + Y * Y       'Default 3rd and 4th dimension

6220 WNEW = (N - 100) / 900: IF N > 1000 THEN WNEW = (N - 1000) / (NMAX - 1000)

6230 IF ODE% <> 2 THEN GOTO 6270

6240    M% = 10

6250    XNEW = A(1) + A(2) * X + A(3) * Y + A(4) * ABS(X) + A(5) * ABS(Y)

6260    YNEW = A(6) + A(7) * X + A(8) * Y + A(9) * ABS(X) + A(10) * ABS(Y)

6270 RETURN
```

Examples of attractors produced by **PROG22** are shown in Figures 7-1 through 7-8. They are displayed as projections onto the *XY* plane to let you observe the higher dimensional representations for the first time on your computer screen. Note that these attractors differ from the cases produced by polynomials in that they tend to have sharp angular corners. The one in Figure 7-3 is not an attractor but is an example of an area-preserving system sometimes called the *gingerbread man* because of its shape.

Figure 7-1. Four-dimensional special map Y

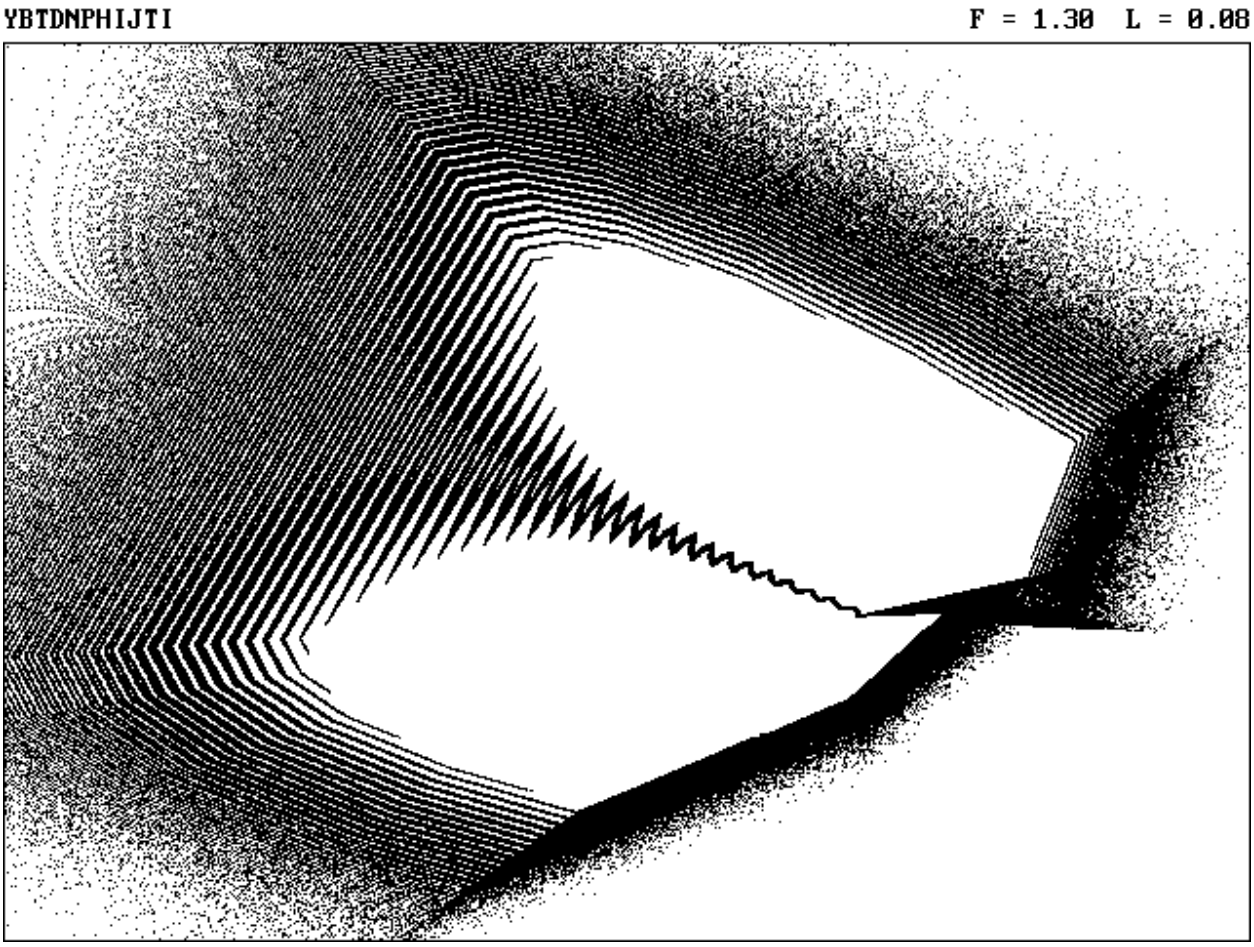


Figure 7-2. Four-dimensional special map Y

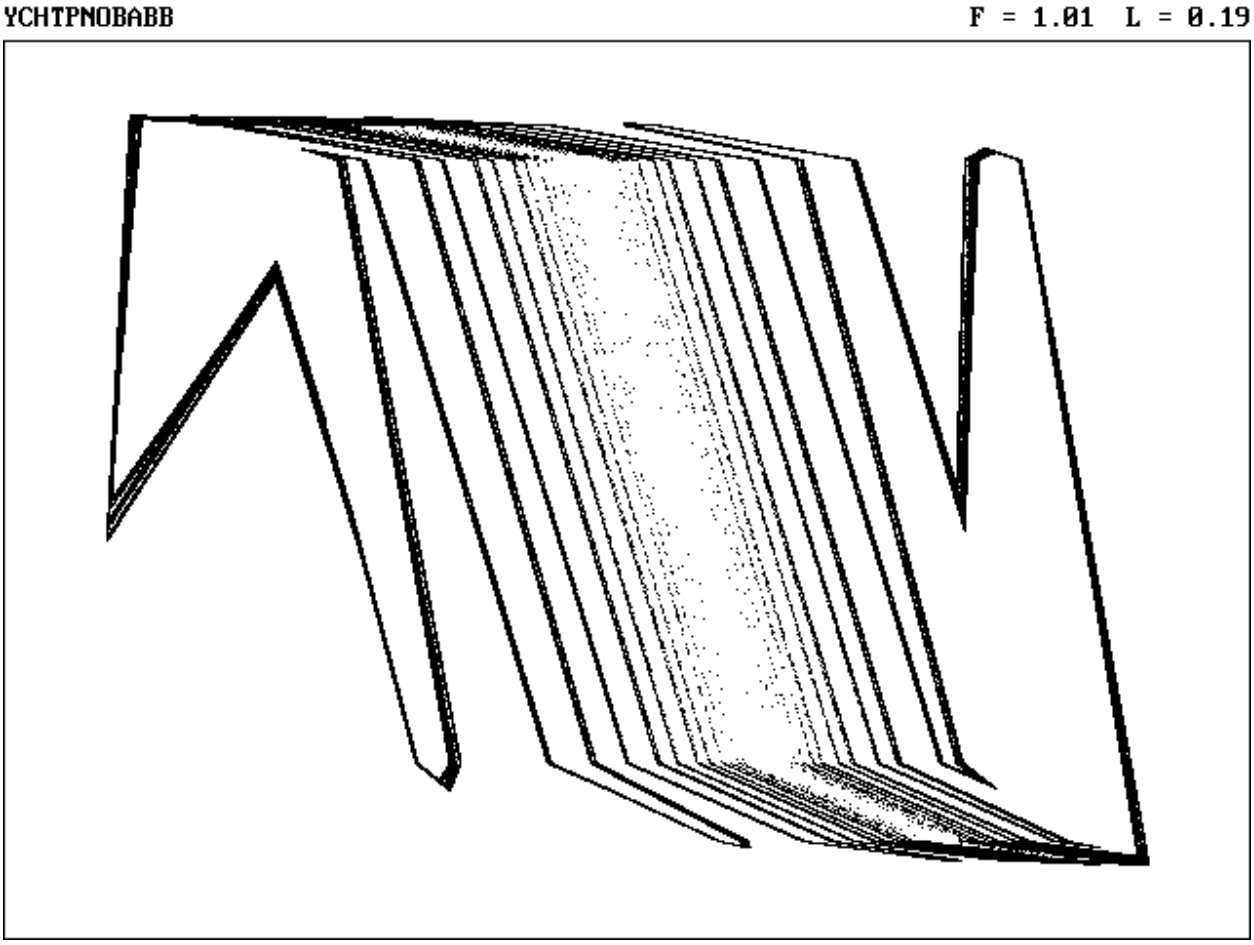


Figure 7-3. Four-dimensional special map Y (gingerbread man)

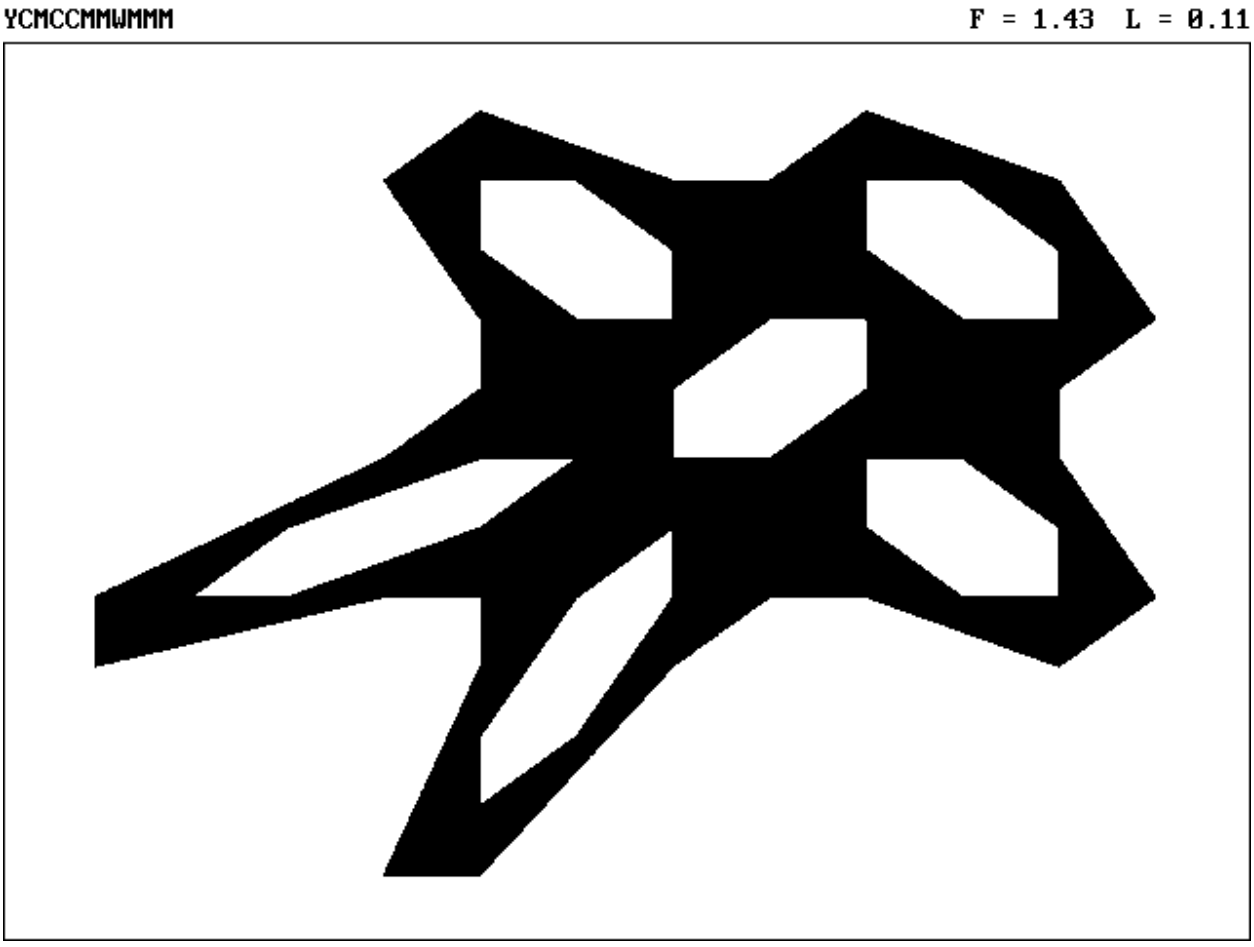


Figure 7-4. Four-dimensional special map Y

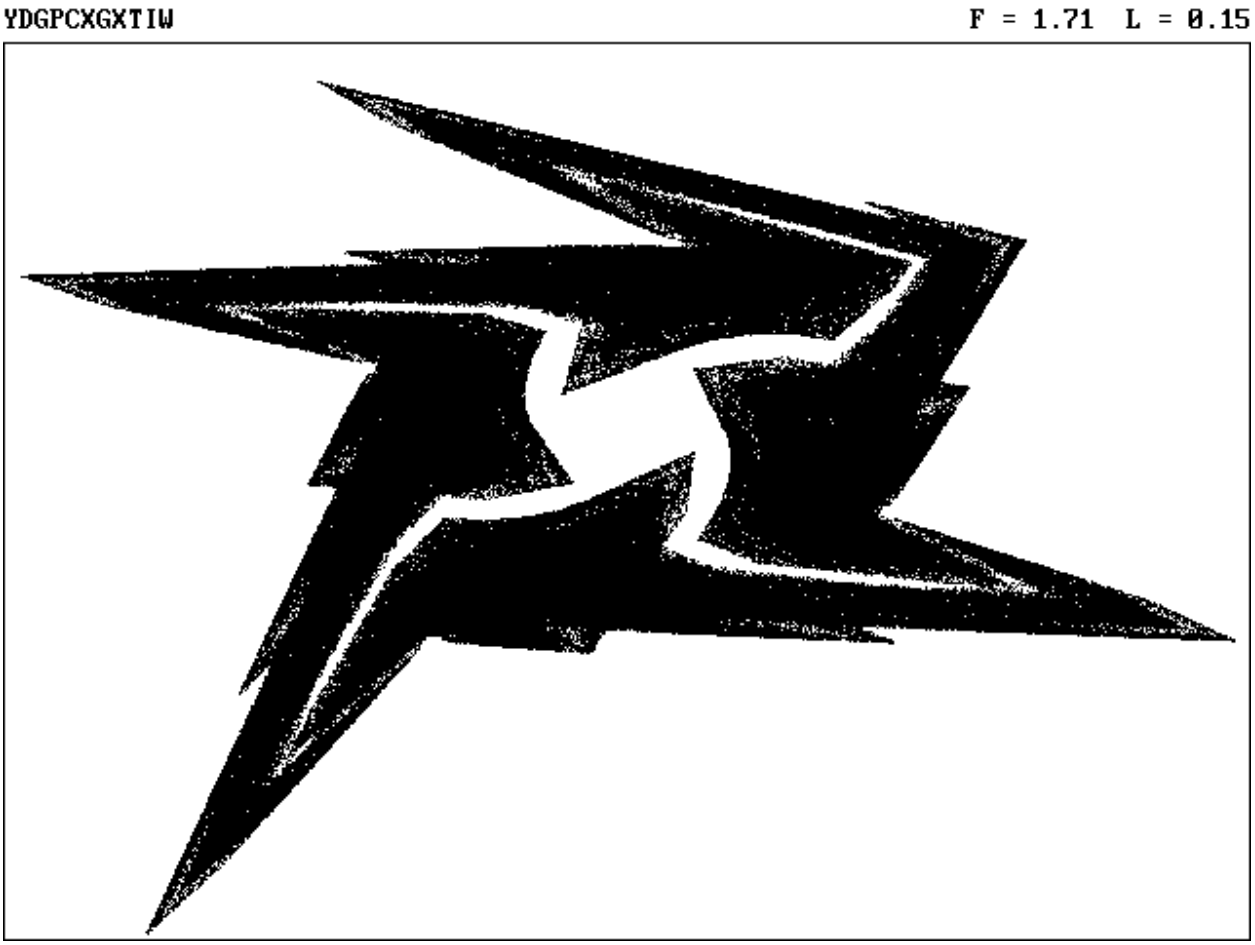


Figure 7-5. Four-dimensional special map Y

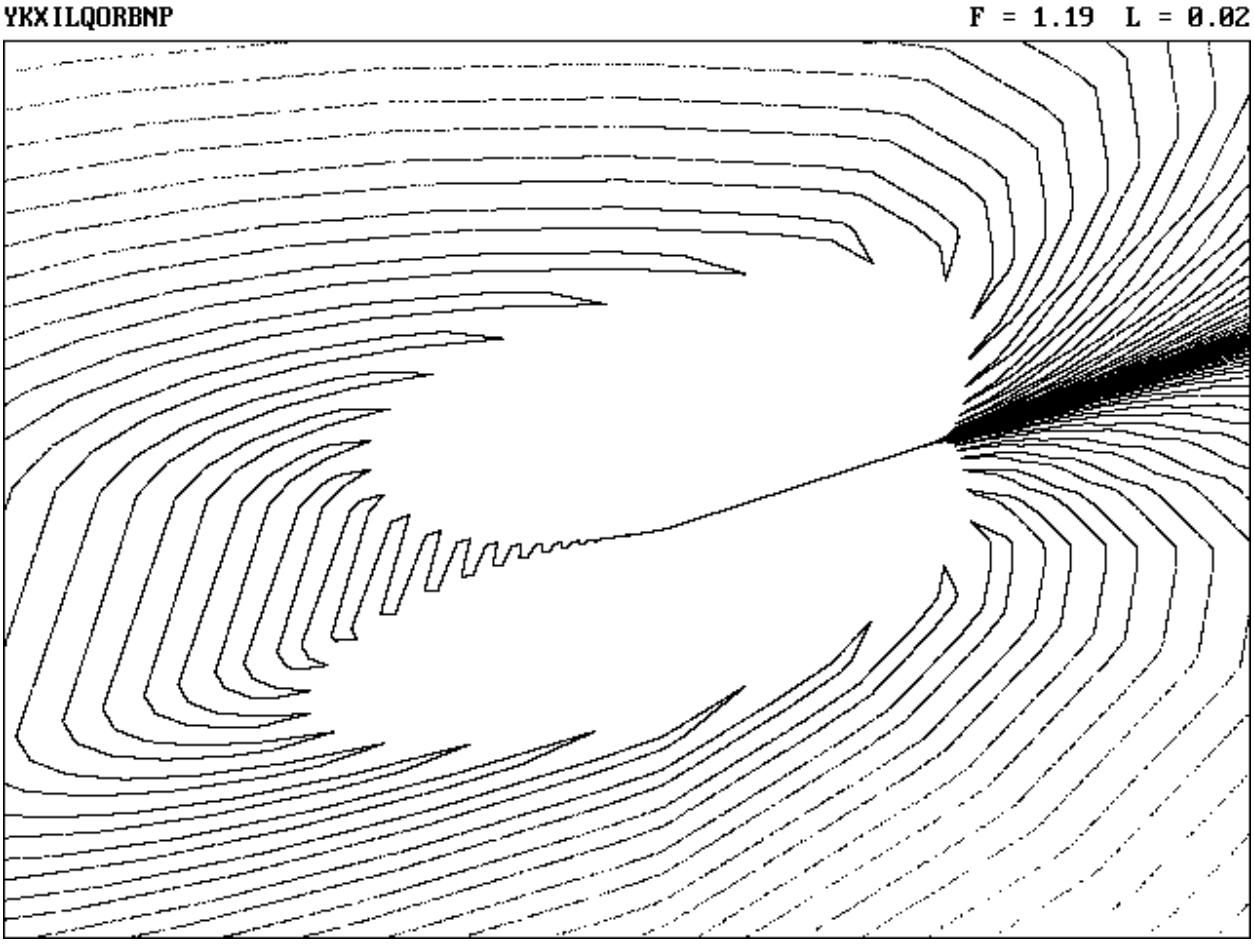


Figure 7-6. Four-dimensional special map Y

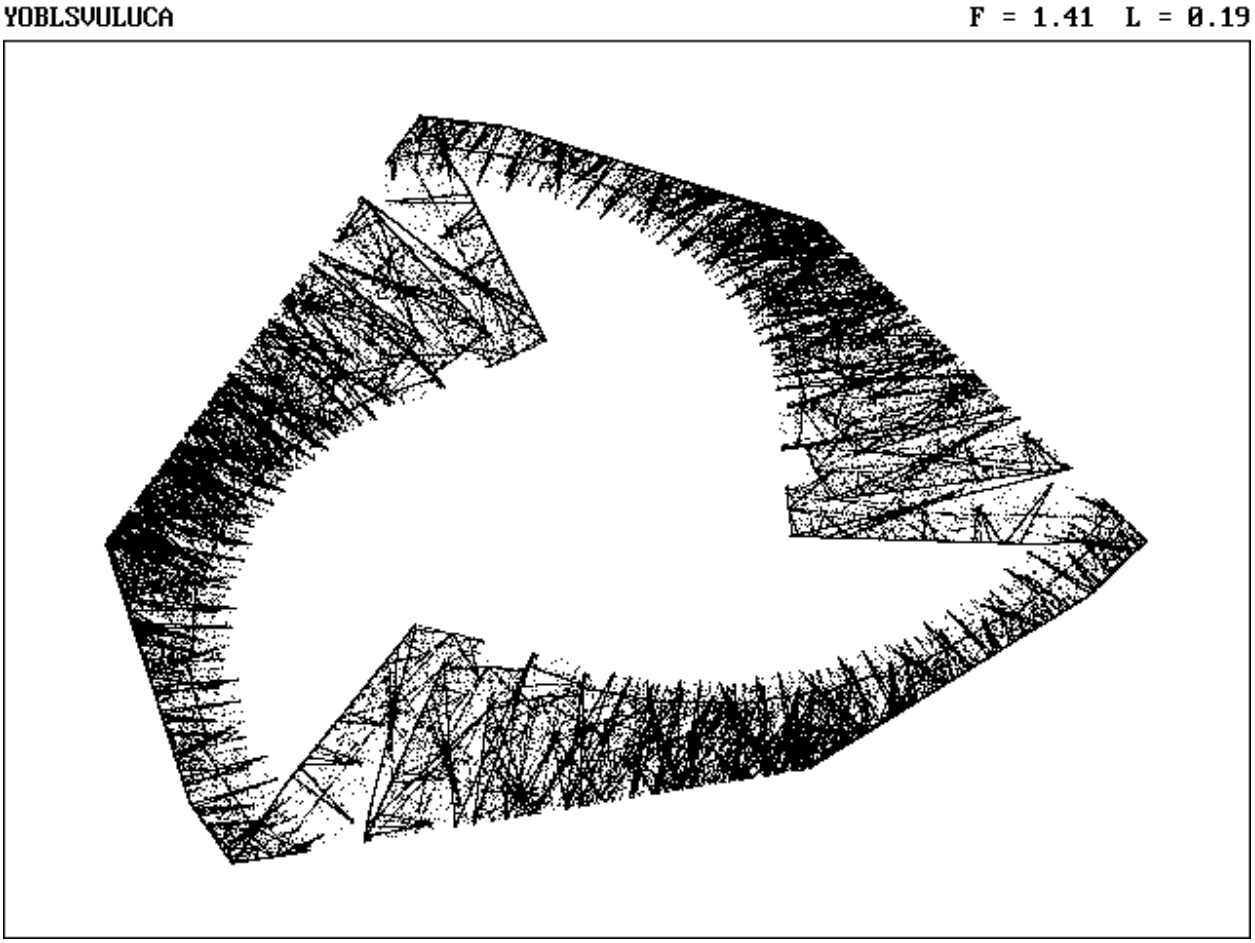


Figure 7-7. Four-dimensional special map Y

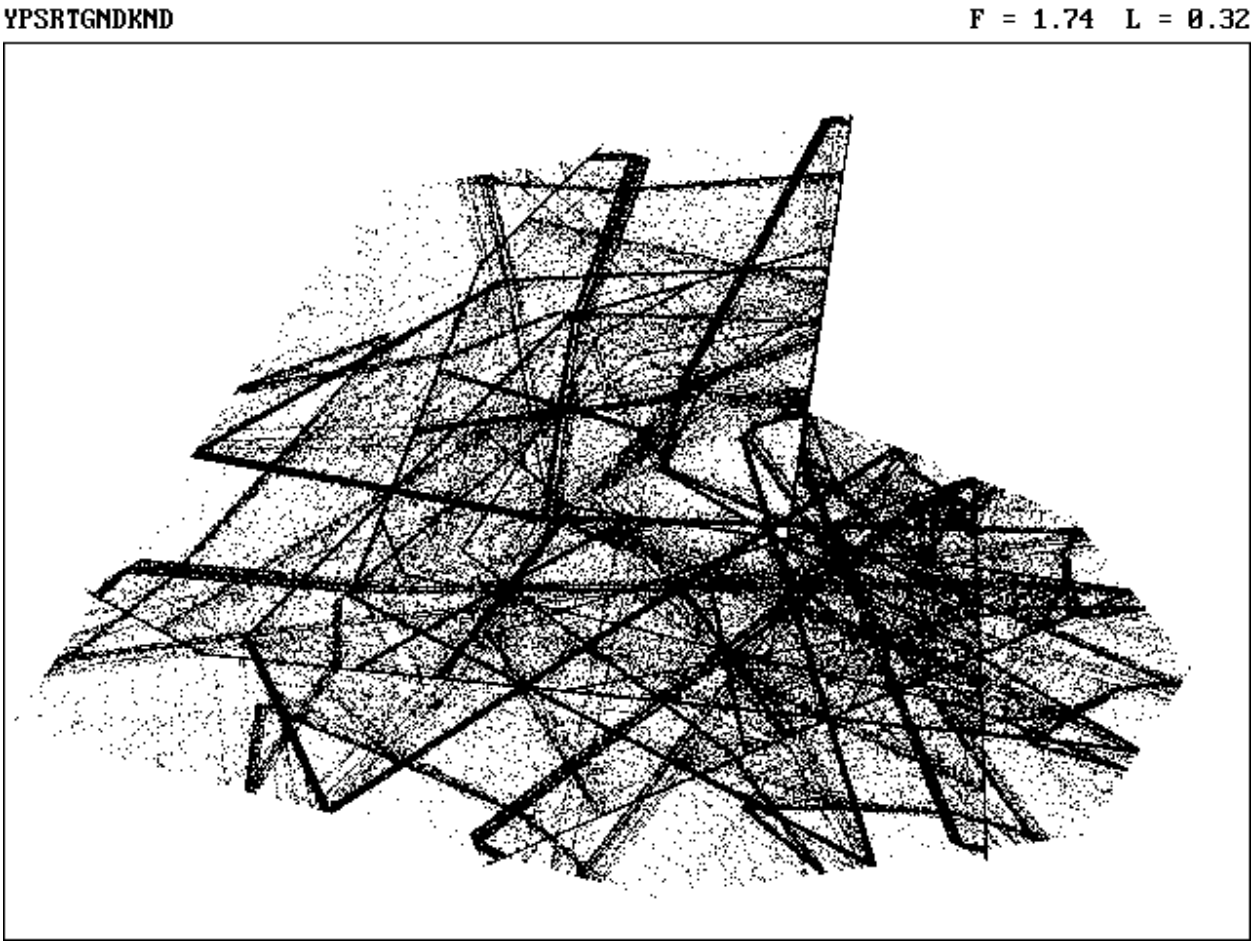
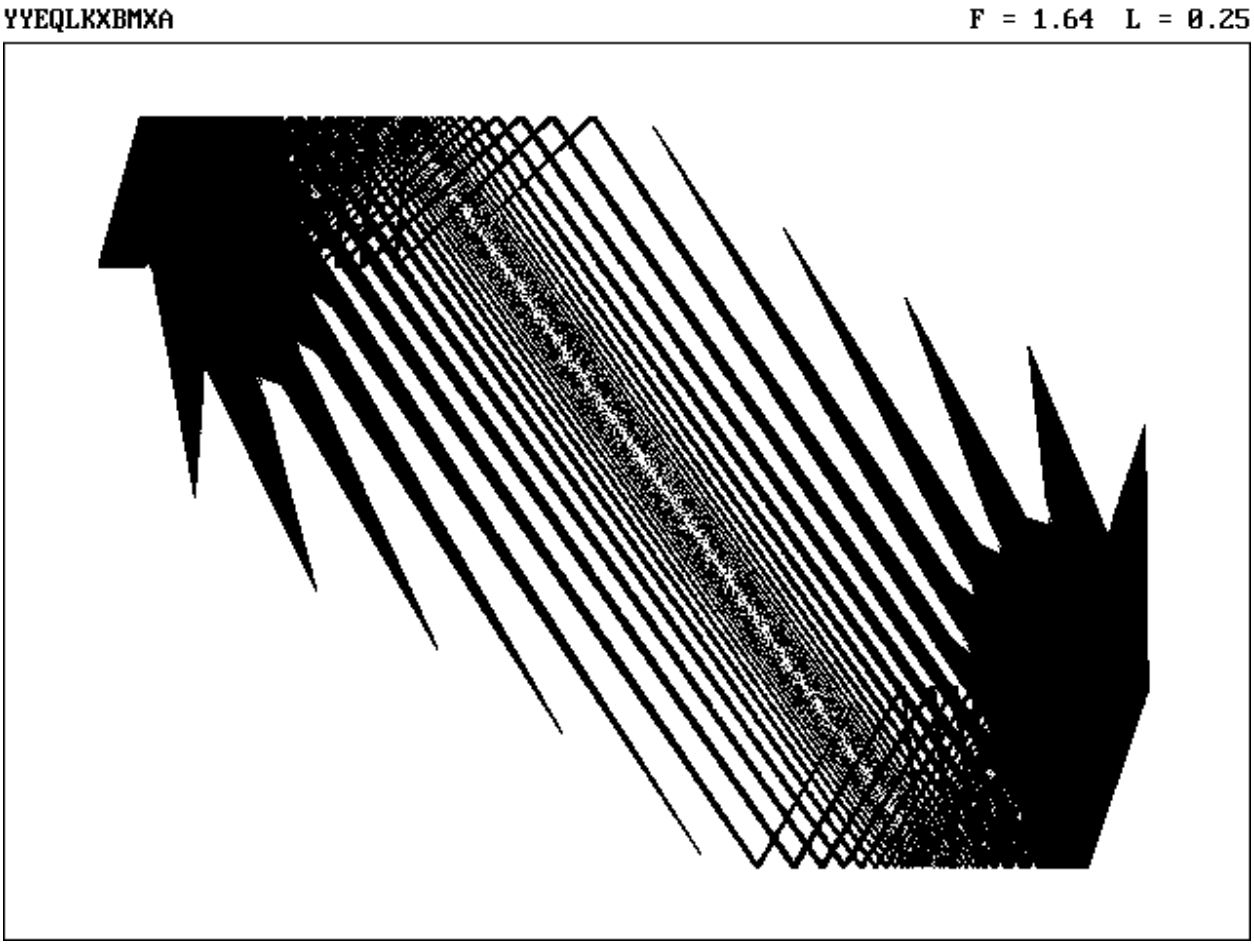


Figure 7-8. Four-dimensional special map Y



7.2 ANDs and ORs

A very different type of nonlinear map can be produced using logical (*Boolean*) operations to manipulate the individual bits of the binary numbers that represent the variables. This is best done after rounding the variables to the nearest integer using the BASIC CINT function. As a variation, you could use the FIX or INT function, both of which truncate rather than round. Most versions of BASIC automatically apply the CINT function before performing logical operations on numbers that are not integers. The conversion of a noninteger to an integer is itself a nonlinear operation, because the graph of CINT(*X*) versus *X* resembles a staircase.

The basic logical operators are AND and OR. If you are not sure what these operations mean, your BASIC manual is a good reference. The operation *X* AND *Y* produces a new number whose bits are 1 if the corresponding bits of *X* and *Y* are both 1, and 0 otherwise. The operation X OR Y produces a new number whose bits are 1 if either (or both) of the corresponding bits of *X* or *Y* are 1, and 0 otherwise. This is also called the *inclusive* OR to distinguish it from the *exclusive* OR (XOR), which produces a number whose bits are 1 if either (but not both) of the corresponding bits of *X* or *Y* are 1, and 0 otherwise.

The following is a general two-dimensional system of equations that includes the AND and OR operators:

$$X_{n+1} = a_1 + a_2X_n + a_3Y_n + a_4X_n \text{ AND } a_5Y_n + a_6X_n \text{ OR } a_7Y_n$$

$$Y_{n+1} = a_8 + a_9X_n + a_{10}Y_n + a_{11}X_n \text{ AND } a_{12}Y_n + a_{13}X_n \text{ OR } a_{14}Y_n$$

(Equation 7D)

The third and fourth dimensions are determined in the same way as in the previous section.

The program modifications required to extend the computer search to such cases are shown in **PROG23**. Codes for this case begin with the letter Z.

PROG23. Changes required in PROG22 to search for special functions of the Z type

```
1000 REM SPECIAL FUNCTION SEARCH (ANDs and ORs)

1090 ODE% = 3                    'System is special function Z
```

```
3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 8): T% = 1



6270 IF ODE% <> 3 THEN GOTO 6310

6280    M% = 14

6290    XNEW = A(1) + A(2) * X + A(3) * Y + (CINT(A(4) * X) AND CINT(A(5) * Y))
+ (CINT(A(6) * X) OR CINT(A(7) * Y))

6300    YNEW = A(8) + A(9) * X + A(10) * Y + (CINT(A(11) * X) AND CINT(A(12) *
Y)) + (CINT(A(13) * X) OR CINT(A(14) * Y))

6310 RETURN
```

Examples of attractors produced by **PROG23** are shown in Figures 7-9 through 7-16. Most of the attractors produced in this way have a streaked or checkered appearance, arising presumably from rounding the variables to integers before performing the logical operations. The ones shown in the figures tend to be the exceptions.

Figure 7-9. Four-dimensional special map Z

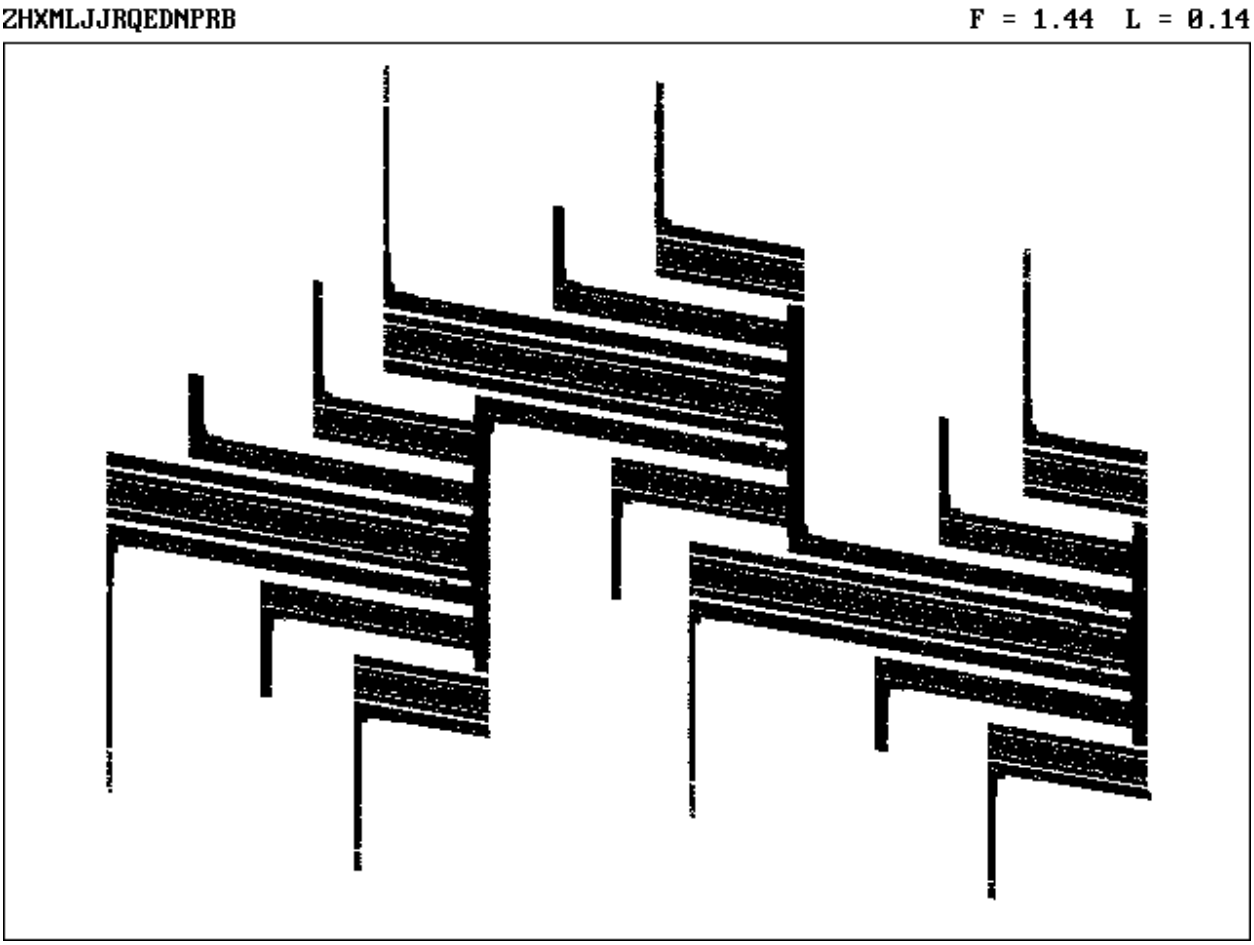


Figure 7-10. Four-dimensional special map Z

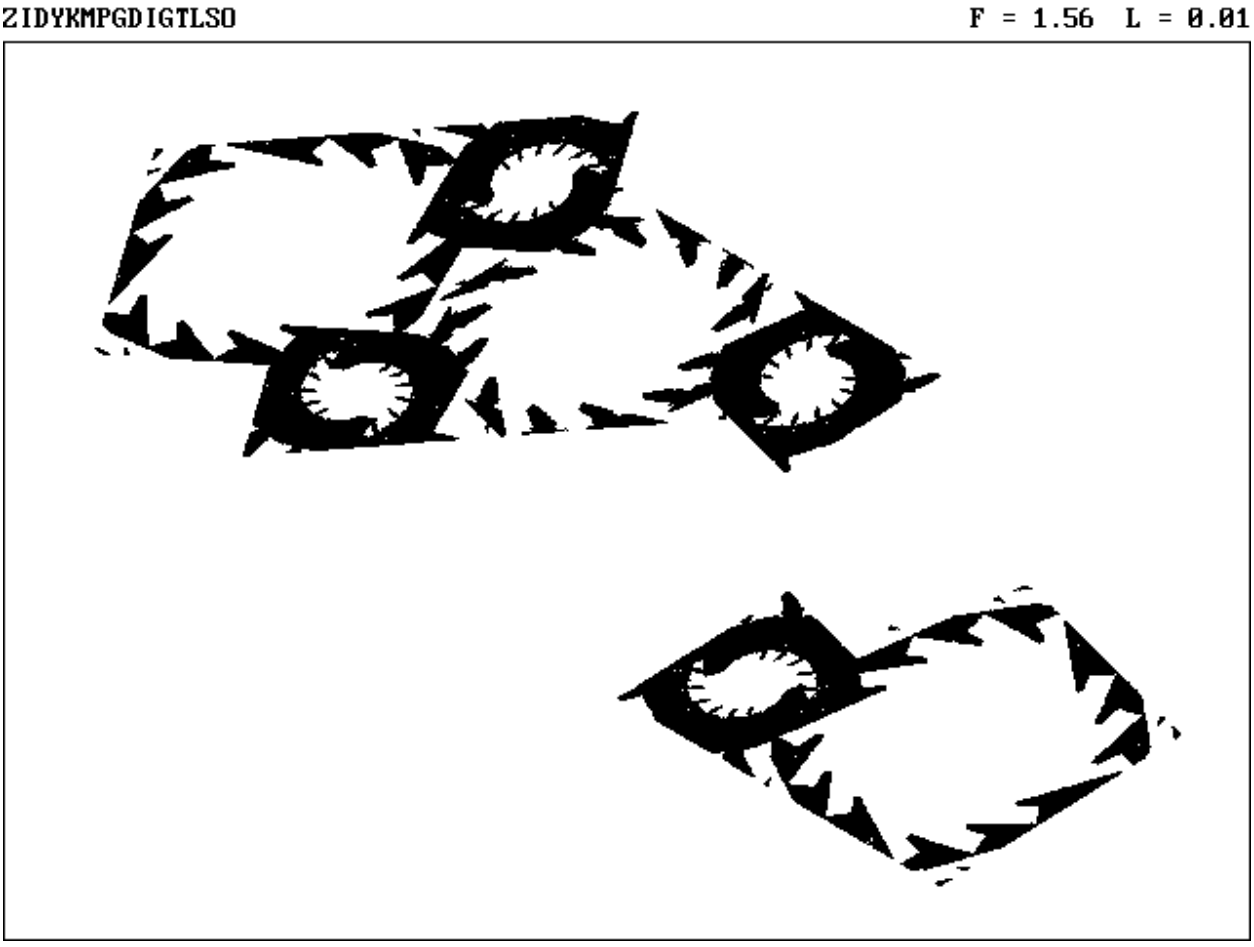


Figure 7-11. Four-dimensional special map Z

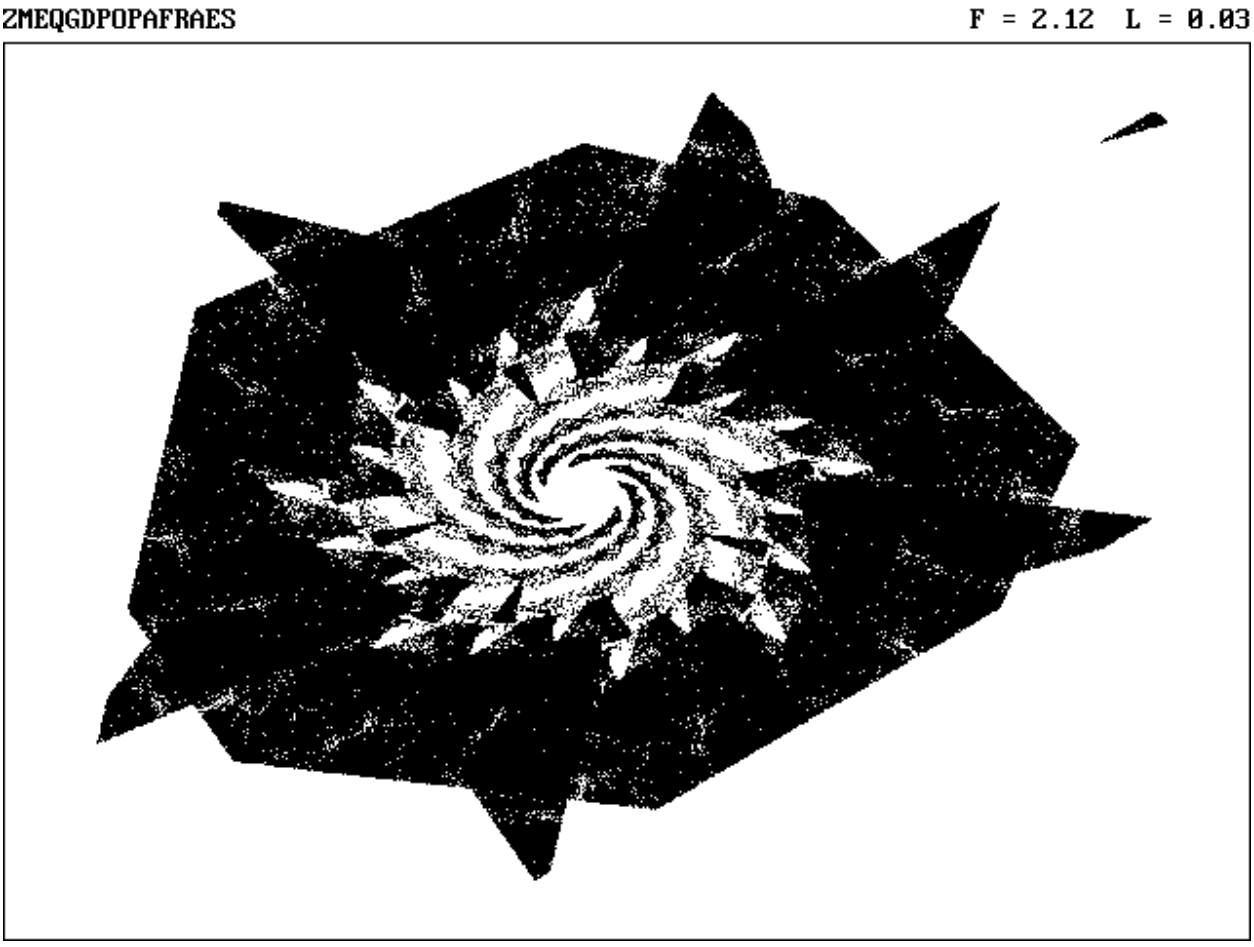


Figure 7-12. Four-dimensional special map Z

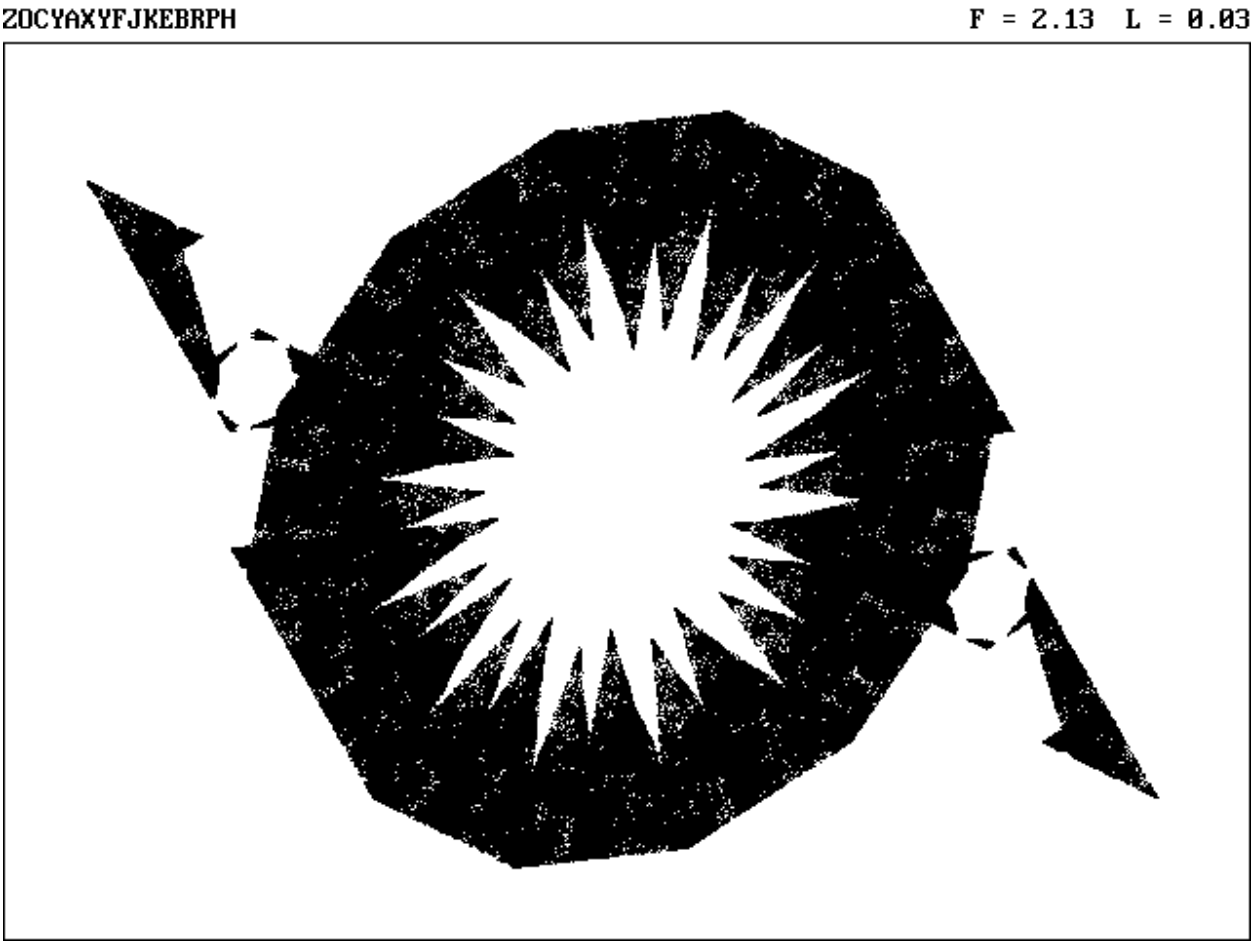


Figure 7-13. Four-dimensional special map Z

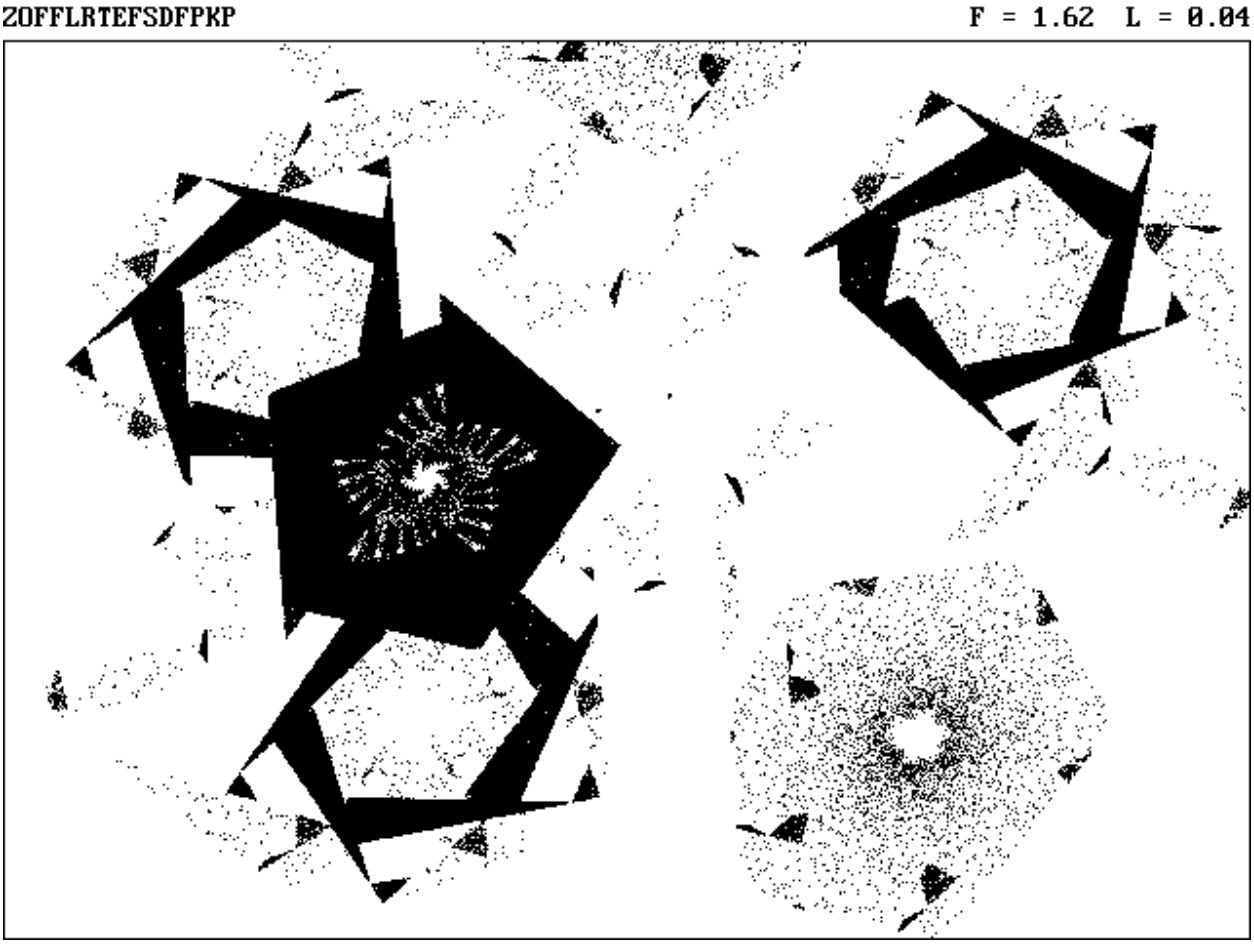


Figure 7-14. Four-dimensional special map Z

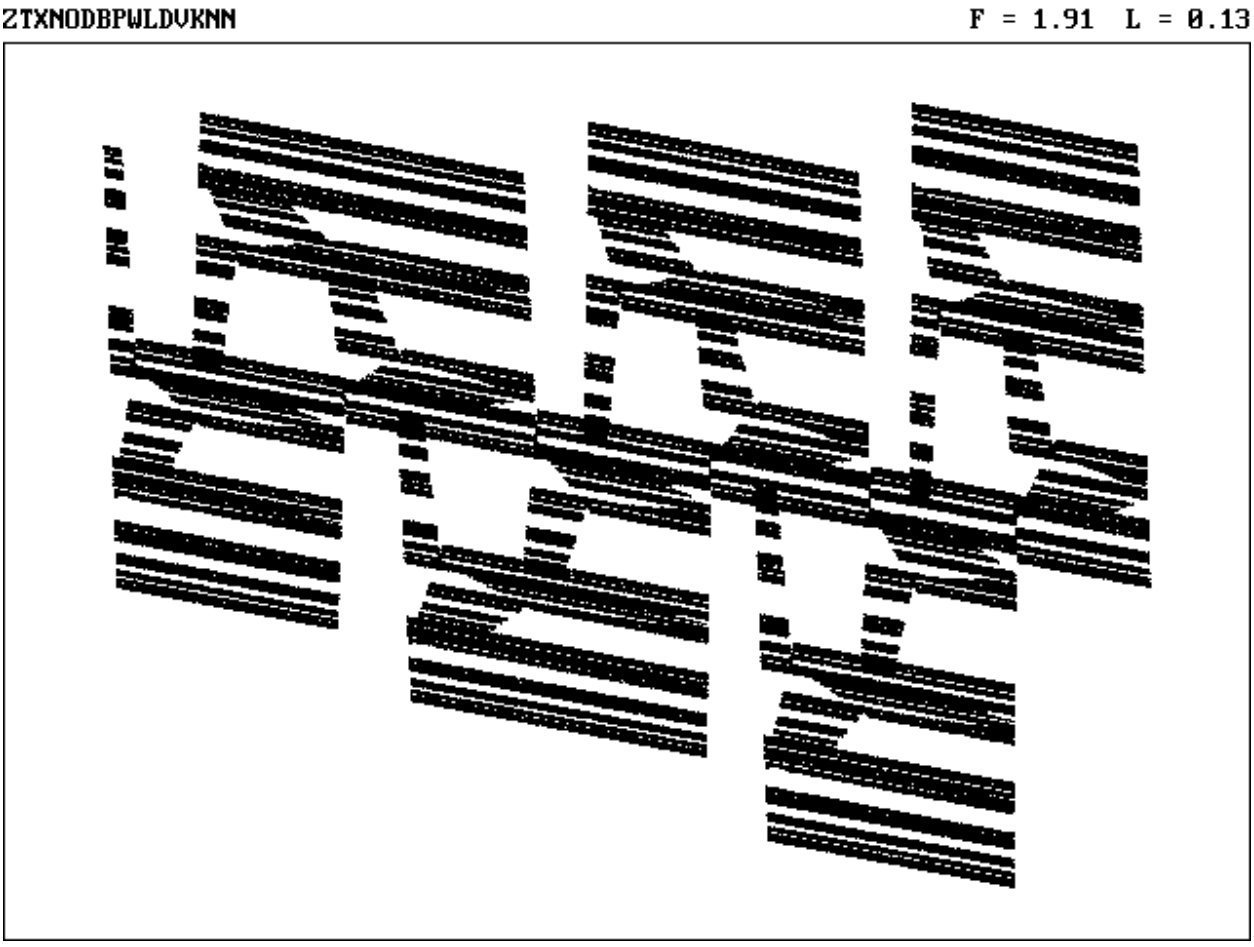


Figure 7-15. Four-dimensional special map Z

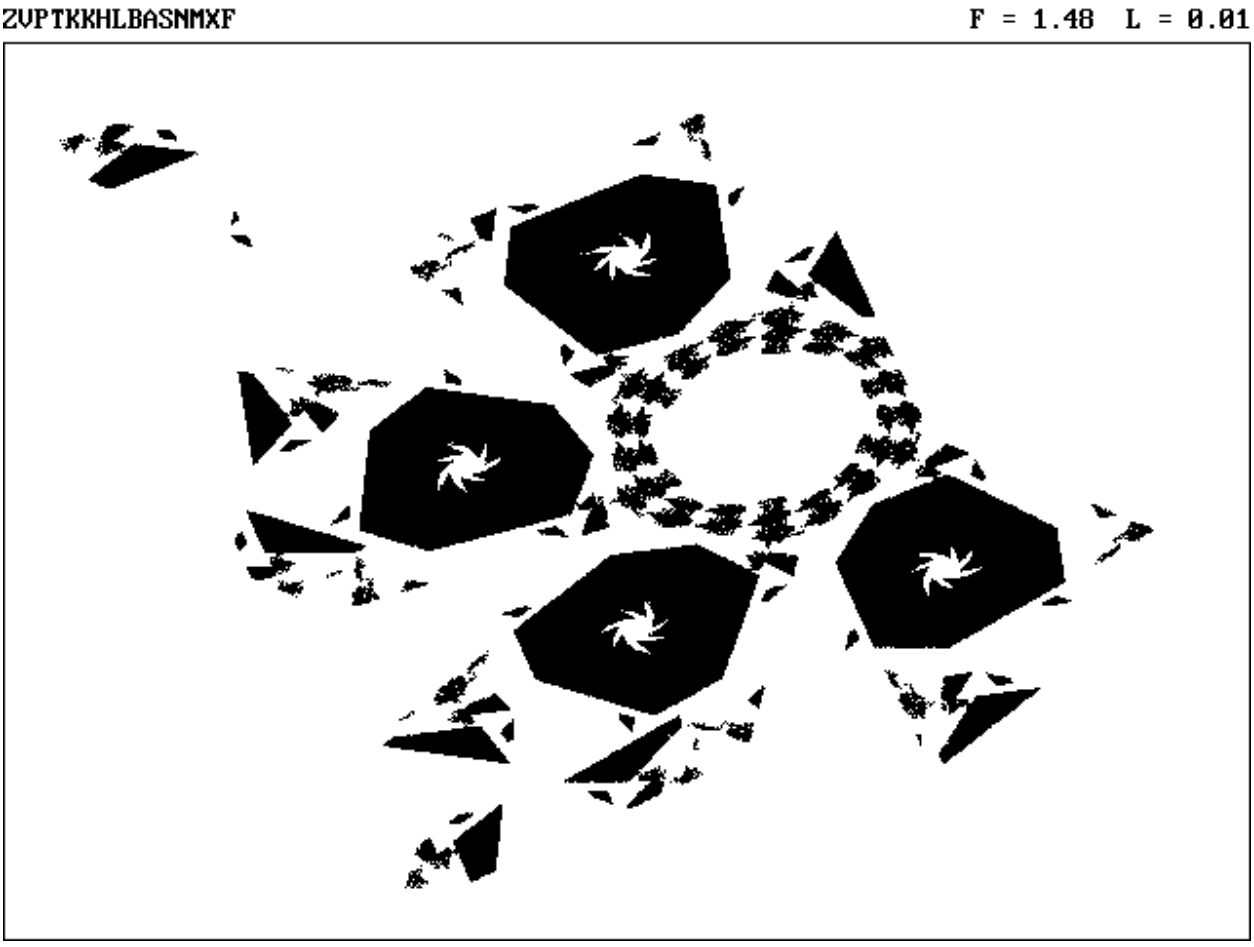
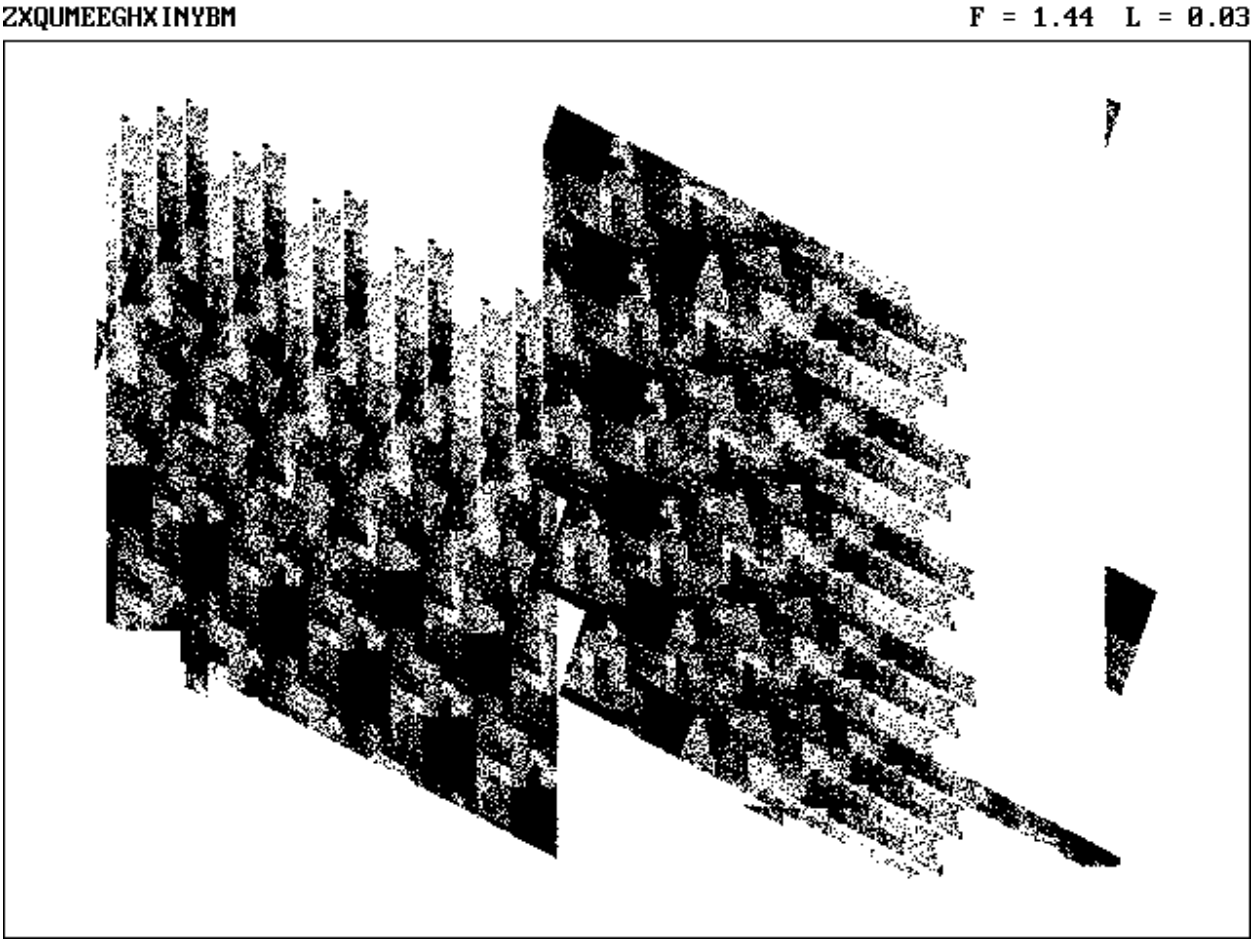


Figure 7-16. Four-dimensional special map Z



7.3 Roots and Powers

Polynomial maps involve powers of the variables that are small positive integers, such as the square (2) and the cube (3). Polynomials exclude such nonlinearities as the square root or the reciprocal of the variables. Roots and powers are mathematically equivalent except for the value of the exponent. The square root of *X* can be written as $X^{0.5}$, and the reciprocal of *X* can be written as *1/X* or as X^{-1}. It is interesting to examine strange attractors that involve fractional and negative powers.

The following is a general two-dimensional system of equations that involves arbitrary powers:

$$X_{n+1} = a_1 + a_2X_n + a_3Y_n + a_4|X_n|^{a_5} + a_6|Y_n|^{a_7}$$

$$Y_{n+1} = a_8 + a_9X_n + a_{10}Y_n + a_{11}|X_n|^{a_{12}} + a_{13}|Y_n|^{a_{14}}$$

(Equation 7E)

The absolute values are needed because BASIC cannot take a root of a negative number. The result would have an imaginary component. Note that if all the exponents happen to be +1, Equation 7E is equivalent to Equation 7B. The third and fourth dimensions are determined in the same way as in Section 7.1.

The program modifications required to extend the computer search to such cases are shown in **PROG24**. Since we have exhausted the capital letter codes, we must invent some new ones. We will continue using the standard ASCII characters beyond Z, as shown in Table 2-1. Thus the codes for this case begin with the left bracket ([), which is ASCII 91.

PROG24. Changes required in PROG23 to search for special functions of the [type

```
1000 REM SPECIAL FUNCTION SEARCH (Roots and Powers)

1090 ODE% = 4                    'System is special function [



3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 9): T% = 1



6310 IF ODE% <> 4 THEN GOTO 6350
```

```
6320    M% = 14

6330    XNEW = A(1) + A(2) * X + A(3) * Y + A(4) * ABS(X) ^ A(5) + A(6) * ABS(Y)
^ A(7)

6340   YNEW = A(8) + A(9) * X + A(10) * Y + A(11) * ABS(X) ^ A(12) + A(13) * ABS(Y)
^ A(14)

6350 RETURN
```

Examples of attractors produced by **PROG24** are shown in Figures 7-17 through 7-24. These attractors are localized mostly to a small region of the *XY* plane with tentacles that stretch out to large distances. If any of the exponents are negative and the attractor intersects the line along which the respective variable is zero, a point on the line maps to infinity. However, large values are visited infrequently by the orbit, so many iterations are required to determine that the orbit is unbounded. For this reason most of the attractors in the figures have holes in their interiors where their orbits are precluded from coming too close to their origins ($X = Y = 0$).

Figure 7-17. Four-dimensional special map [

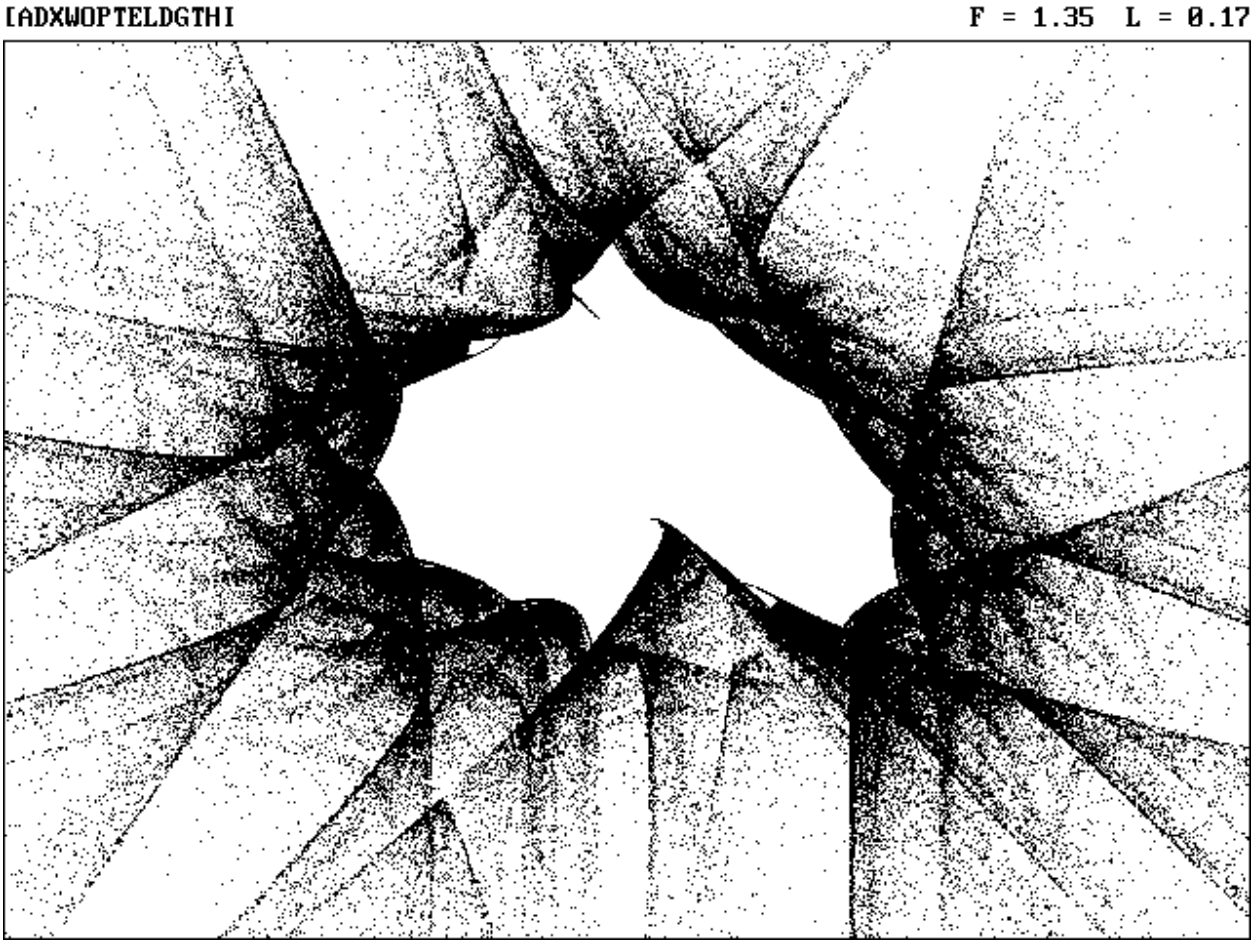


Figure 7-18. Four-dimensional special map [

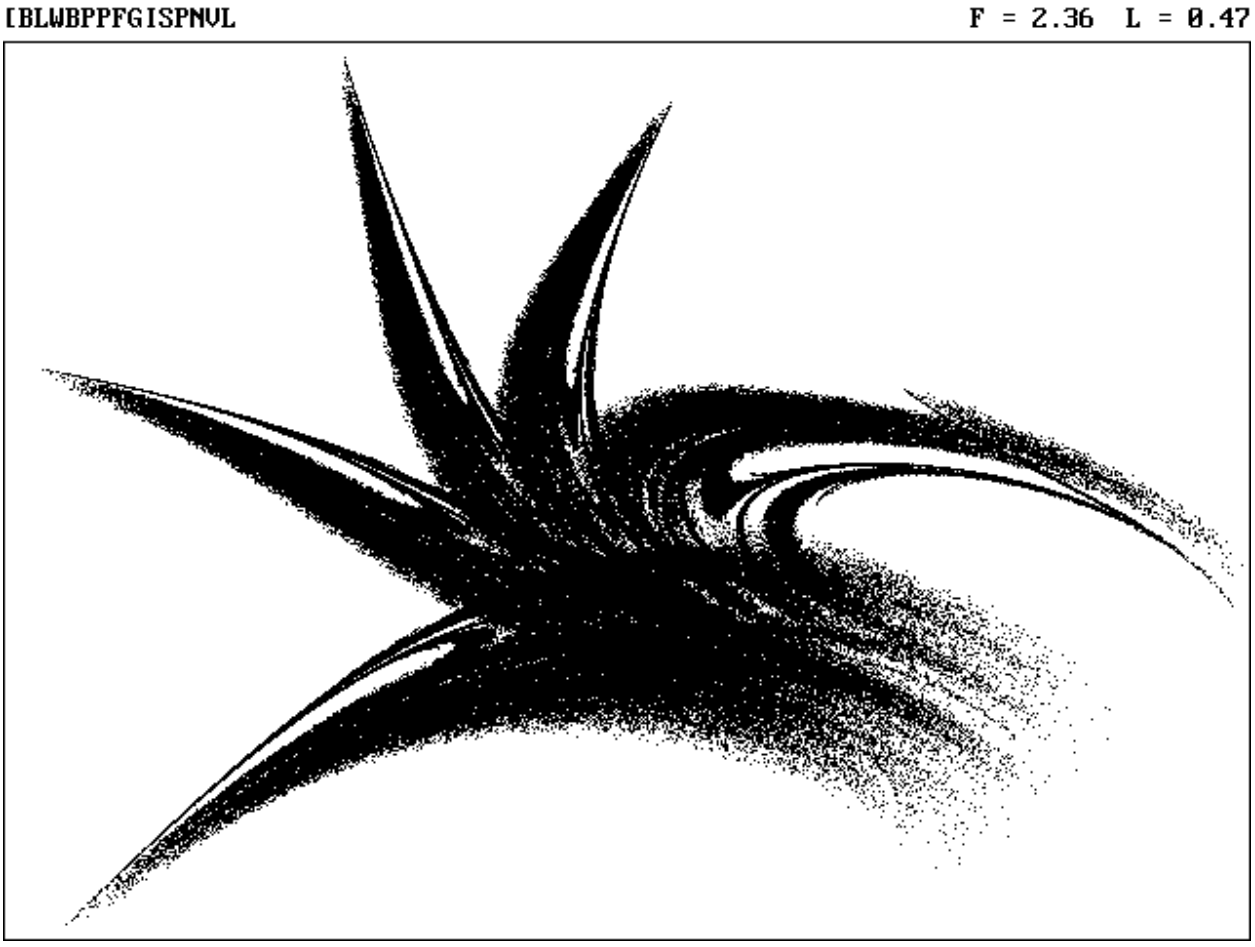


Figure 7-19. Four-dimensional special map [

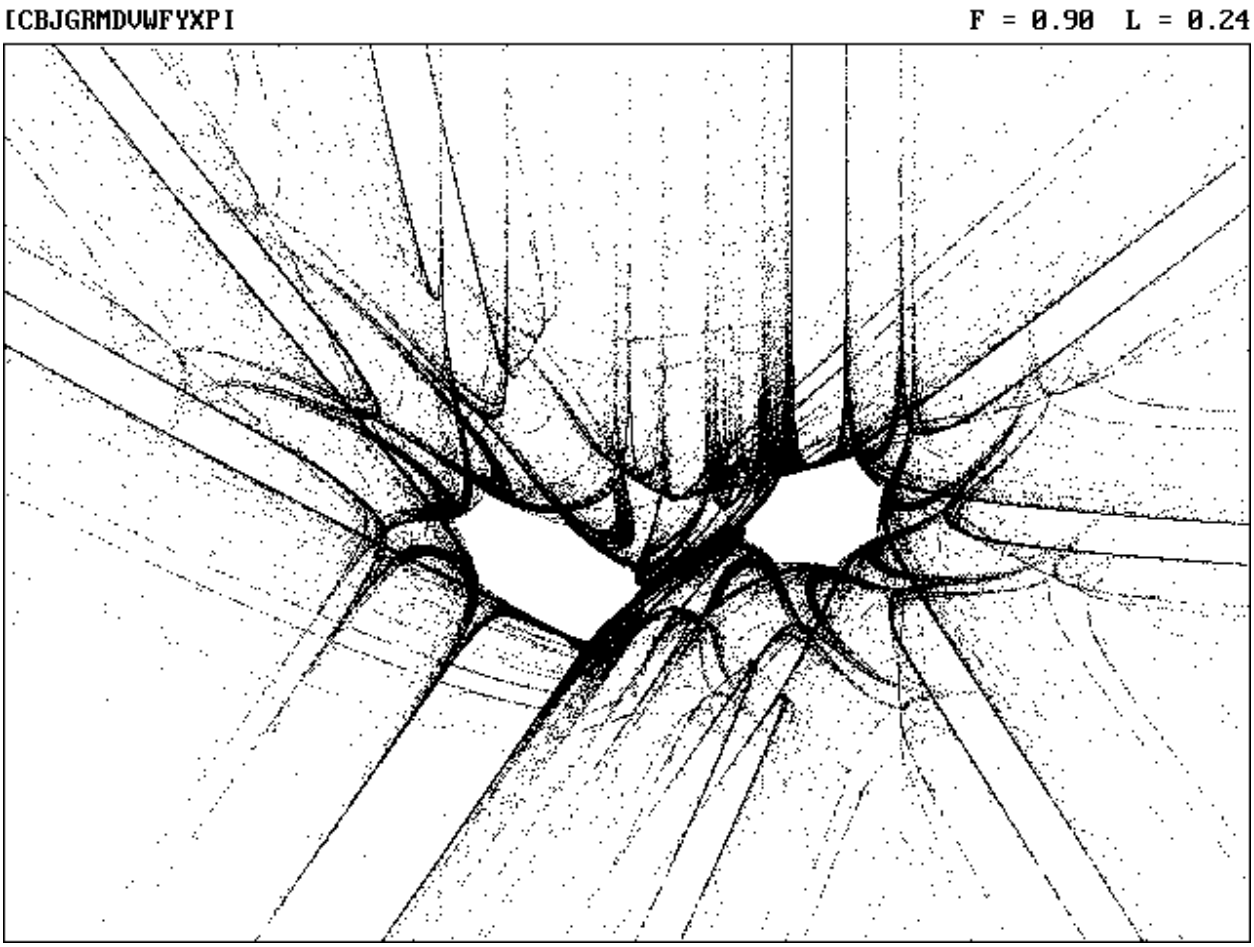


Figure 7-20. Four-dimensional special map [

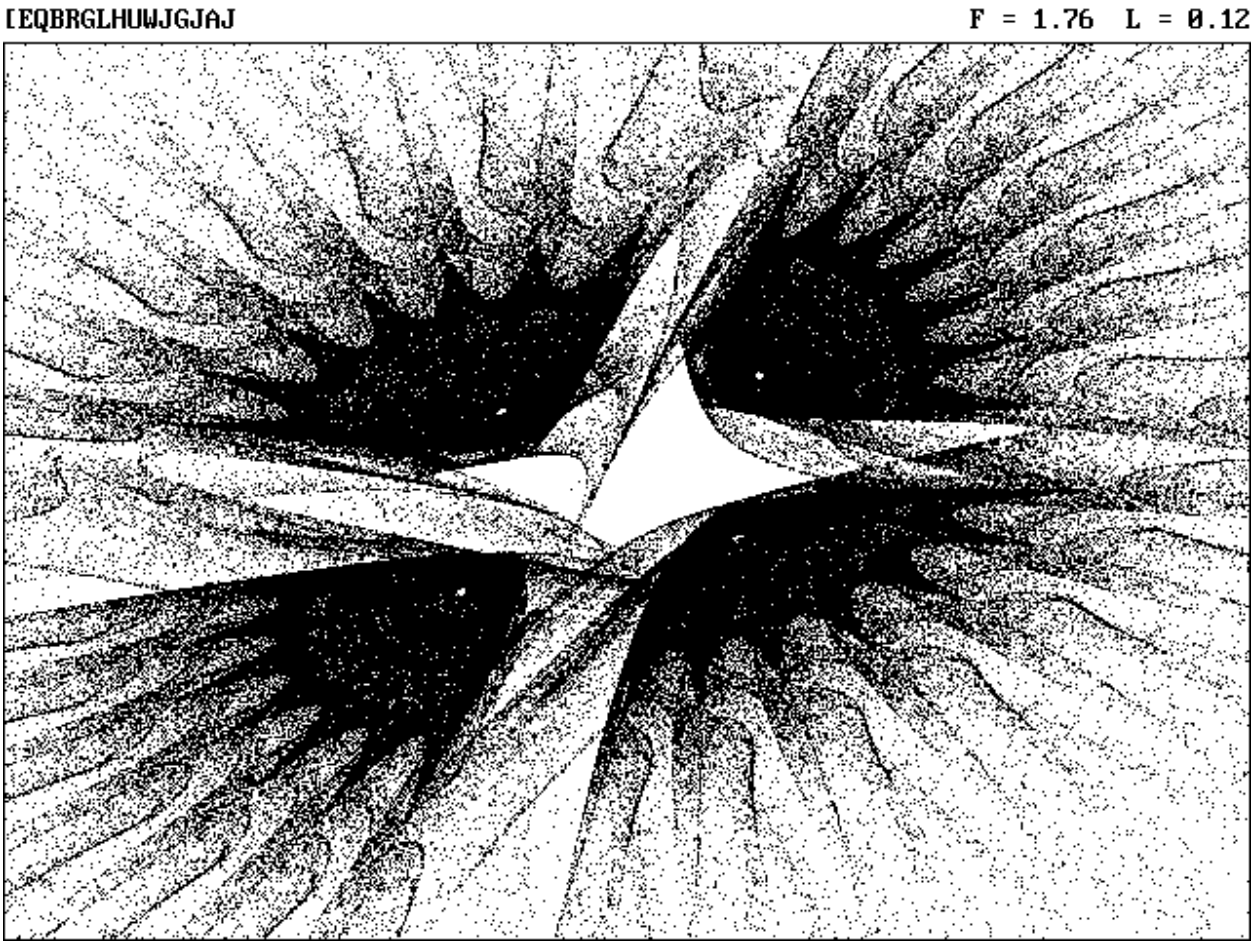


Figure 7-21. Four-dimensional special map [

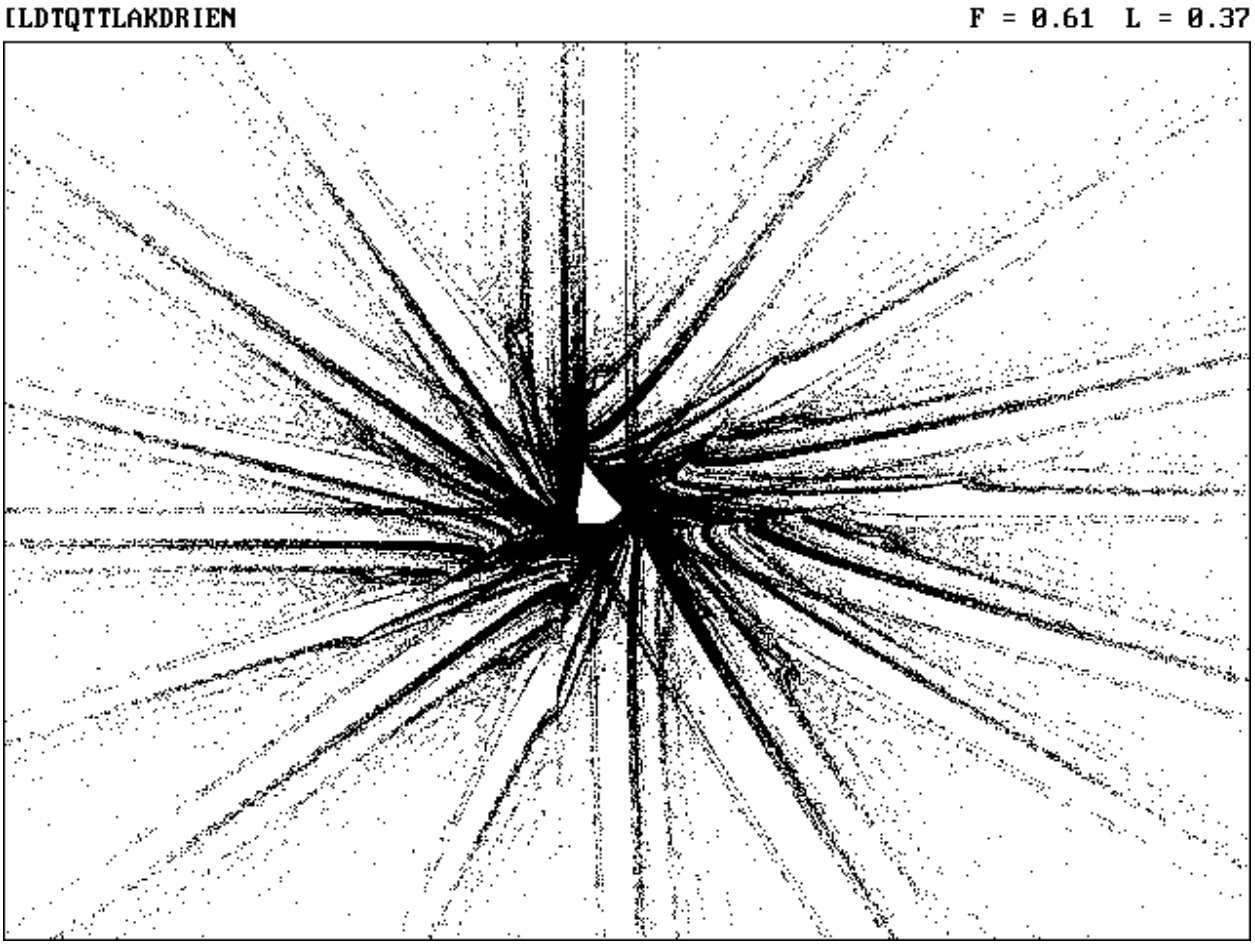


Figure 7-22. Four-dimensional special map [

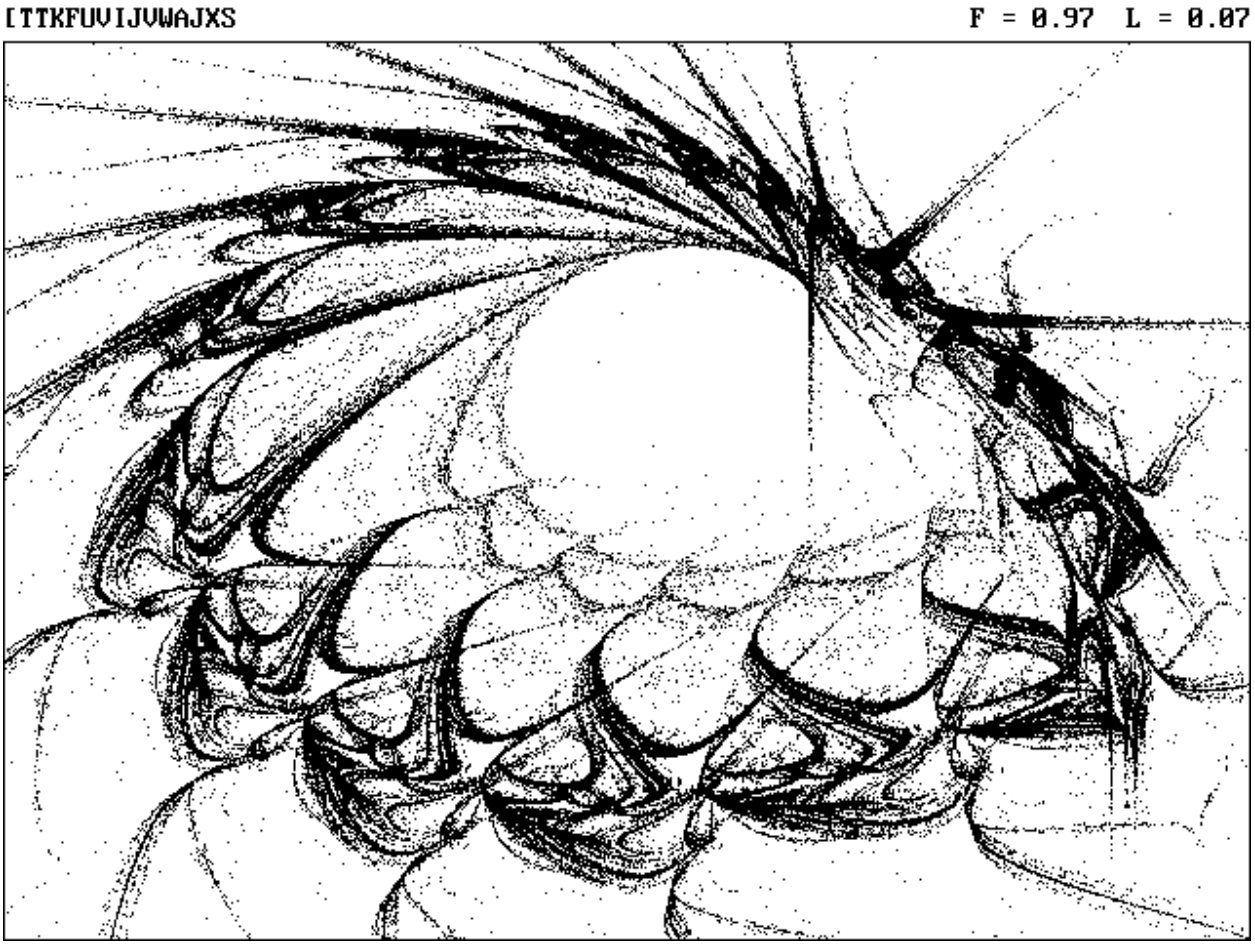


Figure 7-23. Four-dimensional special map [

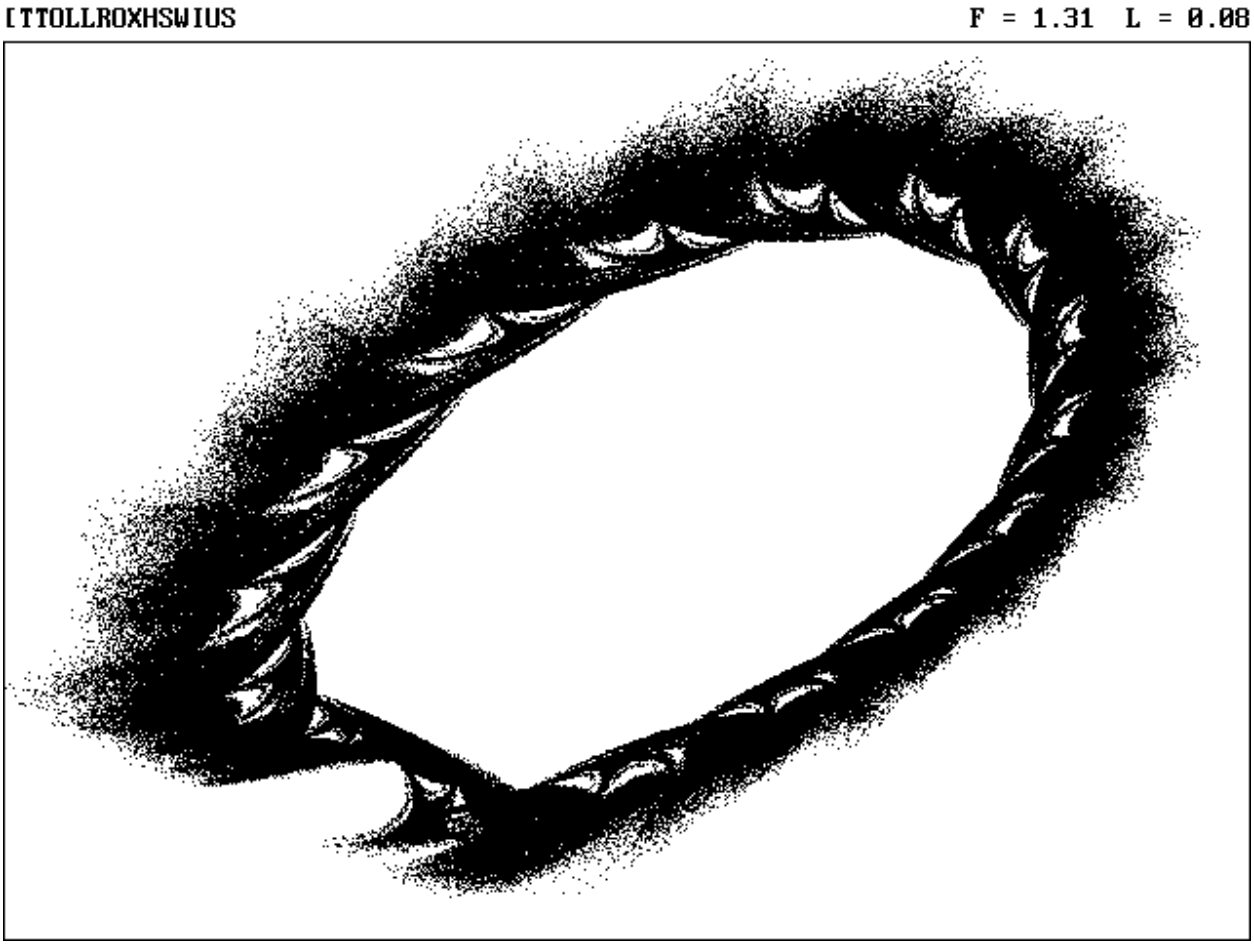
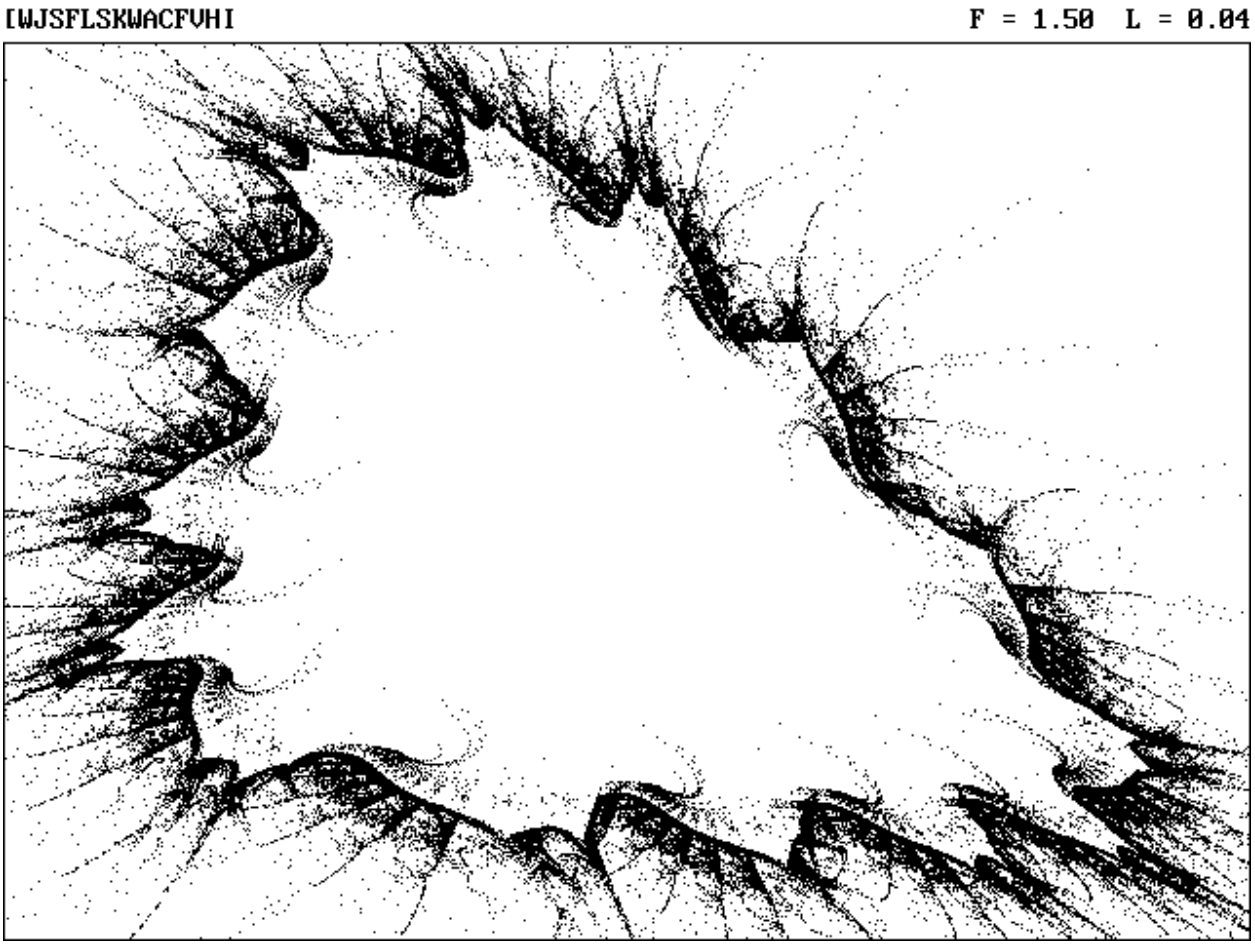


Figure 7-24. Four-dimensional special map [



You may become frustrated seeing a beautiful attractor develop for thousands of iterations and then having the orbit escape. Such behavior is called a *crisis* or *transient chaos*, not to be confused with a catastrophe. For example, the logistic equation with *R* slightly greater than 4.0 is chaotic for many iterations until an iterate happens to exceed 1.0, whereupon the orbit abruptly moves off toward infinity. By contrast, a catastrophe occurs when the solution undergoes a qualitative change at some critical value of a control parameter.

Related to a crisis is another phenomenon called *intermittency*. At certain values of the control parameters, a system exhibits periodic behavior for many cycles and then suddenly becomes chaotic for a while before settling back into periodic behavior. Classic examples of intermittency occur in the logistic equation at about *R* = 3.82812 and in the Lorenz equations at about *r* = 166.2. Intermittency has been observed in many natural systems, and it is a bane to those who try to

make predictions. It is possible that the solar system is intermittently chaotic or even that a crisis can occur leading to a complete loss of its stability, perhaps precipitated by a rare conjunction of a planet with a large asteroid or comet.

Those solutions of Equation 7E that remain bounded tend to have a wispy appearance and to go beyond the frame of the figure because of the occasional large excursions. Artistically, this feature gives the attractors a sense of being connected to the surrounding world rather than being isolated objects suspended in a void. If you frame these cases, a surrounding mat is desirable to provide the illusion that they are being viewed through a window.

7.4 Sines and Cosines

Two of the most common nonlinear functions are the sine and its complement, the cosine. The sine and cosine can be approximated by polynomials as follows:

$$\sin X = X - X^3/6 + X^5/120 - X^7/5040 + \ldots$$

$$\cos X = 1 - X^2/2 + X^4/24 - X^6/720 + \ldots \qquad \text{(Equation 7F)}$$

When the argument *X* is small, the approximations are very accurate using only a few terms in the expansion. The denominator of each term is the *factorial* of the exponent of that term. For example, the factorial of five (written 5!) is equal to 5 x 4 x 3 x 2 x 1 = 120. When *X* is large, many terms are required. In such a case, we would expect to observe dynamics different from those produced by the fifth-order polynomials previously examined.

A general two-dimensional system of equations whose nonlinearity is restricted to the sine function is the following:

$$X_{n+1} = a_1 + a_2X_n + a_3Y_n + a_4\sin(a_5X_n + a_6) + a_7\sin(a_8Y_n + a_9)$$

$$Y_{n+1} = a_{10} + a_{11}X_n + a_{12}Y_n + a_{13}\sin(a_{14}X_n + a_{15}) + a_{16}\sin(a_{17}Y_n + a_{18})$$

(Equation 7G)

It is not necessary to consider the cosine separately; the *phase terms* (a_6, a_9, a_{15}, and a_{18}) have the same effect because cos *X* is equal to sin(*X* + π/2). The third and

fourth dimensions are determined in the same way as in Section 7.1.

The program modifications required to extend the computer search to such cases are shown in **PROG25**. These cases are coded with the backslash (\), which is ASCII 92.

PROG25 Changes Required in PROG24 to Search for Special Functions of the \ Type

```
1000 REM SPECIAL FUNCTION SEARCH (Sines and Cosines)

1090 ODE% = 5                    'System is special function \



3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 10): T% = 1



6350 IF ODE% <> 5 THEN GOTO 6390

6360    M% = 18

6370    XNEW = A(1) + A(2) * X + A(3) * Y + A(4) * SIN(A(5) * X + A(6)) + A(7)
* SIN(A(8) * Y + A(9))

6380    YNEW = A(10) + A(11) * X + A(12) * Y + A(13) * SIN(A(14) * X + A(15)) +
A(16) * SIN(A(17) * Y + A(18))

6390 RETURN
```

Examples of attractors produced by **PROG25** are shown in Figures 7-25 through 7-32.

Figure 7-25. Four-dimensional special map \

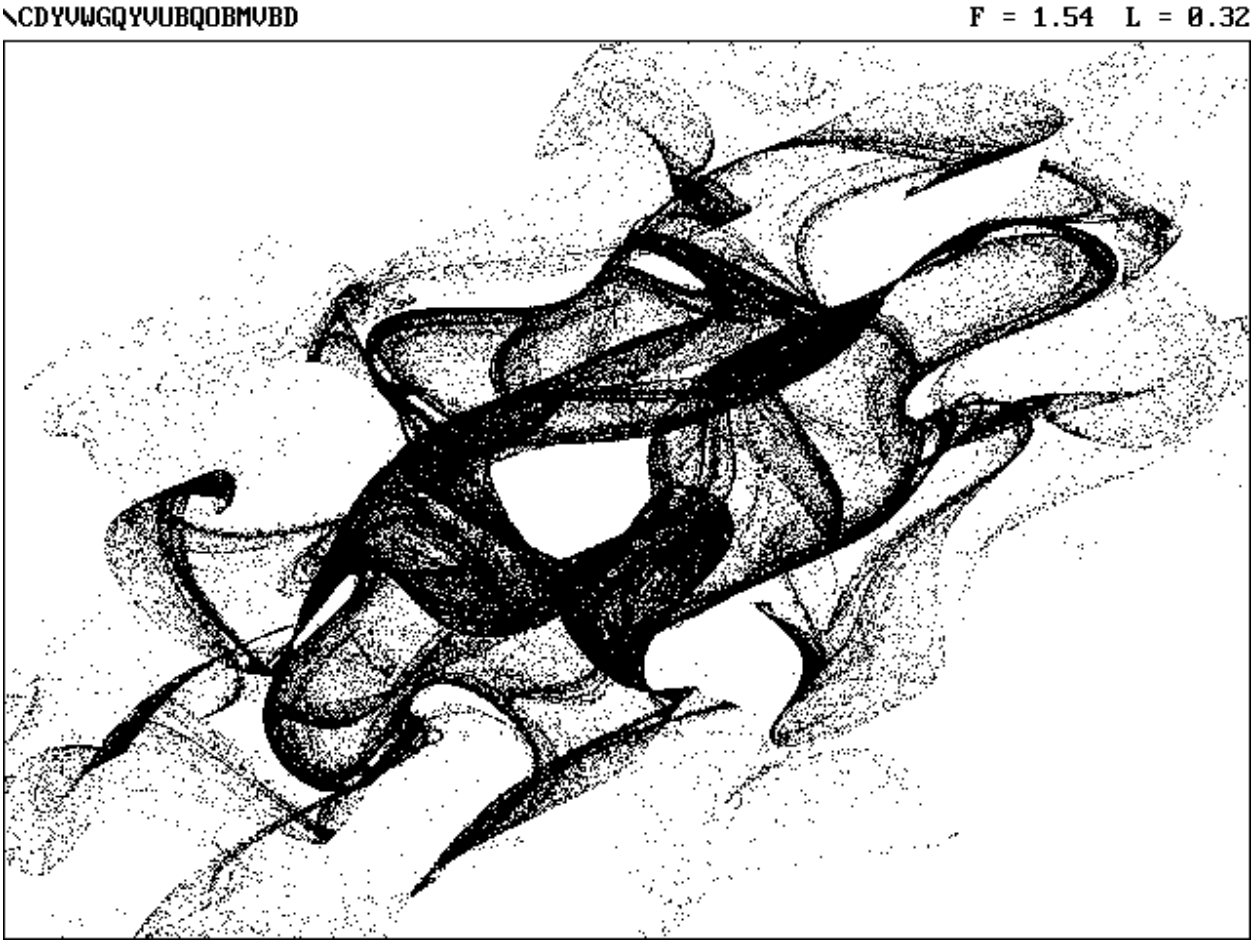


Figure 7-26. Four-dimensional special map \

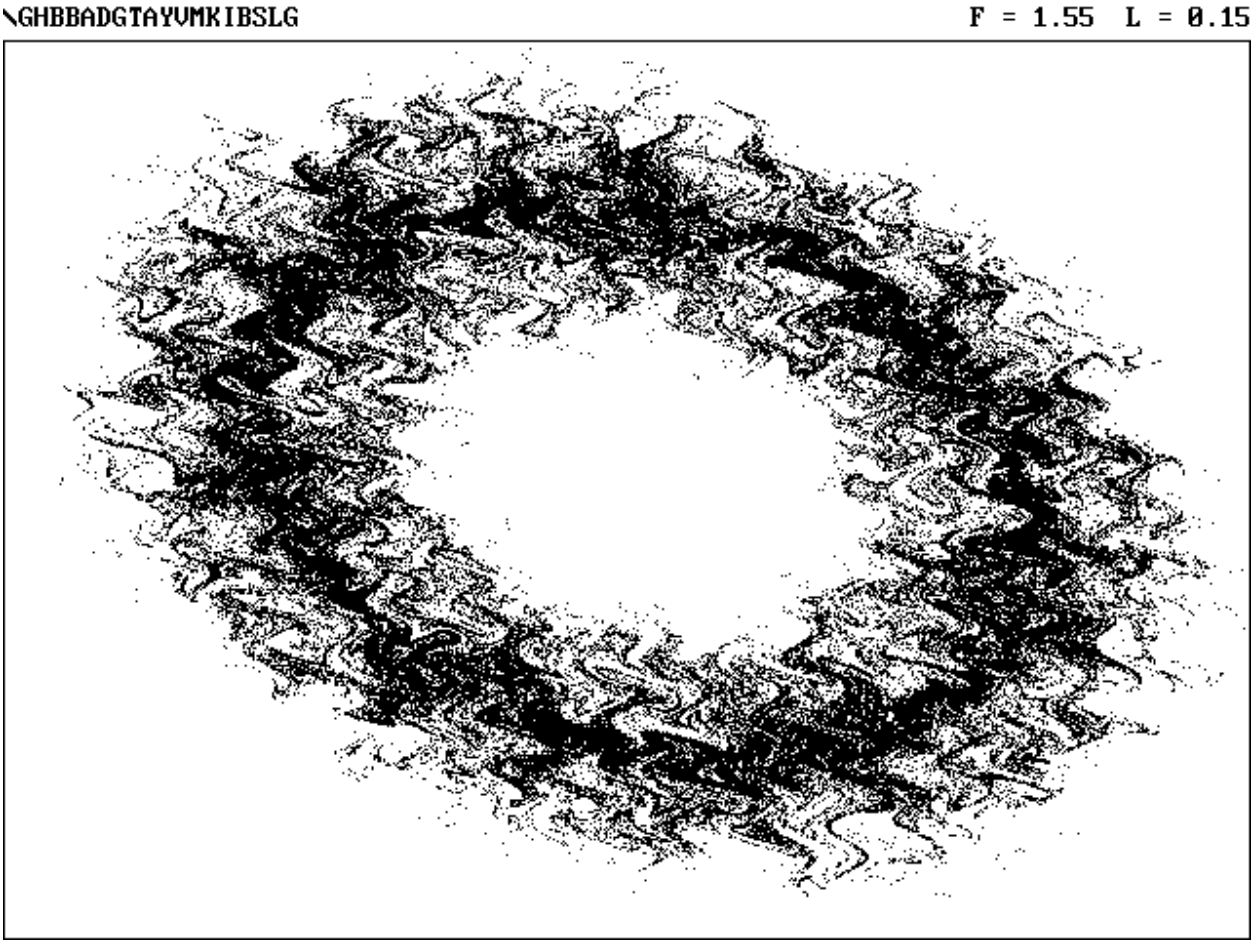


Figure 7-27. Four-dimensional special map \

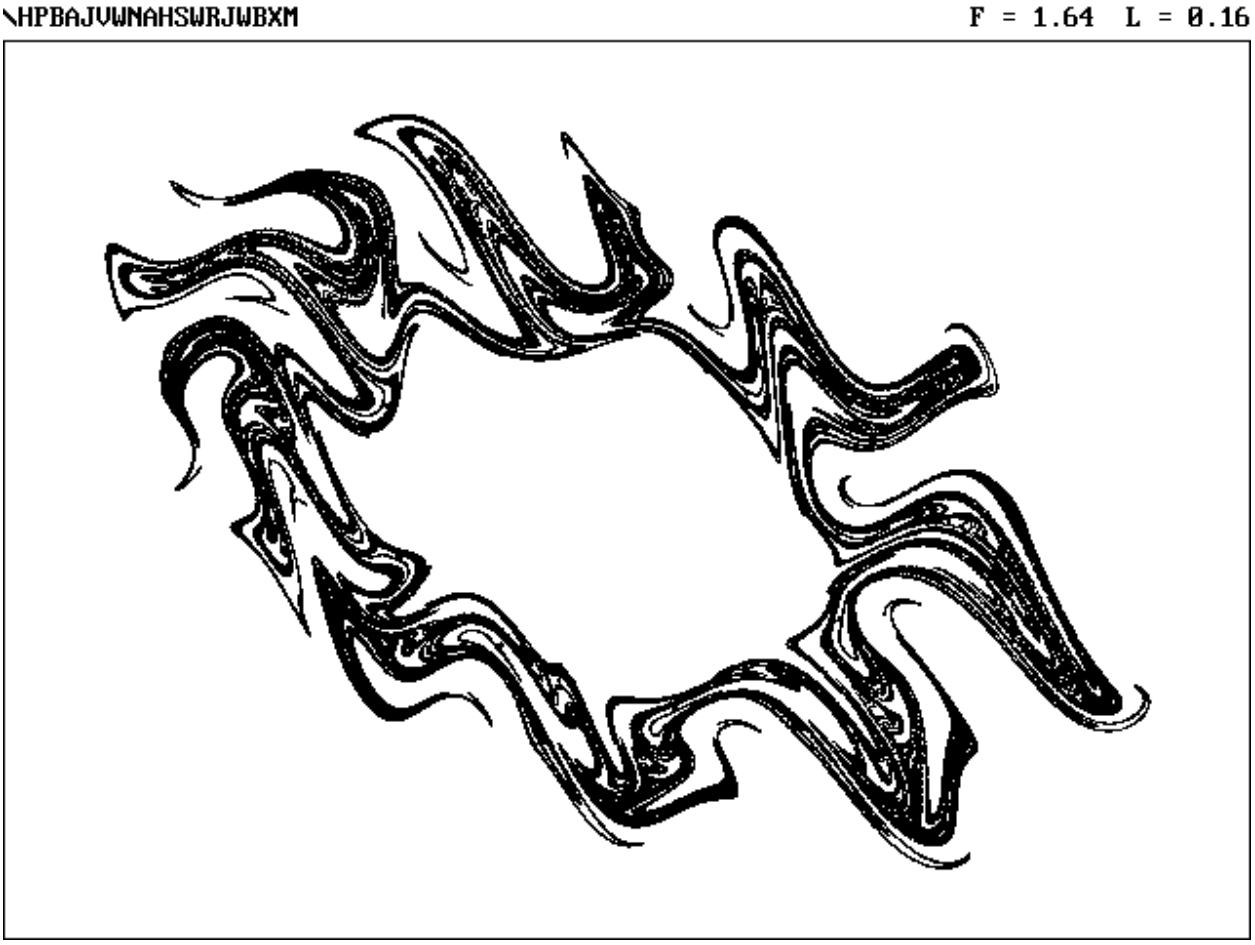


Figure 7-28. Four-dimensional special map \

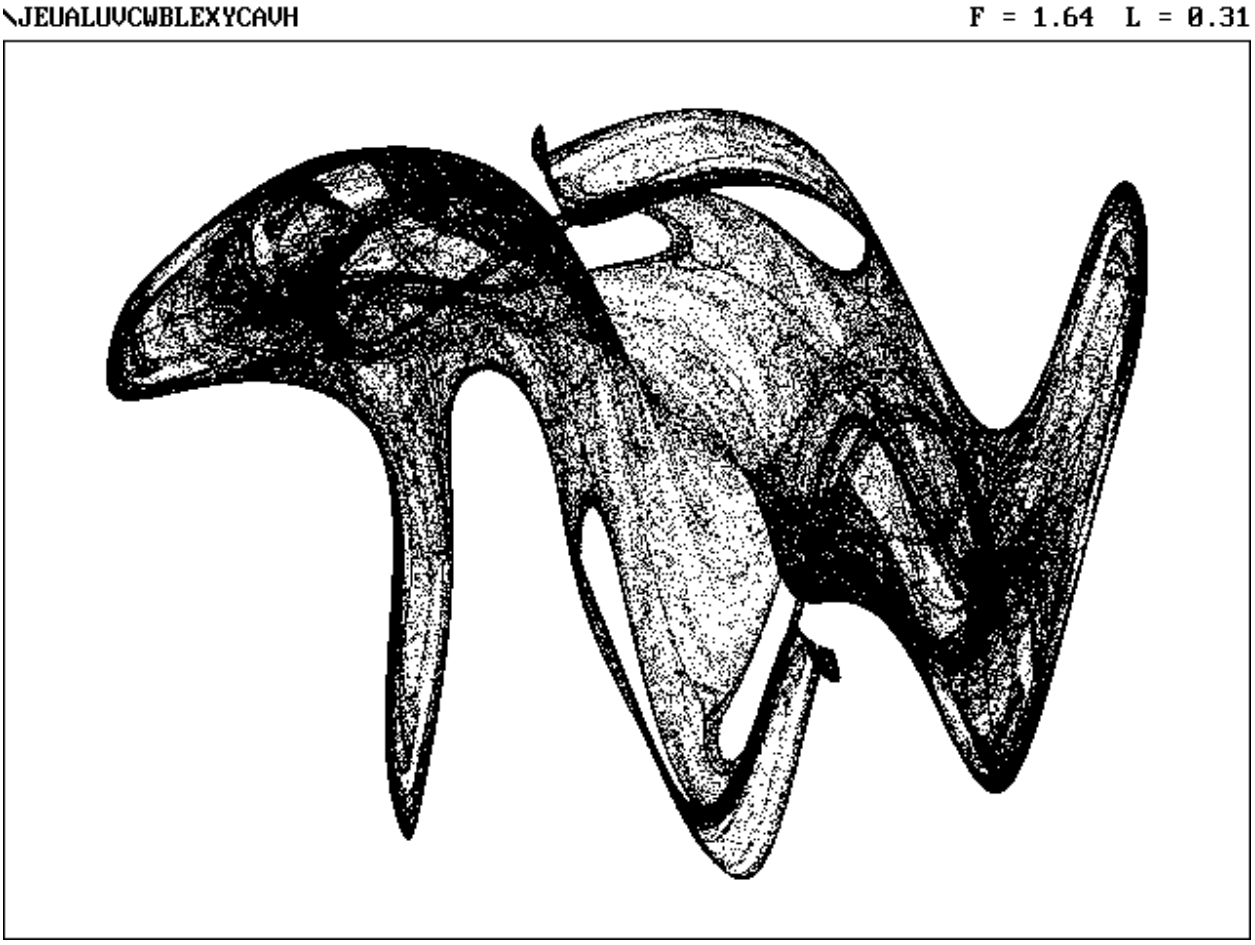


Figure 7-29. Four-dimensional special map \

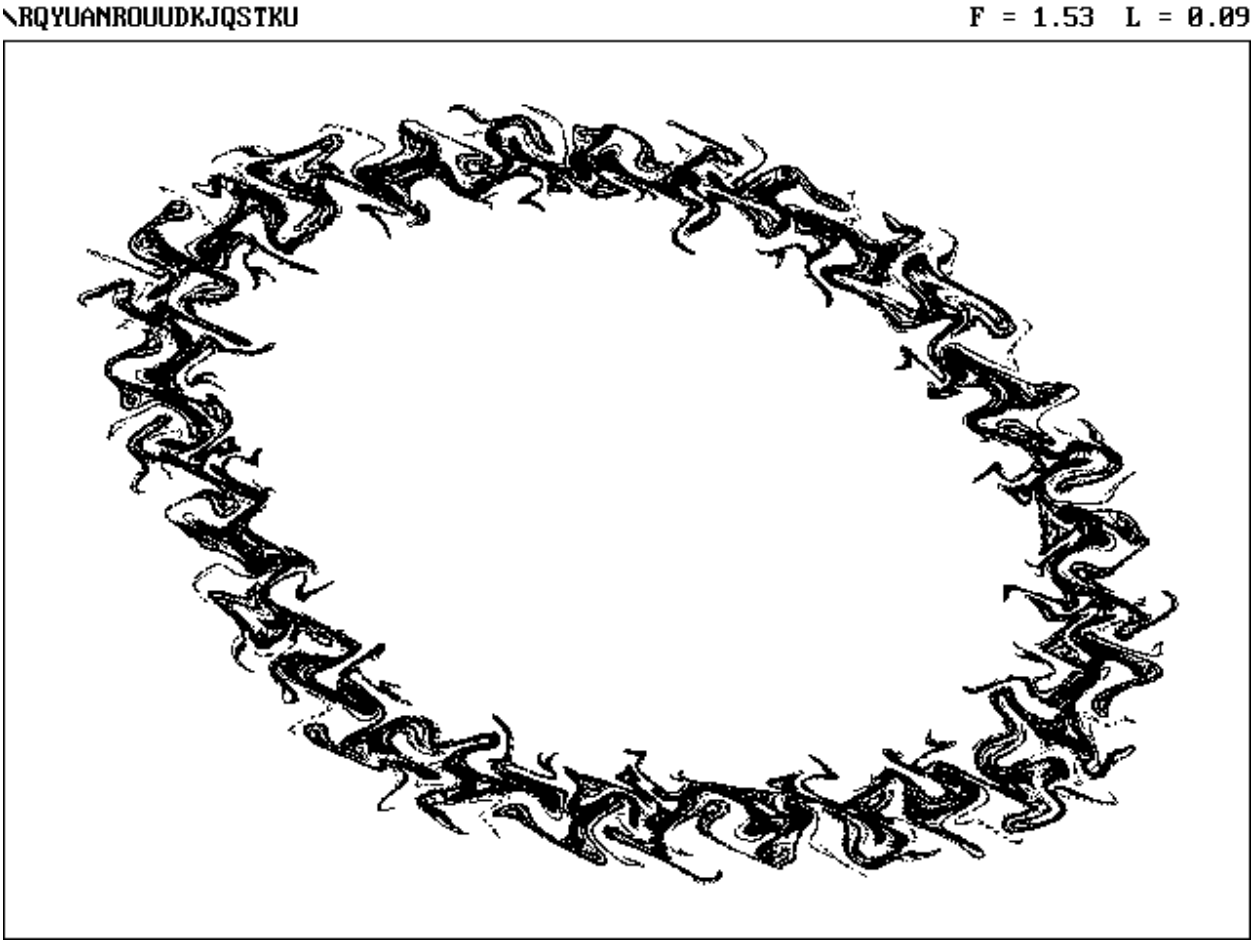


Figure 7-30. Four-dimensional special map \

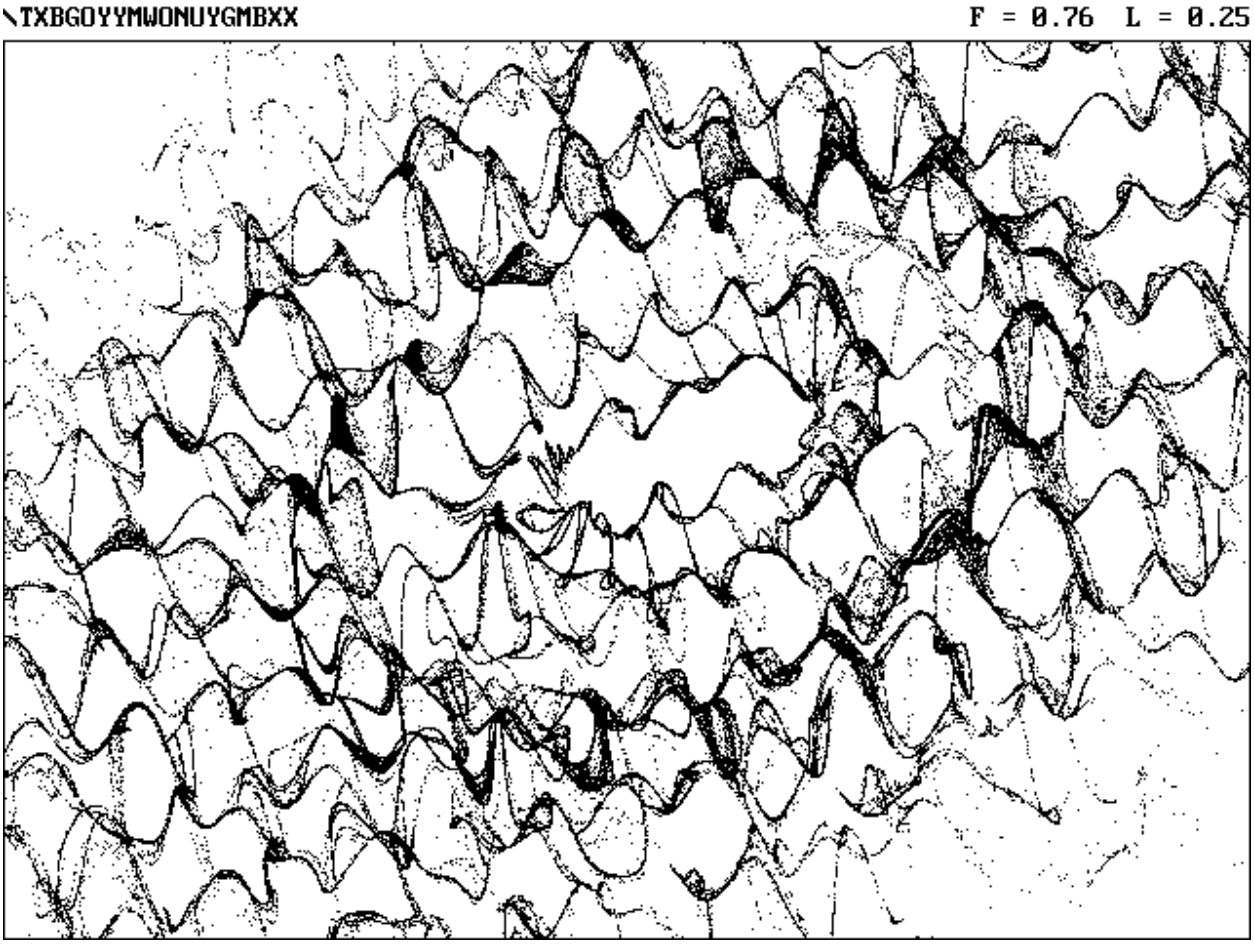


Figure 7-31. Four-dimensional special map \

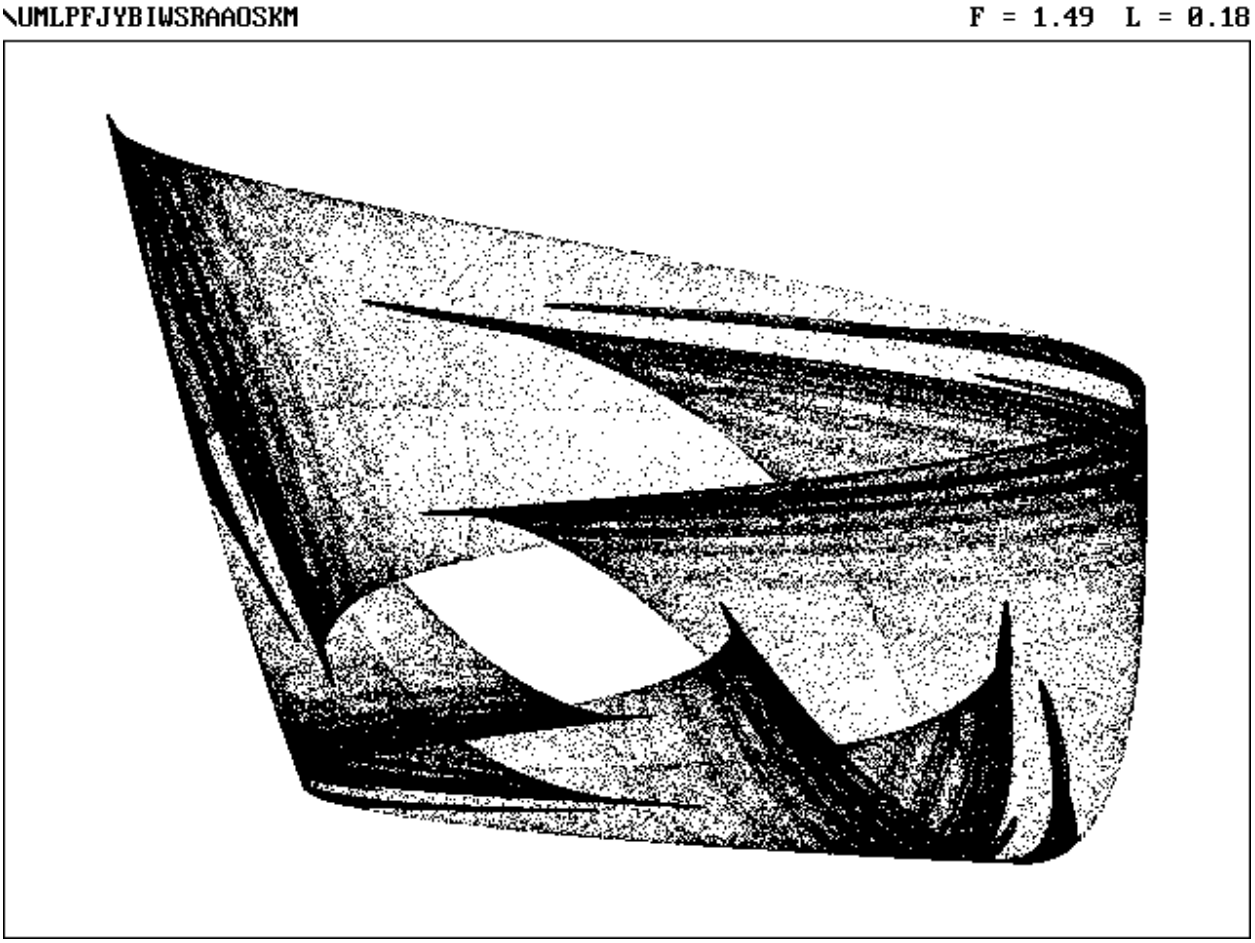
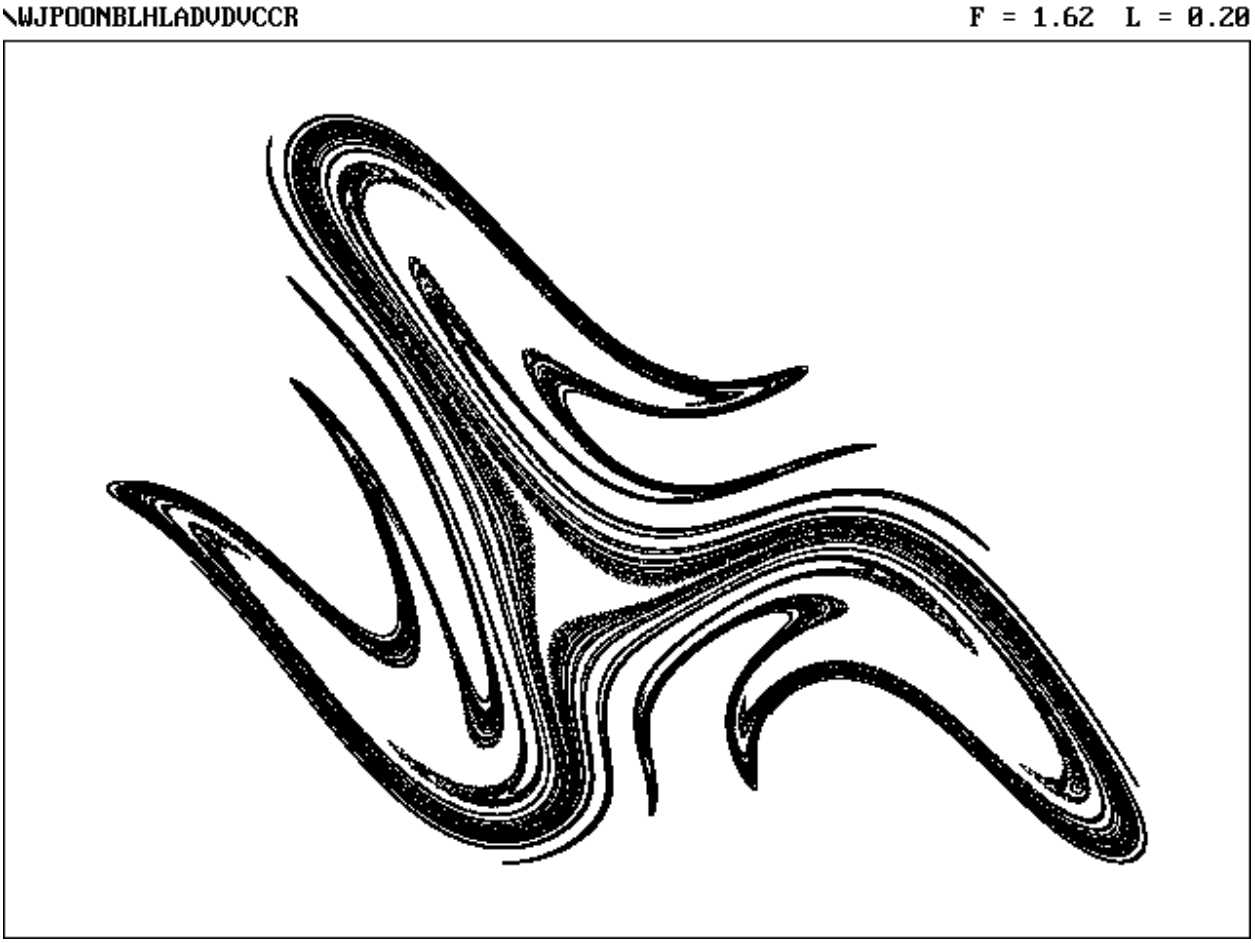


Figure 7-32. Four-dimensional special map \



7.5 Webs and Wreaths

In this section, we consider a special type of map that involves sines and cosines. The solutions are chaotic, but they are not attractors. The systems they describe are Hamiltonian. Such Hamiltonian systems obey the *Liouville theorem*, which states that the phase-space volume occupied by a set of points is conserved as the system evolves in time. Thus the orbit eventually returns arbitrarily close to any initial condition. Contrast this to dissipative systems in which the phase-space volume decreases in time, eventually collapsing all initial conditions within the basin of attraction onto the attractor. In dissipative systems, the basin of attraction is usually much larger than the attractor. The equations are as follows:

$$X_{n+1} = 10a_1 + [X_n + a_2\sin(a_3 Y_n + a_4)]\cos \_ + Y_n \sin \_$$

$$Y_{n+1} = 10a_5 - [X_n + a_2\sin(a_3 Y_n + a_4)]\sin \_ + Y_n \cos \_ \qquad \text{(Equation 7H)}$$

where $\alpha = 2\pi / (13 + 10a_6)$. The third and fourth dimensions are determined in the same way as in Section 7.1.

The special form of Equation 7H guarantees that the solution is not only area-preserving but also has circular symmetry. Furthermore, an inherent periodicity arises from the fact that α is 2π divided by an integer that ranges from 1 to 25. The periodicity is indicated by the last letter of the code (a_6): A for period-1, B for period-2, and so forth. Because of the circular symmetry and infinite extent, it is interesting to project these cases onto a sphere using the **P** command.

The program modifications required to extend the computer search to such cases are shown in **PROG26**. These cases are coded with the right bracket (]), which is ASCII 93.

PROG26. Changes required in PROG25 to search for special functions of the] type

```
1000 REM SPECIAL FUNCTION SEARCH (Webs and Wreaths)

1090 ODE% = 6                    'System is special function ]

1150 TWOPI = 6.28318530717959#  'A useful constant (2 pi)



3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 11): T% = 1
```

```
6390 IF ODE% <> 6 THEN GOTO 6450

6400    M% = 6

6410   IF N < 2 THEN AL = TWOPI / (13 + 10 * A(6)): SINAL = SIN(AL): COSAL = COS(AL)

6420    DUM = X + A(2) * SIN(A(3) * Y + A(4))

6430    XNEW = 10 * A(1) + DUM * COSAL + Y * SINAL

6440    YNEW = 10 * A(5) - DUM * SINAL + Y * COSAL

6450 RETURN
```

As you watch the patterns develop, you can see the orbit wander throughout the *XY* plane along a network of channels. The network is infinite in extent and is called a *stochastic web*. The global wandering is evidence of *minimal chaos*, and it causes the orbit eventually to leave the boundary of the computer screen. If you interrupt the calculation at some point, the resulting structure resembles a wreath or a snowflake. The infinite structure is a *tiling*, but its symmetry is slightly spoiled by the finite thickness of the web. This breaking of the symmetry eliminates the monotony and contributes to the aesthetic appeal of the patterns.

The slow wandering of the orbit throughout the web is an example of *Arnol'd diffusion*, which is named after the Russian mathematician Vladimir Arnol'd. Normally we associate diffusion with a random process in which, for example, the molecules of a gas move slowly from one region to another by countless collisions with other molecules. The presence of diffusion in such simple deterministic systems has many practical consequences such as providing a means for heating a gas of electrically charged particles (a plasma) in a magnetic field using electromagnetic waves.

These stochastic webs contain circular chains of islands, or beads on a necklace, if you prefer a different analogy, whose interiors contain periodic orbits. Surrounding the islands is a stochastic sea in which the orbits are chaotic and connected to all other points in the sea. You will also note that the Lyapunov exponents are small. Since the orbits diffuse slowly in the sea, nearby orbits remain close together for many iterations. For a similar reason, the calculated fractal dimension is lower than it should be. Recall that the dimension calculation is based

on the previous 500 iterates, whose values tend to be nearly equal in this case.

Examples of stochastic webs produced by **PROG26** are shown in Figures 7-33 through 7-40. Because of the slow diffusion of the orbit, these cases provide a good opportunity to exhibit the time variation with colors as shown in Plates 31 and 32. You may want to try different values of *NMAX%* in line 1050 to control the rate at which the colors change. Web maps provide a perfect illustration of how chaos and determinism coexist. The underlying symmetry of the equations is evident in the figures, but the orbit exhibits apparently random motion within the chaotic region.

Figure 7-33. Four-dimensional web map (period-4)

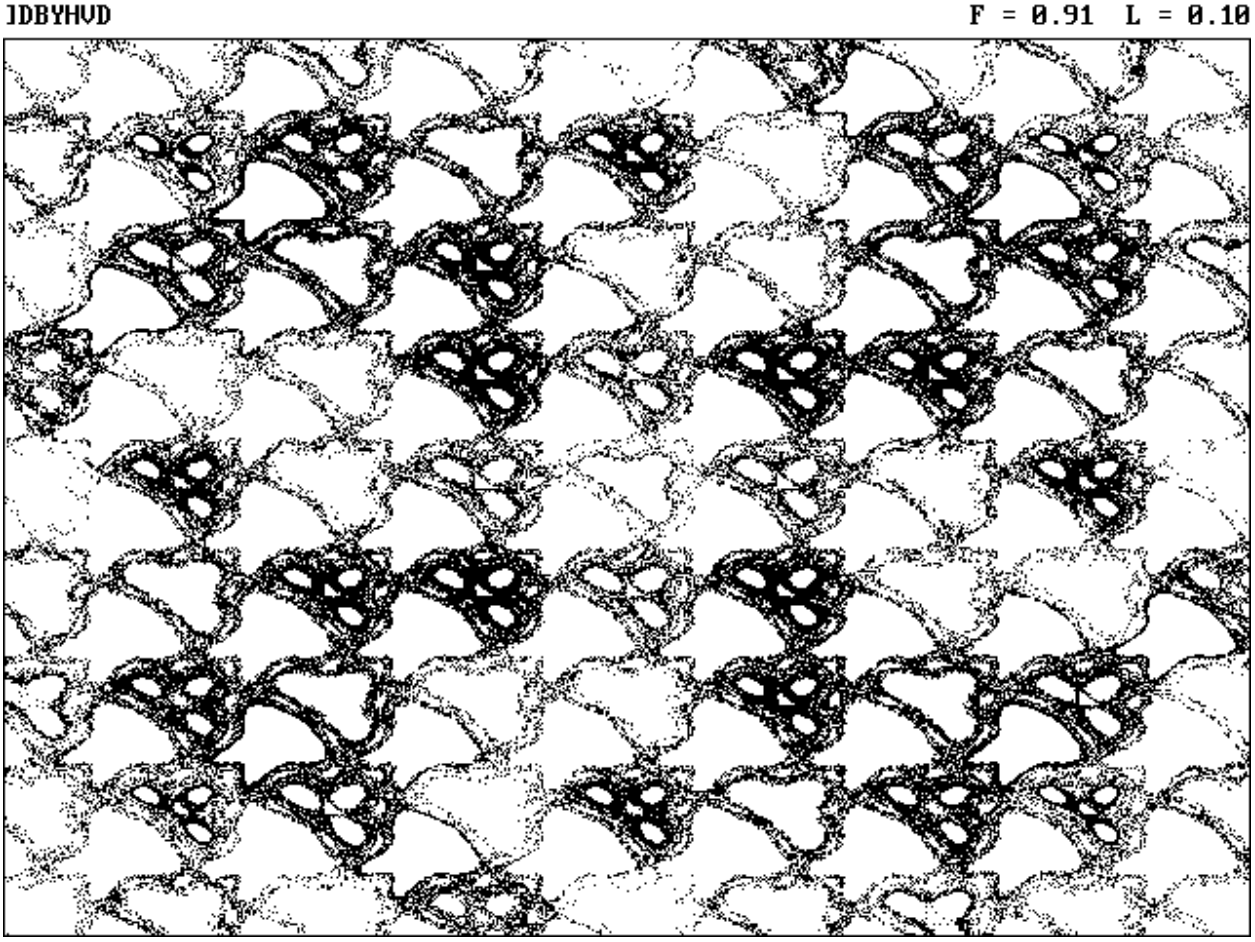


Figure 7-34. Four-dimensional web map (period-13)

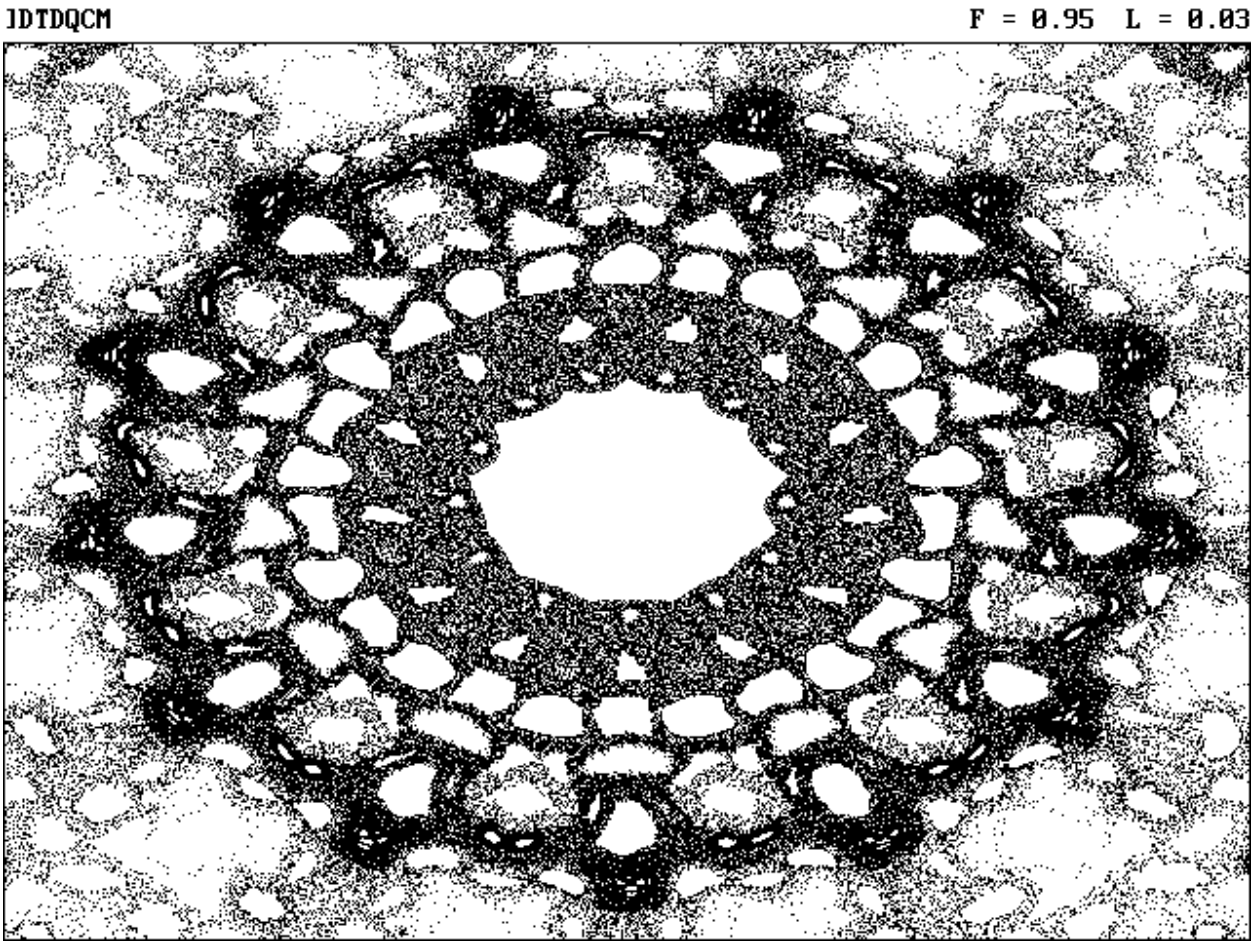


Figure 7-35. Four-dimensional web map (period-5)

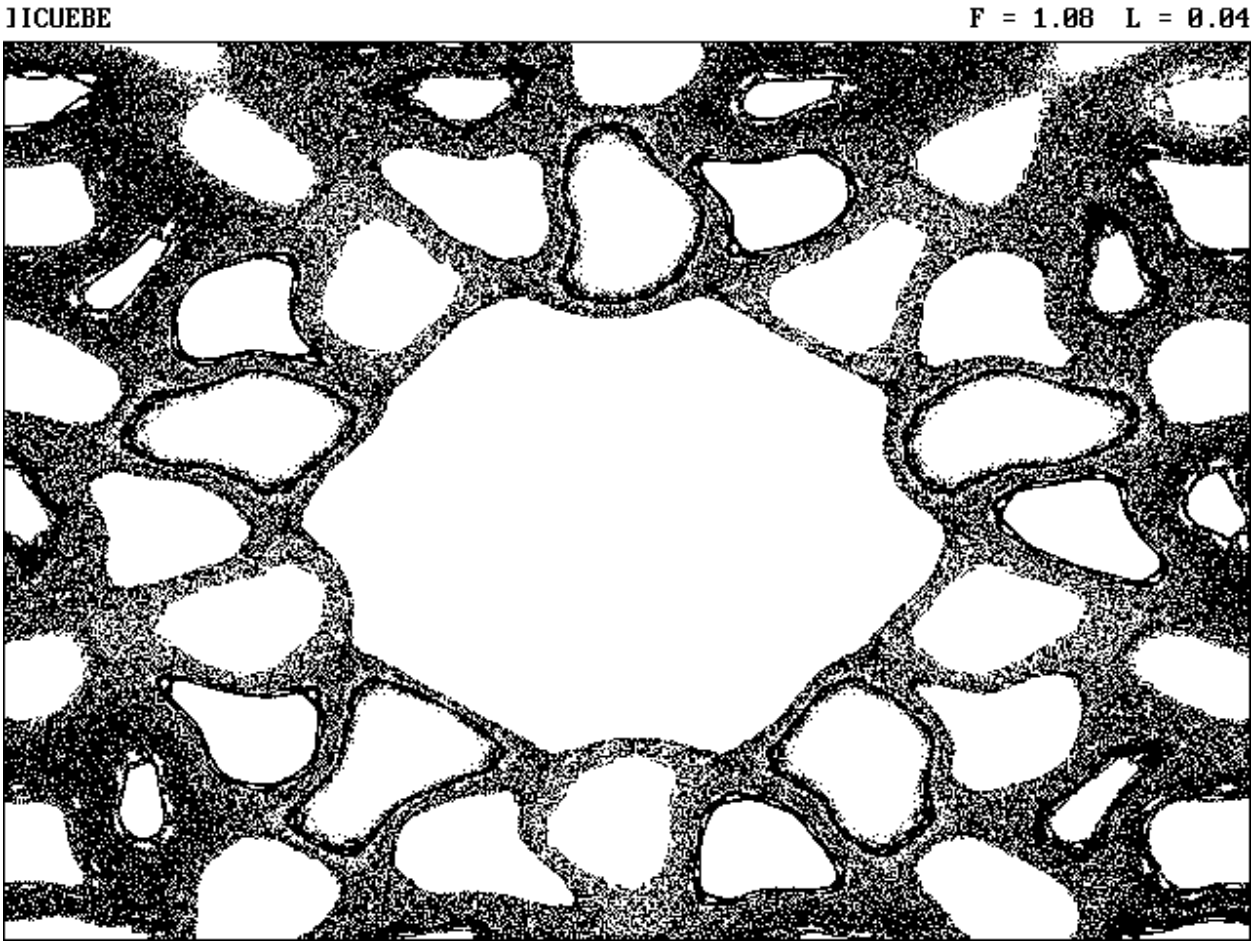


Figure 7-36. Four-dimensional web map (period-12)

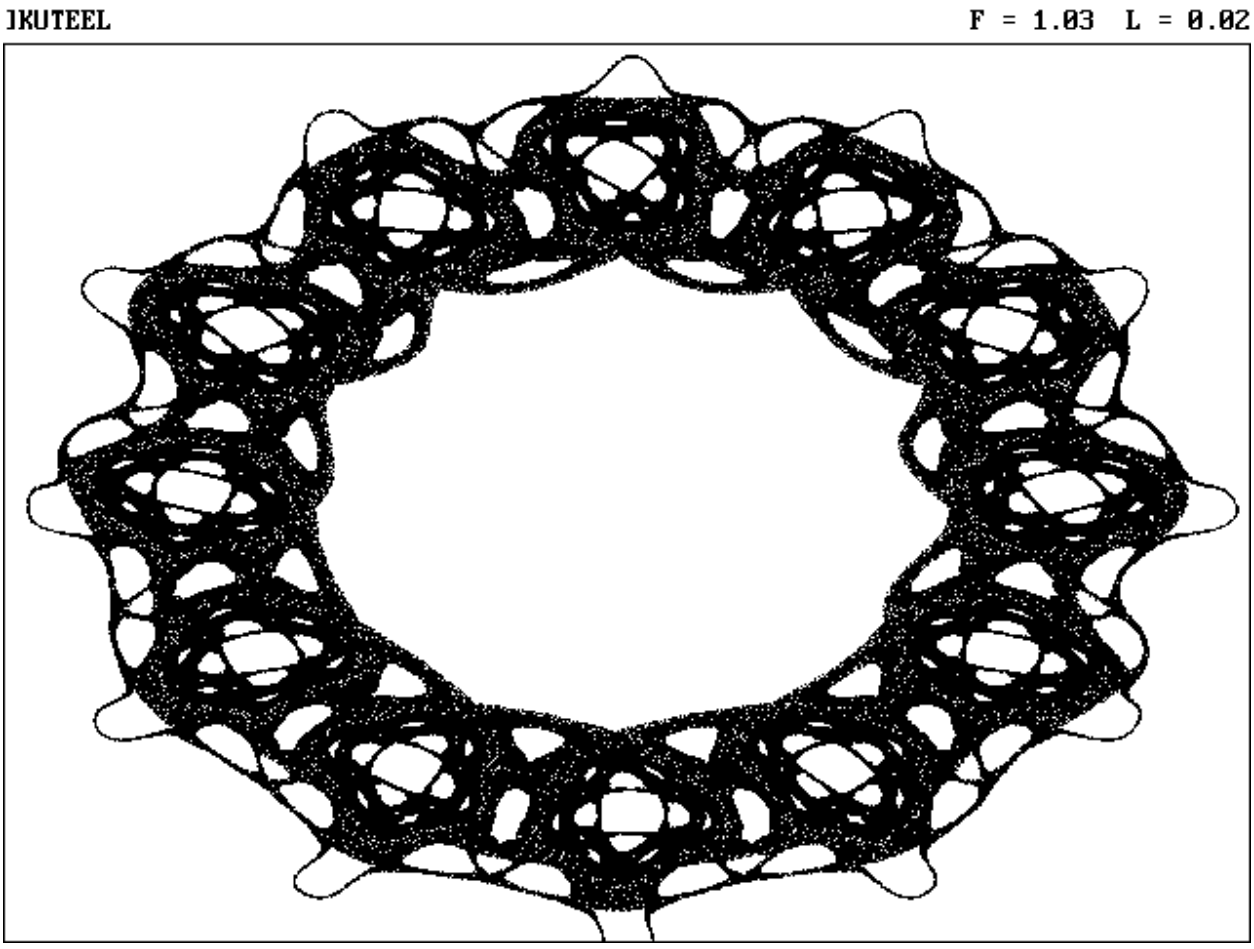


Figure 7-37. Four-dimensional web map (period-12)

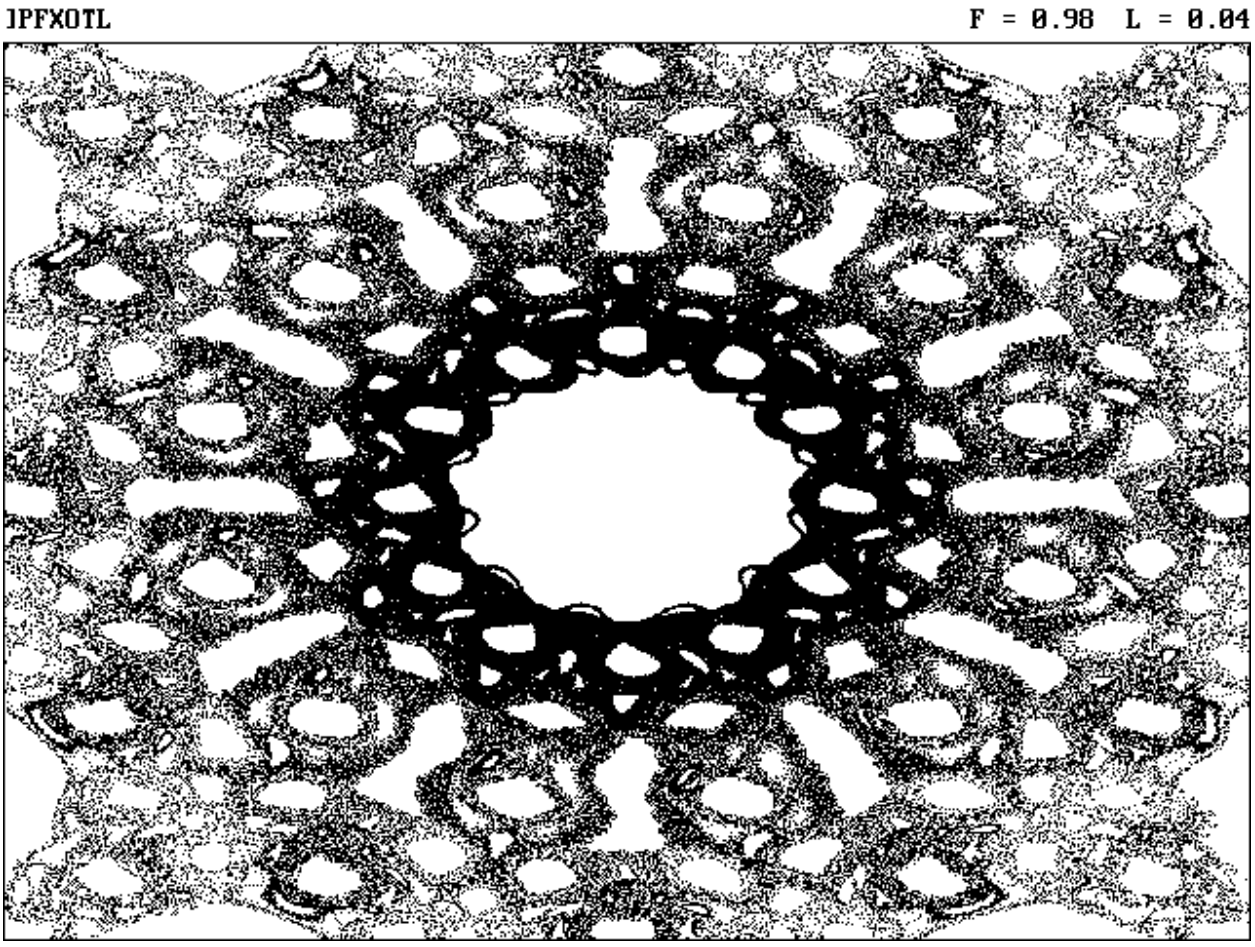


Figure 7-38. Four-dimensional web map (period-7)

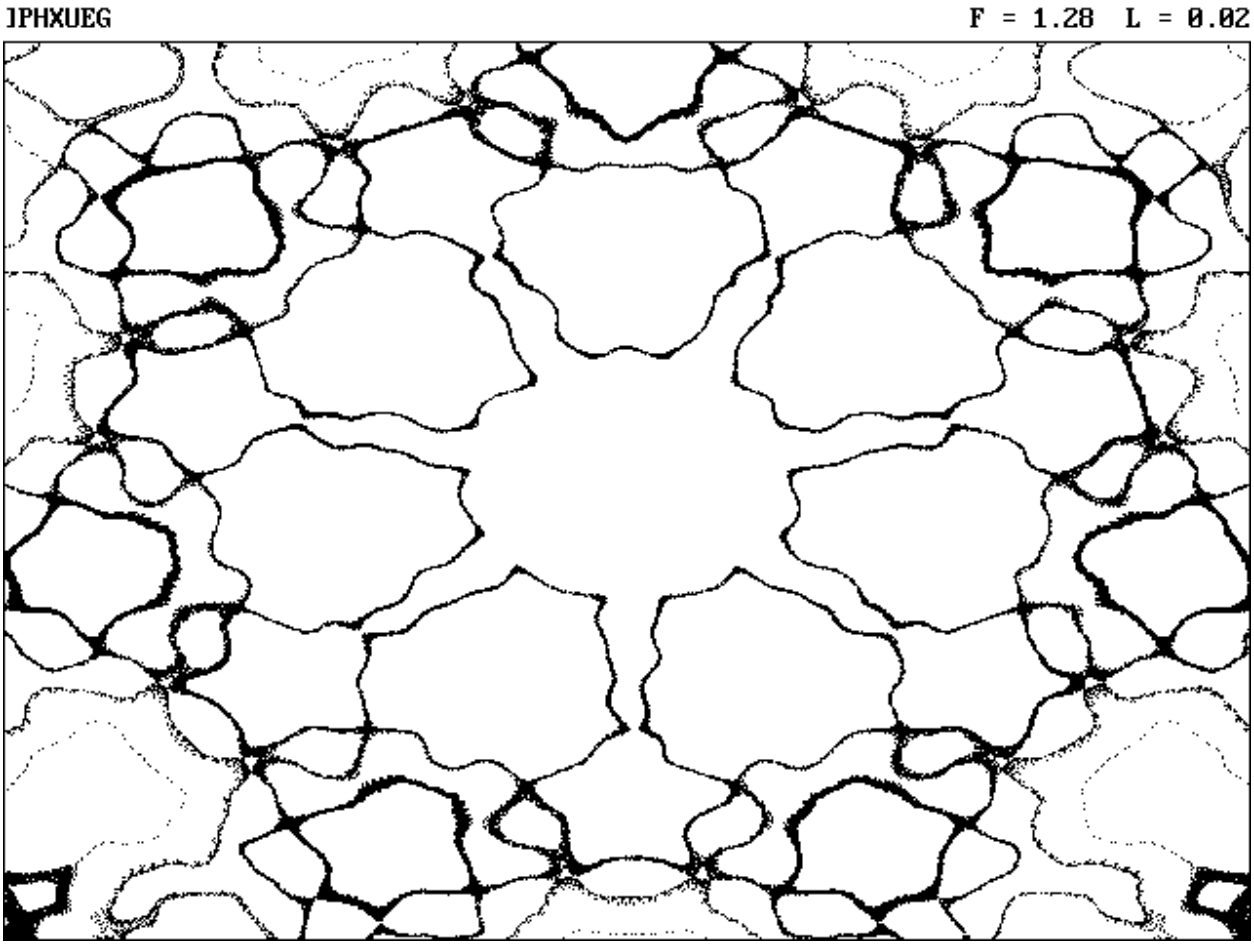


Figure 7-39. Four-dimensional web map (period-23)

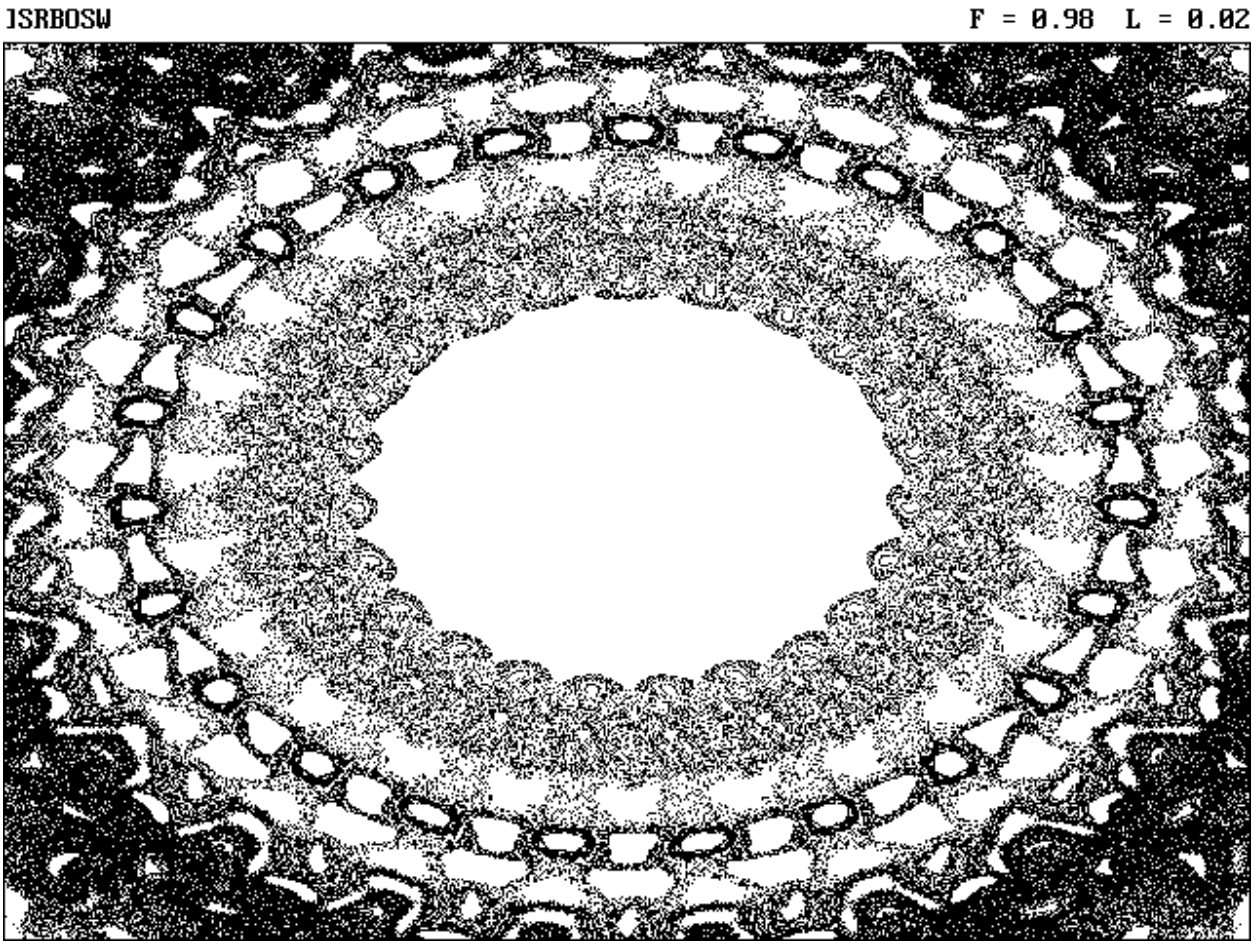
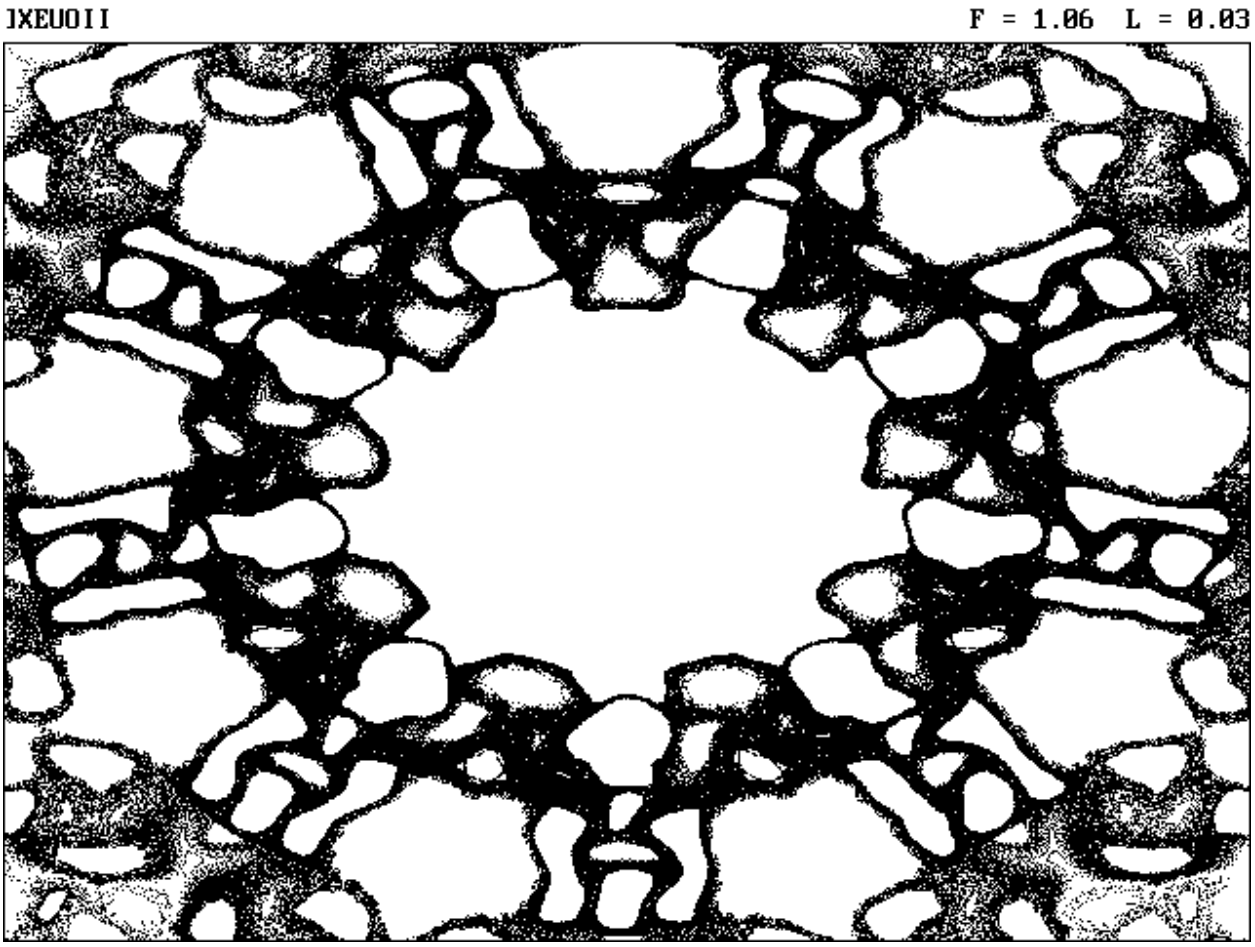


Figure 7-40. Four-dimensional web map (period-9)



7.6 Swings and Springs

All the previous examples of maps and differential equations in this book share the property that the right-hand sides of the equations are independent of the iteration number (*N*) or time (*t*). Such equations are called *autonomous*. For a given set of initial conditions, they produce the same solution for whatever time or iteration number they are started.

Some important physical processes are most conveniently expressed by *nonautonomous* equations. An example is a driven (forced), damped, linear, harmonic oscillator, which is described by the following equations:

X' = Y

Y' = -X - bY + A sin _t (Equation 7I)

In Equation 7I, *b* is the damping constant (friction), A is the amplitude of the drive (forcing) function, and ω is the angular frequency (radians per second) of the drive. The friction force is assumed to be proportional and opposite to the velocity (-Y), although other forms give qualitatively similar results. This type of friction is called *linear damping*.

The usual trick for dealing with nonautonomous equations is to introduce an additional variable (say *Z*) and rewrite Equation 7I, for example, as

X' = Y

Y' = -X - bY + A sin Z

Z' = _ (Equation 7J)

Equation 7J contains a nonlinearity (sin *Z*), but it does not have chaotic solutions. The solution (for positive *b*) is a limit cycle with frequency ω. The limit cycle is largest when the damping is small (*b* positive but much less than 1) and ω is 1, corresponding to *resonance*.

To obtain interesting chaotic solutions, we need additional nonlinear terms. We will restrict these nonlinearities to odd polynomials (*X*, X^3, X^2Y, and so forth) in the *Y'* equation to preserve the symmetry of the oscillation. A general form with odd polynomials up to third order is as follows:

$$X' = a_1 Y$$

$$Y' = a_2 X + a_3 X^3 + a_4 X^2 Y + a_5 X Y^2 + a_6 Y + a_7 Y^3 + a_8 \sin Z$$

$$Z' = a_9 + 1.3 \qquad \text{(Equation 7K)}$$

If the product $a_1 a_2$ is negative, these equations represent the motion of a mass oscillating on a nonlinear spring or a pendulum swinging through a large angle (but not going over the top). If a_3/a_2 is positive, the spring gets stiffer when stretched or compressed (*hard spring*). If a_3/a_2 is negative, the spring gets weaker when stretched or compressed (*soft spring*). If a_3/a_2 is -1/6, the solution approximates a vigorously swinging pendulum. If the product $a_1 a_2$ is positive and a_3/a_2 is negative, the system models a buckled beam and is called the *Duffing two-well oscillator*. The final combination ($a_1 a_2$ and a_3/a_2 both positive) is unstable and has an unbounded solution.

In Equation 7K, we can exploit the fact that *Z* enters only through the term sin *Z*; thus it is periodic with period 2π. Whenever *Z* exceeds 2π, we can subtract 2π from it without changing the result. This trick keeps *Z* bounded in the range 0 to 2π rather than letting it march off to infinity as it would otherwise do. The *Z*-coordinate then becomes the *phase angle* of the drive function. Note that the MOD function in most versions of BASIC works correctly only on integer variables, so it should not be used for the above purpose. The +1.3 term in the *Z* equation ensures that the phase angle always increases in time for $-1.2 \le a_9 \le 1.2$. The fourth variable (*W*) is proportional to time as in the previous examples.

The program modifications required to extend the computer search to such cases are shown in **PROG27**. These cases are coded with the circumflex (^), which is ASCII 94.

PROG27. Changes required in PROG26 to search for special functions of the ^ type

```
1000 REM SPECIAL FUNCTION SEARCH (Swings and Springs)

1090 ODE% = 7                    'System is special function ^


3050    IF ODE% = 1 OR ODE% = 7 THEN L = L / EPS
```

```
3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 12): T% = 1



6450 IF ODE% <> 7 THEN GOTO 6500

6460    M% = 9

6470    XNEW = X + EPS * A(1) * Y

6480    YNEW = Y + EPS * (A(2) * X + A(3) * X * X * X + A(4) * X * X * Y + A(5)
* X * Y * Y + A(6) * Y + A(7) * Y * Y * Y + A(8) * SIN(Z))

6490    ZNEW = Z + EPS * (A(9) + 1.3): IF ZNEW > TWOPI THEN ZNEW = ZNEW - TWOPI

6500 RETURN
```

Examples of attractors produced by **PROG27** are shown in Figures 7-41 through 7-48. These cases are displayed as slices that show the orbit at 16 different drive phases. As you scan across them left to right and top to bottom, you can see the stretching and folding that are characteristics of strange attractors and that account for their fractal microstructure and for the sensitivity to initial conditions.

Figure 7-41. Slices of a four-dimensional special map ^

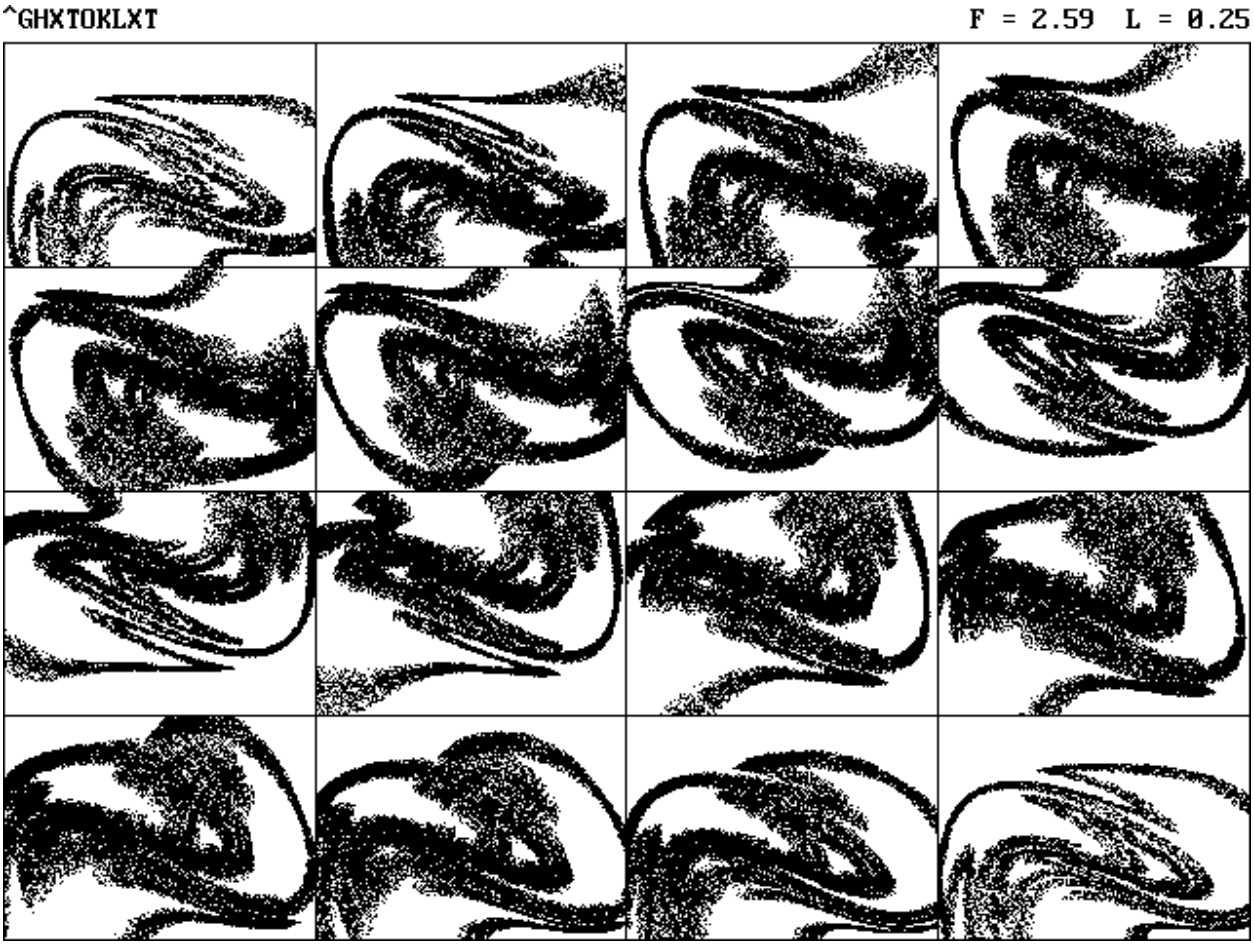


Figure 7-42. Slices of a four-dimensional special map ^

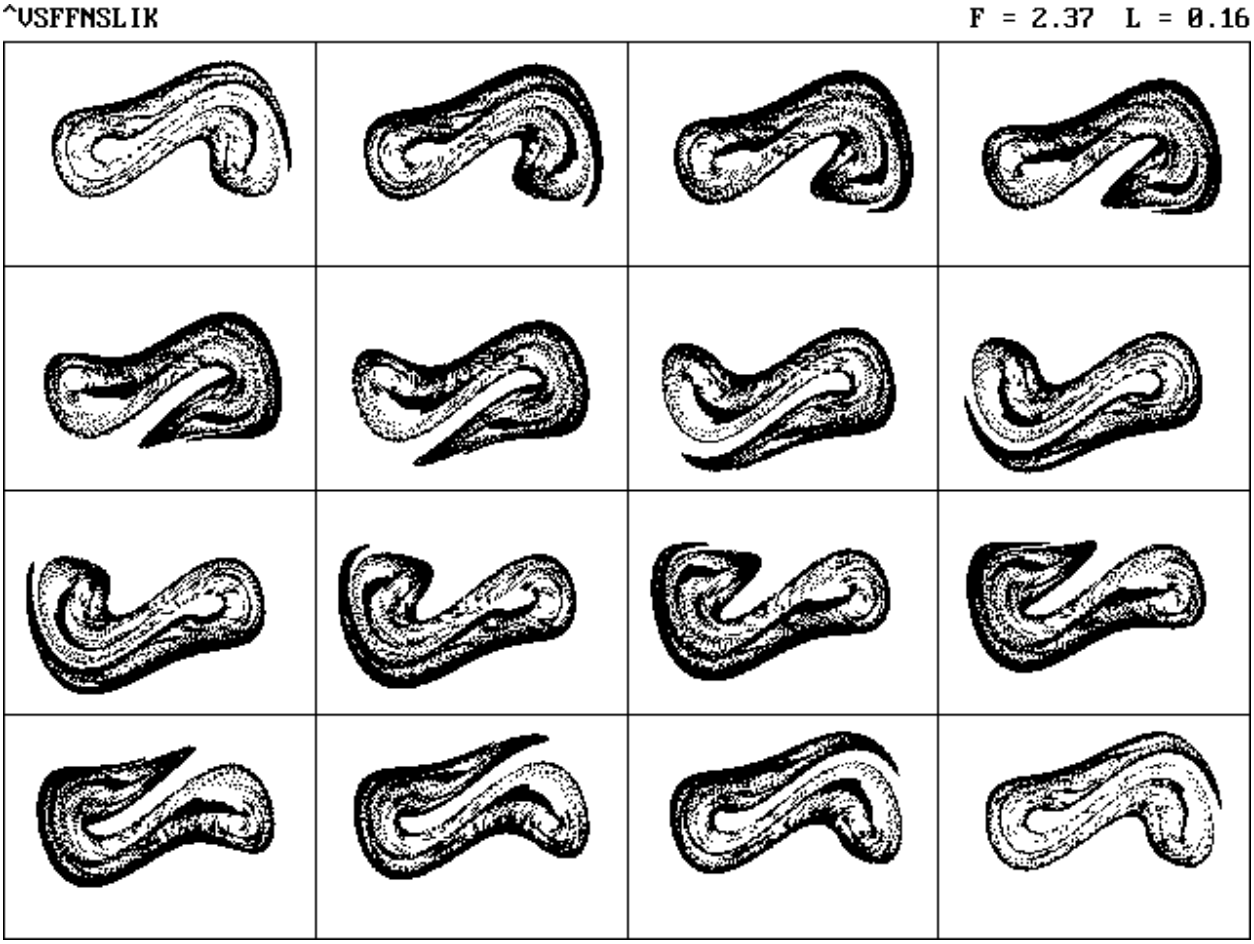


Figure 7-43. Slices of a four-dimensional special map ^

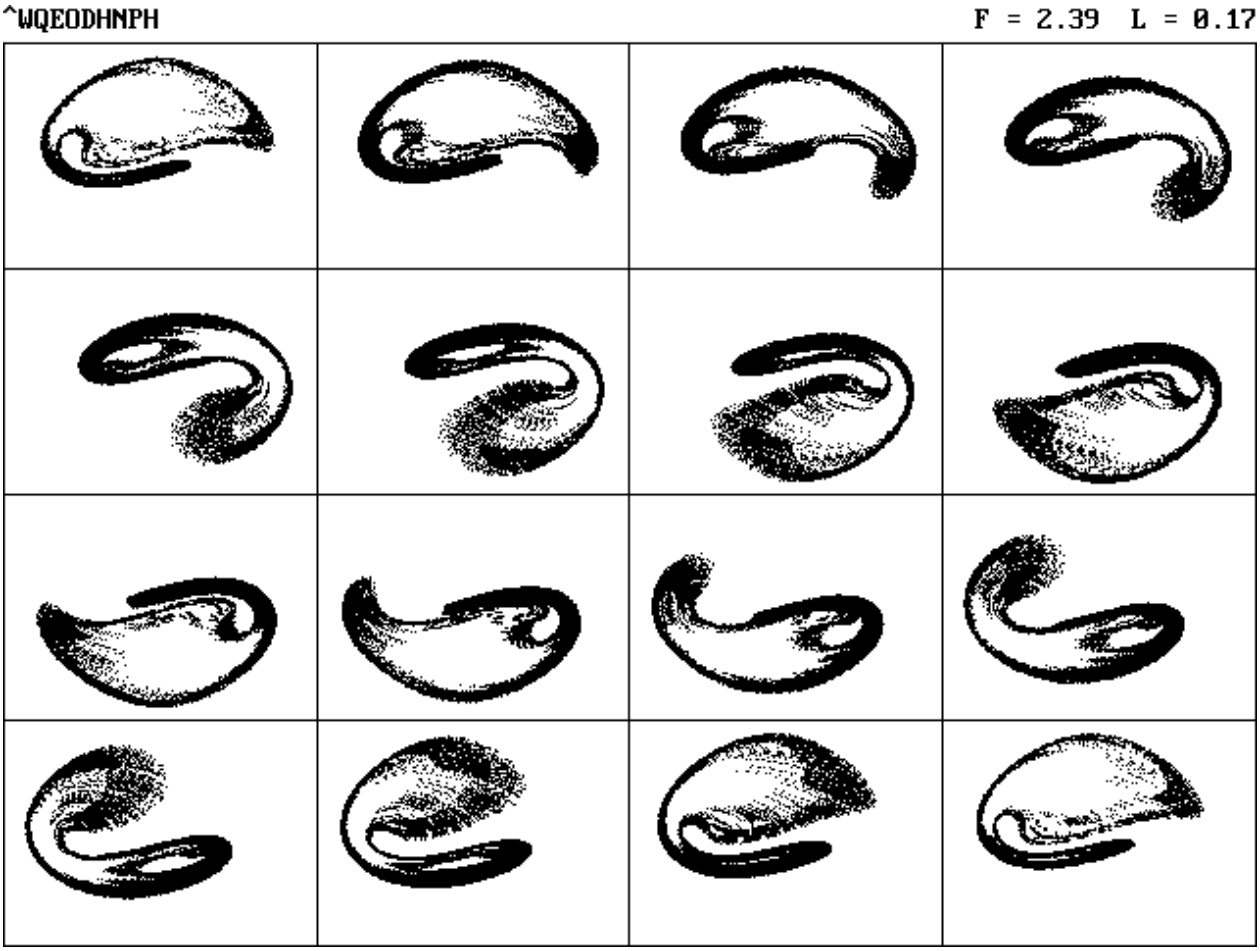


Figure 7-44. Slices of a four-dimensional special map ^

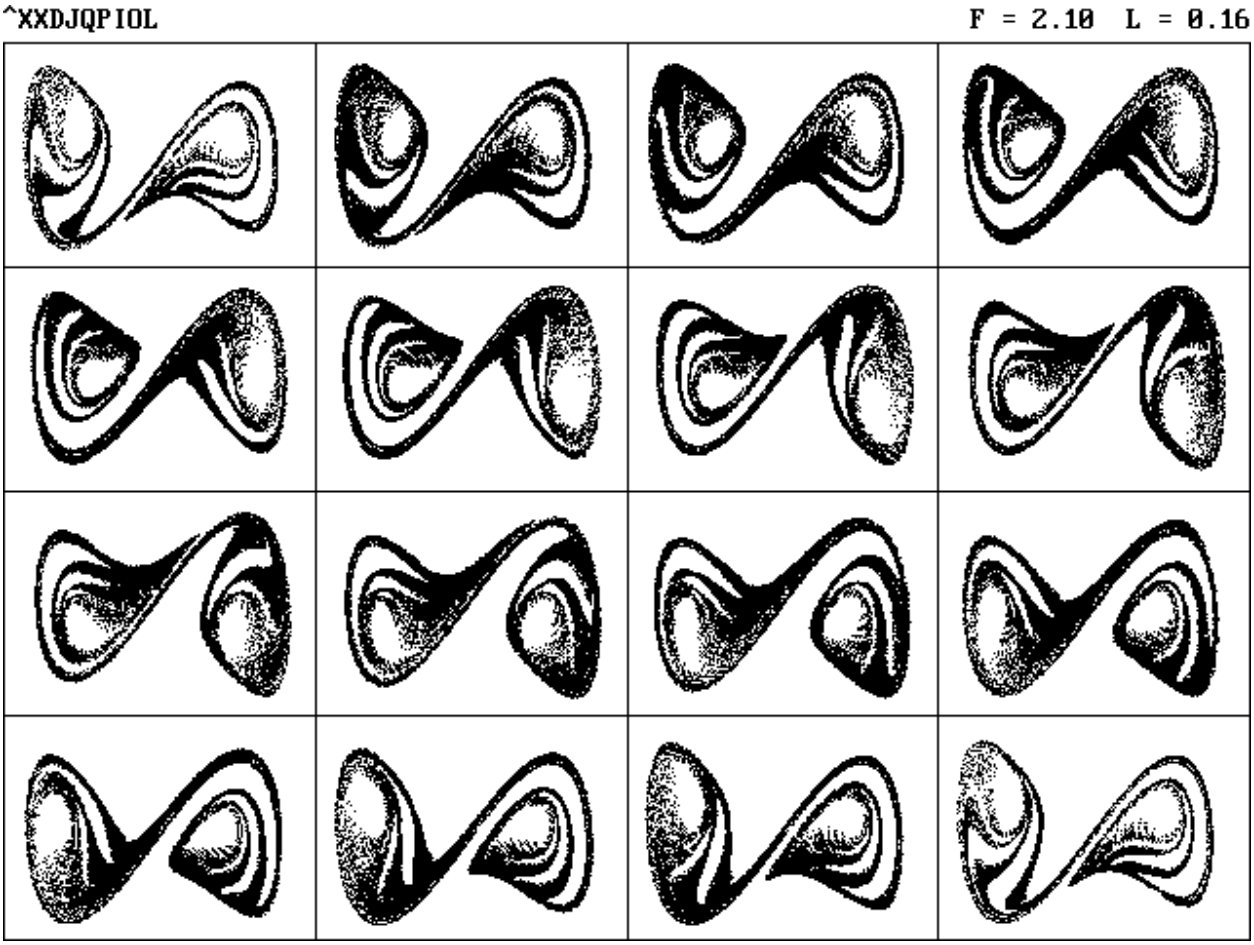


Figure 7-45. Slices of a four-dimensional special map ^

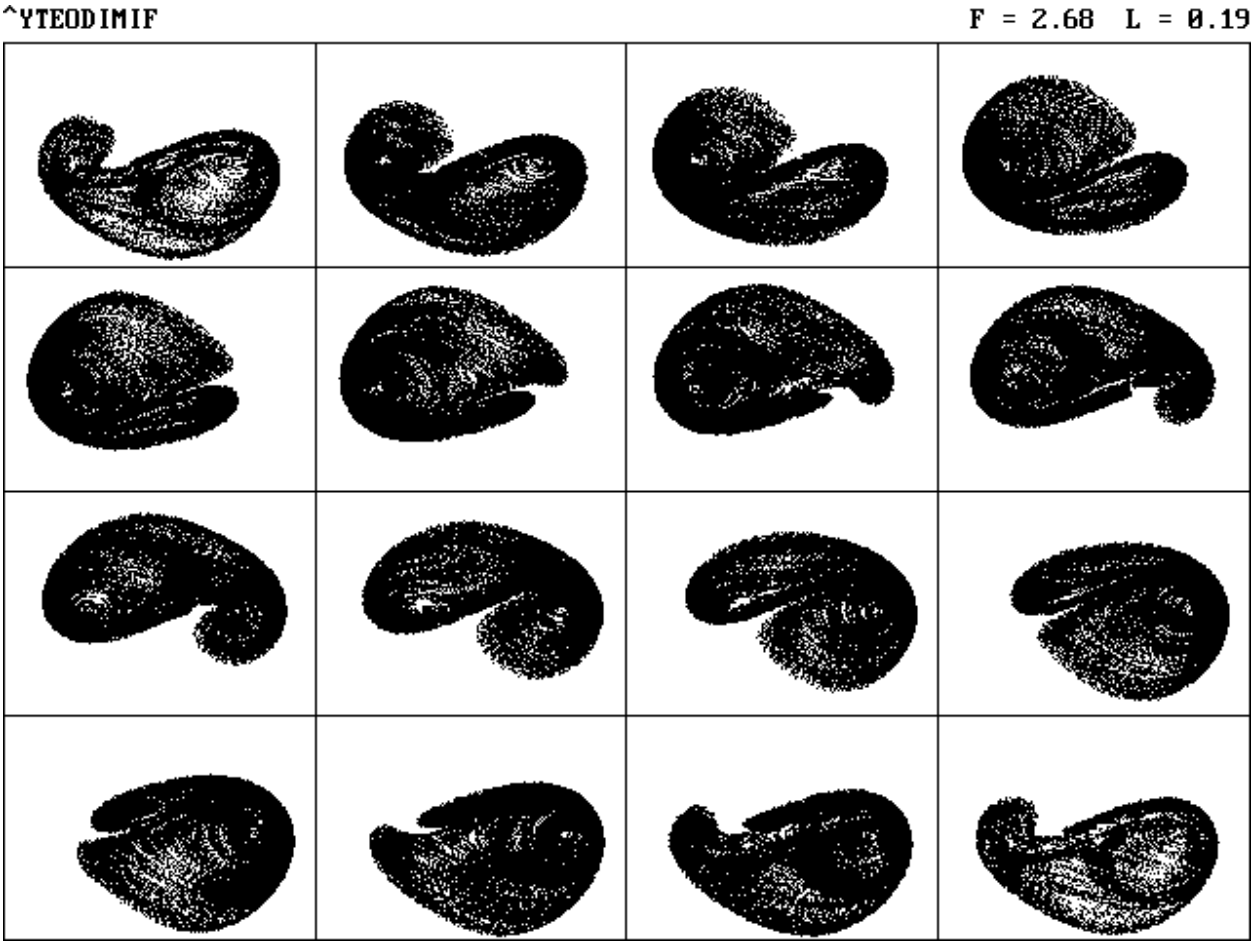


Figure 7-46. Slices of a four-dimensional special map ^

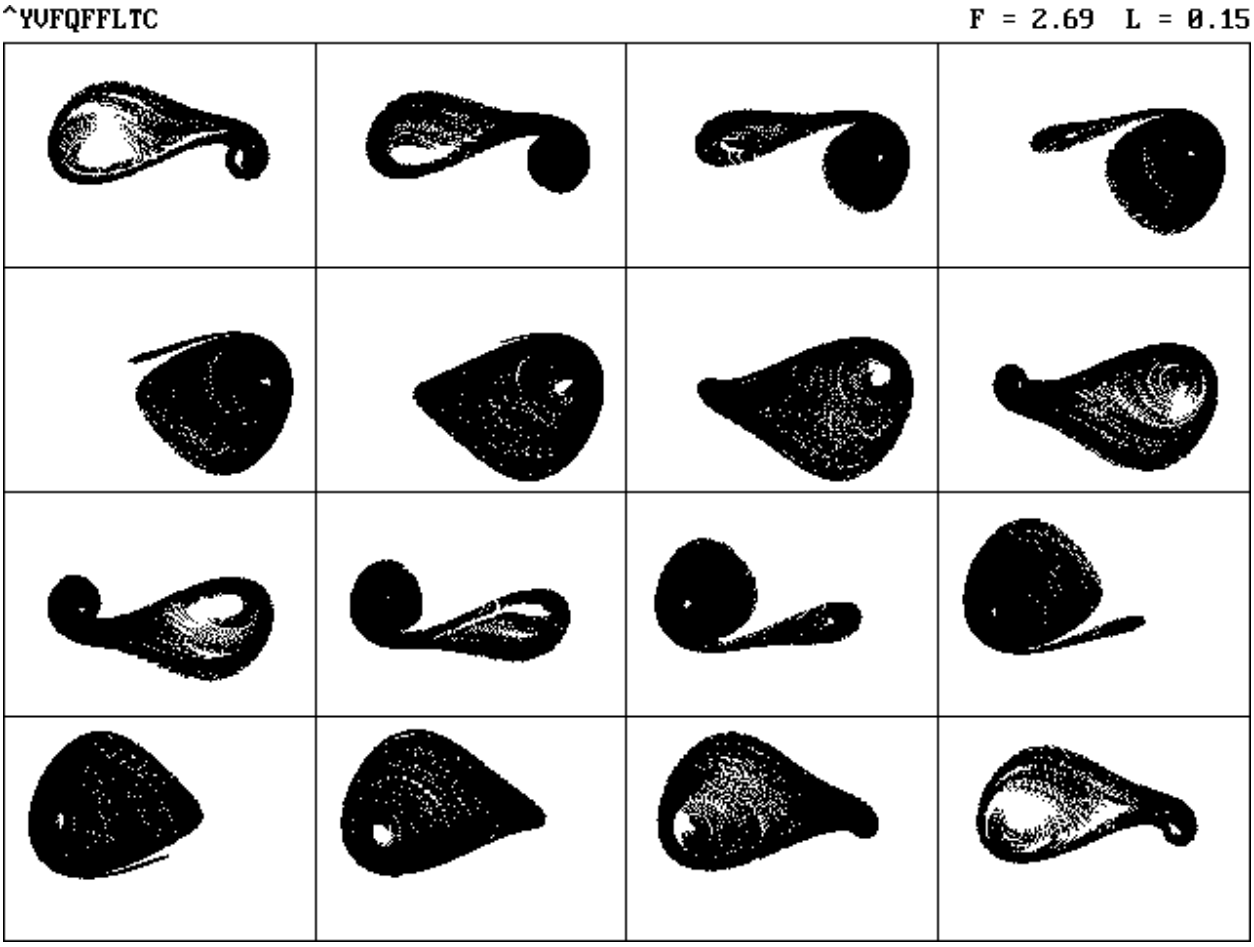


Figure 7-47. Slices of a four-dimensional special map ^

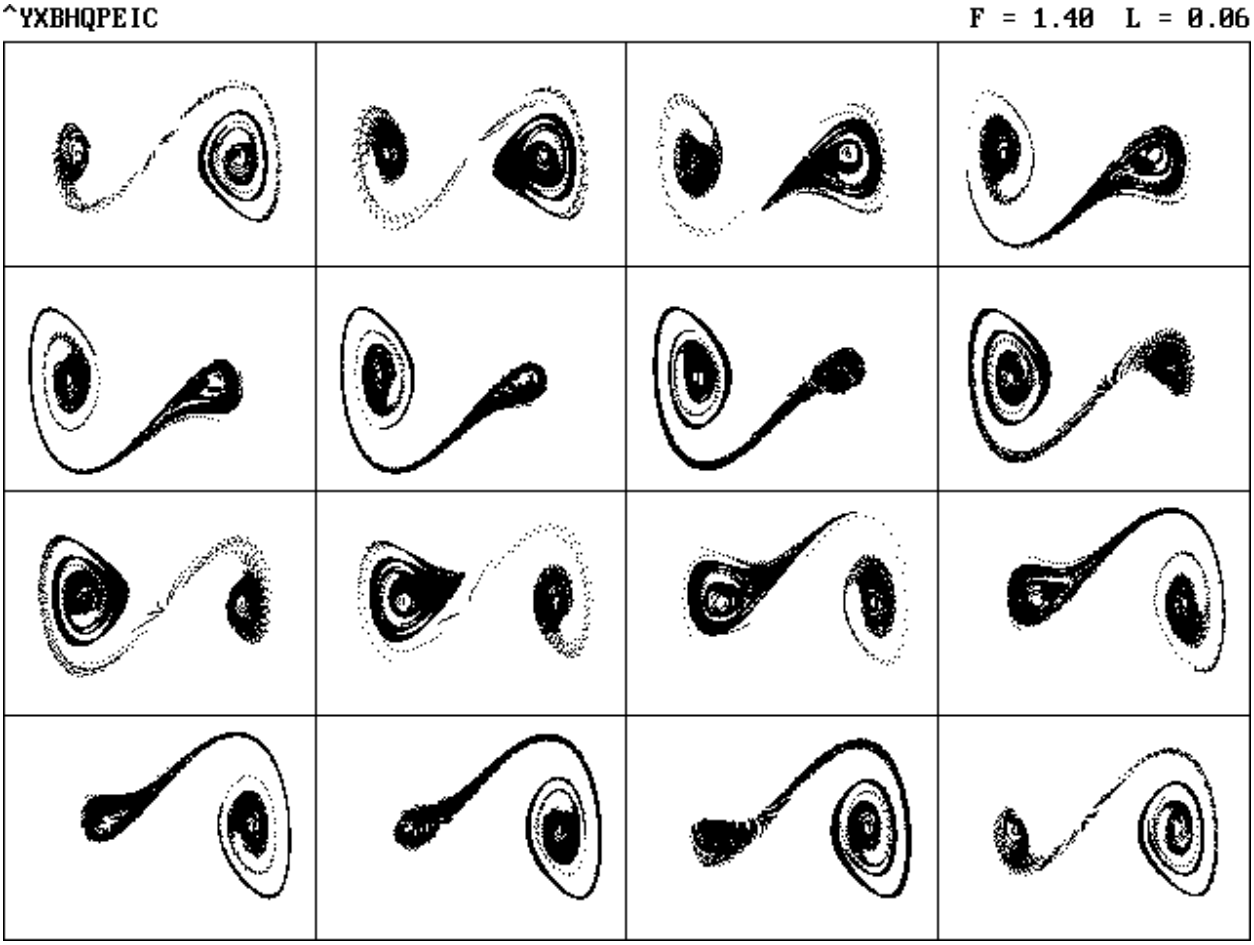
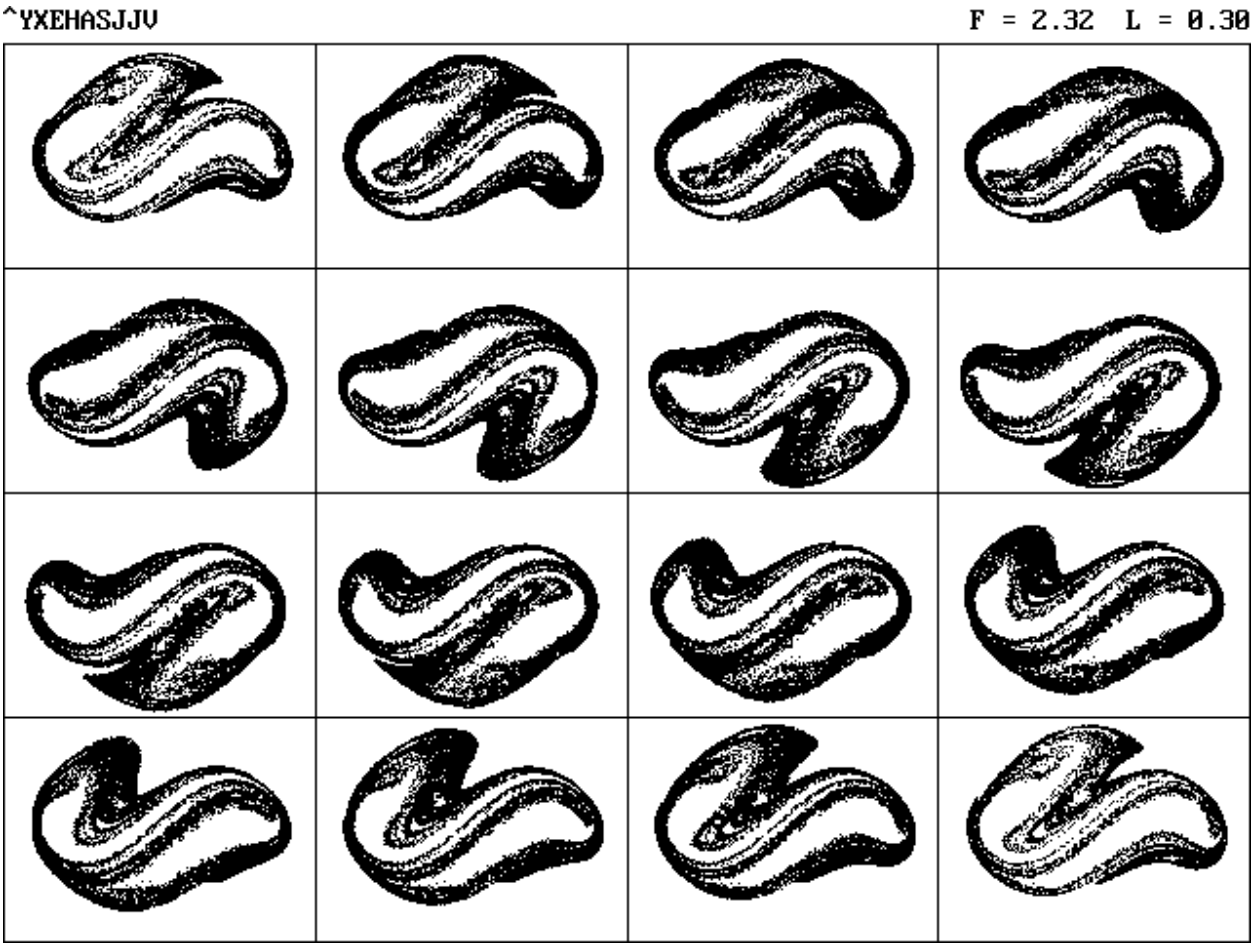


Figure 7-48. Slices of a four-dimensional special map ^



These cases provide an ideal opportunity to animate the third dimension (drive phase). If you have not fanned through the figures in the upper-right-hand corner of the odd pages of the book, do so now. Be sure to fan in the forward direction (low to high page numbers). These figures were produced using 64 phase slices of the attractor `^VYGJBPIJN` with 10 million iterations.

You can see several cycles of stretching and folding. Note that there are three diagonal bands that run from the lower left to the upper right of the attractor. The three bands are stretched and compressed into a single band, while two additional bands enter first from the upper left and then from the lower right. Thus one of the three bands consists of three smaller bands, one of which consists of three even smaller bands, and so forth. Such an infinitely layered band is called a *thick line* or a *Cantor one-manifold*. Viewed in three dimensions, the thick line would appear as a *thick surface* or a *Cantor two-manifold*. There is perhaps no clearer

illustration anywhere in this book of the way strange attractors are formed and acquire their fractal microstructure.

7.7 Roll Your Own

Perhaps this is a good place to leave you with this brief taste of the immense variety of nonlinear functions and equations, nearly all of which admit strange attractors that can be identified and examined using the technique described in this book. The possibilities are limited only by your imagination, and you can be assured that nearly every strange attractor that you discover has never been seen before. There surely exist classes of objects yet to be discovered that are of mathematical and artistic interest.

If you decide to pursue such an exploration, you might start with some of the other nonlinear functions and operators that are built into the BASIC language. Table 7-1 lists a number of interesting possibilities. They are divided into *mathematical functions*, which should be independent of the machine or programming language; *machine functions*, which are dependent on the machine or language; *nonlinear operators*, which are supported by nearly all versions of BASIC; and *advanced functions*, which are supported by VisualBASIC for MS-DOS. You may use them in combinations to invent complicated forms that belong to you alone!

Table 7-1. Nonlinear functions and operators supported by most versions of BASIC

Mathematical Functions	Machine Functions	Nonlinear Operators	Advanced Functions
ABS	FRE	^	DAY
ATN	INP	*	HOUR
CINT	PEEK	/	MINUTE
COS	PLAY	\	MONTH
EXP	POINT	MOD	NOW
FIX	RND	NOT	QBCOLOR
INT	TIMER	AND	RGB
LOG		OR	SECOND
SGN		XOR	WEEKDAY
SIN		EQV	YEAR
SQR		IMP	
TAN			

Chapter 8

Epilogue

It would be an injustice to leave you with the impression that the main use of the ideas in this book is to make pretty pictures. This final chapter describes some of the scientific applications of the method used to generate this large collection of strange attractors. It also suggests some additional explorations you might want to undertake as an extension of both the scientific and artistic aspects of the work described in this book.

8.1 How Common is Chaos?

For hundreds of years, scientists have used equations like those in the previous chapters to describe nature. It is remarkable that almost no one recognized the chaotic solutions to those equations until the last few decades of the 20th century. Now researchers in many disciplines are beginning to see chaos under every rock. It is reasonable to wonder whether chaos is the rule or the exception.

Suppose we had a system of equations of sufficient complexity and with sufficiently many coefficients that it could be used to model most natural processes. We could then attempt to quantify the occurrence of chaos in these equations and draw an inference about the occurrence of chaos in nature. The equations in the previous chapters, especially those involving polynomials of high dimension and high order, might approximate such a system.

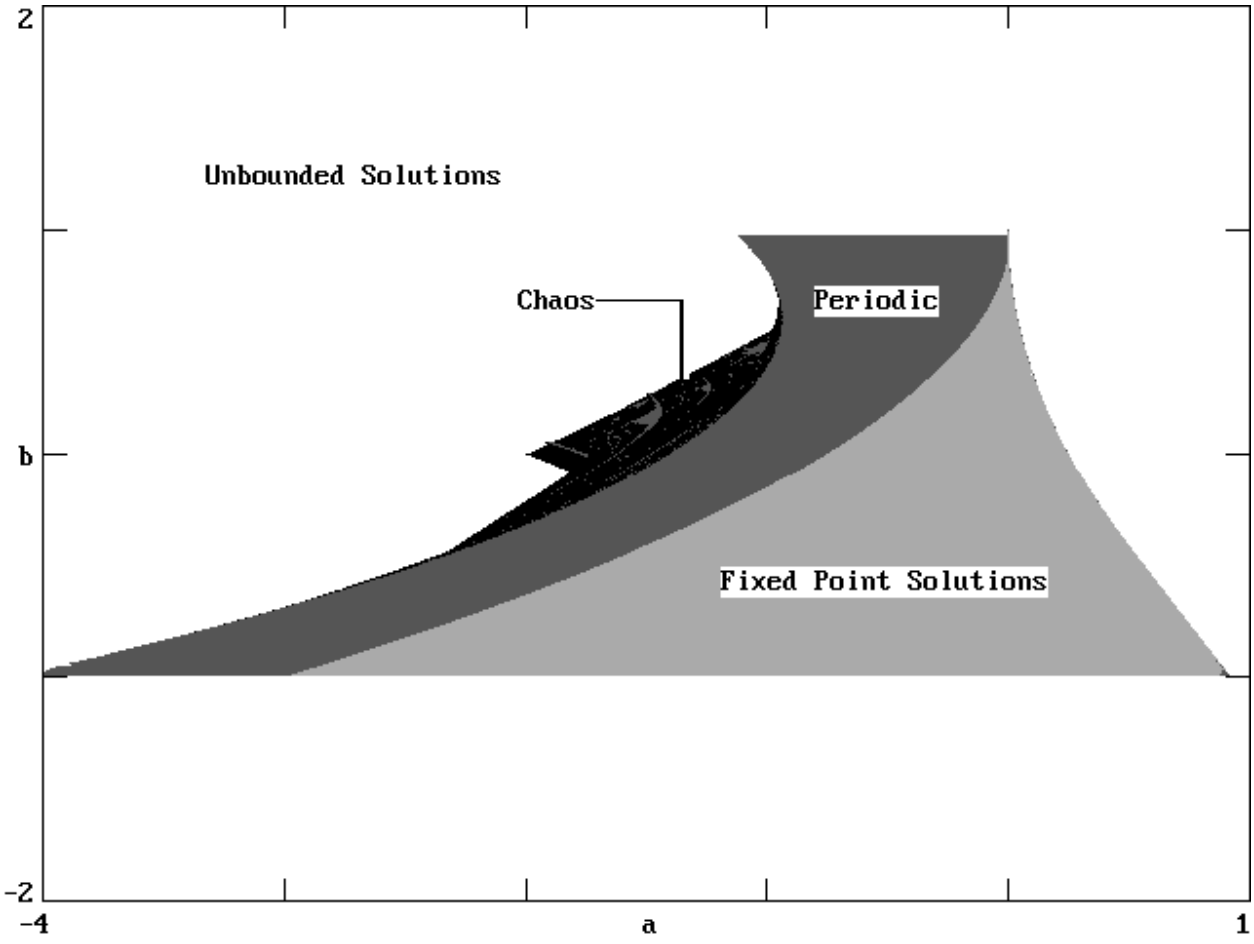
As a simple but very unrealistic example, suppose that all of nature could be modeled by the logistic equation (Equation 1C). This equation has a single parameter *R* that controls the character of the solution. If *R* is greater than 4 or less than -2, the solutions are unbounded, which means that this equation cannot model a physical process under those conditions. Essentially all physical processes are bounded, except perhaps the trajectory of a spacecraft launched with sufficient velocity to escape the galaxy. In the physically realistic range of *R*, there is a band of chaos between about 3.5 and 4, as shown in Figure 1-2 and another identical band between about -1.5 and -2. Careful analysis shows that chaos occurs over about 13% of the range of *R* from -2 to 4.

Since the logistic equation is too simple a model for almost everything, we should examine more complicated models. The Hénon map (Equation 3A) is a two-

dimensional generalization of the logistic map. It has two control parameters, which normally are a = -1.4 and b = 0.3. With b = 0, the Hénon map reduces to the logistic map.

As with the logistic map, the Hénon map has unbounded solutions, chaotic solutions, and bounded nonchaotic solutions, depending on the values of a and b. The bounded nonchaotic solutions may be either fixed points or periodic limit cycles. Figure 8-1 shows a region of the ab plane with the four classes of solutions indicated by different shades of gray. The bounded solutions constitute an island in the ab plane. On the northwest shore of this island is a chaotic beach, which occupies about 6% of the area of the island. The chaotic beach has many small embedded periodic ponds. The boundary between the chaotic and the periodic regions is itself a fractal.

Figure 8-1. Regions of solutions for the Hénon map in the ab plane



The logistic map is chaotic over 13% of its bounded range, and the Hénon map is chaotic over 6% of its bounded range. This result is counterintuitive because it suggests that more complicated (two-dimensional) systems are in some sense less chaotic than simpler (one-dimensional) systems. Is this a general result, or is it peculiar to these two maps? One way to decide is to examine a wider selection of equations, such as the ones used to produce the attractors exhibited throughout this book.

In collecting attractors, we have been discarding interesting information—the number of bounded nonchaotic solutions for each chaotic case that the program finds. Discarding data is offensive to scientists, since experiments are often performed with great effort and at considerable expense. The annals of science are ripe with examples of important discoveries that could have been made sooner or by others if only the right data had been recorded and analyzed.

Table 8-1 shows the results from 30,000 chaotic cases (1000 for each of the 30 types) as identified by the program. This table includes over 400 million cases, of which about 1 million are bounded. Of all the bounded solutions, 2.8% were chaotic according to the criterion described in Section 2.4. The polynomial maps all exhibit a similar occurrence of chaotic solutions. The same is true of ordinary differential equations (ODEs), but the percentage is smaller. The reason for this behavior is not understood.

Table 8-1. Summary of data from 30,000 chaotic cases

Code	D	O	Type	Chaotic	Average F	Average L
A	1	2	Map	3.34%	0.81±0.15	0.53±0.42
B	1	3	Map	5.09%	0.80±0.14	0.50±0.40
C	1	4	Map	8.09%	0.82±0.12	0.52±0.20
D	1	5	Map	7.94%	0.80±0.14	0.51±0.21
E	2	2	Map	7.58%	1.20±0.32	0.27±0.16
F	2	3	Map	7.08%	1.19±0.33	0.27±0.15
G	2	4	Map	6.40%	1.16±0.32	0.27±0.15
H	2	5	Map	5.79%	1.19±0.30	0.28±0.16
I	3	2	Map	6.68%	1.50±0.40	0.16±0.10
J	3	3	Map	5.89%	1.45±0.39	0.15±0.09
K	3	4	Map	5.08%	1.45±0.41	0.15±0.09

Code	D	O	Type	Chaotic	Average F	Average L
L	3	5	Map	4.68%	1.43±0.39	0.14±0.09
M	4	2	Map	4.99%	1.64±0.47	0.10±0.06
N	4	3	Map	4.78%	1.59±0.46	0.09±0.06
O	4	4	Map	5.32%	1.61±0.45	0.09±0.06
P	4	5	Map	5.04%	1.62±0.47	0.10±0.06
Q	3	2	ODE	0.55%	1.28±0.41	0.21±0.33
R	3	3	ODE	1.33%	1.31±0.40	0.73±0.76
S	3	4	ODE	1.23%	1.35±0.40	1.05±0.98
T	3	5	ODE	1.63%	1.38±0.41	1.23±1.16
U	4	2	ODE	1.34%	1.43±0.43	0.16±0.23
V	4	3	ODE	1.84%	1.43±0.43	0.40±0.48
W	4	4	ODE	1.84%	1.46±0.45	0.54±0.62
X	4	5	ODE	1.96%	1.44±0.44	0.66±0.76
Y	4		Special	16.28%	1.37±0.56	0.26±0.26
Z	4		Special	23.19%	1.03±0.44	0.28±0.44
[	4		Special	16.00%	0.63±0.65	0.42±0.23
\	4		Special	1.61%	1.10±0.28	0.16±0.10
]	4		Special	19.80%	1.02±0.16	0.06±0.04
^	4		Special	1.91%	1.80±0.49	0.39±1.03

These results should not be taken too literally because the coefficients have been limited to the range -1.2 to 1.2, the ODEs have not been solved very accurately, and many cases are ambiguous. Chaotic solutions tend to occur at large values of the coefficients where most of the solutions are unbounded. A more careful evaluation, which corrects these difficulties and includes about 35,000 strange attractors but limited to fewer types, shows that the probability that a bounded solution is chaotic for an iterated polynomial map of dimension *D* and order *O* is given approximately by

$$P = 0.349\ D^{-1.69}\ O^{-0.28} \qquad \text{(Equation 8A)}$$

Similarly, the probability that a bounded solution is chaotic for a polynomial ODE of

dimension *D* and order *O* is given approximately by

$$P = 0.0003\ D^2\ O^{0.5} \qquad \text{(Equation 8B)}$$

Maps appear to become less chaotic as they become more complicated (larger *D* and *O*), whereas ODEs become more chaotic.

To assess how common chaos is in nature, we must address the more complicated and subjective issue of whether the equations we have examined are a representative sample of the equations that describe natural processes. Furthermore, we cannot assume *a priori* that nature selects the coefficients of the equations uniformly over the bounded region of control space. It is possible that other constraints mitigate either against or in favor of chaotic behavior.

Another interesting question is how the fractal dimension and the Lyapunov exponent vary with the dimension and order of the system. Table 8-1 includes the average values of these quantities plus or minus (±) the standard deviation for each type of chaotic system. For polynomial maps and ODEs, the fractal dimension varies approximately as the square root of the system dimension. For polynomial maps, the Lyapunov exponent varies inversely with the system dimension. For polynomial ODEs, the Lyapunov exponent increases with the system dimension. The Lyapunov exponent appears to be independent of order for maps, but there is a tendency for the Lyapunov exponent of ODEs to increase with order.

These results are summarized in Figures 8-2 and 8-3. Figure 8-2 shows the relative probability that a strange attractor from a polynomial map or ODE will have a fractal dimension *F* plotted versus $F/D^{0.5}$. The curve is sharply peaked at a value of about 0.8. Almost no attractors have a fractal dimension greater than about $1.3D^{0.5}$. The Lorenz and Rössler attractors (with fractal dimensions slightly above 2.0 in a three-dimensional space) are close to this maximum value. Figure 8-3 shows the relative probability that a strange attractor from a polynomial map will have a Lyapunov exponent *L* plotted versus *LD*. This curve shows a much broader peak at about 0.5. These results hold when the calculations are done more carefully. The reason for this behavior is not understood, but it is potentially important because it gives an indication of the complexity of the system of equations responsible for a strange attractor that one observes in nature.

Figure 8-2. Probability distribution of fractal dimension

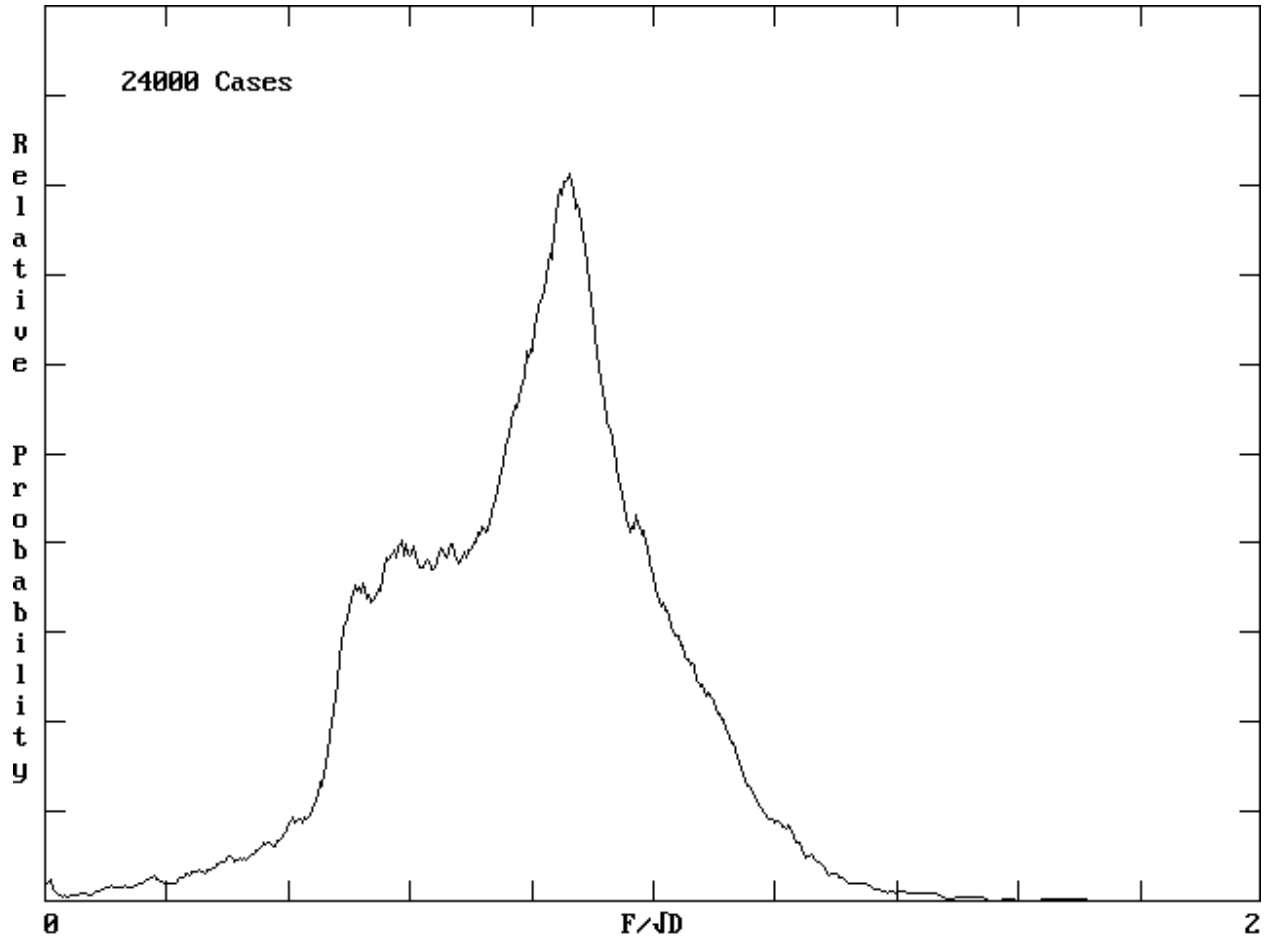
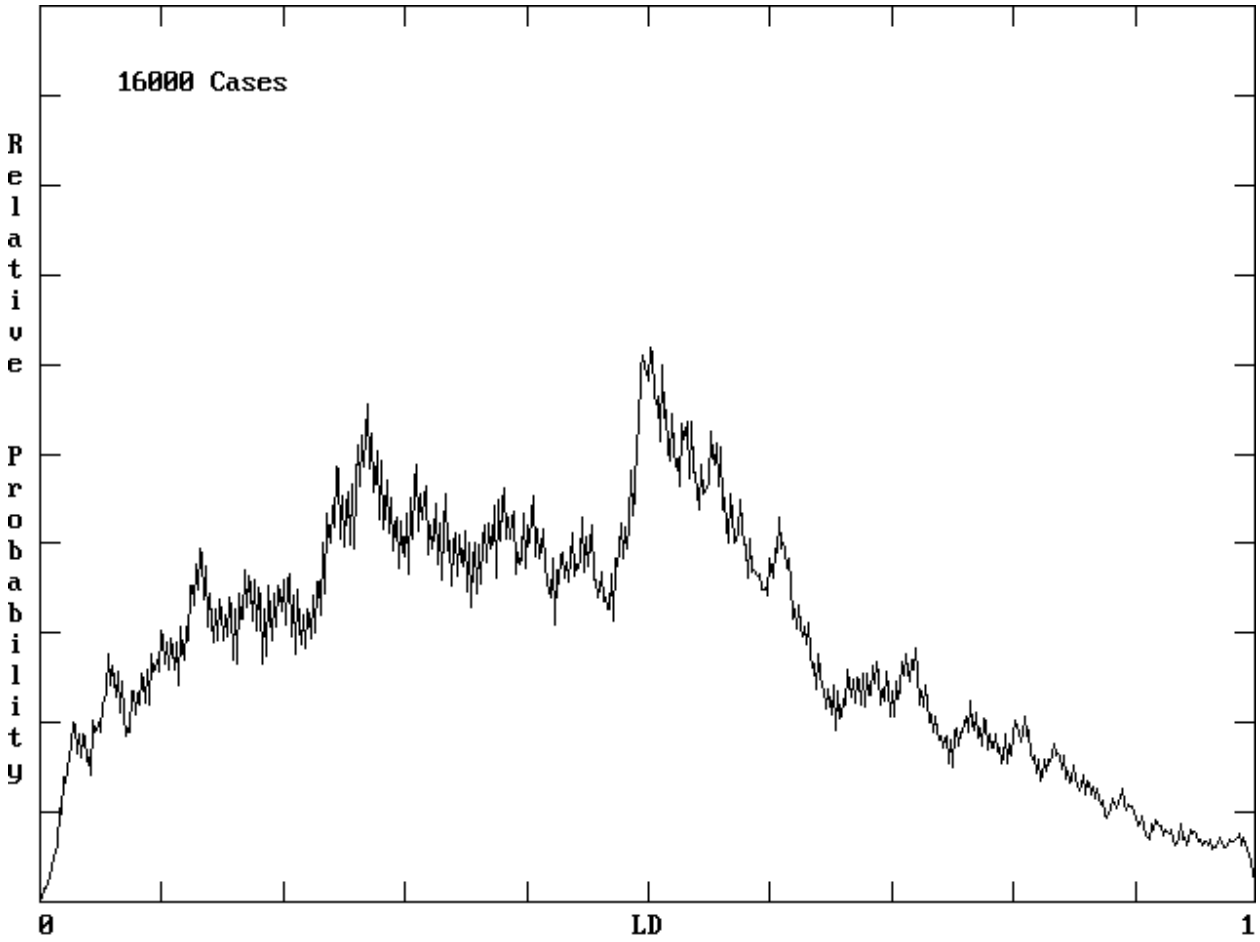


Figure 8-3. Probability distribution of Lyapunov exponent



The similarity of the fractal dimension for attractors produced by polynomials of the same dimensionality raises the concern that they might in some sense all be the same attractor, viewed from different angles and distorted in various ways. There may be a simple mapping that converts one attractor into the others. In such a case, a statistical analysis of the collection would be misleading and meaningless. However, since the Lyapunov exponents are spread over a broad range, it seems likely that the attractors are distinct. In any case, they are visually very different, and thus the technique has artistic if not scientific value.

It is interesting to ask whether the above results are peculiar to polynomials. Table 8-1 includes data for the special functions that were described in Chapter 7. Some of these cases tend to be more chaotic than the polynomials, but the differences are not enormous. Thus is would appear, insofar as nature can be represented by systems of equations of the type described in this book, that chaos

is not the most common behavior, but neither is it particularly rare.

8.2 But Is It Art?

A very different question is whether pictures generated by solving deterministic equations with a computer can legitimately qualify as art. Some people would say that if it was done by a computer without human intervention, it cannot be art. On the other hand, humans chose the equations, built and programmed the computer, decided how the solution would be displayed, and selected from the large number of cases that the computer generated. In this view, the computer is just another tool in the hands of the artist.

At the core of the issue is what we mean by *art*. There are at least two, not necessarily mutually incompatible, views. One is that art is the expression of ideas and emotions—a form of communication between the artist and the observer. The other emphasizes formal design, in which the viewer admires the skill with which the artist manipulated the materials, without reading any particular meaning into it.

Strange attractors qualify by either definition. They are expressions of ideas embodied in the equations, whether it be the dynamics of population growth or a swinging pendulum. These ideas are often abstract and are most apparent to the trained mathematician or scientist, but anyone can see in the patterns the surreal images of plants, animals, clouds, and swirling fluids. The appearance of such familiar images in the solutions of mathematical equations is probably more than coincidental.

Strange attractors also necessarily embody concepts of design. The interplay of determinism and unpredictability ensures that they are neither formless nor excessively repetitious. Furthermore, the skills of an artisan (if not an artist) are required to translate the abstract equations into aesthetically desirable visual forms. These skills are different from (but not inferior to) those possessed by more conventional artists. Renaissance artists, such as Leonardo da Vinci, were often also scientists. We may now be entering a new Renaissance in which art and science are again being drawn together through the visual images produced by computers.

Some artists view art as a creative process whose primary goal is to provide the artist with a sense of satisfaction. The resulting work is merely an inevitable by-product. This view seems especially appropriate for the production of strange attractors, where the programmer's satisfaction is derived from causing the computer to generate the patterns, even if they are never seen by anyone else. Indeed, the computer offers the ideal medium for such conceptual artists, since there need be no material product whatsoever.

Note that visual art need not be beautiful to be good, just as a play need not be humorous. It may be intellectually or emotionally satisfying, or even disturbing. It should capture and hold the interest of the viewer, however. The span of the viewer's attention is one measure of its quality. Art may mix the familiar and the unfamiliar to produce both comfort and dissonance. Furthermore, beauty is at least partially in the eye of the beholder, although recent research indicates that there are absolute universal measures of beauty that form very early in life and may even be genetic.

8.3 Can Computers Critique Art?

The idea that a computer can make aesthetic judgments seems absurd and even offensive to many people. Yet the program developed in this book is already doing this to some degree. For every object (strange attractor) that it identifies, it has discarded many dozens as being uninteresting (nonchaotic). Perhaps the computer could be programmed to be even more discriminating and to select those strange attractors that are likely to appeal to humans. To the extent that aesthetic judgment involves objective as well as subjective criteria, such a proposition is not unreasonable.

A computer lacks emotion, but it can be taught in much the same way that people can be taught. With the help of a human to point out which attractors are visually interesting, the computer can correlate human opinion with various quantitative measures of the attractor. It can then test each new case and assign a probability that it would appeal to a human.

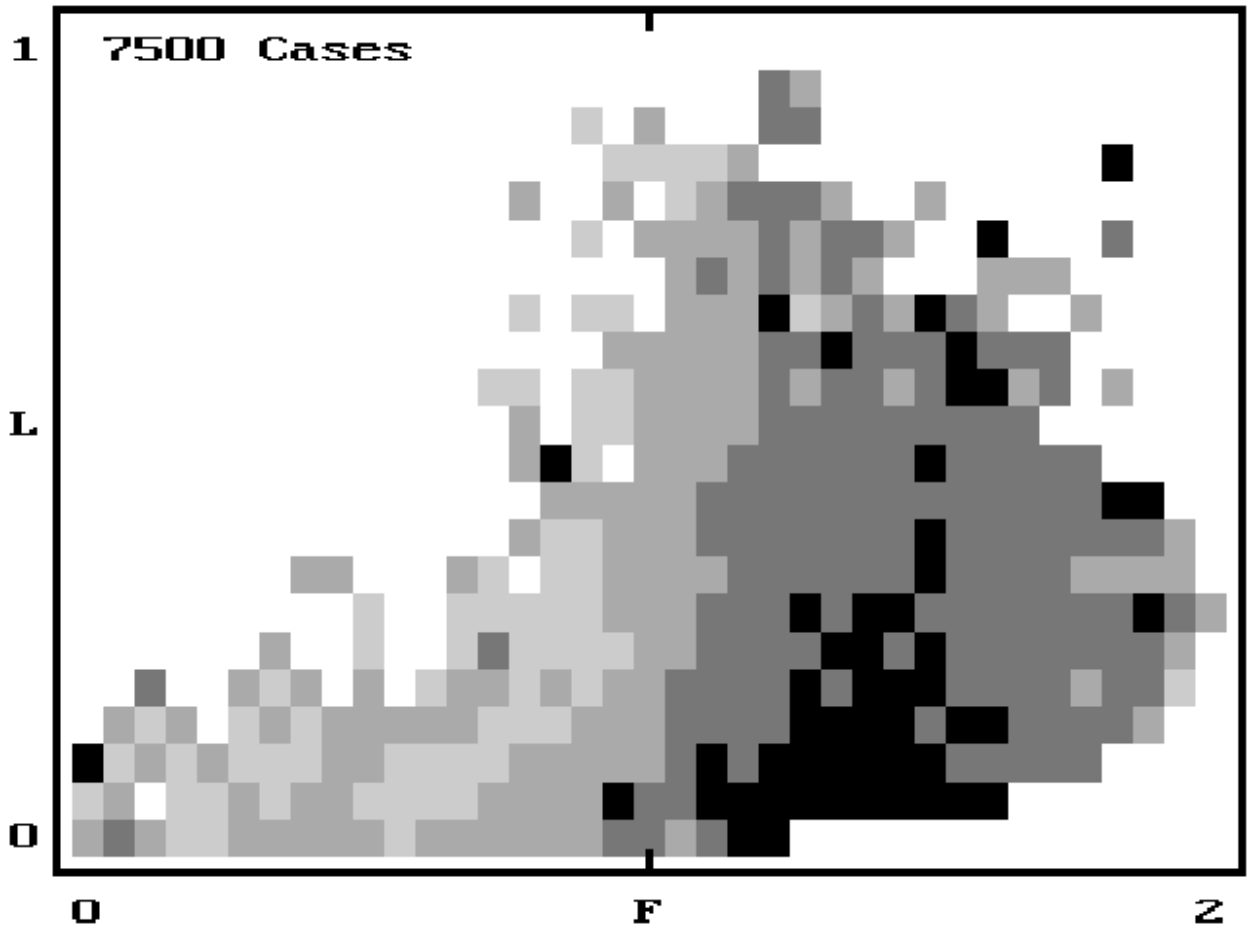
One of the reasons we have been calculating and saving the fractal dimension and Lyapunov exponent for each strange attractor is in anticipation of developing such criteria. You can think of the dimension as a measure of the *strangeness* of an attractor and the Lyapunov exponent as a measure of its chaoticity. These are just two of infinitely many independent quantities we can use to describe each attractor. If we find encouragement from them, it suggests that more can be done.

The first step is to search for a relation between the aesthetic quality and the fractal dimension or Lyapunov exponent. For this purpose, 7500 strange attractors from two-dimensional quadratic maps were evaluated by the author and seven volunteers, including two graduate art students, a former art history major, three physics graduate students, and a former mathematics major. All evaluators were born and raised in the United States. The evaluations were done by choosing attractors randomly and displaying them sequentially on the computer screen without any indication of the quantities that characterize them. The volunteers were asked to evaluate each case on a scale of one to five according to its

aesthetic appeal. It only took a few seconds for each evaluation.

Figure 8-4 displays a summary of all the evaluations as a function of fractal dimension (*F*) and Lyapunov exponent (*L*), using a gray scale in which the darker regions are the most highly rated. The cases examined by particular individuals show a similar trend. All evaluators tended to prefer attractors with dimensions between about 1.1 and 1.5 and Lyapunov exponents between zero and about 0.3. Some of the most interesting cases have Lyapunov exponents below about 0.1. You saw many such examples earlier in this book.

Figure 8-4. Regions of highest aesthetic quality in the FL plane



The dimension preference is not surprising, since many natural objects have dimensions in this range. Nature is strange, but usually not totally bizarre. Some of the attractors that are most universally liked resemble well-known and easily recognizable objects. You can find many such examples in this book.

The Lyapunov exponent preference is harder to understand, but it suggests that strongly chaotic systems are too unpredictable to be appealing. For the 443 cases that were rated 5 (best) by the evaluators, the average dimension was $F = 1.30 \pm 0.20$, and the average Lyapunov exponent was $L = 0.21 \pm 0.13$ bits per iterations, where the errors represent plus or minus one standard deviation. About 28% of the cases evaluated fall within both error bars. Thus this simple criterion would allow the computer to discard nearly three-quarters of the cases that are least likely to be visually appealing. This technique works in practice and was used in some cases to help select attractors to display in this book.

8.4 What's Left to Do?

This book has described a new technique for generating strange attractors in unlimited numbers. A large collection of such objects offers many interesting possibilities to the artist and scientist alike. Some of these uses have already been mentioned. Others may have occurred to you as you read this book. This section leaves you with a few additional suggestions of things you might want to explore on your own.

If your main interest is art, you probably want to produce attractors with improved spatial resolution and more colors. You can experiment with printing on different types of paper or other media. The simple linear correspondence of X, Y, Z, and W to position or color is not essential. Other mappings between the mathematical variables and the points displayed on the screen are possible. You have already seen how to project the attractors onto a sphere. You can project them onto other objects such as cylinders, tori, or even other strange attractors.

When you have generated an attractor that appeals to you, it is natural to want to make small changes to make it even better. The coefficients in the equations are controls that you can adjust. Like knobs on your television set, they allow you to tune the attractor to get just what you want. You can change the colors without replotting the data using the PALETTE command in BASIC. You also can rotate the image to find the best angle from which to view it.

You can produce animated strange attractors with a video camera viewing a monitor connected to the camera, or even more simply, with a photodiode connected to an oscilloscope whose screen illuminates the photodiode. This *video-*

feedback technique has much in common with iterated maps. Each illuminated dot on the screen is mapped back to a different location after a delay determined by the propagation of the electrical and optical signals. The main control parameters are the distance, rotation, focusing, intensity, color, and hue. Some settings produce unbounded solutions—the screen goes solid black or solid white. Other settings produce a fixed-point solution with a stationary pattern. Under other conditions, periodic behavior occurs. The most interesting situation occurs when the pattern constantly changes but is not periodic, corresponding to a chaotic strange attractor. Sometimes you can perturb the system with a flash of light or by moving your hand across the field of view, causing the system to switch from one attractor to another.

Some of the most interesting examples of computer fractals come from plotting the basin boundaries of various attractors. The basin is the set of all initial conditions that are drawn to the attractor. Sometimes these boundaries are smooth; other times they are fractals. By coloring the points just outside the basin according to the number of iterations required for them to leave some (usually large) region surrounding the attractor, beautiful escape-time fractal patterns can be produced.

All of our attractors have such basins, and it's not hard to program the computer to display them, but the calculations are very slow. As an example, Figure 8-5 shows in black the basin for the Hénon map. It is relatively smooth and not particularly interesting. It resembles a thick version of the attractor, but rotated by 90 degrees. Figure 8-6 shows the basin of another two-dimensional quadratic map called the *Tinkerbell map*. Its boundary has obvious fractal structure. In each case, the basin boundary appears to touch the attractor, suggesting that this surprising feature may be common.

Figure 8-5. Basin of attraction for the Hénon map

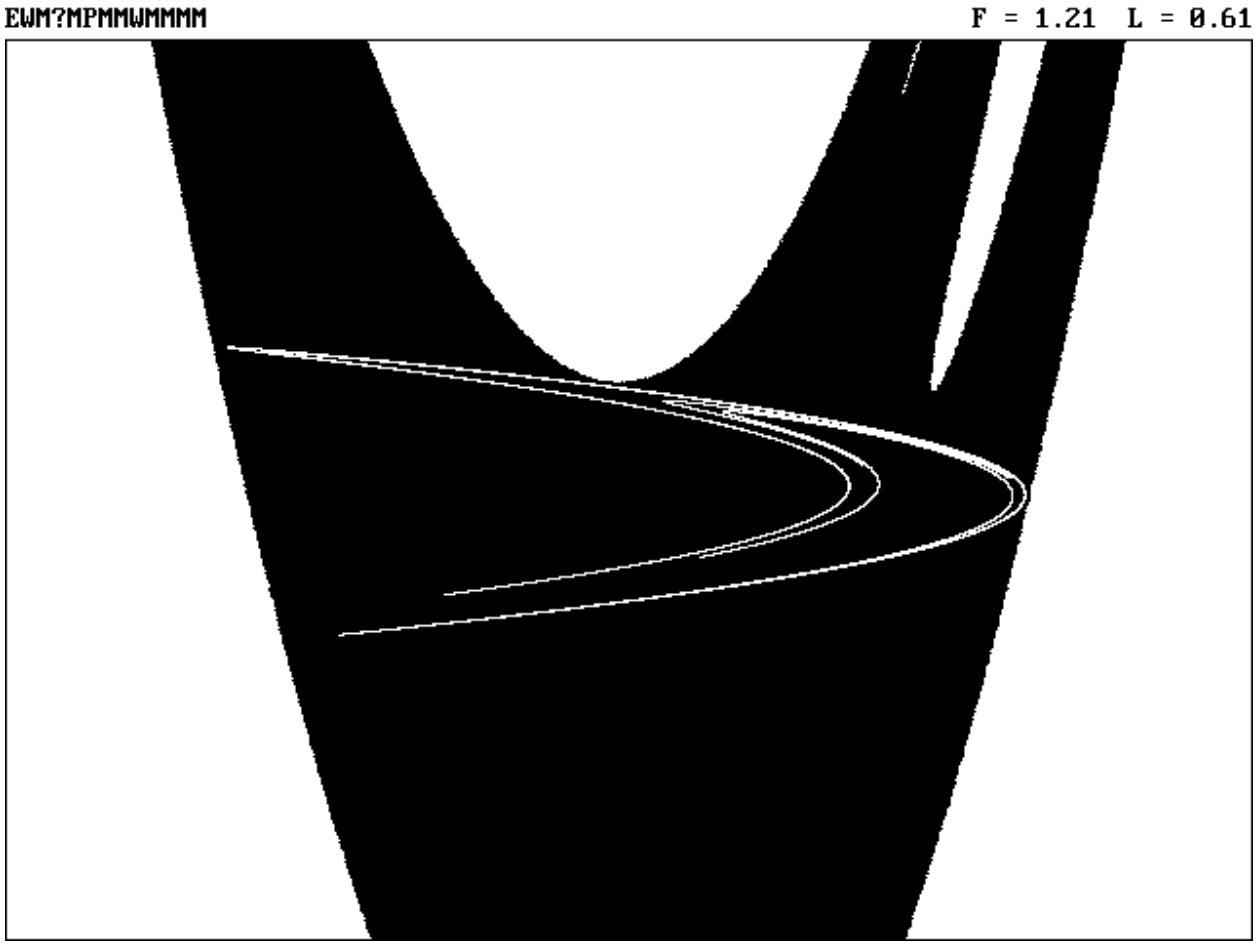
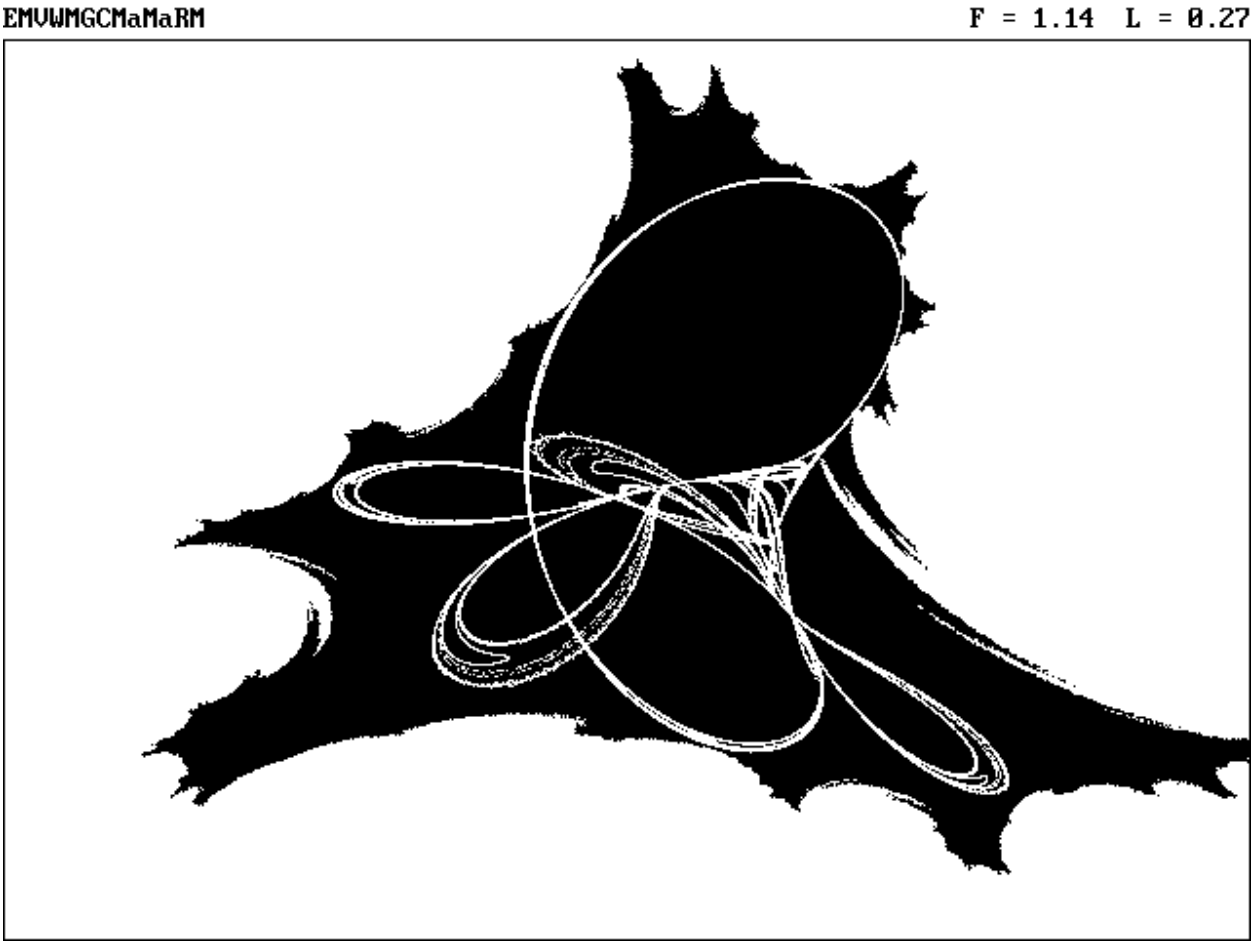


Figure 8-6. Basin of attraction for the Tinkerbell map



Notice that we have now plotted the Hénon map in three different spaces. Figure 3-1 is the usual plot of the orbit in the space of the dynamical variables *X* and *Y*. Figure 8-1 is a plot in the space of the control parameters *a* and *b*. Figure 8-5 is a plot in the space of the initial conditions X_0 and Y_0. All of these plots are necessary to characterize the attractor completely.

The well-known and much-studied Mandelbrot set is the set of bounded solutions in the ab plane of the mapping

$$X_{n+1} = X_n^2 - Y_n^2 + a$$
$$Y_{n+1} = 2X_nY_n + b \qquad \text{(Equation 8C)}$$

The variables *X* and *Y* usually are thought of as the real and imaginary parts of a complex number *Z*. Each point in the *ab* plane has associated with it a *Julia set*, which is the set of all initial conditions X_0 and Y_0 whose solutions are bounded. The Mandelbrot set is the set of Julia sets every point of which is connected to every other point. The Julia set is named after Gaston Julia, a French mathematician who, along with Pierre Fatou, studied iterated maps in the complex plane at about the time of the first world war. Combinations of *a* and *b* at the boundary of the Mandelbrot set produce chaotic solutions, but values inside the set lead to fixed points or limit cycles. For $b = 0$ and $Y_0 = 0$ (the real axis), Equation 8C reduces to a simple one-dimensional quadratic map equivalent to the logistic map.

The Mandelbrot set has been described as the most complicated mathematical object ever seen. It may also be the ultimate computer virus, in that it takes over not only the machine (because of the large computational requirements) but also the mind of the programmer, who often becomes addicted to the beauty and variety of the patterns that it produces.

Like the Mandelbrot set, most of the attractors produced by the programs in this book have intricate structure near the basin boundaries. You can zoom in on these regions to produce patterns that rival those produced by the Mandelbrot set. Calculation times increase markedly as you zoom in ever more closely, however.

Much more can be done to correlate the aesthetic appeal of the attractors with the various numerical quantities that characterize them. Besides the fractal dimension and Lyapunov exponent, the system that produced them has dimension and order. Other measures of an attractor's dimension include the *capacity dimension*, the *information dimension*, and the *Lyapunov dimension*. Related to the Lyapunov exponent is the *entropy*, which is a measure of the disorder of the system. As with the dimension, the entropy can be defined in many ways. These quantities and others are described in many of the books on fractals in the bibliography. Maps and differential equations can also be compared.

You can test for differences between the aesthetic preferences of artists and scientists. Preliminary indications suggest that complexity might appeal more to artists than to scientists, who tend to see beauty in simplicity. There may also be discernible cultural differences. You can see how the various display techniques alter one's preferences. For example, the use of color seems to increase the tolerance for attractors of high fractal dimension.

Many scientific studies can be performed on a sufficiently large collection of strange attractors. The results of Table 8-1, which are merely suggestive and limited to particular types of equations, can be refined with more cases and more examples of each case. For low dimensions, maps are more chaotic than the equivalent ODEs. As the system dimension increases, however, maps become less chaotic, while ODEs become more chaotic. Equation 8A and 8B suggest that for a

system dimension of about six, maps and ODEs will be equally chaotic. Will their roles reverse at higher dimension, or will the chaotic fractions asymptotically approach a universal value of about 2%?

You can ask the same type of questions about bounded, nonchaotic solutions (fixed points, limit cycles, and tori). It is especially interesting to see how common 3-tori are in light of Peixoto's theorem (see Section 6.6), but their production probably requires system dimensions of about 10 according to Figure 8-2, which may or may not apply to tori. Almost nothing is known about such issues.

We have considered the *fraction* of the bounded, hyperdimensional control space over which chaos occurs. It is also interesting to examine the shape, location, and dimension of this chaotic region, as we did for the two-parameter Hénon map in Figure 8-1. It is likely that this region is a fractal. How does its fractal dimension depend on the system dimension and other characteristics of the system of equations? It appears that the dimension of the chaotic region is about half the dimension of the control space for the cases in this book.

The scaling of the average fractal dimension and the Lyapunov exponent with the system dimension is intriguing and should be studied with more cases, better statistics, and more accurate calculations of the fractal dimension. You can examine statistically how the various measures of dimension compare and how these dimensions are related to the spectrum of Lyapunov exponents.

You can look for relations between the geometrical properties of attractors and their respective power spectra. The technique can be used to explore and to quantify the various routes by which a stable solution becomes chaotic. Many such routes have been identified, such as the period-doubling exhibited by the logistic equation, but there are probably others yet to be discovered. Your role resembles that of a biologist confronted with a large variety of species, trying to classify, quantify, and study their similarities, differences, and patterns of behavior.

Another interesting study is to search for previously unknown simple examples of chaos. Variations of the logistic equation are the simplest chaotic polynomial maps. The Lorenz and Rössler attractors are often cited as the simplest examples of chaotic polynomial ODEs. The Lorenz equations (Equation 6D) have seven terms and two quadratic nonlinearities, and the Rössler equations (Equation 6E) have seven terms and one quadratic nonlinearity.

There are at least five different systems of chaotic, three-dimensional quadratic ODEs with five terms and two nonlinearities, and six systems with six terms and one nonlinearity. It will be left as a challenge for you to find them. There does not seem to be any chaotic system of quadratic ODEs with as few as four terms. To whet your appetite, here's a simple chaotic system resembling the Lorenz attractor that has two fewer terms and all its coefficients equal to one:

$$X' = YZ$$

$$Y' = X - Y$$

$$Z' = 1 - XY \qquad \text{(Equation 8D)}$$

You can examine this case using the code $QM^7NM^3NM^3LM^4NM^2LM^6$.

Here's another case with five terms and all unity coefficients that's volume-preserving and thus does not have an attractor, but its solution is either a 2-torus or chaotic depending upon the initial conditions:

$$X' = Y$$

$$Y' = X + YZ$$

$$Z'' = 1 - Y^2 \qquad \text{(Equation 8E)}$$

It can be produced with the code $QM^5NM^5LM^5NM^2NM^5LM^3$.

8.5 What Good Is It?

It is rare that a mathematical concept captivates the interest not only of scientists in diverse fields but of the general public. Chaos has done this, and it has been heralded by some as the next great revolution in science. It has ushered in a new field of *experimental mathematics*, sometimes pejoratively called *recreational mathematics* by the more traditional and often cynical older breed of mathematicians. The use of computers to produce exotic visual patterns has made difficult mathematical ideas accessible to those without extensive formal training.

Yet it is fair to ask what it has done other than to make pretty pictures. One response is to claim that the same things were probably said about the discovery of the atomic nucleus, or electricity, or fire. We must have faith that intellectual advances will eventually yield useful applications.

At the deepest level, an understanding of chaos alters our view of the world. Having seen the complexity that can arise from simple equations, we have reason to hope that simple equations may suffice to describe much of the complexity of the world. Difficult and long-standing problems such as fluid turbulence may eventually be understood using the ideas of chaos. Turbulence is difficult because it involves both temporal and spatial chaos and because its dimension is high.

Chaos may also provide tools for making better predictions of complicated and apparently random systems, such as the weather, the stock market, earthquakes, epidemics, and population growth. It may also have applications in communications and in cryptography for devising secure codes and breaking them.

Since chaotic systems exhibit extreme sensitivity to initial conditions, you might conclude that prediction is hopeless, but this is not the case. Prediction is hopeless for a random system or for an extremely complicated deterministic system. As the physicist Neils Bohr remarked, "Prediction is difficult, especially of the future." However, if the system is simple and chaotic, then the determinism can be exploited to improve short-term predictions.

Furthermore, if the equations produce a strange attractor, as most chaotic systems do, we know that the solution lies somewhere on the attractor. Remember that an attractor occupies a negligible volume of the space in which it is embedded. Although we can't predict where on the attractor the system will be at any particular time in the distant future, we can exclude the vast number of possible states that lie off the attractor. In meteorological terms, this might lead to eliminating the possibility of certain weather conditions with near absolute certainty.

When we see a system in nature that behaves erratically, we are now led to wonder whether it is an example of chaos. Is simple determinism hidden in seemingly random data? This problem is opposite to the one addressed in this book. Here we have started with simple equations and produced complicated patterns. Nature presents us with complicated patterns that may or may not come from simple equations.

Often we are given only a single fluctuating quantity sampled at discrete times—a so-called *time series*. The average daily temperature in New York and the daily closing Dow Jones Industrial Average are two examples of a time series. Sometimes a complicated-looking time series is simply a sum of several sine waves of different unrelated frequencies. In such a case, accurate predictions are possible using linear methods. A good example is the tides, whose behavior is not simply periodic but nevertheless can be predicted with considerable accuracy because it is governed by periodic notions of the earth and moon. More often, the time series has no such simple representation. In such a case, we would like to determine whether the behavior is random or chaotic. Chaotic systems often produce strange attractors.

We usually don't have data for all the variables that describe the system, nor do we even know how many there are, so we don't know the dimension of the space in which an attractor might be embedded. Furthermore, the data record is likely to be corrupted by random noise and measurement errors. The number of data points may be small, and the system may not have reached a steady state. Finally, we usually don't have control over the system, so we cannot directly test its

sensitivity to initial conditions.

Such a situation sounds hopeless, but progress has recently been made in testing for chaos in natural systems. For example, one can plot each data point versus its predecessor, as we did with the one-dimensional maps in Chapter 2. In some cases, this procedure is enough to reveal the determinism. More often, it is necessary to plot each data point versus several of its predecessors in a high-dimensional space. If the system is described by a strange attractor, the result is a version of the attractor called a *diffeomorphism* that is distorted but has the same fractal dimension. Similarly, the Lyapunov exponent can be estimated by selecting nearby points in this space and determining how rapidly they diverge from one another.

The fractal dimension is important because the number of variables and equations is at least as large as the next higher integer, since the attractor has to be embedded in a space with dimension higher than its fractal dimension. These equations are not unique, however. Extracting equations from the data is a difficult if not impossible task, but one whose rewards justify the effort. The equations provide insight into the underlying dynamics and a means for making predictions.

Perhaps the ultimate test for chaos is the accuracy of short-term predictions. With truly random data, prediction is impossible. Conversely, with chaotic data, there is absolute determinism, and prediction is possible in principle, at least on short-time scales dictated by the Lyapunov exponent and the precision of the data. Predictability thus precludes complete randomness and signifies determinism, although randomness and determinism often coexist. For example, the business cycle is mostly random but includes deterministic seasonal changes. The deterministic part is not necessarily chaotic, however.

Dynamical systems are everywhere. Your body contains several. Your heart is a dynamical system whose solution normally approaches a limit cycle, but which can become chaotic when in fibrillation or a fixed point upon death. Recent research suggests that even a healthy heart beats chaotically, and a nearly periodic pattern sometimes precedes cardiac arrest.

Other oscillations occur in your lungs, brain, and muscles. Electrocardiograms and electroencephalograms are time-series records used by doctors to deduce information about the dynamics of the corresponding organ. Some chaos in the brain may be necessary for creativity because it prevents us from repeatedly approaching the same problems in the same way. The fractal dimension of electroencephalograms has been observed to increase when a subject is engaged in mental activity. In psychology, manic depression resembles a limit cycle, and certain types of erratic behavior might be strange attractors.

Evidence for low-dimensional strange attractors has been found in systems

as diverse as the weather and climate, the business cycle, childhood epidemics, sunspots, plasma fluctuations, and even in computer models of the arms race. Some of these results should be viewed skeptically because there are many ways to be misled. The number of data points needed for an accurate determination of the dimension has been variously estimated from a pessimistic and probably overly conservative 42^F to a more optimistic but still large $10^{2+0.4F}$. If the interval between data points is too small, a fictitiously low dimension can result. Certain types of *colored noise* (also called *fractional Brownian motion*) have a degree of determinism and produce time-series records with an apparently low dimension. Many tests have been devised, such as comparing the results with those from a surrogate data set obtained by randomly shuffling the numbers in the time series.

Chaos has a deep philosophical significance. If determinism implied perfect predictability, there would be no room for free will. We would be just small cogs in a large machine over which we have no control. It's easy to feel insignificant in a vast and complicated universe dominated by forces too powerful to resist.

However, chaos illustrates that determinism does not imply predictability. What is bad news for those who want to predict the weather or the stock market is good news for those who want to control them. Recall the hypothetical butterfly flapping its wings in Brazil, which sets off tornadoes in Texas. If the world is really governed by deterministic chaos, we are drastically changing the future with everything we do. We need never feel unimportant or insignificant. Who would have thought that a concept from mathematics could give meaning and purpose to our lives?

Appendix A

Annotated Bibliography

Books and articles on chaos and fractals have proliferated enormously in recent years. A selection of the most useful and readable of these is listed here, along with comments on the content and level of each.

Abbott, E. A. *Flatland: A Romance of Many Dimensions* (New York: Barnes & Noble, 1983). An entertaining classic written in 1884 by a Victorian schoolmaster and minister, this book explains the fourth and higher dimensions by considering the impact of a visit by a three-dimensional creature (A Sphere) to a world inhabited by two-dimensional creatures (A Square and others).

Abraham, F. D. *A Visual Introduction to Dynamical Systems Theory for Psychology* (Santa Cruz, CA: Aerial Press, 1990). This book shows how attractors in dynamical systems have application to a field as unlikely as psychology. It contains numerous sketches of attractors and an explanation of the technical terms used to describe them.

Abraham, R. H. and C. D. Shaw. *Dynamics—The Geometry of Behavior* (Santa Cruz, CA: Aerial Press, 1982). A four-part series that attempts to explain the fundamentals of dynamical behavior without equations using cartoonlike drawings with extended captions.

Barnsley, M. F. *Fractals Everywhere* (San Diego, CA: Academic Press, 1988). This classic text by an expert mathematician describes the mathematics underlying fractals and provides a good source of new fractal types.

Barnsley, M. F., R. L. Devaney, B. B. Mandelbrot, H. O. Peitgen, D. Saupe, and R. F. Voss. *The Science of Fractal Images* (New York: Springer-Verlag, 1988). A readable

collection of articles describing the mathematics that underlies the computer generation of fractals.

Briggs, J. *Fractals: The Patterns of Chaos* (New York: Touchstone, 1992). A non-mathematical but detailed explanation of fractals and chaos with many high-quality drawings and photographs.

Briggs, J. and F. Peat. *The Turbulent Mirror: An Illustrated Guide to Chaos Theory and the Science of Wholeness* (New York: Harper and Row, 1989). This unusual book combines a simple mathematical description of chaos with loosely related philosophical and psychological ramblings.

Burger, D. *Sphereland* (New York: Barnes & Noble, 1983). An entertaining book written in the style of the one by Abbott, it picks up where *Flatland* stops in describing the properties of high-dimensional spaces.

Chernikov, A. A., R. Z. Sagdeev, and G. M. Zaslavsky. "Chaos: How Regular Can it Be?" *Physics Today*, page 27 (November 1988). Written by a group of active Russian nonlinear dynamicists, this review article provides a good background for understanding stochastic webs.

Crutchfield, J. P., J. D. Farmer, N. H. Packard, and R. S. Shaw. "Chaos." *Scientific American* Vol. 255, page 46 (December 1986). This easily readable article presents the fundamentals of chaos theory in a compact form.

Devaney, R. L. *An Introduction to Chaotic Dynamical Systems* (Reading, MA: Addison-Wesley, 1989). This advanced undergraduate text summarizes the mathematics that underlies chaotic systems of equations.

Devaney, R. L. *Chaos, Fractals, and Dynamics: Computer Experiments in Mathematics* (Reading, MA: Addison-Wesley, 1990). This book introduces chaos, fractals, and nonlinear dynamics with numerous beautiful images using computer exercises in BASIC and simple mathematics.

Devaney, R. L. *A First Course in Chaotic Dynamical Systems: Theory and Experiment* (Reading, MA: Addison Wesley, 1992). Written at the undergraduate college level, this book provides an excellent introduction to chaos and fractals.

Dewdney, A. K. "Probing the Strange Attractors of Chaos." *Scientific American*, Vol. 257, page 108 (July 1987). This typical Scientific American presentation provides a readable though abbreviated introduction to the mathematics underlying strange attractors.

Falconer, K. *Fractal Geometry: Mathematical Foundations and Applications* (New York: Wiley, 1990). In somewhat technical terms this book explains the mathematics underlying fractals and describes the various methods of defining and calculating their dimensions.

Feder, J. *Fractals* (New York: Plenum Press, 1988). This introductory technical book describes fractals using geometrical ideas and explains time-series analysis and other related topics from a physical viewpoint.

Feigenbaum, M. J. "Qualitative Universality for a Class of Nonlinear Transformations." *Journal of Statistical Physics* Vol. 19, page 25 (1978). Highly technical but historically important, this paper lays the foundation for many of the principles of chaos by reference to the logistic equation and similar one-dimensional nonlinear maps.

Field, M. and M. Golubitsky. *Symmetry in Chaos: A Search for Pattern in Mathematics, Art, and Nature* (Oxford: Oxford University Press, 1992). A spectacularly illus-

trated book that includes computer-generated images of flowers, graphic logos, motifs, and other artistic mathematical patterns.

Gleick, J. *Chaos: Making a New Science* (New York: Viking Penguin, 1987). This best-selling, historical, and nontechnical account is the starting point for anyone who wants to understand why chaos has excited the imagination of the scientist and nonscientist alike.

Grassberger, P. and I. Procaccia. "Characterization of Strange Attractors." *Physical Review Letters*, Vol. 50, page 346 (1983). A somewhat mathematical paper that was the first to propose the correlation dimension as a convenient method for quantifying strange attractors.

Gulick, D. E*ncounters with Chaos* (New York: McGraw Hill, 1992). A rigorous upper-undergraduate-level introduction to the mathematics of chaos.

Hao, B. L. *Chaos II* (Singapore: World Scientific, 1990). This book reprints many introductory and influential papers on chaos and includes a bibliography of 117 books and 2244 technical papers.

Hénon, M. "A Two-Dimensional Mapping with a Strange Attractor." *Communications in Mathematical Physics*, Vol. 50, page 69 (1976). Although somewhat technical, this article describes one of the first thoroughly studied examples of a chaotic, two-dimensional quadratic map.

Hofstadter, D. R. *Godel, Escher, Bach: An Eternal Golden Braid* (New York: Vintage, 1980). Dealing with, among other things, self-similarity and recursion in music, art, and science, this largely nonmathematical but deeply philosophical book is highly recommended.

Hofstadter, D. R. "Strange Attractors: Mathematical Patterns Delicately Poised Between Order and Chaos." *Scientific American*, Vol. 245, page 22 (November 1981). This article provides an introduction to the mathematics of strange attractors and their significance.

Hofstadter, D. R. *Metamagical Themas* (New York: Basic Books, 1985). A recreational mathematics book that contains many interesting topics, including a chapter on mathematical chaos and strange attractors.

Jackson, E.A. *Perspectives of Nonlinear Dynamics* (Cambridge: Cambridge University Press, 1991). A two-volume graduate-level text that covers all the essentials of chaos and nonlinear dynamics and includes many nice drawings.

Knuth, D. E. *Seminumerical Algorithms*, 2nd ed., Vol. 2 of *The Art of Computer Programming* (Reading, MA: Addison-Wesley, 1981), chap. 3. This serious mathematical book is the classic reference for describing, among other things, the methods by which computers are used to generate pseudorandom numbers and the pitfalls inherent in their misuse.

Krasner, S., ed. *The Ubiquity of Chaos* (Washington, D. C.: American Association for the Advancement of Science, 1990). A collection of 19 scientific papers that illustrates the wide range of fields in which chaotic processes have been discovered and studied.

Levy, S. *Artificial Life: The Quest for a New Creation* (New York: Pantheon Books, 1992). This book takes up where Gleick left off and covers much of the recent history of chaos and its developers.

Li, T. Y. and J. A. Yorke. "Period Three Implies Chaos." *American Mathematical Monthly*, Vol. 82, page 985 (1975). This historical paper contains the first published

reference to the term *chaos*.

Lorenz, E. N. "Deterministic Nonperiodic Flow." *Journal of the Atmospheric Sciences*, Vol. 20, page 130 (1963). Although slow to be appreciated, this historically important paper is now regarded as the first modern example of a strange attractor resulting from the solution of a simple set of ordinary differential equations originally proposed to model atmospheric convection.

Mandelbrot, B. B. *The Fractal Geometry of Nature* (San Francisco: W. H. Freeman, 1982). An extended essay by the father of fractals, this was the seminal work that brought to the attention of the nonspecialist the ubiquity of fractals in nature.

May, R. M. "Simple Mathematical Models with Very Complicated Dynamics." *Nature*, Vol. 261, page 459 (1976). A thought-provoking and mathematically straightforward paper that initiated the widespread interest in the logistic equation and other one-dimensional maps as models for natural processes.

McGuire, M. *An Eye for Fractals* (Reading, MA: Addison-Wesley, 1991). This graphic and photographic essay by a physicist and amateur photographer contains beautiful black and white photographs of natural fractals in the style of Ansel Adams, with a simple mathematical description of fractals.

Moon, F. C. *Chaotic Vibrations* (New York: Wiley-Interscience, 1987). An advanced undergraduate text that emphasizes the applications of chaos theory to real-world engineering problems.

Moon, F. C. *Chaotic and Fractal Dynamics: An Introduction for Applied Scientists and Engineers* (New York: Wiley, 1992). An update of Moon's previous book with many practical engineering applications of chaos theory.

Peitgen, H. O. and P. H. Richter. *The Beauty of Fractals: Images of Complex Dynamical Systems* (New York: Springer-Verlag, 1986). This beautiful, colorful exhibit of computer art emphasizes the Mandelbrot and Julia sets.

Peitgen, H. O., H. Jurgens, and D. Saupe. *Fractals for the Classroom* (New York: Springer-Verlag, 1992). A two-book set that explains the mathematical basis of fractals at a relatively elementary level and contains many BASIC program listings for the production of fractals.

Pickover, C. A. *Computers, Pattern, Chaos and Beauty: Graphics from an Unseen World* (New York: St. Martin's Press, 1990). A how-to book by the master of computer graphic art and visualization filled with original ideas for the computer generation of fractals and other artistic patterns.

Pickover, C. A. *Computers and the Imagination: Visual Adventures Beyond the Edge* (New York: St. Martin's Press, 1991). This extension of Pickover's earlier book by the same publisher takes up where the other stopped and provides additional spectacular examples of computer art and philosophical insight.

Pickover, C. A. *Mazes for the Mind: Computers and the Unexpected* (New York: St. Martin's Press, 1992). The third book in the series that includes puzzles, games, and mazes appealing to computer enthusiasts, artists, and puzzle-solvers, along with incisive commentary and additional dazzling computer images.

Porter, E. and J. Gleick. *Nature's Chaos* (New York: Viking Penguin, 1990). This book of art combines photographs of natural fractals by Porter with a simple, almost poetic, explanation of chaos and fractals by Gleick.

Pritchard, J. *The Chaos Cookbook: A Practical Programming Guide* (Oxford: Butterworth-Heinemann, 1992). A practical and elementary tutorial that includes

programs in BASIC and Pascal to ease the reader into the mathematics of chaos and fractals.

Rössler, O. E. "An Equation for Continuous Chaos." *Physics Letters*, Vol. 57A, page 397 (1976). This short, classic paper by a nonpracticing medical doctor includes stereoscopic views of the Lorenz and Rössler attractors.

Rucker, R. *The Fourth Dimension* (New York: Houghton-Mifflin, 1984). The book to read if you want to understand the philosophical, geometrical, and physical meaning of space-time.

Ruelle, D. "Strange Attractors." (*Mathematical Intelligencer*, Vol. 2, page 126 (1980). This mathematical article includes some of the history of the discovery and understanding of strange attractors and related chaotic phenomena.

Ruelle, D. *Chance and Chaos* (Princeton, NJ: Princeton Univ. Press, 1991). A charming little book by a pioneer in chaos, describing with minimal mathematics the philosophical implications of chaos and randomness.

Schroeder, M. *Fractals, Chaos, and Power Laws: Minutes from an Infinite Paradise* (New York: W. H. Freeman, 1991). A slightly mathematical but highly readable book packed with examples of temporal and spatial chaos in enormously diverse contexts and a wealth of puns.

Schuster, H. G. *Deterministic Chaos* (New York: Springer-Verlag, 1984). This work is aimed at the more mathematically inclined reader who wants to understand chaos and related topics in greater detail, and it includes a good mathematical description of the Lyapunov exponent.

Shaw, R. *The Dripping Faucet as a Model Chaotic System* (Santa Cruz, CA: Aerial Press, 1984). A short but detailed book that describes in nontechnical language how the ideas of chaos and strange attractors can be used to understand a phenomenon as simple as a dripping faucet.

Sparrow, C. T. *The Lorenz Equations, Bifurcations, Chaos, and Strange Attractors* (New York: Springer-Verlag, 1982). This somewhat technical book demonstrates the depth to which a single example of a strange attractor can be studied.

Sprott, J. C. "Simple Programs Create 3-D Images." *Computers in Physics*, Vol. 6, page 132 (1992). This article by the author describes in detail how computers can be programmed to produce anaglyphic images of 3-D objects.

Sprott, J. C. "How Common is Chaos?" *Physics Letters*, Vol. 173A, page 21 (1993). An abbreviated technical paper quantifying the occurrence of chaos in polynomial maps and ODEs that was the inspiration for this book.

Sprott, J.C. "Automatic Generation of Strange Attractors." *Computers &Graphics*, Vol. 17, page 325 (1993). An abbreviated description of the technique that forms the basis of the present book.

Sprott, J. C. and G. Rowlands. *Chaos Demonstrations* (North Carolina State University, Raleigh, NC: Physics Academic Software, 27695-8202). An IBM PC program by the author and a colleague that provides a simple way to learn about chaos, fractals, and related phenomena. It comes with an 84-page user's manual and 3-D glasses.

Stevens, R. T. *Fractal Programming in C* (New York: M&T Books, 1989). Also available in a TurboPascal version, this book provides programs and detailed descriptions for producing most of the standard fractal forms.

Stevens, R. T. *Advanced Fractal Programming in C* (New York: M&T Books, 1990). Picking up where Stevens' previous one left off, this book emphasizes Mandelbrot and Julia sets but also discusses other fractal types, such as L-systems and iterated function systems.

Stewart, I. *Does God Play Dice?: The Mathematics of Chaos* (Oxford: Blackwell, 1989). A charming, light-hearted, but serious book that provides a good introduction to chaos using simple mathematics.

Stewart, I. and M. Golubitsky. *Fearful Symmetry: Is God a Geometer?* (Oxford: Blackwell, 1992). This sequel to the earlier book by Stewart deals with symmetry in nature, art, and science and provides computer programs for producing symmetrical patterns of considerable beauty.

Theiler, J. "Estimating Fractal Dimension." *Journal of the Optical Society of America*, Vol. 7A, page 1055 (1990). A slightly technical paper containing an excellent review of the various ways of calculating fractal dimensions.

Thompson, J. M. T. and H. B. Stewart. *Nonlinear Dynamics and Chaos* (New York: Wiley, 1986). This advanced undergraduate text aimed at physical science students explains the mathematical basis for chaos.

Tsonis, A. A. *Chaos: From Theory to Applications* (New York: Plenum Press, 1992). An advanced undergraduate text that includes recent developments in the applications of chaos theory to real-world problems, such as improved methods of forecasting and distinguishing chaos from noise.

Weeks, J. R. *The Shape of Space: How to Visualize Surfaces and Three-Dimensional Manifolds* (New York: Marcel Dekker, Incorporated, 1985). This book explains in simple language and with clear illustrations the elements of topology that are

fundamental to a deep understanding of strange attractors and other geometrical objects.

Wegner, T. and M. Peterson. *Fractal Creations* (Mill Valley, CA: Waite Group Press, 1991). This book is really a user's manual for the incredible freeware program FRACTINT, which is constantly updated by a dedicated group of volunteer fractal programming enthusiasts.

Wolf, A., J. B. Swift, H. L. Swinney, and J. A. Vastano. "Determining Lyapunov Exponents from a Time Series." *Physica*, Vol. 16D, page 285 (1985). A technical but surprisingly readable article that contains an excellent description of the practical considerations that go into the calculation of Lyapunov exponents.

Zhang, S. Y. *Bibliography on Chaos* (Singapore: World Scientific, 1991). With over 7000 references, this is the most extensive compilation available, and it serves to underscore the popularity of the subject.

Appendix B

BASIC Program Listing

This appendix contains the complete BASIC program that you should have developed if you followed the exercises in this book, along with a few additions. It should run without modification on DOS-based IBM personal computers or compatibles under Microsoft BASICA, GW-BASIC, QBASIC, QuickBASIC, or VisualBASIC for MS-DOS; Borland International Turbo BASIC; and Spectra Publishing PowerBASIC. A disk containing the source program and a version of the program complied with PowerBASIC (**SA.EXE**) and ready to run is included with the book. This is a relatively no-frills program in that it lacks extensive error trapping, fancy menus, and mouse support, but it is fully functional and relatively robust.

The additions to the program are as follows:

1. The program contains a version number (2.0) and a copyright notice. Your purchase of this book and the accompanying disk entitles you to personal use of the program. It is not legal for you to make a copy of the program for someone else, to place it in the public domain, or to incorporate it in whole or in part into programs that are distributed to others. The idea of programming a computer to search automatically for strange attractors based on calculation of the Lyapunov exponent is believed to be original, and proper scientific etiquette requires that you acknowledge the author in any further dissemination of work based on this technique.

2. The program includes a somewhat inelegant but effective test for the graphics capability of the computer on which it is used. It causes the program to run automatically in the highest graphics mode supported by the hardware and by the BASIC version under which it is compiled

or run. The program prints a message and stops if the computer does not have a graphics monitor. Colors are adjusted for CGA MODE 1, and text is properly formatted for screens with 40 columns of text.

3. The program allows you to change the number of iterations that are plotted while in the search mode using the **N** key. Values of a thousand (10^3) to a billion (10^9) are allowed. Note that this value excludes the thousand iterations that are always performed to allow the initial transient to decay.

4. In addition to the planar and spherical projections, the **P** command allows you to project the attractors onto a cylinder with a horizontal or vertical axis or a torus. The toroidal projection is shown looking along the major axis with the doughnut hole at the center of the screen and of negligible size.

5. A **C** command has been added to allow you to clear the screen and restart the calculation with the current values of the variables. This feature allows you to remove the transient in cases when the orbit requires more than a thousand iterations to reach the attractor or to reduce the density of points on the screen, which is sometimes useful, for example, with the 3-D anaglyphic displays.

6. The program allows you to press **V** to save a record of up to 16,000 consecutive iterates of *X, Y, Z*, or *W* in a disk file that can be analyzed in more detail by other programs. To conserve disk space, each new attractor overwrites the data from the previous case. The data files can be read by the companion program *Chaos Data Analyzer*, which allows the data to be displayed in many ways, including phase-space plots, return maps, and Poincaré movies; calculates probability distributions, power spectra, Lyapunov exponents, correlation functions, and capacity and correlation dimensions; and makes predictions based on a novel technique involving singular value decomposition. *Chaos Data Analyzer* is available from The Academic Software Library, Box 8202, North Carolina State University, Raleigh, NC 27695-8202, telephone (800) 955-TASL or (919) 515-7447.

If you have been working systematically through the programs in this book, you will find useful the following list of program lines that require changes to produce the final program **PROG28.BAS**:

```
1000, 1070, 1090, 1140, 1310, 1330, 1340, 1360, 2270-2290, 2500, 3060, 3360, 3400-
3420, 3630, 3670, 3740, 3750, 3780, 4000, 4240, 4250, 4280, 4380, 4390, 4430-4450,
4570-4590, 5650-5710, 5840, 6600-7070
```

PROG28.BAS. Complete BASIC program for producing all the examples in this book and endless variations

```
1000 REM STRANGE ATTRACTOR PROGRAM  BASIC Ver 2.0 (c) 1993 by J. C. Sprott

1010 DEFDBL A-Z                 'Use double precision

1020 DIM XS(499), YS(499), ZS(499), WS(499), A(504), V(99), XY(4), XN(4), COLR%(15)
```

```
1030 SM% = 12                     'Assume VGA graphics
1040 PREV% = 5                    'Plot versus fifth previous iterate
1050 NMAX = 11000                 'Maximum number of iterations
1060 OMAX% = 5                    'Maximum order of polynomial
1070 D% = 2                       'Dimension of system
1080 EPS = .1                     'Step size for ODE
1090 ODE% = 0                     'System is map
1100 SND% = 0                     'Turn sound off
1110 PJT% = 0                     'Projection is planar
1120 TRD% = 1                     'Display third dimension as shadow
1130 FTH% = 2                     'Display fourth dimension as colors
1140 SAV% = 0                     'Don't save any data
1150 TWOPI = 6.28318530717959#  'A useful constant (2 pi)
1160 RANDOMIZE TIMER              'Reseed random-number generator
1170 GOSUB 4200                   'Display menu screen
1180 IF Q$ = "X" THEN GOTO 1250 'Exit immediately on command
1190 GOSUB 1300                   'Initialize
1200 GOSUB 1500                   'Set parameters
1210 GOSUB 1700                   'Iterate equations
1220 GOSUB 2100                   'Display results
1230 GOSUB 2400                   'Test results
1240 ON T% GOTO 1190, 1200, 1210
1250 CLS
```

```
1260 END


1300 REM Initialize
1310 ON ERROR GOTO 6600         'Find legal graphics mode
1320 SCREEN SM%                 'Set graphics mode
1330 ON ERROR GOTO 0            'Resume default error trapping
1340 DEF SEG = 64: WID% = PEEK(74)  'Number of text columns
1350 WINDOW (-.1, -.1)-(1.1, 1.1)
1360 CLS : LOCATE 13, WID% / 2 - 6: PRINT "Searching..."
1370 GOSUB 5600                 'Set colors
1380 IF QM% <> 2 THEN GOTO 1420
1390    NE = 0: CLOSE
1400    OPEN "SA.DIC" FOR APPEND AS #1: CLOSE
1410    OPEN "SA.DIC" FOR INPUT AS #1
1420 RETURN


1500 REM Set parameters
1510 X = .05                    'Initial condition
1520 Y = .05
1530 Z = .05
1540 W = .05
1550 XE = X + .000001: YE = Y: ZE = Z: WE = W
1560 GOSUB 2600                 'Get coefficients
```

```
1570 T% = 3
1580 P% = 0: LSUM = 0: N = 0: NL = 0: N1 = 0: N2 = 0
1590 XMIN = 1000000!: XMAX = -XMIN: YMIN = XMIN: YMAX = XMAX
1600 ZMIN = XMIN: ZMAX = XMAX
1610 WMIN = XMIN: WMAX = XMAX
1620 TWOD% = 2 ^ D%
1630 RETURN

1700 REM Iterate equations
1710 IF ODE% > 1 THEN GOSUB 6200: GOTO 2020     'Special function
1720 M% = 1: XY(1) = X: XY(2) = Y: XY(3) = Z: XY(4) = W
1730 FOR I% = 1 TO D%
1740 XN(I%) = A(M%)
1750 M% = M% + 1
1760 FOR I1% = 1 TO D%
1770 XN(I%) = XN(I%) + A(M%) * XY(I1%)
1780 M% = M% + 1
1790 FOR I2% = I1% TO D%
1800 XN(I%) = XN(I%) + A(M%) * XY(I1%) * XY(I2%)
1810 M% = M% + 1
1820 IF O% = 2 THEN GOTO 1970
1830 FOR I3% = I2% TO D%
1840 XN(I%) = XN(I%) + A(M%) * XY(I1%) * XY(I2%) * XY(I3%)
```

```
1850 M% = M% + 1
1860 IF O% = 3 THEN GOTO 1960
1870 FOR I4% = I3% TO D%
1880 XN(I%) = XN(I%) + A(M%) * XY(I1%) * XY(I2%) * XY(I3%) * XY(I4%)
1890 M% = M% + 1
1900 IF O% = 4 THEN GOTO 1950
1910 FOR I5% = I4% TO D%
1920 XN(I%) = XN(I%) + A(M%) * XY(I1%) * XY(I2%) * XY(I3%) * XY(I4%) * XY(I5%)
1930 M% = M% + 1
1940 NEXT I5%
1950 NEXT I4%
1960 NEXT I3%
1970 NEXT I2%
1980 NEXT I1%
1990 IF ODE% = 1 THEN XN(I%) = XY(I%) + EPS * XN(I%)
2000 NEXT I%
2010 M% = M% - 1: XNEW = XN(1): YNEW = XN(2): ZNEW = XN(3): WNEW = XN(4)
2020 N = N + 1
2030 RETURN

2100 REM Display results
2110 IF N < 100 OR N > 1000 THEN GOTO 2200
2120    IF X < XMIN THEN XMIN = X
```

```
2130    IF X > XMAX THEN XMAX = X
2140    IF Y < YMIN THEN YMIN = Y
2150    IF Y > YMAX THEN YMAX = Y
2160    IF Z < ZMIN THEN ZMIN = Z
2170    IF Z > ZMAX THEN ZMAX = Z
2180    IF W < WMIN THEN WMIN = W
2190    IF W > WMAX THEN WMAX = W
2200 IF N = 1000 THEN GOSUB 3100        'Resize the screen
2210 XS(P%) = X: YS(P%) = Y: ZS(P%) = Z: WS(P%) = W
2220 P% = (P% + 1) MOD 500
2230 I% = (P% + 500 - PREV%) MOD 500
2240 IF D% = 1 THEN XP = XS(I%): YP = XNEW ELSE XP = X: YP = Y
2250 IF N < 1000 OR XP <= XL OR XP >= XH OR YP <= YL OR YP >= YH THEN GOTO 2320
2260    IF PJT% = 1 THEN GOSUB 4100     'Project onto a sphere
2270    IF PJT% = 2 THEN GOSUB 6700     'Project onto a horizontal cylinder
2280    IF PJT% = 3 THEN GOSUB 6800     'Project onto a vertical cylinder
2290    IF PJT% = 4 THEN GOSUB 6900     'Project onto a torus
2300    GOSUB 5000                      'Plot point on screen
2310    IF SND% = 1 THEN GOSUB 3500     'Produce sound
2320 RETURN

2400 REM Test results
2410 IF ABS(XNEW) + ABS(YNEW) + ABS(ZNEW) + ABS(WNEW) > 1000000! THEN T% = 2
```

```
2420 IF QM% = 2 THEN GOTO 2490  'Speed up evaluation mode

2430 GOSUB 2900                 'Calculate Lyapunov exponent

2440 GOSUB 3900                 'Calculate fractal dimension

2450 IF QM% > 0 THEN GOTO 2490  'Skip tests when not in search mode

2460 IF N >= NMAX THEN T% = 2: GOSUB 4900   'Strange attractor found

2470 IF ABS(XNEW - X) + ABS(YNEW - Y) + ABS(ZNEW - Z) + ABS(WNEW - W) < .000001
THEN T% = 2

2480 IF N > 100 AND L < .005 THEN T% = 2    'Limit cycle

2490 Q$ = INKEY$: IF LEN(Q$) THEN GOSUB 3600        'Respond to user command

2500 IF SAV% > 0 THEN IF N > 1000 AND N < 17001 THEN GOSUB 7000 'Save data

2510 X = XNEW                   'Update value of X

2520 Y = YNEW

2530 Z = ZNEW

2540 W = WNEW

2550 RETURN


2600 REM Get coefficients

2610 IF QM% <> 2 THEN GOTO 2640 'Not in evaluate mode

2620    IF EOF(1) THEN QM% = 0: GOSUB 6000: GOTO 2640

2630    IF EOF(1) = 0 THEN LINE INPUT #1, CODE$: GOSUB 4700: GOSUB 5600

2640 IF QM% > 0 THEN GOTO 2730  'Not in search mode

2650    O% = 2 + INT((OMAX% - 1) * RND)

2660    CODE$ = CHR$(59 + 4 * D% + O% + 8 * ODE%)

2670    IF ODE% > 1 THEN CODE$ = CHR$(87 + ODE%)
```

```
2680    GOSUB 4700              'Get value of M%
2690    FOR I% = 1 TO M%        'Construct CODE$
2700        GOSUB 2800          'Shuffle random numbers
2710        CODE$ = CODE$ + CHR$(65 + INT(25 * RAN))
2720    NEXT I%
2730 FOR I% = 1 TO M%           'Convert CODE$ to coefficient values
2740    A(I%) = (ASC(MID$(CODE$, I% + 1, 1)) - 77) / 10
2750 NEXT I%
2760 RETURN


2800 REM Shuffle random numbers
2810 IF V(0) = 0 THEN FOR J% = 0 TO 99: V(J%) = RND: NEXT J%
2820 J% = INT(100 * RAN)
2830 RAN = V(J%)
2840 V(J%) = RND
2850 RETURN


2900 REM Calculate Lyapunov exponent
2910 XSAVE = XNEW: YSAVE = YNEW: ZSAVE = ZNEW: WSAVE = WNEW
2920 X = XE: Y = YE: Z = ZE: W = WE: N = N - 1
2930 GOSUB 1700                 'Reiterate equations
2940 DLX = XNEW - XSAVE: DLY = YNEW - YSAVE
2950 DLZ = ZNEW - ZSAVE: DLW = WNEW - WSAVE
```

```
2960 DL2 = DLX * DLX + DLY * DLY + DLZ * DLZ + DLW * DLW

2970 IF CSNG(DL2) <= 0 THEN GOTO 3070       'Don't divide by zero

2980    DF = 1000000000000# * DL2

2990    RS = 1 / SQR(DF)

3000    XE = XSAVE + RS * (XNEW - XSAVE): YE = YSAVE + RS * (YNEW - YSAVE)

3010    ZE = ZSAVE + RS * (ZNEW - ZSAVE): WE = WSAVE + RS * (WNEW - WSAVE)

3020    XNEW = XSAVE: YNEW = YSAVE: ZNEW = ZSAVE: WNEW = WSAVE

3030    LSUM = LSUM + LOG(DF): NL = NL + 1

3040    L = .721347 * LSUM / NL

3050    IF ODE% = 1 OR ODE% = 7 THEN L = L / EPS

3060   IF N > 1000 AND N MOD 10 = 0 THEN LOCATE 1, WID% - 4: PRINT USING "##.##";
L;

3070 RETURN


3100 REM Resize the screen

3110 IF D% = 1 THEN YMIN = XMIN: YMAX = XMAX

3120 IF XMAX - XMIN < .000001 THEN XMIN = XMIN - .0000005: XMAX = XMAX + .0000005

3130 IF YMAX - YMIN < .000001 THEN YMIN = YMIN - .0000005: YMAX = YMAX + .0000005

3140 IF ZMAX - ZMIN < .000001 THEN ZMIN = ZMIN - .0000005: ZMAX = ZMAX + .0000005

3150 IF WMAX - WMIN < .000001 THEN WMIN = WMIN - .0000005: WMAX = WMAX + .0000005

3160 MX = .1 * (XMAX - XMIN): MY = .1 * (YMAX - YMIN)

3170 XL = XMIN - MX: XH = XMAX + MX: YL = YMIN - MY: YH = YMAX + 1.5 * MY

3180 WINDOW (XL, YL)-(XH, YH): CLS

3190 YH = YH - .5 * MY
```

```
3200 XA = (XL + XH) / 2: YA = (YL + YH) / 2
3210 IF D% < 3 THEN GOTO 3310
3220    ZA = (ZMAX + ZMIN) / 2
3230    IF TRD% = 1 THEN LINE (XL, YL)-(XH, YH), COLR%(1), BF: GOSUB 5400
3240    IF TRD% = 4 THEN LINE (XL, YL)-(XH, YH), WH%, BF
3250    IF TRD% = 5 THEN LINE (XA, YL)-(XA, YH)
3260    IF TRD% <> 6 THEN GOTO 3310
3270        FOR I% = 1 TO 3
3280            XP = XL + I% * (XH - XL) / 4: LINE (XP, YL)-(XP, YH)
3290            YP = YL + I% * (YH - YL) / 4: LINE (XL, YP)-(XH, YP)
3300        NEXT I%
3310 IF PJT% <> 1 THEN LINE (XL, YL)-(XH, YH), , B
3320 IF PJT% = 1 AND TRD% < 5 THEN CIRCLE (XA, YA), .36 * (XH - XL)
3330 TT = 3.1416 / (XMAX - XMIN): PT = 3.1416 / (YMAX - YMIN)
3340 IF QM% <> 2 THEN GOTO 3400         'Not in evaluate mode
3350    LOCATE 1, 1: PRINT "<Space Bar>: Discard       <Enter>: Save";
3360    IF WID% < 80 THEN GOTO 3390
3370    LOCATE 1, 49: PRINT "<Esc>: Exit";
3380    LOCATE 1, 69: PRINT CINT((LOF(1) - 128 * LOC(1)) / 1024); "K left";
3390    GOTO 3430
3400 LOCATE 1, 1: IF LEN(CODE$) < WID% - 18 THEN PRINT CODE$
3410 IF LEN(CODE$) >= WID% - 18 THEN PRINT LEFT$(CODE$, WID% - 23) + "..."
3420 LOCATE 1, WID% - 17: PRINT "F =": LOCATE 1, WID% - 7: PRINT "L ="
```

```
3430 TIA = .05               'Tangent of illumination angle
3440 XZ = -TIA * (XMAX - XMIN) / (ZMAX - ZMIN)
3450 YZ = TIA * (YMAX - YMIN) / (ZMAX - ZMIN)
3460 RETURN

3500 REM Produce sound
3510 FREQ% = 220 * 2 ^ (CINT(36 * (XNEW - XL) / (XH - XL)) / 12)
3520 DUR = 1
3530 IF D% > 1 THEN DUR = 2 ^ INT(.5 * (YH - YL) / (YNEW - 9 * YL / 8 + YH / 8))
3540 SOUND FREQ%, DUR: IF PLAY(0) THEN PLAY "MF"
3550 RETURN

3600 REM Respond to user command
3610 IF ASC(Q$) > 96 THEN Q$ = CHR$(ASC(Q$) - 32)   'Convert to upper case
3620 IF QM% = 2 THEN GOSUB 5800                  'Process evaluation command
3630 IF INSTR("ACDEHINPRSVX", Q$) = 0 THEN GOSUB 4200   'Display menu screen
3640 IF Q$ = "A" THEN T% = 1: QM% = 0
3650 IF ODE% > 1 THEN D% = ODE% + 5
3660 IF ODE% = 1 THEN D% = D% + 2
3670 IF Q$ = "C" THEN IF N > 999 THEN N = 999
3680 IF Q$ = "D" THEN D% = 1 + (D% MOD 12): T% = 1
3690 IF D% > 6 THEN ODE% = D% - 5: D% = 4: GOTO 3710
3700 IF D% > 4 THEN ODE% = 1: D% = D% - 2 ELSE ODE% = 0
```

```
3710 IF Q$ = "E" THEN T% = 1: QM% = 2

3720 IF Q$ = "H" THEN FTH% = (FTH% + 1) MOD 3: T% = 3: IF N > 999 THEN N = 999:
GOSUB 5600

3730 IF Q$ = "I" THEN IF T% <> 1 THEN SCREEN 0: WIDTH 80: COLOR 15, 1: CLS : LINE
INPUT "Code? "; CODE$: IF CODE$ = "" THEN Q$ = " ": CLS :  ELSE T% = 1: QM% = 1:
GOSUB 4700

3740 IF Q$ = "N" THEN NMAX = 10 * (NMAX - 1000) + 1000: IF NMAX > 10 ^ 10 THEN
NMAX = 2000

3750 IF Q$ = "P" THEN PJT% = (PJT% + 1) MOD 5: T% = 3: IF N > 999 THEN N = 999

3760 IF Q$ = "R" THEN TRD% = (TRD% + 1) MOD 7: T% = 3: IF N > 999 THEN N = 999:
GOSUB 5600

3770 IF Q$ = "S" THEN SND% = (SND% + 1) MOD 2: T% = 3

3780 IF Q$ = "V" THEN SAV% = (SAV% + 1) MOD 5: FAV$ = CHR$(87 + SAV% MOD 4): T%
= 3: IF N > 999 THEN N = 999

3790 IF Q$ = "X" THEN T% = 0

3800 RETURN



3900 REM Calculate fractal dimension

3910 IF N < 1000 THEN GOTO 4010         'Wait for transient to settle

3920 IF N = 1000 THEN D2MAX = (XMAX - XMIN) ^ 2 + (YMAX - YMIN) ^ 2 + (ZMAX - ZMIN)
^ 2 + (WMAX - WMIN) ^ 2

3930 J% = (P% + 1 + INT(480 * RND)) MOD 500

3940 DX = XNEW - XS(J%): DY = YNEW - YS(J%): DZ = ZNEW - ZS(J%): DW = WNEW - WS(J%)

3950 D2 = DX * DX + DY * DY + DZ * DZ + DW * DW

3960 IF D2 < .001 * TWOD% * D2MAX THEN N2 = N2 + 1

3970 IF D2 > .00001 * TWOD% * D2MAX THEN GOTO 4010

3980    N1 = N1 + 1
```

```
3990    F = .434294 * LOG(N2 / (N1 - .5))
4000    LOCATE 1, WID% - 14: PRINT USING "##.##"; F;
4010 RETURN


4100 REM Project onto a sphere
4110 TH = TT * (XMAX - XP)
4120 PH = PT * (YMAX - YP)
4130 XP = XA + .36 * (XH - XL) * COS(TH) * SIN(PH)
4140 YP = YA + .5 * (YH - YL) * COS(PH)
4150 RETURN


4200 REM Display menu screen
4210 SCREEN 0: WIDTH 80: COLOR 15, 1: CLS
4220 WHILE Q$ = "" OR INSTR("AEIX", Q$) = 0
4230    LOCATE 1, 27: PRINT "STRANGE ATTRACTOR PROGRAM"
4240    PRINT TAB(27); "IBM PC BASIC Version 2.0"
4250    PRINT TAB(27); "(c) 1993 by J. C. Sprott"
4260    PRINT : PRINT
4270    PRINT TAB(27); "A: Search for attractors"
4280    PRINT TAB(27); "C: Clear screen and restart"
4290   IF ODE% > 1 THEN PRINT TAB(27); "D: System is 4-D special map "; CHR$(87
+ ODE%); " ": GOTO 4320
4300    PRINT TAB(27); "D: System is"; STR$(D%); "-D polynomial ";
4310    IF ODE% = 1 THEN PRINT "ODE" ELSE PRINT "map"
```

```
4320    PRINT TAB(27); "E: Evaluate attractors"
4330    PRINT TAB(27); "H: Fourth dimension is ";
4340        IF FTH% = 0 THEN PRINT "projection"
4350        IF FTH% = 1 THEN PRINT "bands     "
4360        IF FTH% = 2 THEN PRINT "colors    "
4370    PRINT TAB(27); "I: Input code from keyboard"
4380    PRINT TAB(27); "N: Number of iterations is 10^";
4390    PRINT USING "#"; CINT(LOG(NMAX - 1000) / LOG(10))
4400    PRINT TAB(27); "P: Projection is ";
4410        IF PJT% = 0 THEN PRINT "planar   "
4420        IF PJT% = 1 THEN PRINT "spherical"
4430        IF PJT% = 2 THEN PRINT "horiz cyl"
4440        IF PJT% = 3 THEN PRINT "vert cyl "
4450        IF PJT% = 4 THEN PRINT "toroidal "
4460    PRINT TAB(27); "R: Third dimension is ";
4470        IF TRD% = 0 THEN PRINT "projection"
4480        IF TRD% = 1 THEN PRINT "shadow    "
4490        IF TRD% = 2 THEN PRINT "bands     "
4500        IF TRD% = 3 THEN PRINT "colors    "
4510        IF TRD% = 4 THEN PRINT "anaglyph  "
4520        IF TRD% = 5 THEN PRINT "stereogram"
4530        IF TRD% = 6 THEN PRINT "slices    "
4540    PRINT TAB(27); "S: Sound is ";
```

```
4550        IF SND% = 0 THEN PRINT "off"
4560        IF SND% = 1 THEN PRINT "on "
4570    PRINT TAB(27); "V: ";
4580        IF SAV% = 0 THEN PRINT "No data will be saved       "
4590       IF SAV% > 0 THEN PRINT FAV$; " will be saved in "; FAV$; "DATA.DAT"
4600    PRINT TAB(27); "X: Exit program"
4610    Q$ = INKEY$
4620    IF Q$ <> "" THEN GOSUB 3600     'Respond to user command
4630 WEND
4640 RETURN


4700 REM Get dimension and order
4710 D% = 1 + INT((ASC(LEFT$(CODE$, 1)) - 65) / 4)
4720 IF D% > 6 THEN ODE% = ASC(LEFT$(CODE$, 1)) - 87: D% = 4: GOSUB 6200: GOTO
4770
4730 IF D% > 4 THEN D% = D% - 2: ODE% = 1 ELSE ODE% = 0
4740 O% = 2 + (ASC(LEFT$(CODE$, 1)) - 65) MOD 4
4750 M% = 1: FOR I% = 1 TO D%: M% = M% * (O% + I%): NEXT I%
4760 IF D% > 2 THEN FOR I% = 3 TO D%: M% = M% / (I% - 1): NEXT I%
4770 IF LEN(CODE$) = M% + 1 OR QM% <> 1 THEN GOTO 4810
4780    BEEP        'Illegal code warning
4790    WHILE LEN(CODE$) < M% + 1: CODE$ = CODE$ + "M": WEND
4800    IF LEN(CODE$) > M% + 1 THEN CODE$ = LEFT$(CODE$, M% + 1)
4810 RETURN
```

```
4900 REM Save attractor to disk file SA.DIC

4910 OPEN "SA.DIC" FOR APPEND AS #1

4920 PRINT #1, CODE$; : PRINT #1, USING "##.##"; F; L

4930 CLOSE #1

4940 RETURN



5000 REM Plot point on screen

5010 C4% = WH%

5020 IF D% < 4 THEN GOTO 5050

5030    IF FTH% = 1 THEN IF INT(30 * (W - WMIN) / (WMAX - WMIN)) MOD 2 THEN GOTO
5330

5040    IF FTH% = 2 THEN C4% = 1 + INT(NC% * (W - WMIN) / (WMAX - WMIN) + NC%)
MOD NC%

5050 IF D% < 3 THEN PSET (XP, YP): GOTO 5330    'Skip 3-D stuff

5060 IF TRD% = 0 THEN PSET (XP, YP), C4%

5070 IF TRD% <> 1 THEN GOTO 5130

5080    IF D% > 3 AND FTH% = 2 THEN PSET (XP, YP), C4%: GOTO 5110

5090    C% = POINT(XP, YP)

5100   IF C% = COLR%(2) THEN PSET (XP, YP), COLR%(3) ELSE IF C% <> COLR%(3) THEN
PSET (XP, YP), COLR%(2)

5110    XP = XP - XZ * (Z - ZMIN): YP = YP - YZ * (Z - ZMIN)

5120    IF POINT(XP, YP) = COLR%(1) THEN PSET (XP, YP), 0

5130 IF TRD% <> 2 THEN GOTO 5160

5140    IF D% > 3 AND FTH% = 2 AND (INT(15 * (Z - ZMIN) / (ZMAX - ZMIN) + 2) MOD
```

```
2) = 1 THEN PSET (XP, YP), C4%

5150   IF D% < 4 OR FTH% <> 2 THEN C% = COLR%(INT(60 * (Z - ZMIN) / (ZMAX - ZMIN)
+ 4) MOD 4): PSET (XP, YP), C%

5160 IF TRD% = 3 THEN PSET (XP, YP), COLR%(INT(NC% * (Z - ZMIN) / (ZMAX - ZMIN)
+ NC%) MOD NC%)

5170 IF TRD% <> 4 THEN GOTO 5240

5180    XRT = XP + XZ * (Z - ZA): C% = POINT(XRT, YP)

5190    IF C% = WH% THEN PSET (XRT, YP), RD%

5200    IF C% = CY% THEN PSET (XRT, YP), BK%

5210    XLT = XP - XZ * (Z - ZA): C% = POINT(XLT, YP)

5220    IF C% = WH% THEN PSET (XLT, YP), CY%

5230    IF C% = RD% THEN PSET (XLT, YP), BK%

5240 IF TRD% <> 5 THEN GOTO 5280

5250    HSF = 2                 'Horizontal scale factor

5260    XRT = XA + (XP + XZ * (Z - ZA) - XL) / HSF: PSET (XRT, YP), C4%

5270    XLT = XA + (XP - XZ * (Z - ZA) - XH) / HSF: PSET (XLT, YP), C4%

5280 IF TRD% <> 6 THEN GOTO 5330

5290    DZ = (15 * (Z - ZMIN) / (ZMAX - ZMIN) + .5) / 16

5300    XP = (XP - XL + (INT(16 * DZ) MOD 4) * (XH - XL)) / 4 + XL

5310    YP = (YP - YL + (3 - INT(4 * DZ) MOD 4) * (YH - YL)) / 4 + YL

5320    PSET (XP, YP), C4%

5330 RETURN


5400 REM Plot background grid
```

```
5410 FOR I% = 0 TO 15           'Draw 15 vertical grid lines
5420    XP = XMIN + I% * (XMAX - XMIN) / 15
5430    LINE (XP, YMIN)-(XP, YMAX), 0
5440 NEXT I%
5450 FOR I% = 0 TO 10           'Draw 10 horizontal grid lines
5460    YP = YMIN + I% * (YMAX - YMIN) / 10
5470    LINE (XMIN, YP)-(XMAX, YP), 0
5480 NEXT I%
5490 RETURN


5600 REM Set colors
5610 NC% = 15                   'Number of colors
5620 COLR%(0) = 0: COLR%(1) = 8: COLR%(2) = 7: COLR%(3) = 15
5630 IF TRD% = 3 OR (D% > 3 AND FTH% = 2 AND TRD% <> 1) THEN FOR I% = 0 TO NC%:
COLR%(I%) = I% + 1: NEXT I%
5640 WH% = 15: BK% = 8: RD% = 12: CY% = 11
5650 IF SM% > 2 THEN GOTO 5720  'Not in CGA mode
5660    WID% = 80: IF D% < 3 THEN SCREEN 2: GOTO 5720
5670    IF (TRD% = 0 OR TRD% > 4) AND (D% = 3 OR FTH% <> 2) THEN SCREEN 2: GOTO
5720
5680    WID% = 40: SCREEN 1
5690    COLR%(0) = 0: COLR%(1) = 2: COLR%(2) = 1: COLR%(3) = 3
5700    WH% = 3: BK% = 0: RD% = 2: CY% = 1
5710    FOR I% = 4 TO NC%: COLR%(I%) = COLR%(I% MOD 4 + 1): NEXT I%
```

```
5720 RETURN


5800 REM Process evaluation command
5810 IF Q$ = " " THEN T% = 2: NE = NE + 1: CLS
5820 IF Q$ = CHR$(13) THEN T% = 2: NE = NE + 1: CLS : GOSUB 5900
5830 IF Q$ = CHR$(27) THEN CLS : GOSUB 6000: Q$ = " ": QM% = 0: GOTO 5850
5840 IF Q$ <> CHR$(27) AND INSTR("CHNPRVS", Q$) = 0 THEN Q$ = ""
5850 RETURN


5900 REM Save favorite attractors to disk file FAVORITE.DIC
5910 OPEN "FAVORITE.DIC" FOR APPEND AS #2
5920 PRINT #2, CODE$
5930 CLOSE #2
5940 RETURN


6000 REM Update SA.DIC file
6010 LOCATE 11, 9: PRINT "Evaluation complete"
6020 LOCATE 12, 8: PRINT NE; "cases evaluated"
6030 OPEN "SATEMP.DIC" FOR OUTPUT AS #2
6040 IF QM% = 2 THEN PRINT #2, CODE$
6050 WHILE NOT EOF(1): LINE INPUT #1, CODE$: PRINT #2, CODE$: WEND
6060 CLOSE
6070 KILL "SA.DIC"
```

```
6080 NAME "SATEMP.DIC" AS "SA.DIC"

6090 RETURN



6200 REM Special function definitions

6210 ZNEW = X * X + Y * Y       'Default 3rd and 4th dimension

6220 WNEW = (N - 100) / 900: IF N > 1000 THEN WNEW = (N - 1000) / (NMAX - 1000)

6230 IF ODE% <> 2 THEN GOTO 6270

6240    M% = 10

6250    XNEW = A(1) + A(2) * X + A(3) * Y + A(4) * ABS(X) + A(5) * ABS(Y)

6260    YNEW = A(6) + A(7) * X + A(8) * Y + A(9) * ABS(X) + A(10) * ABS(Y)

6270 IF ODE% <> 3 THEN GOTO 6310

6280    M% = 14

6290    XNEW = A(1) + A(2) * X + A(3) * Y + (CINT(A(4) * X) AND CINT(A(5) * Y))
+ (CINT(A(6) * X) OR CINT(A(7) * Y))

6300    YNEW = A(8) + A(9) * X + A(10) * Y + (CINT(A(11) * X) AND CINT(A(12) *
Y)) + (CINT(A(13) * X) OR CINT(A(14) * Y))

6310 IF ODE% <> 4 THEN GOTO 6350

6320    M% = 14

6330    XNEW = A(1) + A(2) * X + A(3) * Y + A(4) * ABS(X) ^ A(5) + A(6) * ABS(Y)
^ A(7)

6340   YNEW = A(8) + A(9) * X + A(10) * Y + A(11) * ABS(X) ^ A(12) + A(13) * ABS(Y)
^ A(14)

6350 IF ODE% <> 5 THEN GOTO 6390

6360    M% = 18

6370    XNEW = A(1) + A(2) * X + A(3) * Y + A(4) * SIN(A(5) * X + A(6)) + A(7)
* SIN(A(8) * Y + A(9))
```

```
6380    YNEW = A(10) + A(11) * X + A(12) * Y + A(13) * SIN(A(14) * X + A(15)) +
A(16) * SIN(A(17) * Y + A(18))

6390 IF ODE% <> 6 THEN GOTO 6450

6400    M% = 6

6410   IF N < 2 THEN AL = TWOPI / (13 + 10 * A(6)): SINAL = SIN(AL): COSAL = COS(AL)

6420    DUM = X + A(2) * SIN(A(3) * Y + A(4))

6430    XNEW = 10 * A(1) + DUM * COSAL + Y * SINAL

6440    YNEW = 10 * A(5) - DUM * SINAL + Y * COSAL

6450 IF ODE% <> 7 THEN GOTO 6500

6460    M% = 9

6470    XNEW = X + EPS * A(1) * Y

6480    YNEW = Y + EPS * (A(2) * X + A(3) * X * X * X + A(4) * X * X * Y + A(5)
* X * Y * Y + A(6) * Y + A(7) * Y * Y * Y + A(8) * SIN(Z))

6490    ZNEW = Z + EPS * (A(9) + 1.3): IF ZNEW > TWOPI THEN ZNEW = ZNEW - TWOPI

6500 RETURN


6600 REM Find legal graphics mode

6610 SM% = SM% - 1

6620 IF SM% = 0 THEN PRINT "This program requires a graphics monitor": STOP

6630 RESUME


6700 REM Project onto a horizontal cylinder

6710 PH = PT * (YMAX - YP)

6720 YP = YA + .5 * (YH - YL) * COS(PH)
```

```
6730 RETURN


6800 REM Project onto a vertical cylinder
6810 TH = TT * (XMAX - XP)
6820 XP = XA + .5 * (XH - XL) * COS(TH)
6830 RETURN


6900 REM Project onto a torus (unity aspect ratio)
6910 TH = TT * (XMAX - XP)
6920 PH = 2 * PT * (YMAX - YP)
6930 XP = XA + .18 * (XH - XL) * (1 + COS(TH)) * SIN(PH)
6940 YP = YA + .25 * (YH - YL) * (1 + COS(TH)) * COS(PH)
6950 RETURN


7000 REM Save data
7010 IF N = 1001 THEN CLOSE #3: OPEN FAV$ + "DATA.DAT" FOR OUTPUT AS #3
7020 IF SAV% = 1 THEN DUM = XNEW
7030 IF SAV% = 2 THEN DUM = YNEW
7040 IF SAV% = 3 THEN DUM = ZNEW
7050 IF SAV% = 4 THEN DUM = WNEW
7060 PRINT #3, CSNG(DUM)
7070 RETURN
```

Appendix C

Other Computers and BASIC Versions

Here are some considerations for using the programs with non-IBM-compatible computers and with different dialects of BASIC.

BASICA and GW-BASIC

These old versions of BASIC are mostly compatible with the program listings in this book. They do not support VGA graphics, and the older versions don't even support EGA. Thus you may have to change *SM%* = 12 in line 1030 to a lower number. **PROG28.BAS** automatically selects an appropriate graphics mode.

These versions of BASIC do not support strings longer than 255 characters. The easiest way to circumvent this problem is to limit the four-dimensional searches to cubic polynomials. Adding the following line to the program after line 2680 accomplishes this:

```
2685    IF M% > 253 THEN GOTO 2650
```

Turbo BASIC and PowerBASIC

The program listings in this book are compatible with Turbo BASIC and PowerBASIC, except for a quirk with the CIRCLE command with VGA graphics that requires the following change in line 3320:

```
3320 IF PJT% = 1 AND TRD% < 5 THEN IF SM% < 11 THEN CIRCLE (XA, YA), .36 * (XH
- XL) ELSE CIRCLE (XA, YA), .5 * (YH - YL)
```

VisualBASIC for MS-DOS

The programs as listed compile and run directly under VisualBASIC for MS-DOS. This version of BASIC makes it easy for you to add pull-down menus, dialog boxes, and mouse support to give the user interface a more modern look and feel.

VisualBASIC for Windows

One way to convert the programs to run as Microsoft Windows applications is to use the utility **TRNSLATE.EXE** supplied with VisualBASIC for MS-DOS to translate the programs into VisualBASIC for Windows. Many program differences must be resolved, however.

A VisualBASIC for Windows version of the listing **PROG06** but without the keyboard strobe and sound capabilities is given in **PROG06VB.BAS**. VisualBASIC for Windows 2.0 does not support the SOUND statement or INKEY$ function, although these capabilities and many others are available through internal Windows drivers. You can use this listing as a starting point for converting the other programs in this book for use with Microsoft Windows. The accompanying disk contains a compiled version of this program in the file **SAWIN.EXE**. To run this program under Windows 3.0 or later, the **VBRUN200.DLL** run-time dynamic link library that comes with VisualBASIC version 2.00, which is included on the accompanying disk, must be in a directory in your search path.

PROG06VB.BAS. VisualBASIC for Windows version of PROG06

```
VERSION 2.00

Begin Form PROG06VB

   Caption         =   "Strange Attractors"

   Height          =   4620

   Left            =   828

   LinkTopic       =   "Form1"

   ScaleHeight     =   4200
```

```
   ScaleWidth      =   6420
   Top             =   1128
   Width           =   6516
End
DefDbl A-Z

Sub Form_Activate ()

1000 Rem TWO-D MAP SEARCH  VisualBASIC Ver 1.0 (c) 1993 by J. C. Sprott
1020 ReDim XS(499), A(504), V(99)
1040 PREV% = 5                ‘Plot versus fifth previous iterate
1050 NMAX = 11000             ‘Maximum number of iterations
1060 OMAX% = 2                ‘Maximum order of polynomial
1070 D% = 2                   ‘Dimension of system
1160 Randomize Timer          ‘Reseed random-number generator
1190 GoSub 1300               ‘Initialize
1200 GoSub 1500               ‘Set parameters
1210 GoSub 1700               ‘Iterate equations
1220 GoSub 2100               ‘Display results
1230 GoSub 2400               ‘Test results
1240 On T% GoTo 1190, 1200, 1210
1250 Cls
1260 End
```

```
1300 Rem Initialize
1320 Cls: Msg$ = "Searching..."
1350 CurrentX = (ScaleWidth - TextWidth(Msg$)) / 2
1360 CurrentY = (ScaleHeight - TextHeight(Msg$)) / 2
1370 Print Msg$
1420 Return

1500 Rem Set parameters
1510 X = .05                    ‘Initial condition
1520 Y = .05
1550 XE = X + .000001: YE = Y
1560 GoSub 2600                 ‘Get coefficients
1570 T% = 3
1580 P% = 0: LSUM = 0:  N = 0:  NL = 0
1590 XMIN = 1000000!: XMAX = -XMIN:  YMIN = XMIN:  YMAX = XMAX
1630 Return

1700 Rem Iterate equations
1720 XNEW = A(1) + X * (A(2) + A(3) * X + A(4) * Y)
1730 XNEW = XNEW + Y * (A(5) + A(6) * Y)
1830 YNEW = A(7) + X * (A(8) + A(9) * X + A(10) * Y)
1930 YNEW = YNEW + Y * (A(11) + A(12) * Y)
```

```
2020 N = N + 1
2030 Return

2100 Rem Display results
2110 If N < 100 Or N > 1000 Then GoTo 2200
2120      If X < XMIN Then XMIN = X
2130      If X > XMAX Then XMAX = X
2140      If Y < YMIN Then YMIN = Y
2150      If Y > YMAX Then YMAX = Y
2200 If N = 1000 Then GoSub 3100       ‘Resize the screen
2210 XS(P%) = X
2220 P% = (P% + 1) Mod 500
2230 I% = (P% + 500 - PREV%) Mod 500
2240 If D% = 1 Then XP = XS(I%): YP = XNEW Else XP = X:  YP = Y
2250 If N < 1000 Or XP <= XL Or XP >= XH Or YP <= YL Or YP >= YH Then GoTo 2320
2300      PSet (XP, YP)                   ‘Plot point on screen
2320 Return

2400 Rem Test results
2410 If Abs(XNEW) + Abs(YNEW) > 1000000! Then T% = 2   ‘Unbounded
2430 GoSub 2900                ‘Calculate Lyapunov exponent
2460 If N >= NMAX Then T% = 2  ‘Strange attractor found
2470 If Abs(XNEW - X) + Abs(YNEW - Y) < .000001 Then T% = 2
```

```
2480 If N > 100 And L < .005 Then T% = 2   'Limit cycle
2490 DoEvents                  'Respond to user command
2510 X = XNEW                  'Update value of X
2520 Y = YNEW
2550 Return

2600 Rem Get coefficients
2650 O% = 2 + Int((OMAX% - 1) * Rnd)
2660 CODE$ = Chr$(59 + 4 * D% + O%)
2680 M% = 1: For I% = 1 To D%:  M% = M% * (O% + I%):  Next I%
2690      For I% = 1 To M%        'Construct CODE$
2700           GoSub 2800          'Shuffle random numbers
2710           CODE$ = CODE$ + Chr$(65 + Int(25 * RAN))
2720      Next I%
2730 For I% = 1 To M%          'Convert CODE$ to coefficient values
2740      A(I%) = (Asc(Mid$(CODE$, I% + 1, 1)) - 77) / 10
2750 Next I%
2760 Return

2800 Rem Shuffle random numbers
2810 If V(0) = 0 Then For J% = 0 To 99: V(J%) = Rnd:  Next J%
2820 J% = Int(100 * RAN)
2830 RAN = V(J%)
```

```
2840 V(J%) = Rnd
2850 Return

2900 Rem Calculate Lyapunov exponent
2910 XSAVE = XNEW: YSAVE = YNEW:  X = XE:  Y = YE:  N = N - 1
2930 GoSub 1700                ‘Reiterate equations
2940 DLX = XNEW - XSAVE: DLY = YNEW - YSAVE
2960 DL2 = DLX * DLX + DLY * DLY
2970 If CSng(DL2) <= 0 Then GoTo 3070      ‘Don’t divide by zero
2980      DF = 1000000000000# * DL2
2990      RS = 1 / Sqr(DF)
3000      XE = XSAVE + RS * (XNEW - XSAVE):  YE = YSAVE + RS * (YNEW - YSAVE)
3020      XNEW = XSAVE:  YNEW = YSAVE
3030      If DF > 0 Then LSUM = LSUM + Log(DF):  NL = NL + 1
3040      L = .721347 * LSUM / NL
3070 Return

3100 Rem Resize the screen
3110 If D% = 1 Then YMIN = XMIN: YMAX = XMAX
3120 If XMAX - XMIN < .000001 Then XMIN = XMIN - .0000005: XMAX = XMAX + .0000005
3130 If YMAX - YMIN < .000001 Then YMIN = YMIN - .0000005: YMAX = YMAX + .0000005
3160 MX = .1 * (XMAX - XMIN): MY = .1 * (YMAX - YMIN)
3170 XL = XMIN - MX: XH = XMAX + MX:  YL = YMIN - MY:  YH = YMAX + MY
```

```
3180 Scale (XL, YL)-(XH, YH): Cls

3460 Return


End Sub
```

QuickBASIC for Apple Macintosh Systems

If you want to run the programs on an Apple Macintosh, the easiest way is to use the Macintosh version of QuickBASIC. Unfortunately, this BASIC (at least in version 1.0) is not very compatible with any of the IBM BASICs. Also, it lacks many important and useful commands, although most of the missing features (such as color) are available through BASIC calls to the Macintosh Toolbox. An alternate though probably equally difficult approach is to convert the C source listing in Appendix D for use with one of the C compilers available for the Macintosh.

The QuickBASIC for Macintosh version of the programs typically executes more slowly than those compiled with the IBM version of QuickBASIC. For example, the program used to produce the data in Table 2-2 finds about 106 attractors per hour when compiled with the Macintosh version of QuickBASIC and run on a 25 MHz Macintosh IIci with a floating-point coprocessor and only 14 per hour when using the QuickBASIC interpreter on the same computer.

To get you started, the listing **PROG06QB.MAC** is a QuickBASIC for Macintosh version of **PROG06**. You can use it as a starting point for converting the other programs in this book for use on the Macintosh. You can use the Apple File Exchange utility to transfer **PROG06QB.MAC** on the accompanying disk to a Macintosh with a high density (1.4-MB) disk drive.

PROG06QB.MAC. Macintosh QuickBASIC version of PROG06

```
1000 REM TWO-D MAP SEARCH  Macintosh QuickBASIC Ver 1.0 (c) 1993 by J. C. Sprott

1010 DEFDBL A-Z                ‘Use double precision

1020 DIM XS(499), A(504), V(99)
```

```
1040 PREV% = 5                  ‘Plot versus fifth previous iterate
1050 NMAX = 11000               ‘Maximum number of iterations
1060 OMAX% = 2                  ‘Maximum order of polynomial
1070 D% = 2                     ‘Dimension of system
1100 SND% = 0                   ‘Turn sound off
1160 RANDOMIZE TIMER            ‘Reseed random-number generator
1190 GOSUB 1300                 ‘Initialize
1200 GOSUB 1500                 ‘Set parameters
1210 GOSUB 1700                 ‘Iterate equations
1220 GOSUB 2100                 ‘Display results
1230 GOSUB 2400                 ‘Test results
1240 ON T% GOTO 1190, 1200, 1210
1250 CLS
1260 END

1300 REM Initialize
1320 WINDOW 1, "Strange Attractors", (0, 36)-(SYSTEM(5), SYSTEM(6)), 1
1350 MENU 2, 0, 1, "Options": MENU 2, 1, SND% + 1, "Sound"
1360 WW = WINDOW(2) / 2: WH = WINDOW(3) / 2: CLS
1370 BUTTON 1, 1, "Searching...", (WW - 45, WH - 10) - (WW + 45, WH + 10)
1420 RETURN

1500 REM Set parameters
```

```
1510 X = .05                    'Initial condition
1520 Y = .05
1550 XE = X + .000001: YE = Y
1560 GOSUB 2600                 'Get coefficients
1570 T% = 3
1580 P% = 0: LSUM = 0: N = 0: NL = 0
1590 XMIN = 1000000!: XMAX = -XMIN: YMIN = XMIN: YMAX = XMAX
1630 RETURN

1700 REM Iterate equations
1720 XNEW = A(1) + X * (A(2) + A(3) * X + A(4) * Y)
1730 XNEW = XNEW + Y * (A(5) + A(6) * Y)
1830 YNEW = A(7) + X * (A(8) + A(9) * X + A(10) * Y)
1930 YNEW = YNEW + Y * (A(11) + A(12) * Y)
2020 N = N + 1
2030 RETURN

2100 REM Display results
2110 IF N < 100 OR N > 1000 THEN GOTO 2200
2120    IF X < XMIN THEN XMIN = X
2130    IF X > XMAX THEN XMAX = X
2140    IF Y < YMIN THEN YMIN = Y
2150    IF Y > YMAX THEN YMAX = Y
```

```
2200 IF N = 1000 THEN GOSUB 3100        ‘Resize the screen
2210 XS(P%) = X
2220 P% = (P% + 1) MOD 500
2230 I% = (P% + 500 - PREV%) MOD 500
2240 IF D% = 1 THEN XP = XS(I%): YP = XNEW ELSE XP = X: YP = Y
2250 IF N < 1000 OR XP <= XL OR XP >= XH OR YP <= YL OR YP >= YH THEN GOTO 2320
2300    PSET (WW * (XP - XL) / (XH - XL), WH * (YH - YP) / (YH - YL))
2310    IF SND% = 1 THEN GOSUB 3500     ‘Produce sound
2320 RETURN

2400 REM Test results
2410 IF ABS(XNEW) + ABS(YNEW) > 1000000! THEN T% = 2    ‘Unbounded
2430 GOSUB 2900                 ‘Calculate Lyapunov exponent
2460 IF N >= NMAX THEN T% = 2   ‘Strange attractor found
2470 IF ABS(XNEW - X) + ABS(YNEW - Y) < .000001 THEN T% = 2
2480 IF N > 100 AND L < .005 THEN T% = 2    ‘Limit cycle
2490 Q$ = INKEY$: IF LEN(Q$) THEN GOSUB 3600        ‘Respond to user command
2500 IF MENU(0) = 2 AND MENU(1) = 1 THEN Q$ = "S": GOSUB 3600
2510 X = XNEW                   ‘Update value of X
2520 Y = YNEW
2550 RETURN

2600 REM Get coefficients
```

```
2650 O% = 2 + INT((OMAX% - 1) * RND)
2660 CODE$ = CHR$(59 + 4 * D% + O%)
2680 M% = 1: FOR I% = 1 TO D%: M% = M% * (O% + I%): NEXT I%
2690    FOR I% = 1 TO M%        'Construct CODE$
2700        GOSUB 2800          'Shuffle random numbers
2710        CODE$ = CODE$ + CHR$(65 + INT(25 * RAN))
2720    NEXT I%
2730 FOR I% = 1 TO M%           'Convert CODE$ to coefficient values
2740    A(I%) = (ASC(MID$(CODE$, I% + 1, 1)) - 77) / 10
2750 NEXT I%
2760 RETURN

2800 REM Shuffle random numbers
2810 IF V(0) = 0 THEN FOR J% = 0 TO 99: V(J%) = RND: NEXT J%
2820 J% = INT(100 * RAN)
2830 RAN = V(J%)
2840 V(J%) = RND
2850 RETURN

2900 REM Calculate Lyapunov exponent
2910 XSAVE = XNEW: YSAVE = YNEW: X = XE: Y = YE: N = N - 1
2930 GOSUB 1700                 'Reiterate equations
2940 DLX = XNEW - XSAVE: DLY = YNEW - YSAVE
```

```
2960 DL2 = DLX * DLX + DLY * DLY
2970 IF CSNG(DL2) <= 0 THEN GOTO 3070       ‘Don’t divide by zero
2980    DF = 1000000000000# * DL2
2990    RS = 1 / SQR(DF)
3000    XE = XSAVE + RS * (XNEW - XSAVE): YE = YSAVE + RS * (YNEW - YSAVE)
3020    XNEW = XSAVE: YNEW = YSAVE
3030    IF DF > 0 THEN LSUM = LSUM + LOG(DF): NL = NL + 1
3040    L = .721347 * LSUM / NL
3070 RETURN

3100 REM Resize the screen
3110 IF D% = 1 THEN YMIN = XMIN: YMAX = XMAX
3120 IF XMAX - XMIN < .000001 THEN XMIN = XMIN - .0000005: XMAX = XMAX + .0000005
3130 IF YMAX - YMIN < .000001 THEN YMIN = YMIN - .0000005: YMAX = YMAX + .0000005
3160 MX = .1 * (XMAX - XMIN): MY = .1 * (YMAX - YMIN)
3170 XL = XMIN - MX: XH = XMAX + MX: YL = YMIN - MY: YH = YMAX + MY
3180 WW = WINDOW(2): WH = WINDOW(3): BUTTON CLOSE 0: CLS
3460 RETURN

3500 REM Produce sound
3510 FREQ% = 220 * 2 ^ (CINT(36 * (XNEW - XL) / (XH - XL)) / 12)
3520 DUR = 1
3540 SOUND FREQ%, DUR: IF PLAY(0) THEN PLAY "MF"
```

```
3550 RETURN


3600 REM Respond to user command
3610 T% = 0
3630 IF ASC(Q$) > 96 THEN Q$ = CHR$(ASC(Q$) - 32)
3770 IF Q$ = "S" THEN SND% = (SND% + 1) MOD 2: T% = 3: MENU 2, 1, SND% + 1, "Sound"
3800 RETURN
```

Appenix D

C Program Listing

This appendix contains a translation of the BASIC program **PROG28.BAS** (Appendix B) into the C language. The C language is preferred by many programmers because of its efficiency, economy, and portability. However, the language is relatively sparse and relies on machine-specific, run-time libraries for most input and output. Although there is a C standard (ANSI C), many necessary extensions are incorporated into various C compilers. These extensions also differ from one platform to another.

The listing here should compile and run without modification under Microsoft QuickC version 2.5 on the IBM PC-compatible platform. Some changes are required to run under other versions of C. The disk included with this book contains the Microsoft QuickC source listing in a file named **PROG28QC.C** and a version **PROG28TC.CPP** that should compile and run with Borland Turbo C++ version 3.0. The programs compile using the small memory model with either compiler. However, you will probably find that the C versions run at about the same speed as the Microsoft QuickBASIC or VisualBASIC for MS-DOS version, and somewhat slower than the PowerBASIC version.

The C versions of the program are fairly literal translations of the BASIC version. All variables are global and retain the same names as in the BASIC version. The BASIC subroutines have been converted into C functions whose names correspond to the BASIC line numbers. No variables are passed to or from any of the functions. The level of indentation is minimal. The program assumes the computer has VGA color graphics. **PROG28QC.C** should compile to a fully functional program, except for the fact that QuickC lacks a sound function.

PROG28QC.C. Microsoft QuickC version of PROG28.BAS

```
/* STRANGE ATTRACTOR PROGRAM  QuickC Ver 2.0 (c) 1993 by J. C. Sprott */


#include <dos.h>

#include <stdio.h>
```

```
#include <graph.h>
#include <math.h>

int PREV, OMAX, D, ODE, SND, PJT, TRD, FTH, SAV, T, WID, QM, P, TWOD;
int M, I, I1, I2, O, I3, I4, I5, J, WH, FREQ, C4, NC, C, RD, CY, BK;
int COLR[16];

char CODE[515], Q;
char FAV[9] = "XDATA.DAT";

double NMAX, EPS, TWOPI, SEG, NE, X, Y, Z, W, XE, YE, ZE, WE, LSUM, N, NL;
double N1, N2, XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX, WMIN, WMAX, XNEW, YNEW;
double ZNEW, WNEW, XP, YP, RAN, XSAVE, YSAVE, ZSAVE, WSAVE, DLX, DLY, DLZ;
double DLW, DL2, DF, RS, L, MX, MY, XL, XH, YL, YH, XA, YA, ZA, TT, PT, TIA;
double XZ, YZ, DUR, D2MAX, DX, DY, DZ, DW, D2, F, TH, PH, XRT, XLT, HSF, AL;
double SINAL, COSAL, DUM, SW, SH;
double XS[500], YS[500], ZS[500], WS[500], A[505], V[100], XY[5], XN[5];

union REGS regs;

FILE *F1, *F2, *F3;

main()
```

```
{
PREV = 5;                    /* Plot versus fifth previous iterate */
NMAX = 11000;                /* Maximum number of iterations */
OMAX = 5;                    /* Maximum order of polynomial */
D = 2;                       /* Dimension of system */
EPS = .1;                    /* Step size for ODE */
ODE = 0;                     /* System is map */
SND = 0;                     /* Turn sound off */
PJT = 0;                     /* Projection is planar */
TRD = 1;                     /* Display third dimension as shadow */
FTH = 2;                     /* Display fourth dimension as colors */
SAV = 0;                     /* Don't save any data */
TWOPI = 6.28318530717959;    /* A useful constant (2 pi) */
srand(time());               /* Reseed random-number generator */
fun4200();                   /* Display menu screen */
T = 1;
if (Q == 'X') T = 0;         /* Exit immediately on command */
while (T) {
    switch (T) {
        case 1: fun1300();   /* Initialize */
        case 2: fun1500();   /* Set parameters */
        case 3: fun1700();   /* Iterate equations */
        case 4: fun2100();   /* Display results */
```

```
        case 5: fun2400();  /* Test results */
    }
}
_clearscreen(_GCLEARSCREEN);    /* Erase screen */
_setvideomode(_DEFAULTMODE);    /* and restore video mode */
}

fun1300() /* Initialize */
{
_setvideomode(_VRES16COLOR);    /* Assume VGA graphics */
WID = 80;                   /* Number of text columns */
_clearscreen(_GCLEARSCREEN);
_settextposition(13, WID / 2 - 6);
printf("Searching...");
fun5600();                  /* Set colors */
if (QM == 2) {
    NE = 0;
    fclose(F1);
    F1 = fopen("SA.DIC", "a");
    fclose(F1);
    F1 = fopen("SA.DIC", "r");
}
}
```

```
fun1500() /* Set parameters */

{

X = .05;                    /* Initial condition */

Y = .05;

Z = .05;

W = .05;

XE = X + .000001;

YE = Y;

ZE = Z;

WE = W;

fun2600();                  /* Get coefficients */

T = 3;

P = 0;

LSUM = 0;

N = 0;

NL = 0;

N1 = 0;

N2 = 0;

XMIN = 1000000;

XMAX = -XMIN;

YMIN = XMIN;

YMAX = XMAX;
```

```
ZMIN = XMIN;

ZMAX = XMAX;

WMIN = XMIN;

WMAX = XMAX;

TWOD = _rotl(1, D);

}


fun1700() /* Iterate equations */

{

if (ODE > 1)

    fun6200();              /* Special function */

else {

    M = 1;

    XY[1] = X;

    XY[2] = Y;

    XY[3] = Z;

    XY[4] = W;

    for (I = 1; I <= D; I++) {

    XN[I] = A[M];

    M = M + 1;

    for (I1 = 1; I1 <= D; I1++) {

    XN[I] = XN[I] + A[M] * XY[I1];

    M = M + 1;
```

```
    for (I2 = I1; I2 <= D; I2++) {
    XN[I] = XN[I] + A[M] * XY[I1] * XY[I2];
    M = M + 1;
    for (I3 = I2; O > 2 && I3 <= D; I3++) {
    XN[I] = XN[I] + A[M] * XY[I1] * XY[I2] * XY[I3];
    M = M + 1;
    for (I4 = I3; O > 3 && I4 <= D; I4++) {
    XN[I] = XN[I] + A[M] * XY[I1] * XY[I2] * XY[I3] * XY[I4];
    M = M + 1;
    for (I5 = I4; O > 4 && I5 <= D; I5++) {
    XN[I] = XN[I] + A[M] * XY[I1] * XY[I2] * XY[I3] * XY[I4] * XY[I5];
    M = M + 1;
    }}}}}
        if (ODE == 1) XN[I] = XY[I] + EPS * XN[I];
    }
    XNEW = XN[1];
    YNEW = XN[2];
    ZNEW = XN[3];
    WNEW = XN[4];
}
N = N + 1;
M = M - 1;
}
```

```
fun2100() /* Display results */
{
if (N >= 100 && N <= 1000) {
    if (X < XMIN) XMIN = X;
    if (X > XMAX) XMAX = X;
    if (Y < YMIN) YMIN = Y;
    if (Y > YMAX) YMAX = Y;
    if (Z < ZMIN) ZMIN = Z;
    if (Z > ZMAX) ZMAX = Z;
    if (W < WMIN) WMIN = W;
    if (W > WMAX) WMAX = W;
}
if ((int)N == 1000) fun3100();  /* Resize the screen */
XS[P] = X;
YS[P] = Y;
ZS[P] = Z;
WS[P] = W;
P = (P + 1) % 500;
I = (P + 500 - PREV) % 500;
if (D == 1) {
    XP = XS[I];
    YP = XNEW;
```

```
}
else {
XP = X;
YP = Y;
}
if (N >= 1000 && XP > XL && XP < XH && YP > YL && YP < YH) {
    if (PJT == 1) fun4100();    /* Project onto a sphere */
    if (PJT == 2) fun6700();    /* Project onto a horizontal cylinder */
    if (PJT == 3) fun6800();    /* Project onto a vertical cylinder */
    if (PJT == 4) fun6900();    /* Project onto a torus */
    fun5000();                  /* Plot point on screen */
    if (SND == 1) fun3500();    /* Produce sound */
}
}

fun2400() /* Test results */
{
if (fabs(XNEW) + fabs(YNEW) + fabs(ZNEW) + fabs(WNEW) > 1000000) T = 2;
if (QM != 2) {              /* Speed up evaluation mode */
    fun2900();              /* Calculate Lyapunov exponent */
    fun3900();              /* Calculate fractal dimension */
    if (QM == 0) {          /* Skip tests unless in search mode */
        if (N >= NMAX) {    /* Strange attractor found */
```

```
            T = 2;

            fun4900();      /* Save attractor to disk file SA.DIC */

        }

        if (fabs(XNEW - X) + fabs(YNEW - Y) + fabs(ZNEW - Z) + fabs(WNEW - W) <
.000001) T = 2;

        if (N > 100 && L < .005) T = 2; /* Limit cycle */

    }

}

if (kbhit()) Q = getch(); else Q = 0;

if (Q) fun3600();           /* Respond to user command */

if (SAV > 0) if (N > 1000 && N < 17001) fun7000();  /* Save data */

X = XNEW;                   /* Update value of X */

Y = YNEW;

Z = ZNEW;

W = WNEW;

}



fun2600() /* Get coefficients */

{

if (QM == 2) {              /* In evaluate mode */

    fgets(CODE, 515, F1);

    if (feof(F1)) {

        QM = 0;

        fun6000();          /* Update SA.DIC file */
```

```
    }
    else {
        fun4700();          /* Get dimension and order */
        fun5600();          /* Set colors */
    }
}
if (QM == 0) {              /* In search mode */
    O = 2 + (int)floor((OMAX - 1) * (float)rand() / 32768.0);
    CODE[0] = 59 + 4 * D + O + 8 * ODE;
    if (ODE > 1) CODE[0] = 87 + ODE;
    fun4700();              /* Get value of M */
    for (I = 1; I <= M; I++) {  /* Construct CODE */
       fun2800();           /* Shuffle random numbers */
       CODE[I] = 65 + (int)floor(25 * RAN);
    }
    CODE[M + 1] = 0;
}
for (I = 1; I <= M; I++) {  /* Convert CODE to coefficient values */
    A[I] = (CODE[I] - 77) / 10.0;
}
}

fun2800() /* Shuffle random numbers */
```

```
{
if (V[0] == 0) for (J = 0; J <= 99; J++) {V[J] = (float)rand() / 32768.0;}
J = (int)floor(100 * RAN);
RAN = V[J];
V[J] = (float)rand() / 32768.0;
}


fun2900() /* Calculate Lyapunov exponent */
{
XSAVE = XNEW;
YSAVE = YNEW;
ZSAVE = ZNEW;
WSAVE = WNEW;
X = XE;
Y = YE;
Z = ZE;
W = WE;
N = N - 1;
fun1700();                  /* Reiterate equations */
DLX = XNEW - XSAVE;
DLY = YNEW - YSAVE;
DLZ = ZNEW - ZSAVE;
DLW = WNEW - WSAVE;
```

```
DL2 = DLX * DLX + DLY * DLY + DLZ * DLZ + DLW * DLW;
if (DL2 > 0) {              /* Check for division by zero */
    DF = 1E12 * DL2;
    RS = 1 / sqrt(DF);
    XE = XSAVE + RS * (XNEW - XSAVE);
    YE = YSAVE + RS * (YNEW - YSAVE);
    ZE = ZSAVE + RS * (ZNEW - ZSAVE);
    WE = WSAVE + RS * (WNEW - WSAVE);
    XNEW = XSAVE;
    YNEW = YSAVE;
    ZNEW = ZSAVE;
    WNEW = WSAVE;
    LSUM = LSUM + log(DF);
    NL = NL + 1;
    L = .721347 * LSUM / NL;
    if (ODE == 1 || ODE == 7) L = L / EPS;
    if (N > 1000 && (int)N % 10 == 0) {
        _settextposition(1, WID - 4);
        printf("%5.2f", L);
    }
}
}
```

```
fun3100() /* Resize the screen */
{
_setcolor(WH);
if (D == 1) {
    YMIN = XMIN;
    YMAX = XMAX;
}
if (XMAX - XMIN < .000001) {
    XMIN = XMIN - .0000005;
    XMAX = XMAX + .0000005;
}
if (YMAX - YMIN < .000001) {
    YMIN = YMIN - .0000005;
    YMAX = YMAX + .0000005;
}
if (ZMAX - ZMIN < .000001) {
    ZMIN = ZMIN - .0000005;
    ZMAX = ZMAX + .0000005;
}
if (WMAX - WMIN < .000001) {
    WMIN = WMIN - .0000005;
    WMAX = WMAX + .0000005;
}
```

```
MX = .1 * (XMAX - XMIN);
MY = .1 * (YMAX - YMIN);
XL = XMIN - MX;
XH = XMAX + MX;
YL = YMIN - MY;
YH = YMAX + 1.5 * MY;
SW = 640 / (XH - XL);
SH = 480 / (YL - YH);
_setvieworg((short)(-SW * XL), (short)(-SH * YH));
_clearscreen(_GCLEARSCREEN);
YH = YH - .5 * MY;
XA = (XL + XH) / 2;
YA = (YL + YH) / 2;
if (D > 2) {
    ZA = (ZMAX + ZMIN) / 2;
    if (TRD == 1) {
        _setcolor(COLR[1]);
        _rectangle_w(_GFILLINTERIOR, SW * XL, SH * YL, SW * XH, SH * YH);
        fun5400();          /* Plot background grid */
    }
    if (TRD == 4) {
        _setcolor(WH);
        _rectangle_w(_GFILLINTERIOR, SW * XL, SH * YL, SW * XH, SH * YH);
```

```
    }

    if (TRD == 5) {

        _moveto_w(SW * XA, SH * YL);

        _lineto_w(SW * XA, SH * YH);

    }

    if (TRD == 6) {

        for (I = 1; I <= 3; I++) {

            XP = XL + I * (XH - XL) / 4;

            _moveto_w(SW * XP, SH * YL);

            _lineto_w(SW * XP, SH * YH);

            YP = YL + I * (YH - YL) / 4;

            _moveto_w(SW * XL, SH * YP);

            _lineto_w(SW * XH, SH * YP);

        }

    }

}

if (PJT != 1) _rectangle_w(_GBORDER, SW * XL + 1, SH * YL - 1, SW * XH - 1, SH
* YH + 1);

if (PJT == 1 && TRD < 5) _ellipse_w(_GBORDER, SW * XL - SH * (YH - YL) / 6, SH
* YH, SW * XH + SH * (YH - YL) / 6, SH * YL);

TT = 3.1416 / (XMAX - XMIN);

PT = 3.1416 / (YMAX - YMIN);

if (QM == 2) {              /* In evaluate mode */

    _settextposition(1, 1);
```

```
    printf("<Space Bar>: Discard       <Enter>: Save");
    if (WID >= 80) {
        _settextposition(1, 49);
        printf("<Esc>: Exit");
        _settextposition(1, 69);
           printf("%d K left", (int)((filelength(fileno(F1)) - ftell(F1)) /
1024.0));
    }}
else {
    _settextposition(1, 1);
    if (strlen(CODE) < WID - 18)
        _outtext(CODE);
    else {
        printf("%*.*s...", WID - 23, WID - 23, CODE);
    }
    _settextposition(1, WID - 17);
    printf("F =");
    _settextposition(1, WID - 7);
    printf("L = ");
}
TIA = .05;                  /* Tangent of illumination angle */
XZ = -TIA * (XMAX - XMIN) / (ZMAX - ZMIN);
YZ = TIA * (YMAX - YMIN) / (ZMAX - ZMIN);
}
```

```
fun3500() /* Produce sound */
{
FREQ = 220 * pow(2, (int)(36 * (XNEW - XL) / (XH - XL)) / 12.0);
DUR = 1;
if (D > 1) DUR = pow(2, (int)floor(.5 * (YH - YL) / (YNEW - 9 * YL / 8 + YH / 8)));
/* A sound statement should be placed here */
}

fun3600() /* Respond to user command */
{
if (Q > 96) Q = Q - 32;     /* Convert to uppercase */
if (QM == 2) fun5800();     /* Process evaluation command */
if (strchr("ACDEHINPRSVX", Q) == 0) fun4200();  /* Display menu screen */
if (Q == ‘A’) {
    T = 1;
    QM = 0;
}
if (ODE > 1) D = ODE + 5;
if (ODE == 1) D = D + 2;
if (Q == ‘C’) if (N > 999) N = 999;
if (Q == ‘D’) {
    D = 1 + D % 12;
```

```
    T = 1;
}
if (D > 6) {
    ODE = D - 5;
    D = 4;
}
else {
    if (D > 4) {
        ODE = 1;
        D = D - 2;
    }
    else ODE = 0;
}
if (Q == ‘E’) {
    T = 1;
    QM = 2;
}
if (Q == ‘H’) {
    FTH = (FTH + 1) % 3;
    T = 3;
    if (N > 999) {
        N = 999;
        fun5600();          /* Set colors */
```

```
    }
}
if (Q == 'I') {
    if (T != 1) {
        _setvideomode(_TEXTC80);
        _settextcolor(15);
        _setbkcolor(1L);
        _clearscreen(_GCLEARSCREEN);
        printf("Code? ");
        I = 0;
        CODE[0] = 0;
        do {
            CODE[I] = getche();
            if (CODE[I] == 8 && I >= 0) I = I - 2;
            if (CODE[I] == 27) {
                I = 0;
                CODE[I] = 13;
            }
        }
        while (CODE[I++] != 13 && I < 506);
        CODE[I - 1] = 0;
        if (CODE[0] == 0) {
            Q = ' ';
```

```
            _clearscreen(_GCLEARSCREEN);}
        else {
            T = 1;
            QM = 1;
            fun4700();
        }
    }
}
if (Q == ‘N’) {
    NMAX = 10 * (NMAX - 1000) + 1000;
    if (NMAX > 1E10) NMAX = 2000;
}
if (Q == ‘P’) {
    PJT = (PJT + 1) % 5;
    T = 3;
    if (N > 999) N = 999;
}
if (Q == ‘R’) {
    TRD = (TRD + 1) % 7;
    T = 3;
    if (N > 999) {
        N = 999;
        fun5600();          /* Get dimension and order */
```

```
    }
}
if (Q == ‘S’) {
    SND = (SND + 1) % 2;
    T = 3;
}
if (Q == ‘V’) {
    SAV = (SAV + 1) % 5;
    FAV[0] = 87 + SAV % 4;
    T = 3;
    if (N > 999) N = 999;
}
if (Q == ‘X’) T = 0;
}

fun3900() /* Calculate fractal dimension */
{
if (N >= 1000) {            /* Wait for transient to settle */
    if ((int)N == 1000) {
        D2MAX = pow(XMAX - XMIN, 2);
        D2MAX = D2MAX + pow(YMAX - YMIN, 2);
        D2MAX = D2MAX + pow(ZMAX - ZMIN, 2);
        D2MAX = D2MAX + pow(WMAX - WMIN, 2);
```

```
    }
    J = (P + 1 + (int)floor(480 * (float)rand() / 32768.0)) % 500;
    DX = XNEW - XS[J];
    DY = YNEW - YS[J];
    DZ = ZNEW - ZS[J];
    DW = WNEW - WS[J];
    D2 = DX * DX + DY * DY + DZ * DZ + DW * DW;
    if (D2 < .001 * TWOD * D2MAX) N2 = N2 + 1;
    if (D2 <= .00001 * TWOD * D2MAX) {
        N1 = N1 + 1;
        F = .434294 * log(N2 / (N1 - .5));
        _settextposition(1, WID - 14);
        printf("%5.2f", F);
    }
}
}

fun4100() /* Project onto a sphere */
{
TH = TT * (XMAX - XP);
PH = PT * (YMAX - YP);
XP = XA + .36 * (XH - XL) * cos(TH) * sin(PH);
YP = YA + .5 * (YH - YL) * cos(PH);
```

```
}

fun4200() /* Display menu screen */
{
_setvideomode(_TEXTC80);
_settextcolor(15);
_setbkcolor(1L);
regs.h.ah = 1;
regs.h.ch = 1;
regs.h.cl = 0;
int86(16, &regs, &regs);    /* Turn cursor off */
_clearscreen(_GCLEARSCREEN);
while (Q == 0 || strchr("AEIX", Q) == 0) {
    _settextposition(1, 27);
    printf("STRANGE ATTRACTOR PROGRAM\n");
    printf("%26cIBM PC QuickC Version 2.0\n", ' ');
    printf("%26c(c) 1993 by J. C. Sprott\n", ' ');
    printf("\n");
    printf("\n");
    printf("%26cA: Search for attractors\n", ' ');
    printf("%26cC: Clear screen and restart\n", ' ');
    if (ODE > 1) {
        printf("%26cD: System is 4-D special map %c \n", ' ', 87 + ODE);}
```

```
    else {
        printf("%26cD: System is %d-D polynomial ", ' ', D);
        if (ODE == 1) printf("ODE\n"); else printf("map\n");
    }
    printf("%26cE: Evaluate attractors\n", ' ');
    printf("%26cH: Fourth dimension is ", ' ');
        if (FTH == 0) printf("projection\n");
        if (FTH == 1) printf("bands     \n");
        if (FTH == 2) printf("colors    \n");
    printf("%26cI: Input code from keyboard\n", ' ');
   printf("%26cN: Number of iterations is 10^%1.0f\n", ' ', log10(NMAX - 1000));
    printf("%26cP: Projection is ", ' ');
        if (PJT == 0) printf("planar   \n");
        if (PJT == 1) printf("spherical\n");
        if (PJT == 2) printf("horiz cyl\n");
        if (PJT == 3) printf("vert cyl \n");
        if (PJT == 4) printf("toroidal \n");
    printf("%26cR: Third dimension is ", ' ');
        if (TRD == 0) printf("projection\n");
        if (TRD == 1) printf("shadow    \n");
        if (TRD == 2) printf("bands     \n");
        if (TRD == 3) printf("colors    \n");
        if (TRD == 4) printf("anaglyph  \n");
```

```
        if (TRD == 5) printf("stereogram\n");
        if (TRD == 6) printf("slices    \n");
    printf("%26cS: Sound is ", ‘ ‘);
        if (SND == 0) printf("off\n");
        if (SND == 1) printf("on \n");
    printf("%26cV: ", ‘ ‘);
        if (SAV == 0) printf("No data will be saved       \n");
       if (SAV > 0) printf("%c will be saved in %cDATA.DAT\n", FAV[0], FAV[0]);
    printf("%26cX: Exit program", ‘ ‘);
    if (kbhit()) Q = getch(); else Q = 0;
    if (Q) fun3600();       /* Respond to user command */
}
}


fun4700() /* Get dimension and order */
{
D = 1 + (int)floor((CODE[0] - 65) / 4);
if (D > 6) {
    ODE = CODE[0] - 87;
    D = 4;
    fun6200();              /* Special function */
}
else {
```

```
    if (D > 4) {
        D = D - 2;
        ODE = 1;
    }
    else ODE = 0;
    O = 2 + (CODE[0] - 65) % 4;
    M = 1;
    for (I = 1; I <= D; I++) {M = M * (O + I);}
    if (D > 2) for (I = 3; I <= D; I++) {M = M / (I - 1);}
}
if (strlen(CODE) != M + 1 && QM == 1) {
   printf("\a");            /* Illegal code warning */
   while (strlen(CODE) < M + 1) strcat(CODE, "M");
   if (strlen(CODE) > M + 1) CODE[M + 1] = 0;
}
}

fun4900() /* Save attractor to disk file SA.DIC */
{
F1 = fopen("SA.DIC", "a");
fprintf(F1, "%s%5.2f%5.2f\n", CODE, F, L);
fclose(F1);
}
```

```
fun5000() /* Plot point on screen */
{
C4 = WH;
if (D > 3) {
   if (FTH == 1) if ((int)floor(30 * (W - WMIN) / (WMAX - WMIN)) % 2) return(0);
   if (FTH == 2) C4 = 1 + (int)floor(NC * (W - WMIN) / (WMAX - WMIN) + NC) % NC;
}
if (D < 3) {                /* Skip 3-D stuff */
    _setpixel_w(SW * XP, SH * YP);
    return(0);
}
if (TRD == 0) {
    _setcolor(C4);
    _setpixel_w(SW * XP, SH * YP);
}
if (TRD == 1) {
    if (D > 3 && FTH == 2) {
        _setcolor(C4);
        _setpixel_w(SW * XP, SH * YP);
    }
    else {
        C = _getpixel_w(SW * XP, SH * YP);
```

```
        if (C == COLR[2]) {
            _setcolor(COLR[3]);
            _setpixel_w(SW * XP, SH * YP);}
        else {
            if (C != COLR[3]) {
                _setcolor(COLR[2]);
                _setpixel_w(SW * XP, SH * YP);
            }
        }
    }
    XP = XP - XZ * (Z - ZMIN);
    YP = YP - YZ * (Z - ZMIN);
    if (_getpixel_w(SW * XP, SH * YP) == COLR[1]) {
        _setcolor(0);
        _setpixel_w(SW * XP, SH * YP);
    }
}
if (TRD == 2) {
    if (D > 3 && FTH == 2 && ((int)floor(15 * (Z - ZMIN) / (ZMAX - ZMIN) + 2) %
2) == 1) {
        _setcolor(C4);}
    else {
        C = COLR[(int)floor(60 * (Z - ZMIN) / (ZMAX - ZMIN) + 4) % 4];
        _setcolor(C);
```

```
    }
    _setpixel_w(SW * XP, SH * YP);
}
if (TRD == 3) {
    _setcolor(COLR[(int)floor(NC * (Z - ZMIN) / (ZMAX - ZMIN) + NC) % NC]);
    _setpixel_w(SW * XP, SH * YP);
}
if (TRD == 4) {
    XRT = XP + XZ * (Z - ZA);
    C = _getpixel_w(SW * XRT, SH * YP);
    if (C == WH) {
        _setcolor(RD);
        _setpixel_w(SW * XRT, SH * YP);
    }
    if (C == CY) {
        _setcolor(BK);
        _setpixel_w(SW * XRT, SH * YP);
    }
    XLT = XP - XZ * (Z - ZA);
    C = _getpixel_w(SW * XLT, SH * YP);
    if (C == WH) {
        _setcolor(CY);
        _setpixel_w(SW * XLT, SH * YP);
```

```
    }
    if (C == RD) {
        _setcolor(BK);
        _setpixel_w(SW * XLT, SH * YP);
    }
}
if (TRD == 5) {
    HSF = 2;                /* Horizontal scale factor */
    XRT = XA + (XP + XZ * (Z - ZA) - XL) / HSF;
    _setcolor(C4);
    _setpixel_w(SW * XRT, SH * YP);
    XLT = XA + (XP - XZ * (Z - ZA) - XH) / HSF;
    _setcolor(C4);
    _setpixel_w(SW * XLT, SH * YP);
}
if (TRD == 6) {
   DZ = (15 * (Z - ZMIN) / (ZMAX - ZMIN) + .5) / 16;
   XP = (XP - XL + ((int)floor(16 * DZ) % 4) * (XH - XL)) / 4 + XL;
   YP = (YP - YL + (3 - (int)floor(4 * DZ) % 4) * (YH - YL)) / 4 + YL;
   _setcolor(C4);
   _setpixel_w(SW * XP, SH * YP);
}
}
```

```
fun5400() /* Plot background grid */
{
_setcolor(0);
for (I = 0; I <= 15; I++) { /* Draw 15 vertical grid lines */
    XP = XMIN + I * (XMAX - XMIN) / 15;
    _moveto_w(SW * XP, SH * YMIN);
    _lineto_w(SW * XP, SH * YMAX);
}
for (I = 0; I <= 10; I++) { /* Draw 10 horizontal grid lines */
    YP = YMIN + I * (YMAX - YMIN) / 10;
    _moveto_w(SW * XMIN, SH * YP);
    _lineto_w(SW * XMAX, SH * YP);
}
}

fun5600() /* Set colors */
{
NC = 15;                    /* Number of colors */
COLR[0] = 0;
COLR[1] = 8;
COLR[2] = 7;
COLR[3] = 15;
```

```
if (TRD == 3 || (D > 3 && FTH == 2 && TRD != 1)) {
    for (I = 0; I <= NC; I++) COLR[I] = I + 1;
}
WH = 15;
BK = 8;
RD = 12;
CY = 11;
}

fun5800() /* Process evaluation command */
{
if (Q == ' ') {
    T = 2;
    NE = NE + 1;
    _clearscreen(_GCLEARSCREEN);
}
if (Q == 13) {
    T = 2;
    NE = NE + 1;
    _clearscreen(_GCLEARSCREEN);
    fun5900();              /* Save favorite attractors to disk */
}
if (Q == 27) {
```

```
    _clearscreen(_GCLEARSCREEN);
    fun6000();              /* Update SA.DIC file */
    Q = ‘ ‘;
    QM = 0;
}
else {
    if (strchr("CHNPRVS", Q) == 0) Q = 0;
}
}

fun5900() /* Save favorite attractors to disk file FAVORITE.DIC */
{
F2 = fopen("FAVORITE.DIC", "a");
fprintf(F2, CODE);
fclose(F2);
}

fun6000() /* Update SA.DIC file */
{
_settextposition(11, 9);
printf("Evaluation complete\n");
_settextposition(12, 8);
printf(" %d cases evaluated", (int)NE);
```

```
F2 = fopen("SATEMP.DIC", "w");
if (QM == 2) fprintf(F2, CODE);
while (feof(F1) == 0) {
    fgets(CODE, 515, F1);
    if (feof(F1) == 0) fprintf(F2, CODE);
}
fcloseall();
remove("SA.DIC");
rename("SATEMP.DIC", "SA.DIC");
}

fun6200() /* Special function definitions */
{
ZNEW = X * X + Y * Y;       /* Default 3rd and 4th dimension */
WNEW = (N - 100) / 900;
if (N > 1000) WNEW = (N - 1000) / (NMAX - 1000);
if (ODE == 2) {
    M = 10;
    XNEW = A[1] + A[2] * X + A[3] * Y + A[4] * fabs(X) + A[5] * fabs(Y);
    YNEW = A[6] + A[7] * X + A[8] * Y + A[9] * fabs(X) + A[10] * fabs(Y);
}
if (ODE == 3) {
    M = 14;
```

```
    XNEW = A[1] + A[2] * X + A[3] * Y + ((int)(A[4] * X + .5) & (int)(A[5] * Y
+ .5)) + ((int)(A[6] * X + .5) | (int)(A[7] * Y + .5));

    YNEW = A[8] + A[9] * X + A[10] * Y + ((int)(A[11] * X + .5) & (int)(A[12] *
Y + .5)) + ((int)(A[13] * X + .5) | (int)(A[14] * Y + .5));

}

if (ODE == 4) {

    M = 14;

  XNEW = A[1] + A[2] * X + A[3] * Y + A[4] * pow(fabs(X), A[5]) + A[6] * pow(fabs(Y),
A[7]);

    YNEW = A[8] + A[9] * X + A[10] * Y + A[11] * pow(fabs(X), A[12]) + A[13] *
pow(fabs(Y), A[14]);

}

if (ODE == 5) {

    M = 18;

   XNEW = A[1] + A[2] * X + A[3] * Y + A[4] * sin(A[5] * X + A[6]) + A[7] * sin(A[8]
* Y + A[9]);

    YNEW = A[10] + A[11] * X + A[12] * Y + A[13] * sin(A[14] * X + A[15]) + A[16]
* sin(A[17] * Y + A[18]);

}

if (ODE == 6) {

    M = 6;

    if (N < 2) {

        AL = TWOPI / (13 + 10 * A[6]);

        SINAL = sin(AL);

        COSAL = cos(AL);

    }

    DUM = X + A[2] * sin(A[3] * Y + A[4]);
```

```
    XNEW = 10 * A[1] + DUM * COSAL + Y * SINAL;

    YNEW = 10 * A[5] - DUM * SINAL + Y * COSAL;

}

if (ODE == 7) {

    M = 9;

    XNEW = X + EPS * A[1] * Y;

    YNEW = Y + EPS * (A[2] * X + A[3] * X * X * X + A[4] * X * X * Y + A[5] * X
* Y * Y + A[6] * Y + A[7] * Y * Y * Y + A[8] * sin(Z));

    ZNEW = Z + EPS * (A[9] + 1.3);

    if (ZNEW > TWOPI) ZNEW = ZNEW - TWOPI;

}

}



fun6700() /* Project onto a horizontal cylinder */

{

PH = PT * (YMAX - YP);

YP = YA + .5 * (YH - YL) * cos(PH);

}



fun6800() /* Project onto a vertical cylinder */

{

TH = TT * (XMAX - XP);

XP = XA + .5 * (XH - XL) * cos(TH);

}
```

```
fun6900() /* Project onto a torus (unity aspect ratio) */

{

TH = TT * (XMAX - XP);

PH = 2 * PT * (YMAX - YP);

XP = XA + .18 * (XH - XL) * (1 + cos(TH)) * sin(PH);

YP = YA + .25 * (YH - YL) * (1 + cos(TH)) * cos(PH);

}

fun7000() /* Save data */

{

if ((int)N == 1000) {

    fclose(F3);

    F3 = fopen(FAV, "w");

}

if (SAV == 1) DUM = XNEW;

if (SAV == 2) DUM = YNEW;

if (SAV == 3) DUM = ZNEW;

if (SAV == 4) DUM = WNEW;

fprintf(F3, "%f\n", DUM);

}
```

Appendix E

Summary of Equations

This appendix contains a complete list, in all its gory detail, of the equations solved by the program to produce the attractors in this book. For simplicity, the subscripts *n+1* and *n* have been omitted on the variables *X*, *Y*, *Z*, and *W*. If it serves no other purpose, this appendix vividly illustrates the power of programming languages in expressing and evaluating lengthy formulas! It is worth emphasizing that the attractors that come from simple equations are every bit as interesting and beautiful as those that come from complicated equations.

Case A: D = **1**, O = **2**, M = 3

$$X = a_1 + a_2X + a_3X^2$$

Case B: D = **1**, O = **3**, M = **4**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3$$

Case C: D = **1**, O = **4**, M = **5**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^4$$

Case D: D = **1**, O = **5**, M = **6**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^4 + a_6X^5$$

Case E: D = **2**, O = **2**, M = **12**

$$X = a_1 + a_2X + a_3X^2 + a_4XY + a_5Y + a_6Y^2$$

$$Y = a_7 + a_8X + a_9X^2 + a_{10}XY + a_{11}Y + a_{12}Y^2$$

Case F: D = **2**, O = **3**, M = **20**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^2Y + a_6XY + a_7XY^2 + a_8Y + a_9Y^2 + a_{10}Y^3$$

$$Y = a_{11} + a_{12}X + a_{13}X^2 + a_{14}X^3 + a_{15}X^2Y + a_{16}XY + a_{17}XY^2 + a_{18}Y + a_{19}Y^2 + a_{20}Y^3$$

Case G: D = **2**, O = **4**, M = **30**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^4 + a_6X^3Y + a_7X^2Y + a_8X^2Y^2 + a_9XY + a_{10}XY^2 + a_{11}XY^3 + a_{12}Y + a_{13}Y^2 + a_{14}Y^3 + a_{15}Y^4$$

$$Y = a_{16} + a_{17}X + a_{18}X^2 + a_{19}X^3 + a_{20}X^4 + a_{21}X^3Y + a_{22}X^2Y + a_{23}X^2Y^2 + a_{24}XY + a_{25}XY^2 + a_{26}XY^3 + a_{27}Y + a_{28}Y^2 + a_{29}Y^3 + a_{30}Y^4$$

Case H: D = **2**, O = **5**, M = **42**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^4 + a_6X^5 + a_7X^4Y + a_8X^3Y + a_9X^3Y^2 + a_{10}X^2Y + a_{11}X^2Y^2 + a_{12}X^2Y^3 + a_{13}XY + a_{14}XY^2 + a_{15}XY^3 + a_{16}XY^4 + a_{17}Y + a_{18}Y^2 + a_{19}Y^3 + a_{20}Y^4 + a_{21}Y^5$$

$$Y = a_{22} + a_{23}X + a_{24}X^2 + a_{25}X^3 + a_{26}X^4 + a_{27}X^5 + a_{28}X^4Y + a_{29}X^3Y + a_{30}X^3Y^2 + a_{31}X^2Y + a_{32}X^2Y^2 + a_{33}X^2Y^3 + a_{34}XY + a_{35}XY^2 + a_{36}XY^3 + a_{37}XY^4 + a_{38}Y + a_{39}Y^2 + a_{40}Y^3 + a_{41}Y^4 + a_{42}Y^5$$

Case I: D = **3**, O = **2**, M = **30**

$$X = a_1 + a_2X + a_3X^2 + a_4XY + a_5XZ + a_6Y + a_7Y^2 + a_8YZ + a_9Z + a_{10}Z^2$$

$$Y = a_{11} + a_{12}X + a_{13}X^2 + a_{14}XY + a_{15}XZ + a_{16}Y + a_{17}Y^2 + a_{18}YZ + a_{19}Z + a_{20}Z^2$$

$$Z = a_{21} + a_{22}X + a_{23}X^2 + a_{24}XY + a_{25}XZ + a_{26}Y + a_{27}Y^2 + a_{28}YZ + a_{29}Z + a_{30}Z^2$$

Case J: D = **3**, O = **3**, M = **60**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^2Y + a_6X^2Z + a_7XY + a_8XY^2 + a_9XYZ + a_{10}XZ + a_{11}XZ^2 + a_{12}Y + a_{13}Y^2 + a_{14}Y^3 + a_{15}Y^2Z + a_{16}YZ + a_{17}YZ^2 + a_{18}Z + a_{19}Z^2 + a_{20}Z^3$$

$$Y = a_{21} + a_{22}X + a_{23}X^2 + a_{24}X^3 + a_{25}X^2Y + a_{26}X^2Z + a_{27}XY + a_{28}XY^2 + a_{29}XYZ + a_{30}XZ + a_{31}XZ^2 + a_{32}Y + a_{33}Y^2 + a_{34}Y^3 + a_{35}Y^2Z + a_{36}YZ + a_{37}YZ^2 + a_{38}Z + a_{39}Z^2 + a_{40}Z^3$$

$$Z = a_{41} + a_{42}X + a_{43}X^2 + a_{44}X^3 + a_{45}X^2Y + a_{46}X^2Z + a_{47}XY + a_{48}XY^2 + a_{49}XYZ + a_{50}XZ + a_{51}XZ^2 + a_{52}Y + a_{53}Y^2 + a_{54}Y^3 + a_{55}Y^2Z + a_{56}YZ + a_{57}YZ^2 + a_{58}Z + a_{59}Z^2 + a_{60}Z^3$$

Case K: D = **3**, O = **4**, M = **105**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^4 + a_6X^3Y + a_7X^3Z + a_8X^2Y + a_9X^2Y^2 + a_{10}X^2YZ + a_{11}X^2Z + a_{12}X^2Z^2 + a_{13}XY + a_{14}XY^2 + a_{15}XY^3 + a_{16}XY^2Z + a_{17}XYZ + a_{18}XYZ^2 + a_{19}XZ + a_{20}XZ^2 + a_{21}XZ^3 + a_{22}Y + a_{23}Y^2 + a_{24}Y^3 + a_{25}Y^4 + a_{26}Y^3Z + a_{27}Y^2Z + a_{28}Y^2Z^2 + a_{29}YZ + a_{30}YZ^2 + a_{31}YZ^3 + a_{32}Z + a_{33}Z^2 + a_{34}Z^3 + a_{35}Z^4$$

$$Y = a_{36} + a_{37}X + a_{38}X^2 + a_{39}X^3 + a_{40}X^4 + a_{41}X^3Y + a_{42}X^3Z + a_{43}X^2Y + a_{44}X^2Y^2 + a_{45}X^2YZ + a_{46}X^2Z + a_{47}X^2Z^2 + a_{48}XY + a_{49}XY^2 + a_{50}XY^3 + a_{51}XY^2Z + a_{52}XYZ + a_{53}XYZ^2 + a_{54}XZ + a_{55}XZ^2 + a_{56}XZ^3 + a_{57}Y + a_{58}Y^2 + a_{59}Y^3 + a_{60}Y^4 + a_{61}Y^3Z + a_{62}Y^2Z + a_{63}Y^2Z^2 + a_{64}YZ + a_{65}YZ^2 + a_{66}YZ^3 + a_{67}Z + a_{68}Z^2 + a_{69}Z^3 + a_{70}Z^4$$

$$Z = a_{71} + a_{72}X + a_{73}X^2 + a_{74}X^3 + a_{75}X^4 + a_{76}X^3Y + a_{77}X^3Z + a_{78}X^2Y + a_{79}X^2Y^2 + a_{80}X^2YZ + a_{81}X^2Z + a_{82}X^2Z^2 + a_{83}XY + a_{84}XY^2 + a_{85}XY^3 + a_{86}XY_2Z + a_{87}XYZ + a_{88}XYZ^2 + a_{89}XZ + a_{90}XZ^2 + a_{91}XZ^3 + a_{92}Y + a_{93}Y^2 + a_{94}Y^3 + a_{95}Y^4 + a_{96}Y^3Z + a_{97}Y^2Z + a_{98}Y^2Z^2 + a_{99}YZ + a_{100}YZ^2 + a_{101}YZ^3 + a_{102}Z + a_{103}Z^2 + a_{104}Z^3 + a_{105}Z^4$$

Case L: D = **3**, O = **5**, M = **168**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^4 + a_6X^5 + a_7X^4Y + a_8X^4Z + a_9X^3Y + a_{10}X^3Y^2 + a_{11}X^3YZ + a_{12}X^3Z + a_{13}X^3Z^2 + a_{14}X^2Y + a_{15}X^2Y^2 + a_{16}X^2Y^3 + a_{17}X^2Y^2Z + a_{18}X^2YZ + a_{19}X^2YZ^2 + a_{20}X^2Z + a_{21}X^2Z^2 + a_{22}X^2Z^3 + a_{23}XY + a_{24}XY^2 + a_{25}XY^3 + a_{26}XY^4 + a_{27}XY^3Z + a_{28}XY^2Z + a_{29}XY^2Z^2 + a_{30}XYZ + a_{31}XYZ^2 + a_{32}XYZ^3 + a_{33}XZ + a_{34}XZ^2 + a_{35}XZ^3 + a_{36}XZ^4 + a_{37}Y + a_{38}Y^2 + a_{39}Y^3 + a_{40}Y^4 + a_{41}Y^5 + a_{42}Y^4Z + a_{43}Y^3Z + a_{44}Y^3Z^2 + a_{45}Y^2Z + a_{46}Y^2Z^2 + a_{47}Y^2Z^3 + a_{48}YZ + a_{49}YZ^2 + a_{50}YZ^3 + a_{51}YZ^4 + a_{52}Z + a_{53}Z^2 + a_{54}Z^3 + a_{55}Z^4 + a_{56}Z^5$$

$$Y = a_{57} + a_{58}X + a_{59}X^2 + a_{60}X^3 + a_{61}X^4 + a_{62}X^5 + a_{63}X^4Y + a_{64}X^4Z + a_{65}X^3Y + a_{66}X^3Y^2 + a_{67}X^3YZ + a_{68}X^3Z + a_{69}X^3Z^2 + a_{70}X^2Y + a_{71}X^2Y^2 + a_{72}X^2Y^3 + a_{73}X^2Y^2Z + a_{74}X^2YZ + a_{75}X^2YZ^2 + a_{76}X^2Z + a_{77}X^2Z^2 + a_{78}X^2Z^3 + a_{79}XY + a_{80}XY^2 + a_{81}XY^3 + a_{82}XY^4 + a_{83}XY^3Z + a_{84}XY^2Z + a_{85}XY^2Z^2 + a_{86}XYZ + a_{87}XYZ^2 + a_{88}XYZ^3 + a_{89}XZ + a_{90}XZ^2 + a_{91}XZ^3 + a_{92}XZ^4 + a_{93}Y + a_{94}Y^2 + a_{95}Y^3 + a_{96}Y^4 + a_{97}Y^5 + a_{98}Y^4Z + a_{99}Y^3Z + a_{100}Y^3Z^2 + a_{101}Y^2Z + a_{102}Y^2Z^2 + a_{103}Y^2Z^3 + a_{104}YZ + a_{105}YZ^2 + a_{106}YZ^3 + a_{107}YZ^4 + a_{108}Z + a_{109}Z^2 + a_{110}Z^3 + a_{111}Z^4 + a_{112}Z^5$$

$$Z = a_{113} + a_{114}X + a_{115}X^2 + a_{116}X^3 + a_{117}X^4 + a_{118}X^5 + a_{119}X^4Y + a_{120}X^4Z + a_{121}X^3Y + a_{122}X^3Y^2 + a_{123}X^3YZ + a_{124}X^3Z + a_{125}X^3Z^2 + a_{126}X^2Y + a_{127}X^2Y^2 + a_{128}X^2Y^3 + a_{129}X^2Y^2Z + a_{130}X^2YZ + a_{131}X^2YZ^2 + a_{132}X^2Z + a_{133}X^2Z^2 + a_{134}X^2Z^3 + a_{135}XY + a_{136}XY^2 + a_{137}XY^3 + a_{138}XY^4 + a_{139}XY^3Z + a_{140}XY^2Z + a_{141}XY^2Z^2 + a_{142}XYZ + a_{143}XYZ^2 + a_{144}XYZ^3 + a_{145}XZ + a_{146}XZ^2 + a_{147}XZ^3 + a_{148}XZ^4 + a_{149}Y + a_{150}Y^2 + a_{151}Y^3 + a_{152}Y^4 + a_{153}Y^5 + a_{154}Y^4Z + a_{155}Y^3Z + a_{156}Y^3Z^2 + a_{157}Y^2Z + a_{158}Y^2Z^2 + a_{159}Y^2Z^3 + a_{160}YZ + a_{161}YZ^2 + a_{162}YZ^3 + a_{163}YZ^4 + a_{164}Z + a_{165}Z^2 + a_{166}Z^3 + a_{167}Z^4 + a_{168}Z^5$$

Case M: D = **4**, O = **2**, M = **60**

$$X = a_1 + a_2X + a_3X^2 + a_4XY + a_5XZ + a_6XW + a_7Y + a_8Y^2 + a_9YZ + a_{10}YW + a_{11}Z + a_{12}Z^2 + a_{13}ZW + a_{14}W + a_{15}W^2$$

$$Y = a_{16} + a_{17}X + a_{18}X^2 + a_{19}XY + a_{20}XZ + a_{21}XW + a_{22}Y + a_{23}Y^2 + a_{24}YZ + a_{25}YW + a_{26}Z + a_{27}Z^2 + a_{28}ZW + a_{29}W + a_{30}W^2$$

$$Z = a_{31} + a_{32}X + a_{33}X^2 + a_{34}XY + a_{35}XZ + a_{36}XW + a_{37}Y + a_{38}Y^2 + a_{39}YZ + a_{40}YW + a_{41}Z + a_{42}Z^2 + a_{43}ZW + a_{44}W + a_{45}W^2$$

$$W = a_{46} + a_{47}X + a_{48}X^2 + a_{49}XY + a_{50}XZ + a_{51}XW + a_{52}Y + a_{53}Y^2 + a_{54}YZ + a_{55}YW + a_{56}Z + a_{57}Z^2 + a_{58}ZW + a_{59}W + a_{60}W^2$$

Case N: D = **4**, O = **3**, M = **140**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^2Y + a_6X^2Z + a_7X^2W + a_8XY + a_9XY^2 + a_{10}XYZ + a_{11}XYW + a_{12}XZ + a_{13}XZ^2 + a_{14}XZW + a_{15}XW + a_{16}XW^2 + a_{17}Y + a_{18}Y^2 + a_{19}Y^3 + a_{20}Y^2Z + a_{21}Y^2W + a_{22}YZ + a_{23}YZ^2 + a_{24}YZW + a_{25}YW + a_{26}YW^2 + a_{27}Z + a_{28}Z^2 + a_{29}Z^3 + a_{30}Z^2W + a_{31}ZW + a_{32}ZW^2 + a_{33}W + a_{34}W^2 + a_{35}W^3$$

$$Y = a_{36} + a_{37}X + a_{38}X^2 + a_{39}X^3 + a_{40}X^2Y + a_{41}X^2Z + a_{42}X^2W + a_{43}XY + a_{44}XY^2 + a_{45}XYZ + a_{46}XYW + a_{47}XZ + a_{48}XZ^2 + a_{49}XZW + a_{50}XW + a_{51}XW^2 + a_{52}Y + a_{53}Y^2 + a_{54}Y^3 + a_{55}Y^2Z + a_{56}Y^2W + a_{57}YZ + a_{58}YZ^2 + a_{59}YZW + a_{60}YW + a_{61}YW^2 + a_{62}Z + a_{63}Z^2 + a_{64}Z^3 + a_{65}Z^2W + a_{66}ZW + a_{67}ZW^2 + a_{68}W + a_{69}W^2 + a_{70}W^3$$

$$Z = a_{71} + a_{72}X + a_{73}X^2 + a_{74}X^3 + a_{75}X^2Y + a_{76}X^2Z + a_{77}X^2W + a_{78}XY + a_{79}XY^2 + a_{80}XYZ + a_{81}XYW + a_{82}XZ + a_{83}XZ^2 + a_{84}XZW + a_{85}XW + a_{86}XW^2 + a_{87}Y + a_{88}Y^2 + a_{89}Y^3 + a_{90}Y^2Z + a_{91}Y^2W + a_{92}YZ + a_{93}YZ^2 + a_{94}YZW + a_{95}YW + a_{96}YW^2 + a_{97}Z + a_{98}Z^2 + a_{99}Z^3 + a_{100}Z^2W + a_{101}ZW + a_{102}ZW^2 + a_{103}W + a_{104}W^2 + a_{105}W^3$$

$$W = a_{106} + a_{107}X + a_{108}X^2 + a_{109}X^3 + a_{110}X^2Y + a_{111}X^2Z + a_{112}X^2W + a_{113}XY + a_{114}XY^2 + a_{115}XYZ + a_{116}XYW + a_{117}XZ + a_{118}XZ^2 + a_{119}XZW + a_{120}XW + a_{121}XW^2 + a_{122}Y + a_{123}Y^2 + a_{124}Y^3 + a_{125}Y^2Z + a_{126}Y^2W + a_{127}YZ + a_{128}YZ^2 + a_{129}YZW + a_{130}YW + a_{131}YW^2 + a_{132}Z + a_{133}Z^2 + a_{134}Z^3 + a_{135}Z^2W + a_{136}ZW + a_{137}ZW^2 + a_{138}W + a_{139}W^2 + a_{140}W^3$$

Case O: D = **4**, O = **4**, M = **280**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^4 + a_6X^3Y + a_7X^3Z + a_8X^3W + a_9X^2Y + a_{10}X^2Y^2 + a_{11}X^2YZ + a_{12}X^2YW + a_{13}X^2Z + a_{14}X^2Z^2 + a_{15}X^2ZW + a_{16}X^2W + a_{17}X^2W^2 + a_{18}XY + a_{19}XY^2 + a_{20}XY^3 + a_{21}XY^2Z + a_{22}XY^2W + a_{23}XYZ + a_{24}XYZ^2 + a_{25}XYZW + a_{26}XYW + a_{27}XYW^2 + a_{28}XZ + a_{29}XZ^2 + a_{30}XZ^3 + a_{31}XZ^2W + a_{32}XZW + a_{33}XZW^2 + a_{34}XW + a_{35}XW^2 + a_{36}XW^3 + a_{37}Y + a_{38}Y^2 + a_{39}Y^3 + a_{40}Y^4 + a_{41}Y^3Z + a_{42}Y^3W + a_{43}Y^2Z + a_{44}Y^2Z^2 + a_{45}Y^2ZW + a_{46}Y^2W + a_{47}Y^2W^2 + a_{48}YZ + a_{49}YZ^2 + a_{50}YZ^3 + a_{51}YZ^2W + a_{52}YZW + a_{53}YZW^2 + a_{54}YW + a_{55}YW^2 + a_{56}YW^3 + a_{57}Z + a_{58}Z^2 + a_{59}Z^3 + a_{60}Z^4 + a_{61}Z^3W + a_{62}Z^2W + a_{63}Z^2W^2 + a_{64}ZW + a_{65}ZW^2 + a_{66}ZW^3 + a_{67}W + a_{68}W^2 + a_{69}W^3 + a_{70}W^4$$

$$Y = a_{71} + a_{72}X + a_{73}X^2 + a_{74}X^3 + a_{75}X^4 + a_{76}X^3Y + a_{77}X^3Z + a_{78}X^3W + a_{79}X^2Y + a_{80}X^2Y^2 + a_{81}X^2YZ + a_{82}X^2YW + a_{83}X^2Z + a_{84}X^2Z^2 + a_{85}X^2ZW + a_{86}X^2W + a_{87}X^2W^2 + a_{88}XY + a_{89}XY^2 + a_{90}XY^3 + a_{91}XY^2Z + a_{92}XY^2W + a_{93}XYZ + a_{94}XYZ^2 + a_{95}XYZW + a_{96}XYW + a_{97}XYW^2 + a_{98}XZ + a_{99}XZ^2 + a_{100}XZ^3 + a_{101}XZ^2W + a_{102}XZW + a_{103}XZW^2 + a_{104}XW + a_{105}XW^2 + a_{106}XW^3 + a_{107}Y + a_{108}Y^2 + a_{109}Y^3 + a_{110}Y^4 + a_{111}Y^3Z + a_{112}Y_3W + a_{113}Y^2Z + a_{114}Y^2Z^2 + a_{115}Y^2ZW + a_{116}Y^2W + a_{117}Y^2W^2 + a_{118}YZ + a_{119}YZ^2$$

$$+ a_{120}YZ^3 + a_{121}YZ^2W + a_{122}YZW + a_{123}YZW^2 + a_{124}YW + a_{125}YW^2 + a_{126}YW^3 + a_{127}Z + a_{128}Z^2 + a_{129}Z^3 + a_{130}Z^4 + a_{131}Z^3W + a_{132}Z^2W + a_{133}Z^2W^2 + a_{134}ZW + a_{135}ZW^2 + a_{136}ZW^3 + a_{137}W + a_{138}W^2 + a_{139}W^3 + a_{140}W^4$$

$$Z = a_{141} + a_{142}X + a_{143}X^2 + a_{144}X^3 + a_{145}X^4 + a_{146}X^3Y + a_{147}X^3Z + a_{148}X^3W + a_{149}X^2Y + a_{150}X^2Y^2 + a_{151}X^2YZ + a_{152}X^2YW + a_{153}X^2Z + a_{154}X^2Z^2 + a_{155}X^2ZW + a_{156}X^2W + a_{157}X^2W^2 + a_{158}XY + a_{159}XY^2 + a_{160}XY^3 + a_{161}XY^2Z + a_{162}XY^2W + a_{163}XYZ + a_{164}XYZ^2 + a_{165}XYZW + a_{166}XYW + a_{167}XYW^2 + a_{168}XZ + a_{169}XZ^2 + a_{170}XZ^3 + a_{171}XZ^2W + a_{172}XZW + a_{173}XZW^2 + a_{174}XW + a_{175}XW^2 + a_{176}XW^3 + a_{177}Y + a_{178}Y^2 + a_{179}Y^3 + a_{180}Y^4 + a_{181}Y^3Z + a_{182}Y^3W + a_{183}Y^2Z + a_{184}Y^2Z^2 + a_{185}Y^2ZW + a_{186}Y^2W + a_{187}Y^2W^2 + a_{188}YZ + a_{189}YZ^2 + a_{190}YZ^3 + a_{191}YZ^2W + a_{192}YZW + a_{193}YZW^2 + a_{194}YW + a_{195}YW^2 + a_{196}YW^3 + a_{197}Z + a_{198}Z^2 + a_{199}Z^3 + a_{200}Z^4 + a_{201}Z^3W + a_{202}Z^2W + a_{203}Z^2W^2 + a_{204}ZW + a_{205}ZW^2 + a_{206}ZW^3 + a_{207}W + a_{208}W^2 + a_{209}W^3 + a_{210}W^4$$

$$W = a_{211} + a_{212}X + a_{213}X^2 + a_{214}X^3 + a_{215}X^4 + a_{216}X^3Y + a_{217}X^3Z + a_{218}X^3W + a_{219}X^2Y + a_{220}X^2Y^2 + a_{221}X^2YZ + a_{222}X^2YW + a_{223}X^2Z + a_{224}X^2Z^2 + a_{225}X^2ZW + a_{226}X^2W + a_{227}X^2W^2 + a_{228}XY + a_{229}XY^2 + a_{230}XY^3 + a_{231}XY^2Z + a_{232}XY^2W + a_{233}XYZ + a_{234}XYZ^2 + a_{235}XYZW + a_{236}XYW + a_{237}XYW^2 + a_{238}XZ + a_{239}XZ^2 + a_{240}XZ^3 + a_{241}XZ^2W + a_{242}XZW + a_{243}XZW^2 + a_{244}XW + a_{245}XW^2 + a_{246}XW^3 + a_{247}Y + a_{248}Y^2 + a_{249}Y^3 + a_{250}Y^4 + a_{251}Y^3Z + a_{252}Y^3W + a_{253}Y^2Z + a_{254}Y^2Z^2 + a_{255}Y^2ZW + a_{256}Y^2W + a_{257}Y^2W^2 + a_{258}YZ + a_{259}YZ^2 + a_{260}YZ^3 + a_{261}YZ^2W + a_{262}YZW + a_{263}YZW^2 + a_{264}YW + a_{265}YW^2 + a_{266}YW^3 + a_{267}Z + a_{268}Z^2 + a_{269}Z^3 + a_{270}Z^4 + a_{271}Z^3W + a_{272}Z^2W + a_{273}Z^2W^2 + a_{274}ZW + a_{275}ZW^2 + a_{276}ZW^3 + a_{277}W + a_{278}W^2 + a_{279}W^3 + a_{280}W^4$$

Case P: D = **4**, O = **5**, M = **504**

$$X = a_1 + a_2X + a_3X^2 + a_4X^3 + a_5X^4 + a_6X^5 + a_7X^4Y + a_8X^4Z + a_9X^4W + a_{10}X^3Y + a_{11}X^3Y^2 + a_{12}X^3YZ + a_{13}X^3YW + a_{14}X^3Z + a_{15}X^3Z^2 + a_{16}X^3ZW + a_{17}X^3W + a_{18}X^3W^2 + a_{19}X^2Y + a_{20}X^2Y^2 + a_{21}X^2Y^3 + a_{22}X^2Y^2Z + a_{23}X^2Y^2W + a_{24}X^2YZ + a_{25}X^2YZ^2 + a_{26}X^2YZW + a_{27}X^2YW + a_{28}X^2YW^2 + a_{29}X^2Z + a_{30}X^2Z^2 + a_{31}X^2Z^3 + a_{32}X^2Z^2W + a_{33}X^2ZW + a_{34}X^2ZW^2 + a_{35}X^2W + a_{36}X^2W^2 + a_{37}X^2W^3 + a_{38}XY + a_{39}XY^2 + a_{40}XY^3 + a_{41}XY^4 + a_{42}XY^3Z + a_{43}XY^3W + a_{44}XY^2Z + a_{45}XY^2Z^2 + a_{46}XY^2ZW + a_{47}XY^2W + a_{48}XY^2W^2 + a_{49}XYZ + a_{50}XYZ^2 + a_{51}XYZ^3 + a_{52}XYZ^2W + a_{53}XYZW + a_{54}XYZW^2 + a_{55}XYW + a_{56}XYW^2 + a_{57}XYW^3 + a_{58}XZ + a_{59}XZ^2 + a_{60}XZ^3 + a_{61}XZ^4 + a_{62}XZ^3W + a_{63}XZ^2W + a_{64}XZ^2W^2 + a_{65}XZW + a_{66}XZW^2 + a_{67}XZW^3 + a_{68}XW + a_{69}XW^2 + a_{70}XW^3 + a_{71}XW^4 + a_{72}Y + a_{73}Y^2 + a_{74}Y^3 + a_{75}Y^4 + a_{76}Y^5 + a_{77}Y^4Z + a_{78}Y^4W + a_{79}Y^3Z + a_{80}Y^3Z^2 + a_{81}Y^3ZW + a_{82}Y^3W + a_{83}Y^3W^2 + a_{84}Y^2Z + a_{85}Y^2Z^2 + a_{86}Y^2Z^3 + a_{87}Y^2Z^2W + a_{88}Y^2ZW + a_{89}Y^2ZW^2 + a_{90}Y^2W + a_{91}Y^2W^2 + a_{92}Y^2W^3 + a_{93}YZ + a_{94}YZ^2 + a_{95}YZ^3 + a_{96}YZ^4 + a_{97}YZ^3W + a_{98}YZ^2W + a_{99}YZ^2W^2 + a_{100}YZW + a_{101}YZW^2 + a_{102}YZW^3 + a_{103}YW + a_{104}YW^2 + a_{105}YW^3 + a_{106}YW^4 + a_{107}Z + a_{108}Z^2 + a_{109}Z^3 + a_{110}Z^4 + a_{111}Z^5 + a_{112}Z^4W + a_{113}Z^3W + a_{114}Z^3W^2 + a_{115}Z^2W + a_{116}Z^2W^2 + a_{117}Z^2W^3 + a_{118}ZW + a_{119}ZW^2 + a_{120}ZW^3 + a_{121}ZW^4 + a_{122}W + a_{123}W^2 + a_{124}W^3 + a_{125}W^4 + a_{126}W^5$$

$$Y = a_{127} + a_{128}X + a_{129}X^2 + a_{130}X^3 + a_{131}X^4 + a_{132}X^5 + a_{133}X^4Y + a_{134}X^4Z + a_{135}X^4W + a_{136}X^3Y + a_{137}X^3Y^2 + a_{138}X^3YZ + a_{139}X^3YW + a_{140}X^3Z + a_{141}X^3Z^2 + a_{142}X^3ZW + a_{143}X^3W + a_{144}X^3W^2 + a_{145}X^2Y + a_{146}X^2Y^2 + a_{147}X^2Y^3 + a_{148}X^2Y^2Z + a_{149}X^2Y^2W + a_{150}X^2YZ + a_{151}X^2YZ^2 + a_{152}X^2YZW + a_{153}X^2YW + a_{154}X^2YW^2 + a_{155}X^2Z + a_{156}X^2Z^2 + a_{157}X^2Z^3 + a_{158}X^2Z^2W + a_{159}X^2ZW + a_{160}X^2ZW^2 + a_{161}X^2W + a_{162}X^2W^2 + a_{163}X^2W^3 + a_{164}XY + a_{165}XY^2 + a_{166}XY^3 + a_{167}XY^4 + a_{168}XY^3Z + a_{169}XY^3W + a_{170}XY^2Z + a_{171}XY^2Z^2 + a_{172}XY^2ZW + a_{173}XY^2W + a_{174}XY^2W^2 + a_{175}XYZ + a_{176}XYZ^2 + a_{177}XYZ^3 + a_{178}XYZ^2W + a_{179}XYZW + a_{180}XYZW^2 + a_{181}XYW + a_{182}XYW^2 + a_{183}XYW^3 + a_{184}XZ + a_{185}XZ^2 + a_{186}XZ^3 + a_{187}XZ^4 + a_{188}XZ^3W + a_{189}XZ^2W + a_{190}XZ^2W^2 + a_{191}XZW + a_{192}XZW^2 + a_{193}XZW^3 + a_{194}XW + a_{195}XW^2 + a_{196}XW^3$$

$$+ a_{197}XW^4 + a_{198}Y + a_{199}Y^2 + a_{200}Y^3 + a_{201}Y^4 + a_{202}Y^5 + a_{203}Y^4Z + a_{204}Y^4W + a_{205}Y^3Z + a_{206}Y^3Z^2 + a_{207}Y^3ZW + a_{208}Y^3W + a_{209}Y^3W^2 + a_{210}Y^2Z + a_{211}Y^2Z^2 + a_{212}Y^2Z^3 + a_{213}Y^2Z^2W + a_{214}Y^2ZW + a_{215}Y^2ZW^2 + a_{216}Y^2W + a_{217}Y^2W^2 + a_{218}Y^2W^3 + a_{219}YZ + a_{220}YZ^2 + a_{221}YZ^3 + a_{222}YZ^4 + a_{223}YZ^3W + a_{224}YZ^2W + a_{225}YZ^2W^2 + a_{226}YZW + a_{227}YZW^2 + a_{228}YZW^3 + a_{229}YW + a_{230}YW^2 + a_{231}YW^3 + a_{232}YW^4 + a_{233}Z + a_{234}Z^2 + a_{235}Z^3 + a_{236}Z^4 + a_{237}Z^5 + a_{238}Z^4W + a_{239}Z^3W + a_{240}Z^3W^2 + a_{241}Z^2W + a_{242}Z^2W^2 + a_{243}Z^2W^3 + a_{244}ZW + a_{245}ZW^2 + a_{246}ZW^3 + a_{247}ZW^4 + a_{248}W + a_{249}W^2 + a_{250}W^3 + a_{251}W^4 + a_{252}W^5$$

$$Z = a_{253} + a_{254}X + a_{255}X^2 + a_{256}X^3 + a_{257}X^4 + a_{258}X^5 + a_{259}X^4Y + a_{260}X^4Z + a_{261}X^4W + a_{262}X^3Y + a_{263}X^3Y^2 + a_{264}X^3YZ + a_{265}X^3YW + a_{266}X^3Z + a_{267}X^3Z^2 + a_{268}X^3ZW + a_{269}X^3W + a_{270}X^3W^2 + a_{271}X^2Y + a_{272}X^2Y^2 + a_{273}X^2Y^3 + a_{274}X^2Y^2Z + a_{275}X^2Y^2W + a_{276}X^2YZ + a_{277}X^2YZ^2 + a_{278}X^2YZW + a_{279}X^2YW + a_{280}X^2YW^2 + a_{281}X^2Z + a_{282}X^2Z^2 + a_{283}X^2Z^3 + a_{284}X^2Z^2W + a_{285}X^2ZW + a_{286}X^2ZW^2 + a_{287}X^2W + a_{288}X^2W^2 + a_{289}X^2W^3 + a_{290}XY + a_{291}XY^2 + a_{292}XY^3 + a_{293}XY^4 + a_{294}XY^3Z + a_{295}XY^3W + a_{296}XY^2Z + a_{297}XY^2Z^2 + a_{298}XY^2ZW + a_{299}XY^2W + a_{300}XY^2W^2 + a_{301}XYZ + a_{302}XYZ^2 + a_{303}XYZ^3 + a_{304}XYZ^2W + a_{305}XYZW + a_{305}XYZW^2 + a_{307}XYW + a_{308}XYW^2 + a_{309}XYW^3 + a_{310}XZ + a_{311}XZ^2 + a_{312}XZ^3 + a_{313}XZ^4 + a_{314}XZ^3W + a_{315}XZ^2W + a_{316}XZ^2W^2 + a_{317}XZW + a_{318}XZW^2 + a_{319}XZW^3 + a_{320}XW + a_{321}XW^2 + a_{322}XW^3 + a_{323}XW^4 + a_{324}Y + a_{325}Y^2 + a_{326}Y^3 + a_{327}Y^4 + a_{328}Y^5 + a_{329}Y^4Z + a_{330}Y^4W + a_{331}Y^3Z + a_{332}Y^3Z^2 + a_{333}Y^3ZW + a_{334}Y^3W + a_{335}Y^3W^2 + a_{336}Y^2Z + a_{337}Y^2Z^2 + a_{338}Y^2Z^3 + a_{339}Y^2Z^2W + a_{340}Y^2ZW + a_{341}Y^2ZW^2 + a_{342}Y^2W + a_{343}Y^2W^2 + a_{344}Y^2W^3 + a_{345}YZ + a_{346}YZ^2 + a_{347}YZ^3 + a_{348}YZ^4 + a_{349}YZ^3W + a_{350}YZ^2W + a_{351}YZ^2W^2 + a_{352}YZW + a_{353}YZW^2 + a_{354}YZW^3 + a_{355}YW + a_{356}YW^2 + a_{357}YW^3 + a_{358}YW^4 + a_{359}Z + a_{360}Z^2 + a_{361}Z^3 + a_{362}Z^4 + a_{363}Z^5 + a_{364}Z^4W + a_{365}Z^3W + a_{366}Z^3W^2 + a_{367}Z^2W + a_{368}Z^2W^2 + a_{369}Z^2W^3 + a_{370}ZW + a_{371}ZW^2 + a_{372}ZW^3 + a_{373}ZW^4 + a_{374}W + a_{375}W^2 + a_{376}W^3 + a_{377}W^4 + a_{378}W^5$$

$$W = a_{379} + a_{380}X + a_{381}X^2 + a_{382}X^3 + a_{383}X^4 + a_{384}X^5 + a_{385}X^4Y + a_{386}X^4Z + a_{387}X^4W + a_{388}X^3Y + a_{389}X^3Y^2 + a_{390}X^3YZ + a_{391}X^3YW + a_{392}X^3Z + a_{393}X^3Z^2 + a_{394}X^3ZW + a_{395}X^3W + a_{396}X^3W^2 + a_{397}X^2Y + a_{398}X^2Y^2 + a_{399}X^2Y^3 + a_{400}X^2Y^2Z + a_{401}X^2Y^2W + a_{402}X^2YZ + a_{403}X^2YZ^2 + a_{404}X^2YZW + a_{405}X^2YW + a_{406}X^2YW^2 + a_{407}X^2Z + a_{408}X^2Z^2 + a_{409}X^2Z^3 + a_{410}X^2Z^2W + a_{411}X^2ZW + a_{412}X^2ZW^2 + a_{413}X^2W + a_{414}X^2W^2 + a_{415}X^2W^3 + a_{416}XY + a_{417}XY^2 + a_{418}XY^3 + a_{419}XY^4 + a_{420}XY^3Z + a_{421}XY^3W + a_{422}XY^2Z + a_{423}XY^2Z^2 + a_{424}XY^2ZW + a_{425}XY^2W + a_{426}XY^2W^2 + a_{427}XYZ + a_{428}XYZ^2 + a_{429}XYZ^3 + a_{430}XYZ^2W + a_{431}XYZW + a_{432}XYZW^2 + a_{433}XYW + a_{434}XYW^2 + a_{435}XYW^3 + a_{436}XZ + a_{437}XZ^2 + a_{438}XZ^3 + a_{439}XZ^4 + a_{440}XZ^3W + a_{441}XZ^2W + a_{442}XZ^2W^2 + a_{443}XZW + a_{444}XZW^2 + a_{445}XZW^3 + a_{446}XW + a_{447}XW^2 + a_{448}XW^3 + a_{449}XW^4 + a_{450}Y + a_{451}Y^2 + a_{452}Y^3 + a_{453}Y^4 + a_{454}Y^5 + a_{455}Y^4Z + a_{456}Y^4W + a_{457}Y^3Z + a_{458}Y^3Z^2 + a_{459}Y^3ZW + a_{460}Y^3W + a_{461}Y^3W^2 + a_{462}Y^2Z + a_{463}Y^2Z^2 + a_{464}Y^2Z^3 + a_{465}Y^2Z^2W + a_{466}Y^2ZW + a_{467}Y^2ZW^2 + a_{468}Y^2W + a_{469}Y^2W^2 + a_{470}Y^2W^3 + a_{471}YZ + a_{472}YZ^2 + a_{473}YZ^3 + a_{474}YZ^4 + a_{475}YZ^3W + a_{476}YZ^2W + a_{477}YZ^2W^2 + a_{478}YZW + a_{479}YZW^2 + a_{480}YZW^3 + a_{481}YW + a_{482}YW^2 + a_{483}YW^3 + a_{484}YW^4 + a_{485}Z + a_{486}Z^2 + a_{487}Z^3 + a_{488}Z^4 + a_{489}Z^5 + a_{490}Z^4W + a_{491}Z^3W + a_{492}Z^3W^2 + a_{493}Z^2W + a_{494}Z^2W^2 + a_{495}Z^2W^3 + a_{496}ZW + a_{497}ZW^2 + a_{498}ZW^3 + a_{499}ZW^4 + a_{500}W + a_{501}W^2 + a_{502}W^3 + a_{503}W^4 + a_{504}W^5$$

Case Q: D = **3**, O = **2**, M = **30**

X = X + 0.1(same as for case I)

Y = Y + 0.1(same as for case I)

Z = Z + 0.1(same as for case I)

Case R: D = **3**, O = **3**, M = **60**

X = X + 0.1(same as for case J)

Y = Y + 0.1(same as for case J)

Z = Z + 0.1(same as for case J)

Case S: D = **3**, O = **4**, M = **105**

X = X + 0.1(same as for case K)

Y = Y + 0.1(same as for case K)

Z = Z + 0.1(same as for case K)

Case T: D = **3**, O = **5**, M = **168**

X = X + 0.1(same as for case L)

Y = Y + 0.1(same as for case L)

Z = Z + 0.1(same as for case L)

Case U: D = **4**, O = **2**, M = **60**

X = X + 0.1(same as for case M)

Y = Y + 0.1(same as for case M)

Z = Z + 0.1(same as for case M)

W = W + 0.1(same as for case M)

Case V: D = **4**, O = **3**, M = **140**

X = X + 0.1(same as for case N)

Y = Y + 0.1(same as for case N)

Z = Z + 0.1(same as for case N)

W = W + 0.1(same as for case N)

Case W: D = **4**, O = **4**, M = **280**

X = X + 0.1(same as for case O)

Y = Y + 0.1(same as for case O)

Z = Z + 0.1(same as for case O)

W = W + 0.1(same as for case O)

Case X: D = **4**, O = **5**, M = **504**

X = X + 0.1(same as for case P)

Y = Y + 0.1(same as for case P)

Z = Z + 0.1(same as for case P)

W = W + 0.1(same as for case P)

Case Y: D = **4**, M = **10**

$X = a_1 + a_2X + a_3Y + a_4|X| + a_5|Y|$

$Y = a_6 + a_7X + a_8Y + a_9|X| + a_{10}|Y|$

$Z = X^2 + Y^2$

$W = (N - 1000) / (N_{MAX} - 1000)$

Case Z: D = **4**, M = **14**

$X = a_1 + a_2X + a_3Y + a_4X \text{ AND } a_5Y + a_6X \text{ OR } a_7Y$

$Y = a_8 + a_9X + a_{10}Y + a_{11}X \text{ AND } a_{12}Y + a_{13}X \text{ OR } a_{14}Y$

$Z = X^2 + Y^2$

$W = (N - 1000) / (N_{MAX} - 1000)$

Case [: D = **4**, M = **14**

$X = a_1 + a_2X + a_3Y + a_4|X|^{a_5} + a_6|Y|^{a_7}$

$Y = a_8 + a_9X + a_{10}Y + a_{11}|X|^{a_{12}} + a_{13}|Y|^{a_{14}}$

$Z = X^2 + Y^2$

$W = (N - 1000) / (N_{MAX} - 1000)$

Case \: D = **4**, M = **18**

$X = a_1 + a_2X + a_3Y + a_4\sin(a_5X + a_6) + a_7\sin(a_8Y + a_9)$

$Y = a_{10} + a_{11}X + a_{12}Y + a_{13}\sin(a_{14}X + a_{15}) + a_{16}\sin(a_{17}Y + a_{18})$

$Z = X^2 + Y^2$

$W = (N - 1000) / (N_{MAX} - 1000)$

Case]: D = **4**, M = **6**

$X = 10a_1 + [X + a_2\sin(a_3Y+a_4)]\cos[2\pi/(13+10a_6)] + Y \sin[2\pi/(13+10a_6)]$

$$Y = 10a_5 - [X + a_2\sin(a_3Y + a_4)]\sin[2\pi/(13+10a_6)] + Y\cos[2\pi/(13+10a_6)]$$

$$Z = X^2 + Y^2$$

$$W = (N - 1000) / (N_{MAX} - 1000)$$

Case ^: D = **4**, M = **9**

$$X = X + 0.1a_1Y$$

$$Y = Y + 0.1(a_2X + a_3X^3 + a_4X^2Y + a_5XY^2 + a_6Y + a_7Y^3 + a_8\sin Z$$

$$Z = [Z + 0.1(a_9 + 1.3)] \bmod 2\pi$$

$$W = (N - 1000) / (N_{MAX} - 1000)$$

Appendix F

Dictionaries of Strange Attractors

Included in this appendix are alphabetical listings of codes for those attractors shown in this book whose figures lack the full codes, a selection of additional interesting cases, and a list of special cases of historical or mathematical significance. The numbers at the end of each code are the fractal dimension (*F*) and the Lyapunov exponent (*L*), respectively.

The cases below represent the ones shown in the book. You should be able to identify them by the first few characters of the code. The disk included with the book contains the file **BOOKFIGS.DIC**, which includes the codes for all the cases shown in the book. You can enter these cases into the program manually using the **I** command, or copy the file **BOOKFIGS.DIC** on the accompanying disk to **SA.DIC** and view them automatically using the **E** command. Note that the contents of any existing **SA.DIC** file will be lost when you do this unless you use the DOS command **COPY SA.DIC** + **BOOKFIGS.DIC SA.DIC** to append the new cases to the end of the **SA.DIC** file.

BOOKFIGS.DIC. Codes for attractors in this book whose full codes were not given

```
JJICKAFXIOXFVGOCIDNIVRPSFYPFGABXKKONQWPAMJGKAGXDBBWFHGXBTPNVD 1.80 0.01

JNRVAPNYFVEDGIVLUUFLVKNVCGQFEHWUISYBJDFJCKLMKRSYVPPKTBFUUAFQQ 1.56 0.13

KIRCGTGYRFOSXCKFOIRNXFLPDLXPISDSOUTOITXGKWSQJGIMOLT
WJUPPCELGSLTRERRKTKJTRUCJQVMNREJTAYYAWJEJISIXDAEUOOIKMV 1.79 0.01

KJJUPXHPMACQRSPYGHTFLGCYTHUSKVNUTTYMMGIQFSKUJYJJHGCQ
YFHLVSCCXAEVHDUNJVNNBUWQRXJEPVLJLDAOBLIRTKJDDDIKQUCSVE 2.33 0.07

KKGKUQRWVRSUTWDGTQHMAYUXCJRDWBPHICJHHSTLBDXQOFFPMNUATX
RCYBMBSWHGBQPGRSOMRTLCYGRLQLTVQJIIVDLTXJEJOHAWUBVRNY 1.97 0.03

KKLXFIMKRYNPNSNVITIBHRMPYHHCNHWLUPKCQQTYNJAKGWVWLMBYFEP
MRUHRARLGHHTLGGYQHEQDHOQTFYRGVUHDCQGQHXYMNFGVGENSJI 1.79 0.02

KKSLVGUQTYKUFHNSDSWCSITGEFKVCTWKSIENMBOLBQQTGOYPHKGPFCYU
UHIOHXVMEFAMMCPRNDJRBQRQILCLELNWEMCYLUHIKCIFERQDBM 1.49 0.01
```

KLGFLNWTWETVNLHHPTLPMMDAJKBJBCCHTRIBKGUEDDDDITBJVIHB
UXIYGTSMGYXTQFASOOTOCDDSYAXGULLLOQCPOXSYVPXBOVOUHRNPQU 1.99 0.03

KLLRVFAKXNTCPWXWENDJVNGQAQEXTSKIDVOYMFMKNFWOFILMGRX
DMKJATLBGFUBOSWJHUDOPYLYWXQSORNXGNLNSWBEESQRTHUKAHBGKSO 1.24 0.04

KLRLPKPHLPXFQRGNCJHUTVSFPFTSIQMIFUSUTCOGFSMQIDJECRT
ATYGZQKDCRBBRTDXFWKHOBMHHIRHNLETJBWAPEUCNSERHFYDLLWNFUD 1.17 0.10

KNRRVWREMTOABPSKHMLDSDCISNDJQQUYKULFQILAWGNTHFSSYTUWH
YFRWHLUUSMCBARBCWHMGALNONSDKSXXBAPKVLXITVKFAFLIYXWBPS 1.25 0.06

KOCWVUCRHIBOOAKVSXHJOPGUQRBHRHNNOJPEMIJNYHVIGIOGPVBDQ
QNEASMCPJVHWGWFWTAKHYNTRBRJLCQTLMLIPJBTGHOWNGNOVOHOWQ 1.69 0.05

KPBMLWTCBCWATDWMVHDHWFKMDLJUERTMLBNNLKKRJAVHUTANQLULI
AYKQNXVBMHVGUEMWKAKECQJMETKFEQNVAVUXKQJRWBRRCSARKVMCT 2.40 0.06

KPGQHOXCGSSBMVWFQEKRLXEATUYYGEGIEIRKLJUYDJTNBRHWPCXGGU
VGFOWMSDXEDKXEJSHIHPPKQDZKMNJGMAFJLVLKWNKKLXPPVNKZEG 1.78 0.01

KQEDOFHXFPJEPRTQBLOOAVOOACBDTUTUPFDGMBVLGYDOFHYTORPWLEYL
NFTUEBOKBDXOPNWNNPKPCFAXEWRTDDLJXMVYHUPMMPPMWDEJDI 2.07 0.09

KQMPLCKLEWODOGMAWCIYBECAQLWEHASTRBPPOMOGEQKIVIEPRCWVEFSRHK
LQBVKCPPHWFVCVGJENSODIYBPYCLHVXOIQIODNHAWUUOFPCT 1.53 0.07

KVFLHQBHLNEGRWRHQKSDSHEIHKOGGUVTRHPIGUPLRHCWBSMKROEFXQMIR
IUYOLMRGUJLHMBNGUCTMOHQUEPPXETTQMNIOIJTTIAISQTBFC 2.08 0.13

KYECWCLFXSOYVGXRWWHJIAAJLNGWPELSLMMRUMTSGJISGDSKSNKTHNBCOG
HCYWLKXGLKBPOPNALOTJQNIMFOSOVGJEGQLILMTFUDYPWNSM 1.57 0.04

LLLRGRWWWLEAGRMETGIDGUKIFLPOPTJPEVSKTMAAWFPFLFTHGJBBFDAXTKKDTNFGALOAHSBYCBSGHKYYTKWP
MCGWDVYEVGBNVWYDJBRSADTVITQSMNIBADRXGKTSDKTAKFWGYOFHMSYYLKNXDMBKCLEWDHDWQVCVTFESSJUXY
1.74 0.06

LLOBPCENCHDFPDMGXGGHXYSNDRNQASITFDLIUXNOFJOMLHRBXQNSQYRPCRINOEBDSVFJAFNWQXMEURSBQKN
XCTOUCFRTVCQHUTYGJBHVKBFMDINQEMLFKHJNGBSRQBWLSLBWSIXOVFVPPYYHIWXDTWAMQDQGONQRNVWMTJGBI
1.31 0.11

LLQEHSEAVYRIDKPRPVPPGFHEQMBKCDXTAJQMWDKCAHPLBRJOFVRSLQPBPLSTXUVHUTXHVMBIKCYXFPXSOMPOEULORCI
HLXYHEFJOBDUTCPYPUCJWOUNDOWAMAMYVEMTHFDSDOFGKKDNNEFEFYUPEUFLRFWBCORHDRDOOGAUXK
1.49 0.05

LLVXALUXCGKASBOJHLPXGYTHEMVEPBGBFUGCPTAIEENWYURWOWJEGRYJFQETREOPQPXQEJDCBLTNTRORXWRCFGHYQK
UBRUAOOJAKHYAYNCCBBQUVJRBIFNWECRAJKXLREYDQUKFUITIMTEBRAUBTIQKABQSJNURTMJQCXADBS
2.01 0.18

LMLGOLTVJPFKQEMLHWIAQGEQNIQXFAQCVHEVKBCNVWNPYMNXKYIHOIDBMJGLOACXSKILBJAHYRPOJJLUYVSOVFUSPF
NMWOTPPXSTSKTOVNNGANGQHKVRBYVDGFCYWDYGJVYHBKLIBSMKBLENIPIPBNINHNOKPUIYQQGIHMXKY

2.07 0.04

LMLMBFUTNIMWITCVCNEFQTGGWNIPARFMGTDKXFMDYYFWVUKBQAFUMKGWXMASSQVIPQOELOBOUFRT
PQYAYQNCLYKCJQTSKLASOQTYTELDLKWQLQJPXLRNTVCVDYOTIFYUCEYAEROJSHYRTAVMLOXVJAHOYPUYJCQMIAKKMJNCQ
1.64 0.04

LNKNTCMVSUJKTQRDDTSHEAOXJHGWIWLOCFOWFRINXSKEDBXULGJFPOXYXLBCPEMRHIUADATTJBLEY
CKFVMPPLVNYVTWWKBNUOYGJPSPAHNTMFFWXFIMYORSYIKEKXYPKAWQAXROWMLDUSRMYQCAMPLLRTUQANETLKIQHBTVTU
2.13 0.16

LNMIWCGDRYTSUWCVPQSEWDLMLPPCCBUBOUWQFOOVRFYPJPEPIUCKHKMXINQPLAKRHBDPYDXCMGDGFXSVP
QNEFQXXJQKTCOQUBSDLVFIVWHCCKHYUDRVYHXATKBQBOMOYNPOUHDBIMCVJRXMEFLWHQCQHGRTHWSBCNWJUTHBDI
1.52 0.11

LOKEHGAFFWVBWLYJBUSENVJSYFTEIVNIQURQXLUTNGNKCPTHIMLDFEAULSHOKOSELHUMAQVBDTLLRINVH
CSQRWXJVPBTUCOWYIHIRUBFCRGOXFFMGKOPILDDMSBYSFJYFDUCXWFVSFFEKUMTKTEEMJGURWERXPJXIUOVFIDXL
1.58 0.04

LPJDQLOHBBPUULIYJLQKECLSLTRDDKLYDGSBCCBEEORIMIEBNVADGTCOEMHEMGDAMLTJVCBQUAFBGPVIKO
VHTIBUHVHQLKBMGVPKXRXNHMQRJWIKFMUDTHFTBGSPOIENGAPPKLMRNVTVTNFXQTFLLCPAANVBRFSIQQJONGSRY
1.59 0.10

LPSONLEWSHNHOGIYSDORACSUHFFRQORDFTEDARDCDUPFSCHCLEDWNEOFBNGVDAKIYNEBQQDHXHMMICDXT
PHLBFBCYOHGCLYOHQTEHXOTOBTJCFKMUNIJYRHTDLTRRWLNWJCPMGTDPWRRGUTUREPUNSSRPYRVINGECTLTCVTQR
2.02 0.16

LQGUEQNPDOHTJIIBLBAHSGBKUBOIOLLBKGLBXTECWGJKBMLXFDGNDVKFKLBRABDOATTFJNRNOLXSEOKFD
UXYHNQELGNGCGICJTNJIKGOLBNJORKRWNLKSWJTETVQFIRBCSYFRFWIPWCALLUQXKQRAQTTBJSQBHQWNIDVBFCXP
2.14 0.11

LTASNEPHQORGLRLCDNJMWHGYYSGHKLMAPHNPOMECDNTENEXDKCRPPWXTPMTBBXPSYSDKEFLBQEVDRNGJLY
FTOAUHVVUGUGONARNWMPCSNKDTYDVMGNUFGPJVOSXMCTFAYYGHXQAPUYIRVBDFRAOHWNALCSKAKPKOANEXNIOGP
1.46 0.06

MKAIUOAYHHCLXYSITCWUNWOLMMDCEQLNQCCMPPHAONXEFCHLXVGVPLRKOFNPS 1.41 0.01

MKWWFTQHWFFBKCFHMUSVFFFANCROGSSPPGOYTTCAONIULPGRITEAQVVQICBKK 1.59 0.05

MMDBTQODXOJLNOSVMTKEICYSWNSLPFNMVLKNSNYNLPFBPKMMYRUYSPDVSJDUH 1.45 0.03

MNGFOFMKCYIFGJSSOVQKUYKUHTITJWEMUHNPXMFABUWXGAMQNQOXMGQTVJQOA 2.05 0.02

NMCSOXFIDMFWROAJLWHYGGVMSKDJHUDLVTYJMPAXVIOQPLTQNLFLPEERFQTWQHSOYPJBI
VEMNBUEURXYUGHMQAIMVRLPNBBQENHGQUDJULMNNFXCKXRXFSGUCSIGLPUHLVXAXHIRTONKH 1.54
0.02

NMCYETOQNYDWARWHDVVLWOFIDJMTXNDKLKYHMKWOBIJFUFRJHVSSWNBMYELRRUIFJKLI
EIFMMHJGNUMKUPKFVTGBYQLRBNHIOJUQWQQWGIMATWYVFQQGPLVJLPOVSHQDRFNTCMFRKALNJ 1.46
0.02

NMGQIXCGPIRHOBACEPHYADFLIFPRLWKUAAOPMQHHBPGAJIXKNHOPRCCRGFXHBOWBEN

XLJLIMQBGEQLYPJREMTOGGSXVXPMADNFRJOVQIPXMRRVWLAIJVLPRQGHFOYEVOWELJOEGNVTSGW.70 0.02

NMHHSPLAXQSKPASBUMAEJFHNQGGFXFJASUJLMPGIPIHAUTXOHXCHOVDDKEKT
HDKLRIISWVEMLQWHVYSTIUWBVDGAAFRWKJHYRAEQTIWEDYMJHEWFHIOTUAKVDPLEUMBRODUBBFCDVNLUI 1.72 0.02

NMHYYGLNLOGHOXYIVNWQDVIMRNHORRRWWNKGMNYFMVVQRNRTMMTLOPCIQRNVNDUN
WMGFIBOMEFWGIKFSNPRRMCTCGRQCQJQECWDQVTOJEOMLIQJMHGJCJCTUSIVRCDNNSLRCEBUFRE2XS7 0.13

NMMQXNNYLNKRNTYJATXWDUVGPEQRRMQFOICLMDUEGKVRQKWMDXRNKWSIHDVBNGOXCP
FPDNPMFVBFKLOSDGWVJVXASWWRQDVCUCPEBAFXMFMGJALVQEKJXSGFIANFGBWSTLQKKFQTLQHIU.69 0.04

NMTICQDDQGFKKPIGLWJDRHOBVHFNIHNEQCYFMJMYASPNVWDECDEPTWEOTRWVQSOOWIPH
HFRMQCVGVEJDCSVFPVMEWMMGEOJQBWLJTWMLTLMNTMRILAQOQBAUYVHHJOVLDLVNYYJDFTENB 1.58 0.03

OKPMLHFLJLOIOOEUWKLFIADMWLEJEEKLCNOGRJKVJIMGQTRWUUILEPNLRJWOEQSMSKWN
LFKKSLLNALSTGWDDIIHMJIYJHIAVSNYUTDSOSJEPYDIWSMQXKBPNHOFPJRLPIISVADUTAHTPSJSSMMNKTEXHEJWRGGLIVOIHXIRKDJ
PMDKGGELUUSABCJJJDHBJOMSWXSFJLPNPMXFOWYKUKOHLUCYWYTNGJODDJKGGYAKFNEGKNDQKEYQR
AQWINBBMBQKTRNDBJBVJQEPXFMXTFOMDCO 1.55 0.05

OMMOXNKSAWSVADFBDLAERLYNSOBBXKHLXKHHVAUUERRTVOYMCUIBQFGJCNSOPJDCWSJBVUNMWSYJH
FAEMDENRIUJMAHYQTTPFIEBBAFEJGTNXLNLPYTBWGEFMPYXLDTPIQTJLOGSAWGTTMMFKDXHESYMDK
GCTDLNXDCVGXYJPBBFHAJGTKYHIWVHFCDHQIKQFQKPXIXRCLQTDMCRTBJMMGXQYRQWEKDWWCYSLFCUQUSFBC
BSMSXMXMQTFGVCNUVJAHCVJQDGMHYSOUUHTGGVPNIWT 1.95 0.02

OMOQRUJSLQIHLLAVGPORHXCLMMXSOOMGQWBEJLBCRJFIWLDFDNABPOUSJVXEIEXBYAHR
OFAMSPIJPMVPNJKDOSBUJKKQRGMMBNYVWQFEMEGSLQVFMUYUHLOQSPAPMIPOBHASJBTSTWEFKMKIJQTQFRVMTFAA
CQQFDBFWFGXSYMHPSVYYCJWEWVDOHXTPSNDJNFPGJUKRSEPQMRGNLPNMODUPOITTMGQNDXVYYBDFE
PHQAGCRJHJISYGPGYIIMASXURGSBLFAMVYUWTOSHOTCEGBKT 1.44 0.05

OMPRETYLSMVWNANPNRJDKCYFJXMPIUFGWTCYXELNGNRHDDPLWLJUMALFPIHTV
LKMFGJNPOCMSWWNMCJUVHBECHDOACRFNMSWNHCOQAVCIWCDPDDJCTJHUIOVRAMIPAOPDRQ
BVWSCLXRTXMEHALMLKEEBVDAVLJIWOHEWBTMWEQNJVLUYVOUTPPXKDOTODRFQWKTNOPVCOJNVWRKLJ
SCMGRKIAGRVHJYISDDECLROVLBWFWCLPYEMMEDJRHMHPNRQWFJUOSSAYFUCYIPPYVSCTSMDP 2.16 0.04

OMXLIBUDPLKNSYDVPOOVLCAIPMVSKGCOECHSOQEPSYBFTAWJHICAVYO
HYQLKRJHTTHOPXPMMAHCRGCWLGFJERQYINJNMKTIYOHIMUQVUNLLCAHHCLJEAGULJBKPJXTGRTSBWGOJ
RTQOIEMFTTJLRIUFITMQFCWWONTSTLDDPVSBJQRPTEYMYXDQUON
MKWALSJSLVCMCNQVNJYIAUOACMYVPCHKPAYFWYRJLKBMYRLGQYUKCD
UDENHRUBIGNFKSJPBABQSDYOXHLMSHKUIIXBUDYJH 1.39 0.02

PMRPHCDNPOKKTATVAYEPWXDJJSCWQNPYQYNJQABCSPUM
FBPDWDSINIEXPVHDSMMRKOOPLGKBEHXIYONCTHMTUJSUCCROEQKNEFDWWQSKUHXOHMKUGXMUJDXRIDDAT
AOMTEKQETBPYBWAWVCXSRVCALBQCVBQJAIVALCCMEWBQKUQVGPLAJLWERAKSUACWGRIEMYMJTKDHCKQRBFLDHIUCBFHRBEUP
WOCWNPCKIHNQORWYMADXLMUPVFMTXFIEMELXHAVEEOEHDOOGMICLEWBPQBGKMQ
HJOLBEMAXONWJYXETOBPOQCLTFFVRYVOVTAPJCXDVEHTQTOIOALNHVTFKVLRNK

MIYMCRCWKFTWLEIAVDPLLOHHIXHFJPMQSKMLXXPDGGMFKAHLCITVI
MOUPHSUSEQQQJEKNYMFPVNKUDITCUELEWP
WWPLPISPMQBYFSUEKQKXNTEVUMROVRAMMIVYAWMYKMDGILFWUCEOLTMBGERMWANBPXFWKREQF 1.93
0.14

RFEJMURNKYSKVHFJXVFDMBKSBJFFUUIBTEFUDKLRKSDLTGJBHHGPHUHLQYSPG 1.61 1.22

RSCVKBOZJYRMFXUTRKLPCNUEACOFHWFSTPFXNDRKGHVWCTXMLRGXVNCBPEUMH 1.69 0.00

SAQKAETCTPFRJAKFEAQRHYFLQZXHNBZGGGJHSDMQQRYYRSWVHKU
ORXKNXPGISSZETJOBELEUEALMZYSDXOOLNPQSAUEBUYAACEBFPMCWJP 1.75 0.24

SFDUKQTRGSVQGTNHJCKNNNCONFTCEJKXEIPRMJMWOZKXHIERPG
SLKDHYAKJKAOLDWSRUIKJMCXAOXTXJFKBORSIAVJLCFOKYOTDUTUEWOA 1.66 1.06

SHBNLCIMXEHDFWCTGGTHMISUIWQVKNOGMRKXUHGAWBSURGPMQGNWIY
HIXDWOEMNSMHNGSNRQUXBSMODGHPAHQXBRNPLGYPFBVPMONHPRDN 1.79 1.13

SQFECQRRYIURKSQUPQJFXYIGXSIBLHXMGPPQIAZNMOYVDLCAWDKMX
OCMDFHPVRROTLEISGDMYLOIIPMIJXMNSZYMOQSTDIRJRHEUTJVULN 1.53 0.11

SSKBHUTGPQHPOEPHRDGFXBALTEPFKACLGRQNWPIIBOGXWOJDLSKQNF
XYITSIRDNIBBFLCDOURLWRFWFEFIOYBHMMTBVRVNJWWFTBULDJEB 1.55 0.15

SUISTHIXFBRILKWHLDWTNCQZSDCNERFJGXAJMEXFVZGTQDDOVECKJQR
SZDYLDBGOEVMGTKYBPSZKXUYKTNPFRMCINCDGDVNMIEXLBDRMKP 1.65 0.45

TBEIESEQRCCJAWWIIMLCJIRCVBWJKCDDRBGOCSLIWUNQNRAGFDOBFOBCRXTVVABBAF
GXLWFFADIGBVOGUROQITIGGDKEUFLVUAJOWBXNIVRAPRTWCXEMMQMDIRWJAICFN
NJKGHBDFFSWPEMBKYKXGRLCSTIAGASCOSPJTFYEU 1.02 0.32

TFRKCBHSZKGAIBHFDVHBWGRWWGWXOPMNPBBEBICJJEVAZDIIVICHNQQPQIRU
MMAPIWNQXEBBOUWGQXJLQFHJPGRDNJFTSCHDLSDTLIHSHEENPJDGFQJJAMEDDILOPUECKN
FIFHWGPSJCOQAPGAEILMOXOQHMRECTRKNCLNHYE 1.90 0.00

THBDSQARKUURSXNOXNTVUHHCALECVHBGQXXACJJJVNCWCXPTXB
FCKGXQSPKXZXCCKWMHVNDAJDJISIOEMRZONFYLVXNLQJNUQCNRZJJSLKDWBAXMOTQAGN
MMGGIIVLOPMTSOCBTXWRDMJJBCBUJVDPEADZWDULHNXDJVFPJFT 2.28 2.36

TIKDSMEKSFJJVOHCDMJQQCRWJAEAEOFLLTVKOUOYRQUGRWRMSRS
VMHKUDOBLKEOARQFQCICONTEARQPCTCVTMLQMUDUUOLVBQUAYOYKWQILMHOJVGL
ADWYQJAXXLJYNXWIGHCTSAETKIEOLGBUQQFJASMQGWNWGCQRLVFADJR 1.89 1.42

TLFBEBGPQAJQLXYMFGYJCXTKGFDZKCSTBCXBFJESQKNOGFGBPWLLDTVL
LPPFBHSKMMFKJTDQQHHSHJJFXYFLNDCXZJXOCQTPAHFLISHTDHFCKIYWSGNWYKEFRW
MAESTCGPRKDYEBSGWBMWQEDVEEKEZJOSQYWVJSLFBKIMPAI 2.08 0.58

TNGLFVWPULUKVAAVNVVQFDYANMFMYAJIJYZVYMGLDVXHCPKAVXF
RMJBCQMKRDETSSUZGVMQNEIYHFTEKLPTKJXUFYGQQRKKPEGHIWZPURYSXOTTT
BMNYWEAUMXLEODVWIUGKNDRRILVUHLFTRLUGFNORBJEFMEEMGGQIHBPWS 0.01-0.00

TQHTPBVXADIMPQSVRDTOTEUGRXODMBJNGAAYTXWW
DDDXDQUONAIGEDVYFXPBGHIDXJOXHBACYQNMHFONNBQXQCGUCRDYDBRYTLBMNXDKNMCQF
NDDYKJTBQRFBTWCKHVMPGJUWOHPYTDNJUMKYLYRNUEBFDBOYSHODYYYAYEXC 1.73 0.31

UGLICIRJDBAEOQBFPHVDSPDIUVHNRQHCVYAEBCNPSFNBQGAQWQNKJDHQTPQPR 1.46 0.09

UUHXPWNCEPURTFNMIOMYQOVRENSHECMCBGEDFHOKXESOIQNUGLDRHTFVNICEE 1.61 0.02

UUJBACFHDEGOKAPREYHRDQAGWPWEWGNTXXWHPPBRCGFOLHOKQLXBIYJPQYYXL 1.55 0.00

VFFSCILFFJKHLPTIXBMAJLOOIXCAIYHILEGQHADVECJ
FHRYWXBXOLMNCUDHJILAGGFMCPSDPKEMOHJAYDWXAGQVHVVHJIIEBO
NJCHSTCCXVHCEVXQGYQPCSJNPUUNHRYPQXBLLUURIXEB 2.15 0.00

VFQGCVARGFCJIGFCJIYQWNAQARFLGRJHCABE
JYGPTBQGWOWEKJTCFUJHJBJSODRBOYEAUFUCNWAJGVRNJNCGKVKB
PBLVTPDELMLMBDAVFUSLIRRNBPWUNGNHUGLCFSXJGKYOSMEBKEHKY 1.55 0.53

VGKEMAGQAQNXJEXJBSMEAXYCMKHYFGCIULBNEMFXSQCCOTXAUIW
SFYKOEQGGJNJQYTTFWAQFIKTYKDBYTNGYKVEBRIABYHWJJLG
USPFUFOHINREPFALYEULXLMCQFBEVYWNXAJCPWVIJC 1.72 0.26

VHTQNWHDQRFGFJXRDQYRWNCRCQFPSBSNVPESANICCHC
SKCQQQMNCYLEEEHJGMKYLRJIWFEORUNBDYQWFHWGNIRSRKEFYOAFM
GCRBBQXDPSCQGQYARKRIMHOPVKFINAHUXOTTLWMMKIBUQ 1.51 0.03

VIPHNKSOEEWSDGCOABBNEJXOBCTAKHTDEHXIFJVHIRTQGXBOS
RNRFRFRSJOJUOERMJTGMJBMMWJIKTMMIHBREBUXDQNEUYYYVL
NIWJLMFQOQUEYEPDJGPJGLDMIHSFCJJLWCXEDDETIHB 2.32 0.27

VJEVASEYAEKHGRQRDLPHAXOGBVIOCSXVLVWXGOBTFUMQTQXW
SMHBQGBOUXRKYEPQKBRMRSPIFPGTBYVWFQRYGXBLBVVVELYDGKQ
FBVGFLJMXMBFJDTSUFPPYAVDCVDQGBMJRUBVLDOSXI 1.80 1.28

VKCLEUHORCIEPAEPGXOXQICXPOMNDFWFOFEOSAQCERPLN
FIQTPROJUBIITHOGKLCIIGENHRPIORNRXDRRISYFBBUFKBDKWSYB
NORDVTFHPHDAURTMHPRVXVFWSGVHOGSCRDIKMXNCMRPI 1.90 0.04

WJCUJYONOYWSFMVSWQXTDCQTPWHWTFDISTVUNKMJGMUWRSGQIBVGU
FVKPURLEQKTDUGOLJFQVVMPVBLMEHQBWHLHAXMQYTLNVMYDRQOCCKXREWDLUYMHONCRWVSEI
UOJJCKIGQVOFQAXIAXCQVNNMJONXITKIDNVQSRSGWRYYBIYDHASUM
AYMCHOUOPKKBKBJSTWSCVDVXDUSWXWOTQMKAYWQNPBG
SHNWSGWRBUSGELGAASIBFBJUPQBIDABHCIRJEKRVSMYDRWQADIAGRNAKYYAD 1.62 0.36

WJUKEUQUEHVWQKIVEHANFQCPGOYIGJCQUKIDROQLRFRHPMYDLFSQIOXUSODQQ
OPBAPNMWDLDNLITGAIXAFKFBKYUHGNPIGWXKFEEADWMAGYGGBWSSFVIBAJWJVFQG
BGBNKYAJOQJADSVEMTHDRINKSTALPSNXORTFLGYYRRBLBMNKDCMW
RPGVRTHHKKSDDTOBLVSGDGORQIMSAUKBIYRGLPJLPC
PHFAXKFAHCGSFRSMYJPTDNGYEWSNESCHRCPLQHNOWVADJXCXFDVMSQRSYOQHEY 2.62 0.00

WKDEDPGPQXLGMOYNFBMPSIYDQNGTDIYKHFLSCRYLVKRESBDQHBRSEHGXGELVCXDVBMQVSAOPDSITCQ

NBNNHUXYFAEIBOGSJMHWHTNKOHBLOHBLQBRDPNKCXEOOHMUSHPGFYNYOTLVUIIEMUNBIFH
SGXOLKGBINCAPFRQVURUHEUIVARQWIWUYNECWWCVCGHINKEKFEATHVRCCHXHIROA
RIAFBWHHQMUTNVSEKQJJQWCGSCULQURQSNHGQAREIPCAJYXOFWJKKSNJDVYEUEPJBHELO 1.76 0.18

WKLAAVLORDCHCPKWGNJLIXUHKAUWAREVFQSGSIKIWRQJLTRHNFUOYOE
VAXJXYSTWXRYGEMQYQKEARQULSQWQIYOXEBXMFPBEOEUFTQDGSBJELDIJTEGAOHRR
XDNOQFTCARAUWJLICIUKHCCAKPMTTRQAWFFWLUTLRNUJTXMALIIKBUBEVNIVMLUJTYW
LPOIORFVADRHKVFCAEWRXOKSIBGUOKKBEREBMJHBEXYMJHXXMTELYSKXYSKVJCSLFS
HLHPTWEGKPPGYYSPFWGNEHAVERBY 1.46 0.41

WNFIDCJHJODNDTYMYSQPICGMDDCQYEODQHCIPAYOIERSSDVFVHQOAC
KDEJIKPOKVVSYAQOJPHJJXUHKYJSTGNSLFFITGPJEAUHJYLEYASIXWXFPGGBGPP
MDERGKEMYNIHXRXHFPMJXEEKRFVJNVDIHGJUMWYTIEQWSAROHRUNDCYSGNQ
EANEWJQXHRPUDCBHXGAOHTWRKKSDAEDJYNSNARSXVUQIVWPGVVGYQQRBGODPJFYDWQQXT
WPJDOKQEQHESDNCOEBMLKELJBMGHAMWETWDU 1.66 0.00

WNHRNECEKBNTNSVBIBKNTHKYDMTDJOPAMRHQVLMOOKKCSSJET
XHQFAPEEMBCJKVKBSFRDOLQJDUYVFBNGWCFNBDHTPCNODDRQUYRGXQEILCNBKRXWIGVLQMIS
JJXRWEUOLOTXQKAIGLOASPRAXQTEVJXGTABDDKTRGYQMBCIOKCXUHGNPXFCIMGFQMV
IGYRHTQGSXOANGWBSKQVIKNCTSEEWDHOWFNWBDRKYTSLGGTKUMAWNUNTONPWMAVO
TUKSYGRQGIIRGELFRKPMVPYKKKGCLL 2.21 0.17

WTQRFNAVWGUNPKGINIXKVCVSOCGUJQDUHTMTKSROOXVKTONNHQJFHVU
GKGDDJCEHHXLSVOMVDDQYMTKRVYTWRWJNGSXHDHGNRVFXNNSWINEDCAAFQBQJMRSUL
FUIBEKGHMEQDDRKAGWNGDCWQLXVTQABHNMOBBGHHERNQVNVWQLMAQQBPFRLAUCOQYBBSUEISAEK
RJOHWMHNWTKFPKGJBLHVBGVNWSEQYXBRAFBXVJWRXRJOECFSLIQBVGXRJVVDAKIE
YCCJHEERAYSCRLEVNHGER 1.78 0.47

XJUSDQKDEJYCDRBCBQFKKINFBFCIHLYWDDGHDSYQPKWIAGKVETECIW
FODMNAIJKATEBDCFMIYBYSGPFNLFPNCCDVATVLWBJCPBEXWVUPBWTJSTUUYEXOBEDDUXJNIF
UUWAXIJEMTPCXKILNVLYOSNTLFTBTFMFWLBJYUHLJMEPKJXWTYIWJDPCLJIRQSROTUBHQWBMBDIYIEBCPM
LBSOVXKDDKTOHRLFGJKKLGJAONUQXJYXHEPXMGROTIMOPSWHQJDELFHMAGFRPLBJIQQUKNNEWQG
HRGSWNNUNIMNALFLPBGNHFYDKPDOWPKUGIMYHYTPAOHXUOTCPAVAQLOADUHOVFDSJRMI
DHBPXYJDLRIPCVJBFXEQYIDDYFEICPTPLGUNRSWXJYH
GQKNETCYEFRNYUSSDGXSWRBAPNFKNMAUHMHXYTUQMPXQMGSJSVFK
TMJKNINYUFOBEFYGKCWIJWAEMOMTTRWBJTYOITXIJWGVGAPHMENULICMSSQ 1.65 0.33

XNOYBECRSMOTWDWDPQGPJPPEOSJLUBTKKGOPWQGQTSPBMISYTUVHN
WWXUIGGJRFKSMRAEHMLYHQCILLLKBKJJDWSUFQYAUETYMVJLCAPSUSCBUHYFTTTAG
OYFVEEJXIDKCPOGBUSRXIOOAUAAKTEUXIGICEGWVULNLHBHJOLDETAXFICCSLIT
VXTJREJGDTSHIEJGNVHLRYTEQVNKHJBVAFGJMVQFSFPGDEFCQAEFQXFGPEROFUTWBPRFUEBNYBYMYMINMO
LACTXNBWFVCHETYYKNCISHRRSUUTFWKXMUALWPESAGWQHTRBWFATKQKEAEBRGTQHDUEYMJYGIGMNPULLC
FVYLMXINBUASIBHYRMKDXHMSAGSCHLVRUDLJOVIHQVTKOCCUCQSWWJKMPDROP
WXFPSJLKDEIDMFXUHWFQSBLBNAISWNSWXEPCUXUAHCSVUW
YGLLXNPCRXJFQYSYJJXMIWTWVPINIRPUMSDWPCTMQGWVELQMOFEBUB 1.50 0.58

XPOOCSENDLUDHWQRTQDCQDHPWSTUYDLBKQKWIAKNPVLAFNNANJXBYNSYPASAGDEDHTDMWKRVUM
RRNQIOFDKBJYQTHFYALOAPCYUORXLYQKKBDFKFYSXDIHSPKGUDUVVMGDAIQHEDMURPYAQRONAFMJI
XKGLQVNXDYBKYMMHHHYVVGYDAKBLMDJMYOYFCIPCVYCCTDREHEPFEYUHUHUVRQCMBYPXUIDL
UIVDLJJBNDNGERYWLFDNLGKDRYUDPVWCWNACGGKYBIEIKVKNRNFVFEVMXMHIMJGKSMQEOQFVVCBP
JTYTXYVSPOQYOAPEOMXOQBHDLIBWPNHOUPUUXJXJPYOSUGQHJCEQDNAFAXDEGRJ

```
PMBKSUNCHBELOANHOQUYSWITSYNKYUMUKFPNMKGXPMTRHYIADGKKFPHAKGMON
ALAXAICFVKURUQGSEPDAIUPMQSEOFIDUQUEKBKGTPBYRMWUEFGEHSUDJJKBDVK
CDXNIXEARYBSNEAXGCAK 1.38 0.14

XRDYAOEYFENFXMQMXXPSSCWORVLAKLAFXSADHCQGWDRGUXUKURRLOAQTDYUAVCPJPUPC
TPLWVIOXMCCHRUYNNPGYYYBTSOHALFLBLBGQVIXKUJDSCTPFDLXJOBXLKDBTDANERYUD
ESXIIRMJFBOSGBMIAJKCRXXNLYQAFVBACVXQVMRTJFUUGIGOBPCKAKOKPBWMRQQYVYGJ
WPOKTEHEBWCYYIBXVOLKVFGHDPQAOQKQUSCLDCLRBDPIIFRCMLULATVPCDSXRDIOYILUL
OUUCFRUGWWXAJRHNOIEHIUPMMCLPJSOLVOTTKGQNRNALLDKIIOVAQNAKEQLETILABC
ROPKFXXTPHGVPGUQLQIHJGODMSVDWCGPKYKJAJNMMJBIUJKYPHFYBH
PEAOBETXFGKPJKTDOVEPQMHCWNARVTVIIMKUPCMUNTFLQAVWPYABQLNYEAHFIMQRLGHJ
YDLIGQFAGAYKNQVOSXWSLHUFTQRWIHWFDJCELYGHXFMQ 2.31 0.24

XTPXAPOKYSXEJCLEHGQGYXTGILQANWASHNBSHSWNQHHUPOSBWSXDDRPUQPTJVPGA
NCRXHBFOMHXHEJKHRTIHHPBRPVEXOCFFNUUUUHUHKSHYCTJGAJMJAMUKIMQJOBQDGPDBLBN
LMVAFNYGWCJUPWHNMWOLTINFJUEMSNMKUMABSHYBRTAHJRDTFGIYXOMFAYVXXSSXIUQFBGOETJK
KTNVQPBFPQOCXNQJVROLJOSXIQFUAGANAJYQIXHCTJLLQAHEJODVVDGJMPBNR
GFFJOYGADQESGDHSCORYQCGPYNTBMUDDOGLEKUOMQTNEXSXOQKMEHFFAGTQ
YIFVDRCGDTSUKEBNPPXDHHNKNOTEKMWDHDLGNEJUITQNGXDLBN
GWSYJARIIIGCQGGGQUTTCJYUQNFWWTWBDIHXARWNFIFSMUNWLAPIYTBDVGVCWCPUIMFWWKYQOKKO
PGXCRFSEUPOKHYTPOTECMIOHCSFGSWYCUBPAAQEDAMITQSHTG 1.81 0.45
```

The cases below represent a collection of additional strange attractors chosen for their beauty and diversity. They would have been appropriate for inclusion in this book if space had permitted. You can enter these cases into the program manually using the **I** command, or copy the file **SELECTED.DIC** on the accompanying disk to **SA.DIC** and view them automatically using the **E** command. Note that the contents of any existing **SA.DIC** file will be lost when you do this unless you use the DOS command **COPY SA.DIC** + **SELECTED.DIC SA.DIC** to append the new cases to the end of the **SA.DIC** file.

SELECTED.DIC. A selection of additional visually interesting attractors

```
EAEUBNVIAHERQ 1.36 0.16
EAHSVIGTJKOTB 1.22 0.12
EAMTMNQQXUYGA 1.27 0.10
EBDNOAXZJNRSG 1.41 0.20
ECDJXIYLSYQUM 1.43 0.11
EDSZHYZHEKUNJ 1.45 0.13
EENWUQSLHYSAT 1.46 0.15
EEURCEQVLRNSF 1.04 0.09
EFFRXAXMGLFNI 1.31 0.19
EFHUPPBRKIWHO 1.43 0.05
EFIRCDERRPVLD 1.37 0.05
EFMMMYWCFUMMM 1.59 0.18
EFOKRIROFDMPQ 1.01 0.09
```

```
EGLXOESFTTPSV 1.77 0.12
EGOSXRBRCBSPM 1.26 0.19
EHGUHDPHNSGOH 1.46 0.03
EHNHBMWMSVEPL 1.18 0.10
EIFLJUWMAICIB 1.17 0.02
EIFVQJFOOLVDV 1.58 0.11
EIJYRRWOBTMEN 1.44 0.09
EIXOFHUZSBQHK 1.35 0.10
EJOYHUWIVDACF 1.31 0.12
EJXAICXIXFRHI 1.37 0.12
EKCBIUIPLETRR 1.20 0.03
EKJXKXKKDYTLK 1.36 0.07
EKMMMYSUKEMMM 1.50 0.12
ELIRZLTCPNHOX 1.44 0.11
ELRWEFKFHUBHS 1.33 0.06
ELUFBBFISGJYS 1.59 0.18
EMCRBIPOPHTBN 1.39 0.05
EMFPGVXTIIDKB 1.48 0.35
EMJDSFTVHGEEV 1.31 0.02
EMLDRMQYIQWQD 1.34 0.21
EMQPUKNVAGCBE 1.64 0.43
EMTGETXEJWCUR 1.44 0.07
EMTQIBOXSCMRC 1.24 0.16
ENDVDPLVKBXEF 1.15 0.15
EODGQCNXODNYA 1.31 0.07
EODSTPMSDFIAO 1.46 0.08
EOOHVSVPDBGXW 1.26 0.01
EPKBNVOONOTTC 1.36 0.30
EQEGJUASEDNUJ 1.41 0.25
EQFFVSLMJJGCR 1.37 0.11
EQHVHQHYTEYQA 1.60 0.22
EQVEUTIPLADHO 1.26 0.09
ERLKHGBBDLIKJ 1.48 0.20
ESOKMLEVUMKDW 1.26 0.07
ETAPDHJKMTUBD 1.33 0.03
EUATWVBSHJIWR 1.35 0.11
EVAVMXOETHDMQ 1.23 0.11
EVBUQHNYPGJDF 1.34 0.37
EVBWNBDELYHUL 1.47 0.13
EWNCSLFLGIHGL 1.24 0.05
EWQKCSBRBQDJX 1.28 0.17
EWQLIJJHEXMPP 1.05 0.05
EWRAHGMIKMCHF 1.07 0.07
EWXHJEGNRHQFP 1.26 0.26
EXXFGRHFTPDWD 1.32 0.22
EZMCQGIXPJMJB 1.21 0.28
FBOJESOSHMVVWMHJMQIIQ 1.40 0.09
FEROJRKQNOWMUXJOIYLIM 1.21 0.07
FFEWXYQONQQJNVELRTBPS 1.43 0.21
FHGVGMHYSSJSHFWIBTSUA 1.16 0.06
FHHOLCTHYWJITRPCYLEHO 1.08 0.16
```

```
FHWHHMFMKFBEKIKSXLMYS 1.25 0.02
FIWOJAIIXSPLVYKSUHUEW 1.21 0.12
FJMANGGKMZMNRBNKCJGCK 1.34 0.07
FKDPVHZGDBDFPTCQHFLXL 1.06 0.03
FKMIABSKYNEQUXZZHSMPC 1.53 0.16
FLAJYVHGVIHGBPQWXYSRJ 1.19 0.29
FLUUCTCPDSRYFJWBQOCYJ 1.21 0.03
FLWAQNQVNQOOHIQQMBWSC 1.25 0.05
FLWIAFTKMWQJDIUCXKEKU 1.23 0.14
FMDEEAVWKUXTHEYOHOICN 1.19 0.03
FMKWIQQVZNEBRWNCJTLON 1.20 0.01
FNFIVJTOSUSTJWCJAYCLM 1.49 0.08
FNHWLFXVXQGPFILBICUJB 1.14 0.04
FNOYESLFRWEWERNDUKOCI 1.19 0.19
FODKJNNRHWCSUAJTUGHBP 1.29 0.17
FOGUKSYQYMBLEUWIHRKTQ 1.34 0.14
FOMMMMMMMKCXIDMMMMMMM 1.16 0.09
FQFEOIKPUKEIBJOTDQAMY 1.35 0.02
FRDLDHYKHKPWENJTOWOXD 1.63 0.31
FRHBYPULLAQSYMQYGREQP 1.27 0.13
FRMMMMMMMSYPVKMMMMMMM 1.22 0.19
FSCEYJHFADPMQDMCUWKVW 1.12 0.19
FVFOTKULQICUAFLQHUKPX 1.05 0.18
FXHDPILBCVMHPOJFVXIWK 1.10 0.17
FYAFYNGBDDHNVBPSPAIDF 1.39 0.14
GGTLDSNGVTBTEINQKQDPQEGNVFMOKES 1.44 0.05
GIGXPNBBVEMODKOCLXERMUNVVPWECYS 1.38 0.07
GISPJWDOAOFTCUBLOMEWXFCXDJQDRDU 1.30 0.17
GJNQRXOIMQLYVLQUNWKDCMXQAOYNZOT 1.52 0.15
GKDSEBTOQIOKPQYHKJUGSWYHMKICRYS 1.30 0.09
GKJWTVIJQTOGJDROEPYKTNWTGABNVBN 1.13 0.18
GKPQYRIDMPDGDKTTRRQPHOCFAOWRABQ 1.23 0.06
GMEHSXHFEMMYNXCJPRTKWADSAPIBBWR 1.44 0.19
GMTBIQYTGHEOJFMLDRXEIFJHLUPBBUY 1.56 0.24
GMVMMPSBXJMZCOFROXCKAFGVDQOKGIK 1.06 0.01
GNCDXEXXMVEOQBKWKEIMXJMLTDEDVSR 1.23 0.11
GNVIKAVSGBOYGFMTIXSOQNOTCVDOZCR 1.36 0.04
GOSLDJIOXHGVDTGGNCNOQYNWJBFHYLX 1.35 0.15
GQHRLGBFRLNFSNXGUKGWEJKFMQTGRDC 1.41 0.07
GQIWKNIRBQKIYSLXLCCKGEUQVUELLLJ 1.14 0.03
GQXEHVCCLDWBHSHHMUUABJRSPJLTTFC 1.51 0.16
GRLKHETMSIVYWMBBQODMFNGRISGHBSI 1.41 0.10
GSELTBLWMEENVFBSFAXMVDXKKEDGYXL 1.40 0.04
GTUFSHTHCLKCTBYWNMCISKERQISFVAV 1.32 0.20
GUMMMMMMMMMMMWEODFMMMMMMMMMMM 1.02 0.08
GUOBMPCJRXSHHSCPMQFZVNESALEKOHY 1.49 0.24
GXEMONYFKDJMDTPNSLGHQLHOOTOQBUN 1.20 0.18
HCJBKUPMMMMMMMMMWMMMMICMMMMMMMMMMMMMMMM 1.39 0.07
HHANQRENHONYATQYPTNXKNMNQEGDWKYPNSMMMODAOBC 1.13 0.01
HIFZLMPJUBERQBKLRRDOWMOLICDPVRJOTHOBSFUKGVL 1.39 0.24
HJVVHSHPOLDTXUCNMYRAYOFBMAAJFMLNAHJQOBBNXQX 1.48 0.06
```

```
HKTSDLOMMMMMMMMMWMMMMDCMMMMMMMMMMMMMMMMMM 1.41 0.06
HLONENJVYCXAJUMHKRSEUAOANAQGULOQBHKNWQOVOJN 1.22 0.13
HMOAQRKMMMMMMMMMMWMMMMYCMMMMMMMMMMMMMMMMMM 1.35 0.10
HNBCTKKSQMGKBDNBXSLIFLJWOONPAPCGTYTUXWRVMQU 1.07 0.15
HNJCYVPMFLGQMPHCQEOMHMMEIHIJJJDDCRVYEGUXBSF 1.36 0.14
HOLTJASMMMMMMMMMMWMMMMWCMMMMMMMMMMMMMMMMMM 1.16 0.03
HPYIESIXFDBTDVPNOUFHZZPNFOCNTRINYYVMVLHHZLE 1.33 0.25
HQJDYFYUMIRBQRDGPLHOBEHGBHDNKVCRSSYVJIFIVLF 1.25 0.18
HQMXVDGMMMMMMMMMMWMMMMBCMMMMMMMMMMMMMMMMMM 1.39 0.06
HROIARTMMMMMMMMMMWMMMMFCMMMMMMMMMMMMMMMMMM 1.15 0.03
HRPMBJSMMMMMMMMMMWMMMMKCMMMMMMMMMMMMMMMMMM 1.41 0.05
HRUTHBNTTDGJJJDTVZBMVMIFSIDNDFEGWLJINVMMVXM 1.21 0.16
HSAXIXDPKULROIQBVOECGZQNWTAEYJAKDETXIOCHJQV 1.24 0.04
HUCXVGLMMMMMMMMMMWMMMMGCMMMMMMMMMMMMMMMMMM 1.40 0.07
HWBVVHLMMMMMMMMMMCMMMMSWMMMMMMMMMMMMMMMMMM 1.05 0.02
IGHCMVFESUFJJGIUOHWFROOIGMGFBEC 1.41 0.05
IHESJWNBMQQEOYLGPPBXGNHTJTPNTUG 1.23 0.05
IIIQMSNRHWELIGAGIURCSRIWABJSESC 1.74 0.22
IIUWOTLCIVQNMKGLXCBGELWUUWUQKIT 1.48 0.08
IIXYMLIVIVWOAXXXLEHKDPICIGQUQGF 1.33 0.02
IJVSBDFVNDDWWNMDMHDOPCFNYRIYWIU 1.08 0.03
IJWWHJKPHMMKUVMKFSRHKJCYOISSQNB 1.98 0.02
IKLEDDBHYJKFRYPBNYCVPRSVJILWEFP 1.50 0.10
IKTOSTVLDYEKWJRCTSIHIQJQVBSSEWG 1.32 0.03
IKTSIRHICWJQSSEVWGBJQKVLDSYETTO 1.33 0.03
IKUELCPYRWJFNDCNNRBVQKQREITYMIY 1.51 0.06
ILGEBMRRGWSRRFOQCLRDOOEARWYJBVE 1.56 0.05
ILLMEVWJKOGMOIVHTISBKJGYYEFWSEK 1.59 0.08
ILOTMOQYJBPLDUWTSWJQDQJVAQLEDQF 1.49 0.21
IMNGCLHTMPFKYEQXNXVUETBDSSWOOGN 1.47 0.04
INJWFGVSOPUNATNJMNRWDQMFKIGMRSB 1.39 0.10
INKRCPBNOMEMVQQKSKYEIJOCQWEYOFP 1.48 0.02
INLJYYNNEIORHAKLKJKOVJFTFGGSMQY 1.52 0.04
INNRCKWREIASTBGRGPADGMGSHPKMPHU 1.33 0.03
IOCVGJFNYEVPTEQLASRSELPUHOTDBXP 1.50 0.07
IOGBGSHOUTDPTRFKCORFDLNKOSPNPHA 1.61 0.10
IPBMEFIUKEKPDTZEJMPXSJTUFZLFRJA 1.20 0.03
IPIIOOVNXNHPAUADBROXSSACJSXGMKX 1.52 0.07
IPNBJINWBKXSIISTQCVRQNUPKSCLTXS 1.45 0.04
IQETFNJNAHIINXFKUHXYHMTTBNJSIII 1.34 0.08
IQNBDVISXIPPLGVLRMKNCMORMJOCIHX 1.60 0.19
IRGOUVHFMIJQBAKEWDJOVQNUSGCNPDU 1.63 0.04
IRIVIMLQBPFVPSLIKHJNDSPMWMCBGMK 1.95 0.02
IUFPFQLVOLTUAVQYFLEVREPQLSNQRCD 1.40 0.08
IUIMPUSPSEJNDPKKENDVSEHCVWDVEGQ 1.36 0.11
IWUBBBVGSWOQFPMBKOPLQKUEIKHSVHM 1.62 0.14

JLDMKNSOMROLNJUIPQVIFKDIYJMYLSUIWGJFWHTIPRVUTBSGQKMHYPGIDKLPH 1.19 0.11
JONRJIUFLWTQFQFTUOCWQLPSSEQGBNLGMCKKEXSFGLHWKSDPJYIQNMJQBAIMT 1.33 0.02
JUDGRDVXMOHNMOYGPJIXMNMJPKXEAIIHNLJKHKBWHJIQUCGCVGQRSQGQDENMX 1.95 0.06
```

KFQXCEQYBQSYWDRSDPXYWOOSLJLQQURWRIRMOONBFJJDUQVR
JBFRIRCDOODRYIBWPNXSGREJIRQJWDKAOFNLUIPMCAIILSONYUNZDDNRFZ 1.72 0.16

KKWOOHREEEJUQAVCEEBRTVVBVPUJHIUDNQUUNHZDTHDMYPUGJE
GUPAJGXNMESDDGCDEMYNCDYINJQQDGYHMRMCPHSVJZPQLTVFFFSWODAS 1.35 0.01

KLPQQUQYSIXVMQDFDAEIDXBLHEQNKPWWSFUIKWRECWGRWNTRKQ
SNZTQRAQODNEYETSLOBYLSFGGTDWBAYPUOSAYDSPUHJAYTEPVPSYMITP 1.24 0.04

KNVXSIBPKCSPMTCYGJGBKBACKTUEARUPKKHGNUCXUWEMOCLDKTNJ
YVIHPDQRSDVGHXDRGCTPXUJIHNFFPPNDHFWFIYBINJBTDCIUATHBXG 2.00 0.02

KPUMNHSVGHFQTTXWQLRACVNNRSOROTNEXGQIJMFGPDIJOUYATCHV
NHOUBMQCGYJLGPHMBBULJUSPSVRNBKODHHLGXFYUYSANFVFVRTITDE 1.72 0.05

LJSWOSYKITFNRTXIFBSANRCXSEGVINFAFXVHHJJTUHOUIWJTDYOGTXANV
IDRAAXUMHHJTJXVLJOFHDIJMJMAIJABSJFIFKAGSEYHSJWSIEARJBSKNKGKVOJHIG
FVRFTGOLXANNQVSIIEPXDKBKKCJJLPIRHLJFTKISATMTSJP 1.55 0.13

LJUUKNADVUXNGAFRQDMKIHUDSFRPARHYAGGMEOJWSXOGTRYQTJRFRGDWP
QHKQWVVDXMSDHXKAAYFUPVTODYHVTUIRSAOBMLPYQUCWOPFTEGSRAQNWJDHVVRWHD
RQRFWXTYLKIULWCBAXKGRDCFREDNPJMNDNMAOJMEOLLNDJM 1.43 0.04

MFFJSKNPJKXDRXXJIJJGLJAAMGNLVJTMPLNHEBDFIEMMHRNVSGIRVTVOIPFMI 1.48 0.14
MGRWGVPWQGWJCPIRJWMNTPPPGTQFHNLMLUPPLJVWESGISIHPHRYLXPAWOAPHN 1.54 0.06

MINXSAGQQMMODEJKPNPITOHHGAQNNPJSUMWCQYQHHFPXIJMSFWPWLNWNQVLRH 1.73 0.18
MKSOKQDGVGUFHWQYLSFUYFQBDHTBSXEMHOUSNDCJPGIDNKNGPKQLSLDQWPJNQ 1.72 0.05
MLQALFYIJMJKKCOBFIVNXFELUGSGBNWHAYWDIQYDRJOUWNLMLOPAKJFBFQHHD 1.69 0.06
MLRLDVIRFJRJGILUQYYYSFKHMQQOWRMIIPRGNQJHRKKHYDMXSIYMFPLQRWSSD 1.51 0.04
MMACCBHNSGDQNRKVMRDNJWQBFSLRBGBMLPBRKMNMIORGMDMNTHAVXNEHCLPJA 1.76 0.07
MMDDKCBROKYJMXXQKPBMJSSMSURNIKSONBTWUCCOPRAGMFPNNTVGRNEQDQYNL 1.57 0.05
MMHOQKISLPQJCPKHMRKWOCRJYWDDHCCMOGJRFCIXILWTJQMUXFLCRKNBGQNFX 1.79 0.01
MMLIYSWAWFBMXHXQMMWBHKOECJXSHCGMFJSGCGVQFFNBQXMNFMLIRCGTNHSPJ 1.42 0.03
MMLOLQBGOTTARNTOMGDQKMUPTBDFTBNMEAWCOWUXLXSIACMNBDWPTMARNKEFN 1.06 0.01
MMMNLPBOWPGCVVDNMPQURKHKNXKSVKNMTVMUQLJYKOJFNWMUVCIVGROFBTKNK 1.36 0.02
MMMTHIGVYGFFABIQMTDXTWMBDVACEUXMHHBJLSXROGSPJUMCNYFRYMTEWRXOX 1.30 0.02
MMOUOUYJTCQPFCPFMXWQCKIWBKFXTMKMCUNEFQEQOGAKEHMODHRGLONHWMCRN 1.36 0.16
MMQSQNAJVTXENLDPMEUWPPGVLAIGBWOMWTSAFWOBIKWMYQMGBTOVXDILQKJOE 1.64 0.05
MMWYKARXJGXRDLMKKESHNSJEQHJRVEERSWWFXEELSGGMUAOEHWYXLMHNLMLON 1.73 0.01
MPGBTHKTSSWCKASIJIUFPXNLESSLJNUNICHOHEETONYWOGJSWMNXHOXEBLFLG 1.71 0.03
MQCQGOBGVFLHSXPTPDOOVIREBODHGUMPIYTYPHFWBENBNKOFFBLOLOAPMEHUI 1.85 0.02
MQVEIINRFUOYTBNCLVDSANJJXUTTSAWFRMGWDETGAKRMRDVAVSIKBLBUWLCHQ 1.14 0.01
MSOJMJRQYAJSCATEFVCPNKHPIKKNKQOIWADJDWGJBJFISJNSBVNPHWYMOGINL 1.47 0.07

NMKAIYSQDHFABDWNQMFPVRJYYVNXPNDGTIOYMTGDIRTAUEXPXITIQPIKUJHOONVEEPICMHUM
IDQLBCXFOFCPIACOFYJMICGPJTBAQTRFPAMQMSUNUXTSFHOOMGUNCTEHPTHPCJIAKSKBD 1.85 0.08

NMQJNUVMGQFSHIJOYQCMLIOGXAXUCDXMCPKNMDGFQUKRFBVPKWPBLKTUFEREGFHOYOXR
KMAMAYUNAGEECHJDELBJWAGLDINGXIJXEPSGHPMPLFMCGQWAAOLMHYPATAKUYYTPQFIEEITNH 1.58
0.14

QCGPVUQQJPHUYULGKOJNBJEAZPENSZH 1.91 0.02
QDPHNYGLPMVYYQOCUIYOLVKFVFIDMIN 1.52 0.01
QJFPNELMYBGVUOLEEMJJRICCDKLQMUK 1.80 0.33
QJRFPORWBILHPHKQMRUDPUDXIJYGBGF 1.29 0.25

REVUKOGCERMHXEKBCQTJRHCJVXRRIBQAQYHVCXEXIOTVIVKAKKECPWFGPTHXF 1.45 1.99
RFTGHODREUPELRRMEVVELSQBHTDDHYYHFQNAFLQRSEAMVAJGPYWYQBUEHFHRF 1.39 0.25
RIAFAHLTKWQMGNCDYLEJNEDYULQPHJXKUAMSMDQWKZYDFMVBREESSEGMGOJHK 1.96 1.37
RJYQGRWLLATYZGCMHPEZNQNBEDGRLXBSVCZBXHRKLJMERQMCFKTKGKKBLOECH 1.56 1.18
RLBVMDKPEEHKIDLRFCJPLXOYTVEUPSTSLNJZLANDNYPWYRGCPGRWNUXTWYRVJ 1.16 0.17
RNDZJLUGWZKQUSBPXESCPJZDIJCEDQEBYGCFPNQIKBYDFYHTOPXMKTZLFHNNN 1.24 0.07

RQPXHNHUENPJJOICSJJMEHIEJEDRVVMSTQASROWWMADDPNGXMCMYFCTRXODNA 1.30 0.47
RVKNTJDLHNLHODWEIGXXHCCOGAZQIFOFXWJPUUFUQOVYSFENYXJSUNDQESHDL 1.17 0.11
RYQAALUJBCNPFEHQWYTQIUSPFGLLCGHDXGAJWWOKENEFCYOVHGXGHQXURHTUC 1.36 0.99

SABKETAXTDIXENYTOTGIFVLULGFOQQJEIYGPAMRRRYBSCXNNXTYDWUIDR
BFOGPFMRUUFDRLUFJILAIGYNNSMVHRGEOATODWVLDISWXMMND 1.65 0.22

SDHVJMXJNYTYHPMPGZXFRUCGPONXGAGCVKNORVWCWDFHRJWJWDCBHIEY
XJJGAKNRCTJMKMTQEHIJUUPEFAGQHINMBRWDUTYIDWHLJGFFFV 1.41 0.91

SDKUSYQVHCLINLBFHWYDYBYIAAKVIBEUTKGIIHQCENPICOYNWAKJJXOYTYL
KVEQMWRAJITCEINHLIYLOWDMHNUHWPWQQJLNMWTLOQLMFUI 1.37 0.43

SFJBUTEWDSLJBIGEASAPGLXTCPQPBNBFMMQJFYECVALMNYGKEVZUMQRKUNUFMBPHUKD
KHGJAJKLTTORHEBIFDTQLCMNGGNNJISOBIOJVTL 1.30 0.06

SSAHQOJHWOCLGRCGOGUBHOJRGAVXWQJCSFKLBYWGBBKOKONDNQFQUEASCISKXJWBGINOUM
YHRULGSPEATARYGOMQVRUQYKPSNVFTLFADPW 1.19 0.03

SUTYTGVRGAPBEYAVPTRWRNFJSDDMUAVSCIOOXVLDHLOMNCLYGELIMXACINABQQAIKELSKDBHO
GWAUPTCTMSBFEWYOPHTOPIXHTUESYNLDU 1.39 0.37

TSTDKLCSQYLOIQFVLHHXTBDGYMGKRJJBZAFDDDUGEKKWUWWVJVPJFTDBSBSNNIIXWMXLMPJEL
CNGTQHOCWFOYHLHIKZGSGQXSAQJGJJNPUQLKPCEQQWFKVQQOPSLYGPRWUQICYUFQ
IQWKBVGKRZCDTRSDMOQMENFDIXATYCBH 1.88 3.85

UBLFBKKFNATJVTJUKJFGALBIPQHVRUMAROTNVHBLAQVSHVHRGLFJAAABFRJFW 1.81 0.07
UKDGJLEXISFBNTDVQVMXQFXCGMQKOSUHDOFPYOOECJEEOPMNLSQCPEYJEYIGO 1.10 0.16
ULYMXJCIVBIBCHWVWSEEKQVKQIOJBQHQSHJOFBPPDWSMBTVSERGGYPPPATWSB 1.52 0.13
UMALDKMYNVYSDIAQVVHAWOJGDLFHKPSQEDJWPSOQCBFHGYMIVIXGXYCITMAXH 1.33 0.10

VBDIYXAGPEUVXXPCKVYCLPPRBAYDQTEIRDGMWBDIHIJAPGSLDYDLMTANMNMHNCLTYBJGXAYBJ
BLFJXDJRKOHCNPBIRTSLGSYSDOLUNAQTIJMNVOEIUOKAHLIXWPEMPQQFTMVPUFECMACT 1.69 0.60

VHETJPMKNMHQNUVBIOTFADJUXXIQRSGDSNXAGNEKPMCJRIDEHOFTVTPLWUFLNDCWLKHXKKE
LUMBDOHSIBSDEWWSPLQVMWLQERMCANDUBCXULQWYGOTLGLLQFSJGVQEUIEQQXXWGEKVTPA1.030.14

VMMKMRMREPCPYAFRGJBOTPHNQRFXVNSNYQVJNBXXPKRPVHFQAF

```
GFSTHYFKCIDWXOQJAKRFKHPYHNENTDQLJMQGMXTFPBDFUIPAIODWYAMTXJDIWGERTHD
OKWFXPLSWFYPXNQMNOBKILSG 1.84 0.44

YAMHDWIODTE 1.49 0.09
YAQCXBTDRHF 1.43 0.36
YBJSRTYCQLD 1.50 0.22
YDNBSGGTQDW 1.39 0.15
YEBYSXDGLVB 1.54 0.42
YELRVNTLOBU 1.34 0.21
YHKTEILDYRK 1.20 0.01
YLNVRNDFOTX 1.30 0.12
YOJMOEAKWYV 1.00 0.02
YSMLSCXTGBR 1.20 0.03
YVBTGBLJNUB 1.42 0.41
YWOABOXTDFD 1.48 0.32
ZAMWBPMDUAXGGLB 1.47 0.13
ZBBCMQDQKTJPSUR 1.15 0.02
ZCBINUXAKUFLEAK 1.59 0.06
ZCTFTHMKITVNFFG 1.69 0.08
ZEIVGPLCKGALTXT 1.41 0.01
ZFPEIBFLSVWGFPA 1.09 0.01
ZKRFMRHHAVUOIKM 1.16 0.02
ZLFBPYYLFUKMSED 1.18 0.01
ZMEROSWMYGDQTKO 1.87 0.01
ZMIVEQTDVEEOWVN 1.70 0.03
ZNFEMVRCISEHSIN 1.28 0.03
ZNVIDLJDFUUQUBK 1.04 0.03
ZOCGMCVDMPDFMHJ 1.64 0.06
ZRLDWMOJKXHBLAN 1.73 0.03
ZSVGLBHQEXQQMEE 1.21 0.01
[IAWNTNHKJOCOSY 1.32 0.33
[JJMLWFVTUKDTTQ 1.97 0.20
\ASEFAHYBJUVPHIMWYT 1.54 0.27
\BBTXCFCJUGDJXKONUH 1.56 0.12
\IUPGKKXXOTAUMPEPDN 1.30 0.29
\JISPFOVANDEBNXUWBY 1.48 0.21
\JJIASHWXGPXCGROCWG 1.36 0.17
\KBDABMSIBRWMQNMUKW 1.69 0.21
\LLXGLCOXAJDQJOICVC 1.70 0.10
\MEOANCSXDVGCVLGHQT 1.65 0.10
\NBIAYCYGUSVJFXEJGA 1.32 0.29
\NGPKOPDSBDCBGVBBPP 1.58 0.12
\SCNPBHGXICIDBPKQSI 1.48 0.20
\TOFXBCFVQIXXQUWDXB 1.41 0.19
\TTKFATYUYRXVWLWHQB 1.06 0.19
]CDUHTE 1.10 0.04
]FDDLCE 1.06 0.03
]HDFNTE 1.20 0.03
]HGEWRK 1.17 0.02
]KFUDSM 1.21 0.02
]KSFJWS 1.02 0.01
```

```
]LTDADG 1.09 0.03
]MSDXEI 1.03 0.03
]OTYGNE 1.43 0.05
]QFWSDE 1.16 0.02
]RBFMSH 1.10 0.02
]SBSMKG 1.34 0.04
]VEIVIK 1.01 0.01
]WXSAGJ 1.07 0.03
]YDUCBE 1.07 0.01
^EAVTPDJJI 2.28 0.20
^UTKLLWIDM 2.22 0.21
^VRDSNKGRT 2.28 0.24
```

The cases below represent a collection mostly of attractors, some of which are not strange, but which are special examples discussed in this book or other cases of historical or mathematical significance. You can enter them into the program manually using the **I** command or copy the file **SPECIAL.DIC** on the accompanying disk to **SA.DIC** and view them automatically using the **E** command. Note that the contents of any existing **SA.DIC** file will be lost when you do this unless you use the DOS command **COPY SA.DIC** + **SPECIAL.DIC SA.DIC** to append the new cases to the end of the **SA.DIC** file.

SPECIAL.DIC A list of important special cases

```
Logistic Map: AMu% 0.86 1.00

Delayed Logistic Map: EM4WM48bM 0.99 0.00

Tinkerbell Map: EMVWMGCMaMaRM 1.14 0.27

Hénon Map: EWM?MPM2WM4 1.20 0.60

Three-Fold Symmetric Icon: FMXUEM2IMEM5C=MXMG 0.79 0.32

Four-Fold Symmetric Icon: GMBMYOM2AMAM4OM5UAM3EBMYM 1.21 0.35

Five-Fold Symmetric Icon: HM4MjQLM2WM5M2jMHM3QM7H=MjMWM2]M4MjML 1.13 0.48

Lorenz Attractor: QMCM3WM5iM2L2M7NM4JM 2.21 0.15

Point Attractor: QMLM3NM5LM3LM14 0.00 -0.14

Rössler Attractor: QM5IM2IM2QM3NM4NM3QM30M 1.94 0.08

Point Repellor: QM5NM5LM18 1.00 0.00
```

Van der Pol Equation: RM11OM9KM2KM6OM28 1.04 0.00

Symmetric Limit Cycle: RMNMAM3AM3NM9LM2AM6NMAM26 1.04 0.00

2-Torus: VMNMAM4AM7NM19LM2AM11NMAM42NMAM2AOM28LM2AM2NMA 1.04 0.00

2-Torus (rotated): VMNMAM8AM13NM24NMAM6AM6OM3LM3AM20NMAM22LM3AM11NMA 1.04 0.00

Gingerbread Man: YCMC2M2WM3 1.42 0.11

Tent Map: YM2WM2VM39 1.23 1.00

Lozi Attractor: YWMW<M2RM3 1.67 0.68

Chirikov (Standard) Map: \MWM4dWqMW2M3dWq 1.13 0.04

Circle Map: \NWMyWM4NM2yWM4 1.16 1.28

Birkhoff's Bagel: ^BVKBDVHIK 1.90 0.22

Forced Damped Hard Spring: ^WCLM2KMWJ 1.01 -0.01

Forced Van der Pol Oscillator: ^WCMCMWMSD 1.68 0.00

Forced Damped Linear Spring: ^WCM3KMWJ 0.92 -0.07

Forced Damped Soft Spring: ^WCNM2JMWJ 1.01 -0.20

Duffing Two-Well Oscillator: ^W2CM2KMRD 2.84 0.15