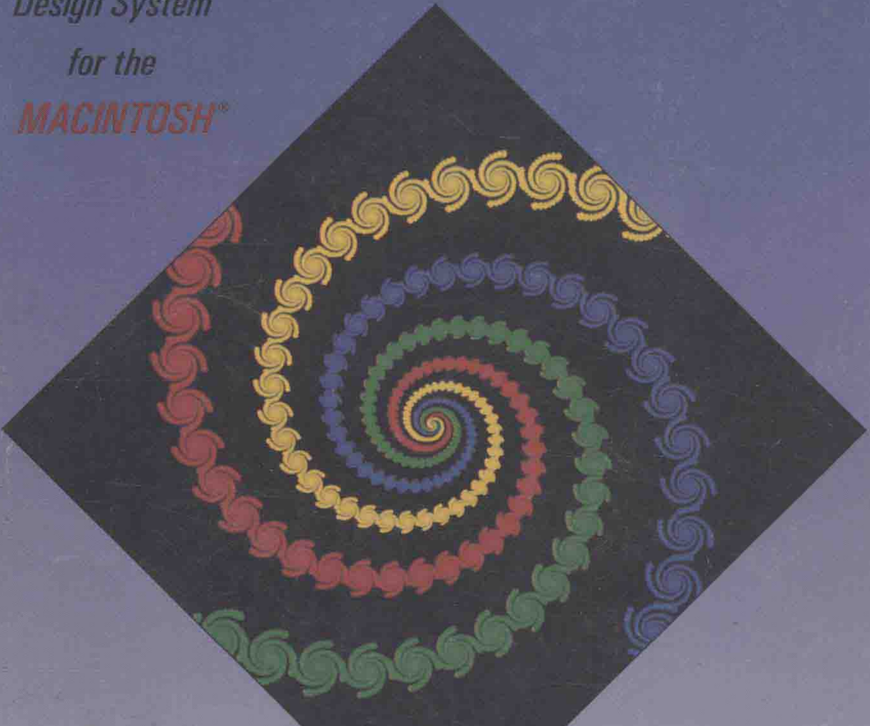
# FRACTAL ATTRACTION™

A Fractal Design System for the MACINTOSH®

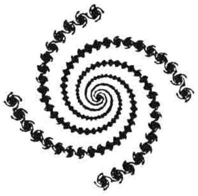


KEVIN D. LEE

YOSEF COHEN

# *Fractal Attraction*™

*A Fractal Design System for the **Macintosh**®*



KEVIN D. LEE

College of St. Catherine
St. Paul, Minnesota

*and*

Sandpiper Software Inc.
St. Paul, Minnesota

YOSEF COHEN

University of Minnesota
St. Paul, Minnesota

ACADEMIC PRESS

*Harcourt Brace Jovanovich, Publishers*

Boston San Diego New York
London Sydney Tokyo Toronto

This book is printed on acid-free paper. ♾

Produced in the United States of America.

ISBN 0-12-440740-4

# *Fractal Attraction*™
*A Fractal Design System for the* ***Macintosh***®

Some states do not allow limitation on how long an implied warranty lasts, not exclusions or limitations of incidental or consequential damages, so the above limitations and exclusions may not apply to you. This Warranty gives you specific legal rights, and you may also have other rights which vary from jurisdiction to jurisdiction.

THE RE-EXPORT OF UNITED STATES ORIGIN SOFTWARE IS SUBJECT TO THE UNITED STATES LAWS UNDER THE EXPORT ADMINISTRATION ACT OF 1969 AS AMENDED. ANY FURTHER SALE OF THE PRODUCT SHALL BE IN COMPLIANCE WITH THE UNITED STATES DEPARTMENT OF COMMERCE ADMINISTRATION REGULATIONS. COMPLIANCE WITH SUCH REGULATIONS IS YOUR RESPONSIBILITY AND NOT THE RESPONSIBILITY OF AP OR SANDPIPER.

# Table of Contents

§1. What is Fractal Attraction? .......... 1

*§1.1. Hardware and System Software Required* .......... *1*

*§1.2. Installing Fractal Attraction* .......... *1*

*§1.3. A Quick Tour* .......... *2*

§2. Mathematical Background .......... 7

*§2.1. Affine Transformations* .......... *7*

*§2.2. Iterated Function Systems (IFS)* .......... *13*

*§2.3. How to Design an Attractor* .......... *17*

§3. The Random Algorithm .......... 27

§4. Using the Design Window .......... 31

§5. Applying the Collage Theorem .......... 39

§6 Color Rendering Example .......... 47

§7. The Fractal Attraction Application .......... 51

*§7.1. The Windows* .......... *51*

*§7.2. The Menus* .......... *53*

§8. Literature Cited .......... 61

Appendix A: IFS Examples .......... 63

Appendix B: Rotation, Scale, & Translation Form .......... 65

Appendix C: Changing Fractal Window Size .......... 67

# §1. What is Fractal Attraction?

*Fractal Attraction* is a Macintosh application with which you use Iterated Function Systems (IFS) to design your own fractals. As you will see, the ideas behind IFS, which were developed by Barnsley (1988) and co-workers are surprisingly simple, and yet, can produce pictures of incredible complexity and beauty.

The following pages first give a quick overview to using *Fractal Attraction*, then explain what IFS is, and what it does, and show you how to use *Fractal Attraction.* If you are familiar with IFS, you can certainly skip the appropriate sections (e.g. Mathematical Background), you may, however, want to skim through the sections which explain how to design your own fractals. If you are familiar with both IFS and the Macintosh you may want to skip the manual altogether, although *it is suggested that you read the sections on using the **Design** window, the collage example and, if you are using a color Mac, the section on color rendering.*

No apologies are made for the very basic and intuitive explication of the ideas. Throughout it is assumed that you are familiar with basic Macintosh concepts such as clicking, dragging, and menus.

## *§1.1. Hardware and System Software Required*

Two versions of *Fractal Attraction* are included on the master disk, standard and enhanced. The two versions are functionally equivalent except for speed, the enhanced version takes advantage of the math coprocessor and the faster processor on the newer machines. The standard version will run on any Macintosh from the Plus up, but it is intended for use only with the Plus,SE, or LC (or Mac IIsi if it does not have the optional math coprocessor). If you own an SE or Plus rest assured that you can still enjoy *Fractal Attraction* , much time was spent optimizing the standard version for speed.

If you are using the standard version on the LC or IIsi, it will automatically adjust to be in color.

It is suggested that you use system software 6.0 or later.

## *§1.2. Installing Fractal Attraction*

To install *Fractal Attraction* simply drag the appropriate version of the application for your machine and the two the folders named IFS Examples and Initial Figures onto your hard disk. For a floppy disk system copy to a floppy disk.

## *§1.3. A Quick Tour*

To quickly introduce you to the capabilities of *Fractal Attraction* we will create the so-called Sierpinski triangle. Hopefully this will encourage you to read further and find out why the twenty-one numbers that constitute our IFS example produce the Sierpinski triangle.

Start a session with *Fractal Attraction* by double-clicking on the *Fractal Attraction* icon. Three windows will appear. They are titled **Untitled** (no pun intended), **Design**, and **Fractal.** The **Untitled** window is where you enter and edit the IFS code in a spreadsheet like fashion; we will refer to this window as the **code** window. The title will change to match the name of the file. The **design** window also allows you to edit the IFS code but in a draw/paint like fashion. Using this window you can quickly and easily design a fractal. The final window, **Fractal,** is where the image specified by the IFS is plotted.

To illustrate the ease with which *Fractal Attraction* lets you design fractals we will start with the **design** window. Make the **design** window the active window either by clicking on it or choosing **Design** under the **Fractal** menu. You will notice two rectangles, a large one with a dotted outline and a smaller one inside the large one (see figure 1). This smaller one is actually a transformation (reduced copy) of the large rectangle (the result of an equation, but you can read about that later). Fractal Attraction automatically supplies one transformation. In the **design** window you will add two more to create the Siepinski triangle.

First move the first transformation to a new location, click on any edge of the small rectangle, it now becomes selected. Click again inside the figure but this time keep pressing the button and drag the rectangle to the position shown in figure 1.2.

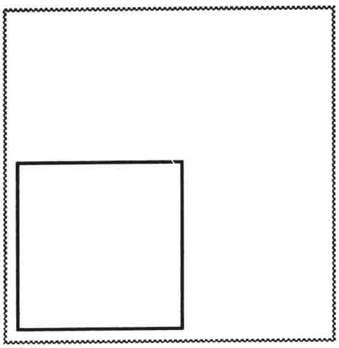


**Figure 1.1**

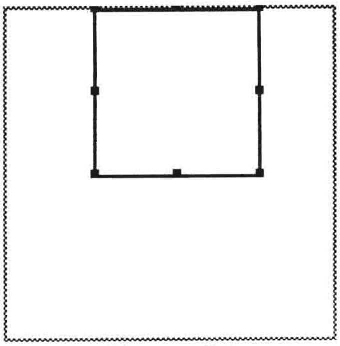


**Figure 1.2**

Now add a new transformation by selecting **New Transformation** under the **design** menu (see figure 1.3). Using your moving skills, position this new transformation as shown in figure 1.4.

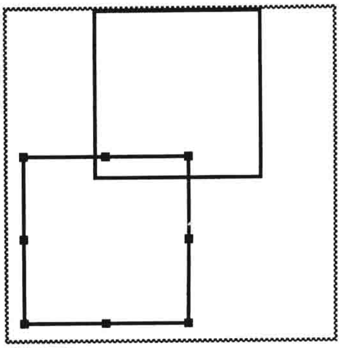


**Figure 1.3**

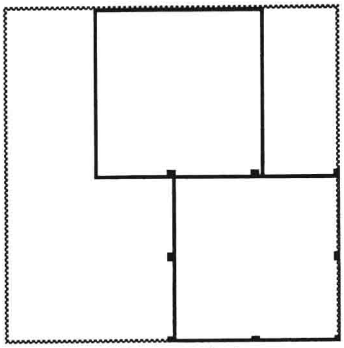


**Figure 1.4**

Add one more transformation. Position it as shown in figure 1.5. You have now built an IFS code (system of equations, i.e. the transformations) for generating a Sierpinski triangle.

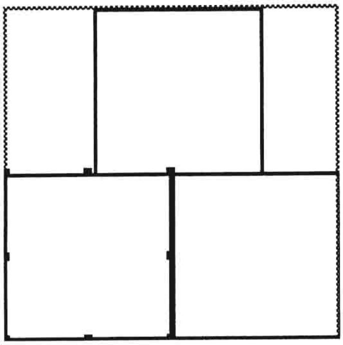
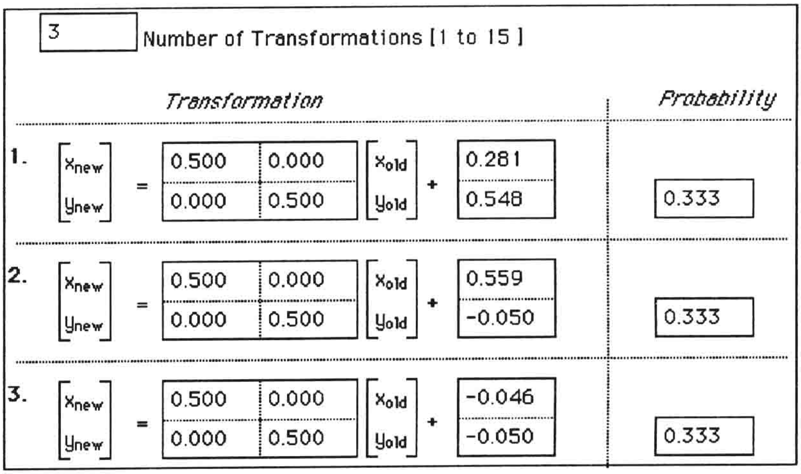


**Figure 1.5**

**Working in the Code Window:** Make the **code** window the active window and observe the IFS code you created. Your numbers will probably be slightly different from the numbers shown here. In this window you can edit the parameters (numbers) by pointing and clicking. Most of the time you will do your creating in the **design** window and use the **code** window for fine tuning. By the way, you have now specified a set of 3 equations which act on points in an initial picture (to be explained later).



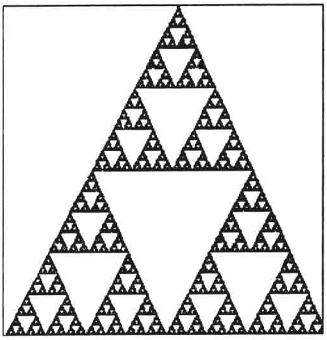
3 Number of Transformations [1 to 15]

| | Transformation | Probability |
|---|---|---|
| 1. | $\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} 0.500 & 0.000 \\ 0.000 & 0.500 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} 0.281 \\ 0.548 \end{bmatrix}$ | 0.333 |
| 2. | $\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} 0.500 & 0.000 \\ 0.000 & 0.500 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} 0.559 \\ -0.050 \end{bmatrix}$ | 0.333 |
| 3. | $\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} 0.500 & 0.000 \\ 0.000 & 0.500 \end{bmatrix} \begin{bmatrix} x_{old} \\ y_{old} \end{bmatrix} + \begin{bmatrix} -0.046 \\ -0.050 \end{bmatrix}$ | 0.333 |

The next item to observe in the **code** window is the coordinate system; that is the maximum and minimum values for the *x* and *y* in the picture to be produced. By default, *Fractal Attraction* scales the coordinate system for you (you will learn how to scale your own coordinates later). Scroll to the bottom of the **Untitled** window. You should see four cells labeled *x min*, *x max*, *y min*, and *y max*. They should be set to -0.1, 1.1, -0.1 and 1.1, respectively.

**Plotting the Image in the Fractal Window:** You may now proceed to plot the Sierpinski triangle. Bring the Fractal window to the front by either clicking on it or by choosing Fractal from under the Window menu. There are two algorithms implemented for plotting the fractal (that is, the picture specified by the equations): random and deterministic.

The random algorithm is faster—we will use it. Make sure **Random** is chosen (checked) under the **Fractal** menu. Now choose **Do 10,000**, also under the **Fractal** menu. *Fractal Attraction* is now drawing in the **Fractal** window. Eventually you should see the Sierpinski triangle as shown here. If you want a crisper image, choose **Do 10,000** again.



You may be wondering about the numbers in the bottom of the **fractal** window. Move the mouse over the fractal image and watch how the numbers change. The first pair of numbers tells you the *x* and *y* location of the mouse in the current coordinate system. The third number tells you how many times the point has been 'hit' (explained later). The fourth number tells how many iterations (more about iterations later) have been calculated. The final number is the number of seconds it took to generate the last set of iterations.

## More of the Design Window.

This is a good time to show off the power of *Fractal Attraction.* Make the **design** window the active window. Click on the the top rectangle, it will become selected and eight very small rectangles (handles) will appear. While pressing the option key click and drag one of the corner handles. This allows you to rotate the transformation, try rotating 45 degrees (see figure 1.6). Now make the **fractal** window active, and try running another 10000 points. (you may want to choose **clear** under the **edit** menu first). Did you get a fractal similar to figure 1.7? In the **design** window you can shrink, expand, rotate, flip and shear your transformation. Section 4, "Using the Design Window", explains all of these features in detail.

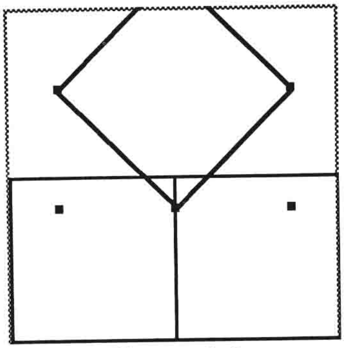


**Figure 1.6**

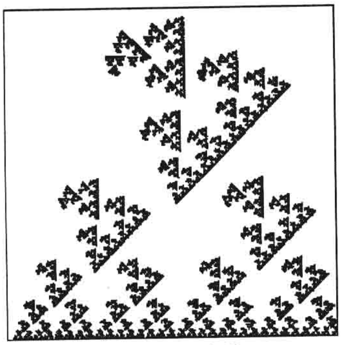
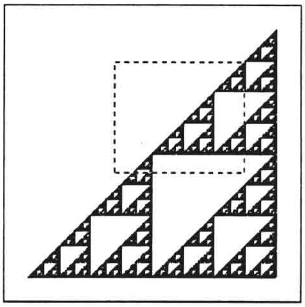
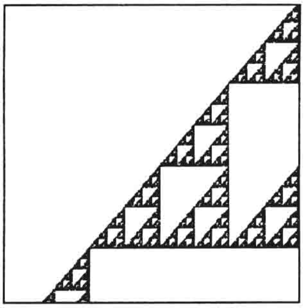


**Figure 1.7**

**Zooming In:** Once you have generated a fractal you can zoom in on a portion of it.


Before **Zoom In**


After **Zoom In**

To zoom in, select a portion of the fractal using the mouse (left drawing above). Choose **Zoom In** from under the **Fractal** menu. This will rescale the coordinate system so that the lower left and upper right coordinates of the selection rectangle now become the lower left and upper right coordinates of the window. The image looks rough (right drawing above) until you run the algorithm again at the higher magnification. Try it: run the algorithm for another 10,000 iterations. Do you see the self-similarity property of this picture? This self similarity is one of the distinguishing characteristics of fractals.

**Saving and Loading an IFS Code:** Make the **Untitled** window the active window. You can save this window by choosing **Save** under the **File** menu.

There are several example files in the folder titled IFS Examples. To access a file (which contains the IFS code) you must first close the active **code** window by choosing **Close** under the **File** menu or clicking in the close box of the window. After closing the **code** window (file), you can use the **Open** command under the **File** menu to look at another file. Choose another file and try 10,000 iterations (after making the **Fractal** window active, and making sure **Random** is checked). You may want to clear the **Fractal** window first by choosing **Clear** under the **Edit** menu.

This completes your quick introduction to *Fractal Attraction.* Hopefully you will find constructing IFS codes and generating images as fascinating as we have. To understand the theory behind the images and more details about the *Fractal Attraction* application read the next section, Mathematical Background. If you already have a background in transformational geometry and IFS codes you may want to skip to the section 4, "Using the **Design** Window".

# §2. Mathematical Background

This section will explain the ideas and principles behind generating fractals based on sets of equations. The presentation is at a very basic and intuitive level, and as you will hopefully find out for yourself, does not require sophisticated mathematics.

## *§2.1. Affine Transformations*

We shall start from the very beginning. Our discussion will be limited to the familiar *spaces* we are all used to: points, lines, planes, and so on. These are called *Euclidean* spaces. Consider a single point which constitutes a space. Since the point is all of the space, we need not describe its position. We then say that the *dimension* of a single-point space is 0. Next, consider a line, with an origin and a direction (Figure 2.1). The position of a point on the line can be described by a single "number": its distance from the origin, shown as $x_0$ in Figure 2.1. This is the familiar idea of a number-line from high school algebra. Since you need a single number to describe the position of a point on a line, you say, figuratively, that a number-line has a *dimension* of 1. You should be aware of the fact that mathematician sometimes think of spaces, numbers, distance, and dimensions in terms more abstract than their everyday use.

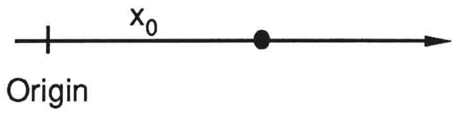


**Figure 2.1.**

Denote the distance of the point from the origin by $x_0$, and let us apply the following *transformation* which moves our point to a new location on the line:

$$x_1 = a\,x_0 \qquad (1)$$

where $a$ is a *parameter* (a real number) and $x_1$ is the new distance from the origin. The next position of the point (that is $x_1$) depends on the value of $a$. We have the following possibilities:

- if $a < -1$, $x_1$ will move to the left of the origin, and will be further away from the origin than $x_0$ was;
- if $a = -1$, $x_1$ will move to the left of the origin, and will be at the same distance as $x_0$ was;

- if $-1 < a < 0$, $x_1$ will move to the left of the origin and will be closer to it than $x_0$ was;
- if $a = 0$, $x_1$ will be at the origin;
- if $0 < a < 1$, $x_1$ will be to the right of and closer to the origin than $x_0$;
- if $a = 1$, the position of the point remains unchanged, that is $x_1 = x_0$; finally,
- if $a > 1$, $x_1$ will move further away from the origin and to the right.

We can now generalize our transformation to more than one step, and write equation (1) as:

$$x_{n+1} = a\, x_n \;,\quad x_0 \text{ given.} \tag{2}$$

Equation (2) allows us to calculate the new position of each point based on its previous position, given that the *initial* position, $x_0$, is known. The transformation (2) is also called a *map*, or an *iteration*, and it is linear in a sense that no powers of $x$ (other than 0 and 1, of course) are involved.

Next, suppose we want to move the point around on the line as in equation (2), but after calculating its new position we want to "nudge" it by a constant distant. We call this nudging *translation*, and we achieve it by adding a free parameter to equation (2) to get

$$x_{n+1} = a\, x_n + b \;,\quad x_0 \text{ given.} \tag{3}$$

where $b$ is a real number. For example, if $b = 1$, $x_{n+1}$ would be moved one unit to the right.

To describe the position of a point on a plane, we need two coordinates, $x$ and $y$, as shown in Figure 2.2. The distances $x_0$ and $y_0$ are also called the $x$,$y$ coordinates of the point, and by convention are written as $(x,y)$. Thus, if the position of the point shown in Figure 2.2 is, for example, 10 units along the $x$ axis and 15 units along the $y$ axis we write (10,15) for its coordinates. The position of a point is then described by its distance from the origin along the $x$-axis, denoted as $x_0$ in Figure 2.2, and along the $y$-axis, denotes as $y_0$ in Figure 2.2. We now have two independent directions in our space (plane), and we conclude that the dimensionality of a plane is 2. Next, let us see what we can do with transformations of points on the plane.

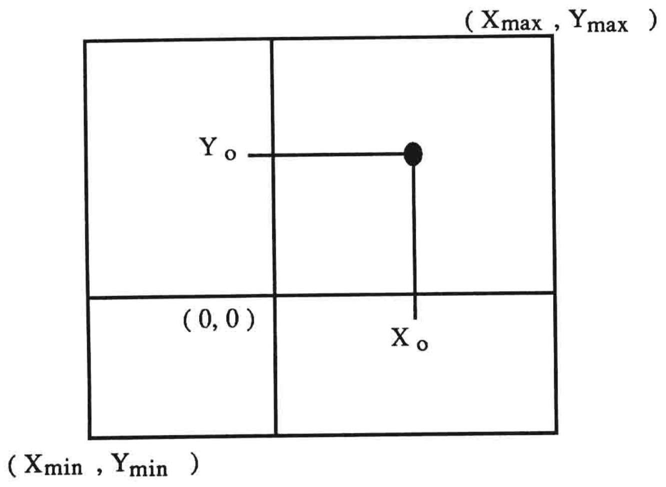


**Figure 2.2.**

Similar to equation (3), we write

$$\left.\begin{aligned} x_{n+1} &= a\,x_n + b\,y_n + e \quad x_0 \text{ given} \\ y_{n+1} &= c\,x_n + d\,y_n + f \quad y_0 \text{ given} \end{aligned}\right\} \qquad (4)$$

where the parameters *a*, *b*, *c*, *d*, *e*, and *f* are all real numbers. We can now move the point (shown in Figure 2.2) around on the plane to wherever we desire by playing with the parameter values. For example, if we set all of the parameters to zero, one iteration suffices to move any point in the plane to the origin, where the point will stay for as many iterations as we apply (that is for any *n*). We can also move to the origin "slowly" by choosing *a*, *b*, *c*, and *d* to all be strictly between -1 and 1, and setting $e = f = 0$. We can also translate our original point to any place we wish on the plane by choosing $-1 < a, b, c,$ and $d < 1$ and setting *e* and *f* to the appropriate values (found by solving $x^* = ax^* + by^* + e$ and $y^* = cx^* + dy^* + f$ with $(x^*, y^*)$ being our final point, often called the fixed point) . Note also that—with an appropriate choice of the parameters *a*, *b*, *c*, and *d*—we can move the point $(x,y)$ to the *fixed point* rapidly or slowly. Finally, note that if we choose the parameters as described here, no matter where we start on the plane, we will wind up at the same point (provided that we apply the iteration (4) enough times).

We can now write equation (4) more generally in matrix notation as

$$W\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} e \\ f \end{pmatrix} \qquad (5)$$

where $W$ is an *affine transformation*. Equation (5) is just a short hand notation for equation (4). As you will see momentarily, affine transformations preserve images in space, but in general contract or expand, translate and rotate these images. A contraction can be achieved, for example, by setting the absolute values of parameters $a$, $b$, $c$, and $d$ all $< 1$. The translation and rotation are achieved by appropriate values of all of the parameters. To produce fractals we shall use contractive transformations (in a sense that pairs of points get closer to each other as the number of iterations increases).

Thus far, we discussed transformations on a single point. But there is nothing to prevent us from applying $W$ to a *set* (collection) of points. If $W$ is a *contraction mapping* in a sense that it successively contracts the size of the initial set after each iteration, this will result in a new set of points closer to each other. Applying $W$ many times, we should eventually obtain a single point, called the *fixed-point* of the affine transformation $W$. To see this, consider the following 2 examples.

Start with a set of points which form the sides of a square, and denote that square by $S_0$ (Figure 2.3). Now to each point on the edge of the square (keep in mind that the interior of the square is not part of our initial set; the edges are) apply the transformation

$$W\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.75 & 0.00 \\ 0.00 & 0.75 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} 0.00 \\ 0.00 \end{pmatrix} \qquad (6)$$