(d) $\det(A+B)=\det(A)+\det(B)$

(e) If matrix $B$ is obtained from $A$ by replacing one row of $A$ by a number $k$ times the corresponding row of $A$, then $\det(B)=k\det(A)$.

(f) If one row of $A$ is a constant multiple of another row of $A$ then $\det(A)=0$.

(g) Two of these identities are not quite correct, in general, but they can be corrected using another of these identities that is correct. Elaborate on this.

**Suggestion:** For part (f) automate the experiment as follows: After a random integer matrix $A$ is built, randomly select a first row number $r1$ and a different second row number $r2$. Then select randomly an integer $k$ in the range

$[-9, 9]$. Replace the row $r2$ with $k$ times row $r1$. This will be a good possible way to create your test matrices. Use a similar selection process for part (e).

11. (a) Prove that matrix addition is commutative $A+B=B+A$ whenever $A$ and $B$ are two matrices of the same size. (This is identity (5) in the text.)
(b) Prove that matrix addition is associative, $(A+B)+C=A+(B+C)$ whenever $A$, $B$, and $C$ are matrices of the same size. (This is the first part of identity (6) in the text.)

12. (a) Prove that the distributive laws for matrices: $A(B+C)=AB+AC$ and $(A+B)C=AC+BC$, whenever the matrices are of appropriate sizes so that a particular identity makes sense. (These are identities (7) in the text.)
(b) Prove that for any real number $\alpha$, we have that $\alpha(A+B)=\alpha A+\alpha B$, whenever $A$, $B$, and $C$ are matrices of the same size and that $\alpha(AB)=(\alpha A)B=A(\alpha B)$ whenever $A$ and $B$ are matrices with $AB$ defined. (These are identities (8) in the text.)

13. Prove that matrix multiplication is associative, $(AB)C=A(BC)$ whenever $A$, $B$, and $C$ are matrices so that both sides are defined. (This is the second part of identity (6) in the text.)

14. (*Discovering Facts about Matrices*) As we have seen, many matrix rules closely resemble corresponding rules of arithmetic. But one must be careful since there are some exceptions. One such notable exception we have encountered is that, unlike regular multiplication, matrix multiplication is not commutative; that is, in general we cannot say that $AB=BA$. For each of the statements below about matrices, either give a counterexample, if it is false, or give a proof if it is true. In each identity, assume that the matrices involved can be any matrices for which the expressions are all defined. Also, we use 0 to denote the zero matrix (i.e., all entries are zeros).
(a) $0A=0$.
(b) If $AB=0$, then either $A=0$ or $B=0$.
(c) If $A^2=0$, then $A=0$.
(d) $(AB)'=B'A'$ (recall $A'$ denotes the transpose of $A$).
(e) $(AB)^2=A^2B^2$.
(f) If $A$ and $B$ are invertible square matrices, then so is $AB$ and $(AB)^{-1}=B^{-1}A^{-1}$.
**Suggestion:** If you are uncertain of any of these, run some experiments first (as shown in some of the preceding exercises). If your experiments produce a counterexample, you have disproved the assertion. In such a case you merely record the counterexample and move on to the next one.

## 7.2: INTRODUCTION TO COMPUTER GRAPHICS AND ANIMATION

**Computer graphics** is the generation and transformation of pictures on the computer. This is a hot topic that has important applications in science and business as well as in Hollywood (computer special effects and animated films). In this section we will show how matrices can be used to perform certain types of geometric operations on "objects." The objects can be either two- or three-dimensional but most of our illustrations will be in the two-dimensional plane. For two-dimensional objects, the rough idea is as follows. We can represent a basic object in the plane as a MATLAB graphic by using the command `plot(x,y)`, where $x$ and $y$ are vectors of the same length. We write $x$ and $y$ as row vectors, stack $x$ on top of $y$, and we get a $2\times n$ matrix $A$ where $n$ is the common length of $x$ and $y$. We can do certain mathematical operations to this matrix to change it into a new matrix $A1$, whose rows are the corresponding vertex vectors $x1$ and $y1$. If we look at `plot(x1,y1)`, we get a transformed version of the original geometrical object. Many interesting geometric transformations can be realized by simple matrix multiplications, but to make this all work nicely we will need to introduce a new artificial third row of the matrix $A$, that will simply consist of 1's. If we work instead with these so-called homogeneous coordinates, then all of the common operations of scaling (vertically or horizontally), shifting, rotating, and reflecting can be realized by matrix multiplications of these homogeneous coordinates. We can mix and repeat such transformations to get more complicated geometric transformations; and by putting a series of such plots together we can even make movies. Another interesting application is the construction of fractal sets. Fractal sets (or fractals) are beautiful geometric objects that enjoy a certain "self-similarity property," meaning that no matter how closely one magnifies and examines the object, the fine details will always look the same.

Polygons, which we recall are planar regions bounded by a finite set of line segments, are represented by their vertices. If we store the $x$-coordinates of these vertices and the $y$-coordinates of these vertices as separate vectors (say the first two rows of a matrix) preserving the order of adjacency, then MATLAB's `plot` command can easily plot the polygon.

**EXAMPLE 7.3:** We consider the following "CAT" polygon shown in Figure 7.2. Store the $x$-coordinates of the vertices of the CAT as the first row vector of a matrix $A$, and the corresponding $y$-coordinates as the second row of the same matrix in such a way that MATLAB will be able to reproduce the cat by plotting the second row vector of $A$ versus the first. Afterwards, obtain the plot from MATLAB.

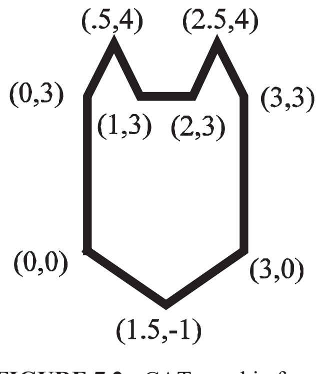


**FIGURE 7.2:** CAT graphic for Exampe 7.3.

SOLUTION: We can store these nine vertices in a $2\times 10$ matrix $A$ (the first vertex appears twice so the polygon will be closed when we plot it). We may start at any vertex we like, but we must go around the cat in order (either clockwise or counterclockwise). Here is one such matrix that begins at the vertex (0,0) and moves clockwise around the cat.

```
>> A=[0  0  .5  1  2  2.5  3  3  1.5  0; ...
     0  3  4   3  3   4   3  0  -1   0];
```

To reproduce the cat, we plot the second row of $A$ (the $y$'s) versus the first row (the $x$ 's):

```
>> plot(A(1,:), A(2,:))
```

In order to get the cat to fit nicely in the viewing area (recall, MATLAB always sets the view area to just accommodate all the points being plotted), we reset the viewing range to $-2 \le x \le 5, -3 \le y \le 6,$ and then use the `equal` setting on the axes so the cat will appear undistorted.

```
>> axis([-2 5 -3 6])
>> axis('equal')
```

The reader should check how each of the last two commands changes the cat graphic; we reproduce only the final plot in Figure 7.3(a). Figure 7.3 actually contains two cats, the original one (white) as well as a gray cat. The gray cat was obtained in the same fashion as the orginal cat, except that the `plot` command was replaced by the `fill` command, which works specifically with polygons and whose syntax is as follows:

| | |
|---|---|
| `fill(x,y,color)` → | Here `x` and `y` are vectors of the $x$- and $y$-coordinates of a polygon (preserving adjacency order); `color` can be either one of the predefined plot colors (as in Table 1.1) in single quotes, (e.g., `k` would be a black fill) or an RGB-vector $[r\ g\ b]$ (with $r$, $g$, and $b$ each being numbers in [0,1]) to produce any color; for example, `[.5 .5 .5]` gives medium gray. |

The elements $r$, $g$, and $b$ in a color vector determine the amounts of red, green, and blue to use to create a color; any color can be created in this way. For example, $[r\ g\ b] = [1\ 0\ 0]$ would be a pure-red fill; magenta is obtained with the rgb vector [1 0 1], and different tones of gray can be achieved by using equal amounts of red, green, and blue between [0 0 0] (black) and [1 1 1] (white).

For the gray cat in Figure 7.3(b), we used the command `fill(A(1,:), A(2,:), [.5 .5 .5])`. To get a black cat we could either set the rgb vector to [0 0 0] or replace it with `k`, which represents the preprogrammed color character for black (see Table 1.1).

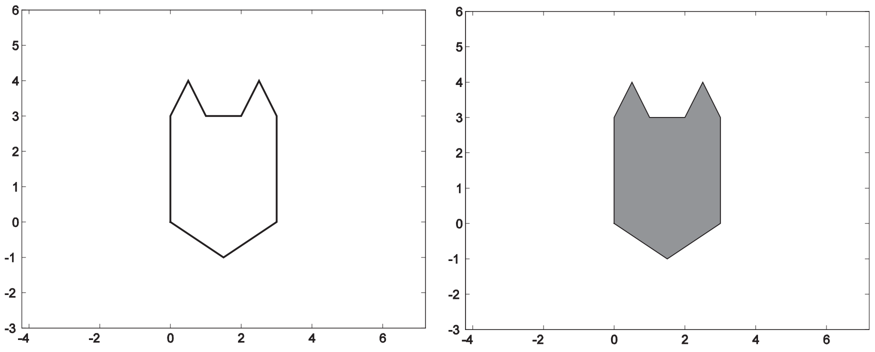


**FIGURE 7.3:** Two MATLAB versions of the cat polygon: (a) (left) the first white cat was obtained using the `plot` command and (b) (right) the second with the `fill` command.

EXERCISE FOR THE READER 7.3: After experimenting a bit with *rgb* color vectors, get MATLAB to produce an orange cat, a brown cat, and a purple cat. Also, try and find the rgb color vector that best matches the MATLAB built-in color cyan (from Table 1.1, the symbol for cyan is `c`).

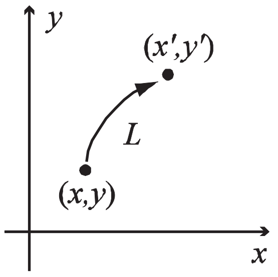


**FIGURE 7.4:** A linear tranformation $L$ in the plane.

A **linear transformation** $L$ on the plane $R^2$ corresponds to a $2\times 2$ matrix $M$ (Figure 7.4). It transforms any point $(x, y)$ (represented by the column vector $\begin{bmatrix} x \\ y \end{bmatrix}$) to the point $M\begin{bmatrix} x \\ y \end{bmatrix}$ obtained by multiplying it on the left by the matrix $M$. The reason for the terminology is that a linear transformation preserves the two important linear operations for vectors in the plane: addition and scalar multiplication. That is, letting $P_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$, $P_2 = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$ be two points in the plane (represented by column vectors), and writing $L(P) = MP$, the linear transformation axioms can be expressed as follows:

$$L(P_1 + P_2) = L(P_1) + L(P_2), \tag{9}$$

$$L(\alpha P_1) = \alpha L(P_1). \tag{10}$$

Both of these are to be valid for any choice of vectors $P_i (i = 1,2)$ and scalar $\alpha$. Because $L(P)$ is just $MP$ (the matrix $M$ multiplied by the matrix $P$), these two identities are consequences of the general properties (7) and (8) for matrices. By the same token, if M is any $n\times n$ matrix, then the transformation $L(P) = MP$

defines a linear transformation (satisfying (9) and (10)) for the space $\mathbf{R}^n$ of $n$-length vectors. Of course, most of our geomtric applications will deal with the cases $n = 2$ (the plane) or $n = 3$ (3-dimensional $(x, y, z)$ space).

Such tranformations and their generalizations are a basis for what is used in contemporary interactive graphics programs and in the construction of computer videos. If, as in the above example of the CAT, the vertices of a polygon are stored as columns of a matrix $A$, then, because of the way matrix multiplication works, we can transform each of the vertices at once by multiplying the matrix $M$ of a linear transformation by $A$. The result will be a new matrix containing the new vertices of the transformed graphic, which can be easily plotted.

We now move on to give some important examples of transformations on the plane $\mathbf{R}^2$.

(1) Scalings: For $a > 0,\ b > 0$ the linear transformation

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} ax \\ by \end{bmatrix}$$

will scale the horizontal direction with respect to $x = 0$ by a factor of $a$ and the vertical direction with respect to $y = 0$ by a factor of $b$. If either factor is < 1, there is contraction (shrinkage) toward 0 in the corresponding direction, while factors > 1 give rise to an expansion (stretching) away from 0 in the corresponding direction. As an example, we use $a = 0.3$ and $b = 1$ to rescale our original CAT (Figure 7.5). We assume we have left in the graphics window the first (white) cat of Figure 7.3(a).

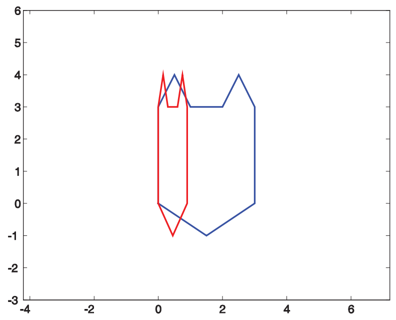


**FIGURE 7.5:** The scaling of the original cat using factors $a$ = 0.3 for horizontal scaling and $b$ = 1 (no change) for vertical scaling has produced the narrow-faced cat.

```
>>M=[.3 0; 0 1]; %store scaling matrix
>>A1=M*A; %create the vertex matrix of the transformed cat;
>>hold on  %leave the original cat in the window so we can compare
>>plot(A1(1,:), A1(2,:), 'r') %new cat will be in red
```

**Caution:** Changes in the axis ranges can also produce scale changes in MATLAB graphics.

(2) Rotations: For a rotation angle $\theta$, the linear tranformation that rotates a point $(x, y)$ an angle $\theta$ (counterclockwise) around the origin (0,0) is given by the following linear tranformation:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

(See Exercise 12 for a justification of this.) As an example, we rotate the original cat around the origin using angle $\theta = -\pi/4$ (Figure 7.6). Once again, we assume the graphics window initially contains the original cat of Figure 7.3 before we start to enter the following MATLAB commands:

```
>> M=[cos(-pi/4) -sin(-pi/4); sin(-pi/4) cos(-pi/4)];
>> A1=M*A;, hold on,plot(A1(1,:), A1(2,:), 'r')
```

(3) Reflections: The linear tranformation that reflects points over the $x$-axis is given by

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -x \\ y \end{bmatrix}.$$

Similary, to reflect points across the $y$-axis, the linear transformation will use the matrix $M = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$. As an example, we reflect our original CAT over the $y$-axis (Figure 7.7). We assume we have left in the graphics window the first cat of Figure 7.3.

```
>> M=[-1 0; 0 1];
>> A1=M*A; hold on, plot(A1(1,:), A1(2,:), 'r')
```

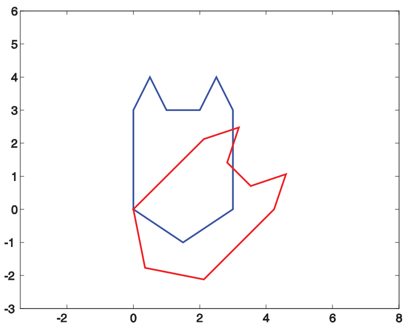


**FIGURE 7.6:** The rotation (red) of the original CAT (blue) using angle $\theta = -\pi/4$. The point of rotation is the origin (0,0).

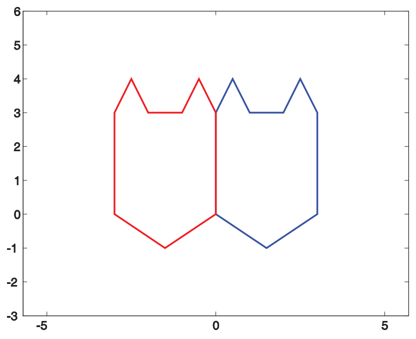


**FIGURE 7.7:** The reflection (left) of the original CAT across the y-axis.

(4) Shifts: Shifts are very simple and important transformations that are not linear transformations. For a fixed (shift) vector $V_0 = (x_0, y_0) \neq (0,0)$ that we identify, when convenient, with the column vector $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$, the **shift transformation** $T_{V_0}$ associated with the shift vector $V_0$ is defined as follows:

$$(x', y') = T_{V_0}(x, y) = (x, y) + V_0 = (x + x_0, y + y_0).$$

What the shift transformation does is simply move all $x$-coordinates by $x_0$ units and move all $y$-coordinates by $y_0$ units. As an example we show the outcome of applying the shift transformation $T_{(1,1)}$ to our familiar CAT graphic. Rather than a matrix multiplication with the $2\times10$ CAT vertex matrix, we will need to add the corresponding $2\times10$ matrix, each of whose 10 columns is the shift column vector $\begin{bmatrix}1\\1\end{bmatrix}$ that we are using (Figure 7.8). Once again, we assume the graphics window initially contains the original (white) CAT of Figure 7.3 before we start to enter the following MATLAB commands (and that the CAT vertex matrix $A$ is still in the workspace).

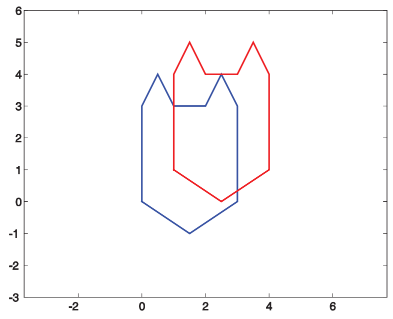


**FIGURE 7.8:** The shifted CAT (upper right) came from the original CAT using a shift vector (1,1). So the cat was shifted one unit to the right and one unit up.

```
>>size(A) %check size of A    →ans = 2    10
>> V=ones(2,10); A1=A+V; hold on, plot(A1(1,:),A1(2,:), 'r')
```

EXERCISE FOR THE READER 7.4: Explain why the shift transformation is never a linear transformation.

It is unfortunate that the shift transformation cannot be realized as a linear transformation, so that we cannot realize it as using a $2\times2$ matrix multiplication of our vertex matrix. If we could do this, then all of the important transformations mentioned thus far could be done in the same way and it would make combinations (and in particular making movies) an easier task. Fortunately there is a way around this using so-called homogeneous coordinates. We first point out a more general type of transformation than a linear transformation that includes all linear transformations as well as the shifts. We define it only on the two-dimensional space $\mathbf{R}^2$, but the definition carries over in the obvious way to the three-dimensional space $\mathbf{R}^3$ and higher-dimensional spaces as well. An **affine transformation** on $\mathbf{R}^2$ equals a linear tranformation and/or a shift (applied together). Thus, an affine transformation can be written in the form:

$$\begin{bmatrix}x'\\y'\end{bmatrix} = M\begin{bmatrix}x\\y\end{bmatrix} + V_0 = \begin{bmatrix}a & b\\c & d\end{bmatrix}\begin{bmatrix}x\\y\end{bmatrix} + \begin{bmatrix}x_0\\y_0\end{bmatrix}. \tag{11}$$

The **homogeneous coordinates** of a point/vector $\begin{bmatrix}x\\y\end{bmatrix}$ in $\mathbf{R}^2$ is the point/vector $\begin{bmatrix}x\\y\\1\end{bmatrix}$ in $\mathbf{R}^3$. Note that the third coordinate of the identified three-dimensional point is always 1 in homogeneous coordinates. Geometrically, if we identify a point $(x, y)$ of $\mathbf{R}^2$ with the point $(x, y, 0)$ in $\mathbf{R}^3$ (i.e., we identify $\mathbf{R}^2$ with the plane $z = 0$ in $\mathbf{R}^3$), then homogeneous coordinates simply lift all of these points up one unit to the plane $z = 1$. It may seem at first glance that homogeneous coordinates are making things more complicated, but the advantage in computer graphics is given by the following result.

**THEOREM 7.2:** (*Homogeneous Coordinates*) Any affine transformation on $\mathbf{R}^2$ is a linear transformation if we use homogeneous coordinates. In other words, any affine transformation $T$ on $\mathbf{R}^2$ can be expressed using homogeneous coordinates in the form:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T\left(\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right) = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{12}$$

(matrix multiplication), where $H$ is some $3 \times 3$ matrix.

*Proof:* The proof of the theorem is both simple and practical; it will show how to form the matrix $H$ in (12) from the parameters in (11) that determine the affine transformation.

*Case 1:* $T$ is a linear transformation on $\mathbf{R}^2$ with matrix $M = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, i.e., $T\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = M\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}$ (no shift). In this case, the transformation can be expressed in homogeneous coordinates as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T\left(\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right) = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{13}$$

To check this identity, we simply perform the matrix multiplication:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + 0 \\ cx + dy + 0 \\ 0 + 0 + 1 \end{bmatrix} \Rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} ax + by \\ cx + dy \end{bmatrix} = M\begin{bmatrix} x \\ y \end{bmatrix},$$

as desired.

*Case 2:* T is a shift transformation on $\mathbf{R}^2$ with shift vector $V_0 = \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$, that is, $T\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$ (so the matrix $M$ in (12) is the identity matrix). In this case, the transformation can be expressed in homogeneous coordinates as:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T\left(\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right) = \begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{14}$$

We leave it to the reader to check, as was done in Case 1, that this homogeneous coordinate linear transformation does indeed represent the shift.

*Case 3:* The general case (linear transformation plus shift);

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = T\left(\begin{bmatrix} x \\ y \end{bmatrix}\right)\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix},$$

can now be realized by putting together the matrices in the preceding two special cases:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T\left(\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right) = \begin{bmatrix} a & b & x_0 \\ c & d & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{15}$$

We leave it to the reader check this (using the distributive law (7)).

The basic transformations that we have so far mentioned can be combined to greatly expand the mutations that can be performed on graphics. Furthermore, by using homogeneous coordinates, the matrix of such a combination of basic transformations can be obtained by simply multiplying the matrices by the individual basic transformations that are used, in the correct order, of course. The next example illustrates this idea.

**EXAMPLE 7.4:** Working in homogeneous coordinates, find the transformation that will rotate the CAT about the tip of its chin by an angle of $-90°$. Express the transformation using the $3\times 3$ matrix $M$ for homogeneous coordinate multiplication, and then get MATLAB to create a plot of the transformed CAT along with the original.

SOLUTION: Since the rotations we have previously introduced will always rotate around the origin (0,0), the way to realize this transformation will be by combining the following three transformations (in order):
(i) First shift coordinates so that the chin gets moved to (0,0). Since the chin has coordinates $(1.5, -1)$, the shift vector should be the opposite so we will use the shift transformation

$$T_{(-1.5,1)} \sim \begin{bmatrix} 1 & 0 & -1.5 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} = H_1$$

(the tilde notation is meant to indicate that the shift transformation is represented in homogeneous coordinates by the given $3\times 3$ matrix $H_1$, as specified by (14)).

(ii) Next rotate (about (0,0)) by $\theta = -90°$. This rotation transformation $R$ has matrix

$$\begin{bmatrix} \cos(-90°) & -\sin(-90°) \\ \sin(-90°) & \cos(-90°) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix},$$

and so, by (13), in homogeneous coordinates is represented by

$$R \sim \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} = H_2 .$$

(iii) Finally we undo the shift that we started with in (i), using

$$T_{(1.5,-1)} \sim \begin{bmatrix} 1 & 0 & 1.5 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix} = H_3 .$$

If we multiply each of these matrices (in order) on the left of the original homogeneous coordinates, we obtain the transformed homogeneous coordinates:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H_3 H_2 H_1 \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix},$$

that is, the matrix $M$ of the whole transformation is given by the product $H_3 H_2 H_1$. We now turn things over to MATLAB to compute the matrix $M$ and to plot the before and after plots of the CAT.

```
>> H1=[1 0 -1.5; 0 1 1; 0 0 1]; H2=[0 1 0; -1 0 0; 0 0 1];
>> H3=[1 0 1.5;0 1 -1; 0 0 1];
>> format rat %will give a nicer display of the matrix M
>> M=H3*H2*H1
→M =       0        1        5/2
          -1        0        1/2
           0        0        1
```

We will multiply this matrix $M$ by the matrix $AH$ of homogeneous coordinates corresponding to the matrix $A$. To form $AH$, we simply need to tack on a row of ones to the bottom of the matrix $A$. (See Figure 7.9.)

```
>> AH=A; %start with A
>> size(A) %check the size of A
→ans = 2   10
>> AH(3,:)=ones(1,10); %form the
>> %appropriately sized third row
>> %for AH
>> size(AH)  →ans = 3    10
>> hold on, AH1=M*AH;
>> plot(AH1(1,:), AH1(2,:), 'r')
```

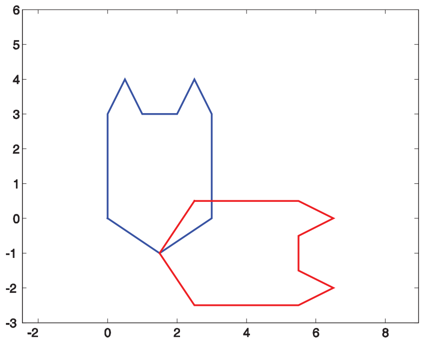


**FIGURE 7.9:** The red CAT was obtained from the blue cat by rotating –90° about the chin. The plot was obtained using homogeneous coordinates in Example 7.3.

EXERCISE FOR THE READER 7.5: Working in homogeneous coordinates, (a) find the transformation that will shift the CAT one unit to the right and then horizontally expand it by a factor of 2 (about $x = 0$) to make a "fat CAT". Express the transformation using the $3\times 3$ matrix $M$ for homogeneous coordinate multiplication, and then use MATLAB to create a plot of the transformed fat cat along with the original.
(b) Next, find four transformations each shifting the cat by one of the following shift vectors $(\pm 1, \pm 1)$ (so that all four shift vectors are used) after having rotated the CAT about the central point (1.5, 1.5) by each of the following angles: $30°$ for the upper-left CAT, $-30°$ for the upper-right CAT, $45°$ for the lower-left cat, and $-45°$ for the lower-right cat. Then fill in the four cats with four different (realistic cat) colors, and include the graphic.

We now show how we can put graphics transformations together to create a movie in MATLAB. This can be done in the following two basic steps:

**STEPS FOR CREATING A MOVIE IN MATLAB:**

| |
|---|
| *Step 1:* Construct a sequence of MATLAB graphics that will make up the frames of the movie. After the *j*th frame is constructed, use the command `M(:,j)= getframe;` to store the frame as the *j*th column of some (movie) matrix $M$. |
| *Step 2:* To play the movie, use the command `movie(M, rep, fps)`, where `M` is the movie matrix constructed in step 1, `rep` is a positive integer giving the number of times the movie is to be (repeatedly) played, and `fps` denotes a positive integer giving the speed, in "frames per second," at which the movie is to be played. |

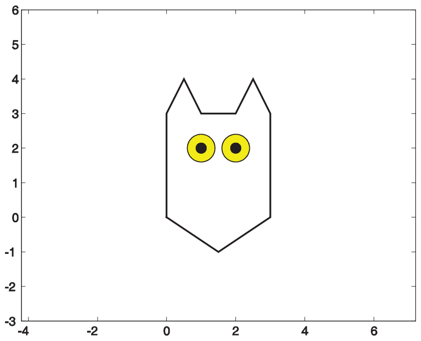


**FIGURE 7.10:** The original CAT of Example 7.3 with eyes added, the star of our first cat movie.

Our next example gives a very simple example of a movie. The movie star will of course be the CAT, but this time we will give it eyes (Figure 7.10). For this first example, we do not use matrix transformations, but instead we directly edit (via a loop) the code that generates the graphic. Of course, a textbook cannot play the movie, so the reader is encouraged to rework the example in front of the computer and thus replay the movie.

**EXAMPLE 7.5:** Modify the CAT graphic to have a black outline, to have two circular eyes (filled in with yellow), with two smaller black-filled pupils at the center of the eyes. Then make a movie of the cat closing and then reopening its eyes.

SOLUTION: The strategy will be as follows: To create the new CAT with the specified eyes, we use the "hold on" command after having created the basic CAT. Then we `fill` in yellow two circles of radius 0.4 centered at (1, 2) (left eye) and at (2, 2) (right eye); after this we fill in black two smaller circles with radii 0.15 at the same centers (for the pupils). The circles will actually be polygons obtained by parametric equations. To gradually close the eyes, we use a for loop to create CATs with the same outline but whose eyes are shrinking only in the vertical direction.

This could be done with homogeneous coordinate transforms (that would shrink in the $y$ direction each eye but maintain the centers—thus it would have to first shift the eyes down to $y = 0$, shrink and then shift back), or alternatively we could just directly modify the $y$ parametric equations of each eye to put a shrinking scaling factor in front of the sine function to turn the eyes both directly into a shrinking (and later expanding) sequence of ellipses. We proceed with the second approach. Let us first show how to create the CAT with the indicated eyes. We begin with the original CAT (this time with black line color rather than blue), setting the `axis` options as previously, and then enter `hold on`. Assuming this has been done, we can create the eyes as follows:

```
>> t=0:.02:2*pi;  %creates time vector for parametric equations
>> x=1+.4*cos(t); y=2+.4*sin(t); %creates circle for left eye
>> fill(x,y,'y') %fills in left eye
>> fill(x+1,y, 'y') %fills in right eye
>> x=1+.15*cos(t); y=2+.15*sin(t); %creates circle for left pupil
>> fill(x,y,'k') %fills in left pupil
>> fill(x+1,y,'k') %fills in right pupil
```

To make the frames for our movie (and to "get" them), we employ a for loop that goes through the above construction of the "CAT with eyes", except that a factor will be placed in front of the sine term of the $y$-coordinates of both eyes and pupils. This factor will start at 1, shrink to 0, and then expand back to the value of 1 again. To create such a factor, we need a function with starting value 1 that decreases to zero, then turns around and increases back to 1. One such function that we can use is $(1+\cos x)/2$ over the interval $[0, 2\pi]$. Below we give one possible implementation of this code:

```
>>t=0:.02:2*pi; counter=1;
>>A=[0  0  .5  1  2  2.5  3  3  1.5  0;... ...
     0  3  4   3  3   4   3  0  -1   0];
>>x=1+.4*cos(t); xp=1+.15*cos(t);
>>for s=0:.2:2*pi
 factor = (cos(s)+1)/2;
 plot(A(1,:), A(2,:), 'k')
 axis([-2 5 -3 6]), axis('equal')
 y=2+.4*factor*sin(t); yp=2+.15*factor*sin(t);
 hold on
 fill(x,y,'y'), fill(x+1,y, 'y'), fill(xp,yp,'k'), fill(xp+1,yp,'k')
 M(:, counter) = getframe;
 hold off, counter=counter+1;
end
```

The movie is now ready for screening. To view it the reader might try one (or both) of the following commands.

```
>> movie(M,4,5) %slow playing movie, four repeats
>> movie(M,20,75) %much faster play of movie, with 20 repeats
```

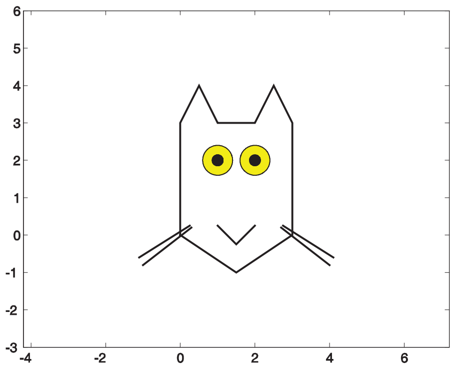


**FIGURE 7.11:** The more sophisticated cat star of the movie in Exercise for the Reader 7.7 (b).

EXERCISE FOR THE READER 7.6: (a) Create a MATLAB function M-file, called `mkhom(A)`, that takes a $2\times m$ matrix of vertices for a graphic (first row has $x$-coordinates and second row has corresponding $y$-coordinates) as input and outputs a corresponding $3\times m$ matrix of homogeneous coordinates for the vertices.
(b) Create a MATLAB function M-file, called `rot(Ah,x0,y0,theta)` that has inputs, `Ah`, a matrix of homogeneous coordinates of some graphic, two real numbers, `x0`, `y0` that are the coordinates of the center of rotation, and `theta` , the angle (in radians) of rotation. The output will be the homogeneous coordinate vertex matrix gotten from `Ah` by rotating the graph an angle `theta` about the point $(x0, y0)$.

EXERCISE FOR THE READER 7.7: (a) Recreate the above movie working in homogeneous coordinate transforms on the eyes.
(b) By the same method, create a similar movie that stars a more sophisticated cat, replete with whiskers and a mouth, as shown in Figure 7.11. In this movie, the cat starts off frowning and the pupils will shift first to the left, then to the right, then back to center and finally up, down and back to center again, at which point the cat will wiggle its whiskers up and down twice and change its frown into a smile.

**Fractals** or **fractal sets** are complicated and interesting sets (in either the plane or three-dimensional space) that have the **self-similarity property** that if one magnifies a certain part of the fractal (any number of times) the details of the structure will look exactly the same.

The computer generation of fractals is also a hot research area and we will look at some of the different methods that are extensively used. Fractals were gradually discovered by mathematicians who were specialists in set theory or function theory, including (among others) the very famous Georg F. L. P. Cantor (1845–1918, German), Waclaw Sierpinski (1882–1969, Polish), Gaston Julia (1893–1978, French) and Giuseppe Peano (1858–1932, Italian) during the late nineteenth and early twentieth centuries. Initially, fractals came up as being pathological objects without any type of unifying themes. Many properties of factals that have shown them to be so useful in an assortment of fields were discovered and popularized by the Polish/French mathematician Benoit Mandelbrot(Figure 7.12).[3] The precise definition of a fractal set takes a lot of preliminaries; we refer to the references, for example, that are cited in the footnote on this page for details. Instead of this, we will jump into some examples. The main point to keep in mind is that all of the examples we give (in the text as well as in the exercises) are actually impossible to print out exactly because of the self-similarity property; the details would require a printer with infinite resolution. Despite this problem, we can use loops or recursion with MATLAB to get some decent renditions of fractals that, as far as the naked eye can tell (your printer's resolution permitting), will be accurate illustrations.

Fractal sets are usually best described by an iterative procedure that runs on forever.

**EXAMPLE 7.6:** (*The Sierpinski Gasket*) To obtain this fractal set, we begin with an equilateral triangle that we illustrate in gray in Figure 7.13(a); we call this set the *zeroth generation*. By considering the midpoints of each of the sides of this triangle, we can form four (smaller) triangles that are similar to the original. One is upside-down and the other three have the same orientation as the original. We delete this central upside down subtriangle from the zeroth generation to form the *first generation* (Figure 7.13(b)).

**FIGURE 7.12**: Benoit Mandelbrot (b. 1924) Polish/French mathematician.

[3] Mandelbrot was born in Poland in 1924 and his family moved to France when he was 12 years old. He was introduced to mathematics by his uncle Szolem Mandelbrojt, who was a mathematics professor at the Collège de France. From his early years, though, Mandelbrot showed a strong preference for mathematics that could be applied to other areas rahter than the pure and rather abstruse type of mathematics on which his uncle was working. Since World War II was taking place during his school years, he often was not able to attend school and as a result much of his education was done at home through self-study. He attributes to this informal education the development of his strong geometric intuition. After earning his Ph.D. in France he worked for a short time at Cal Tech and the Institute for Advanced Study (Princeton) for postdoctoral work. He then returned to France to work at the Centre National de la Recherche Scientifique. He stayed at this post for only three years since he was finding it difficult to fully explore his creativity in the formal and traditional mathematics societies that dominated France in the mid-twentieth century (the "Bourbaki School"). He returned to the United States, taking a job as a research fellow with the IBM research laboratories. He found the atmosphere extremely stimulating at IBM and was able to study what he wanted. He discovered numerous applications and properties of fractals; the expanse of applications is well demonstrated by some of the other joint appointments he has held while working at IBM. These include Professor of the Practice of Mathematics at Harvard University, Professor of Engineering at Yale, Professor of Economics at Harvard, and Professor of Physiology at the Einstein College of Medicine. Many books have been written on fractals and their applications. For a very geometric and accessible treatment (with lots of beautiful pictures of fractals) we cite [Bar-93], along with [Lau-91]; see also [PSJY-92]. More analytic (and mathematically advanced) treatments are nicely done in the books [Fal-85] and [Mat-95].

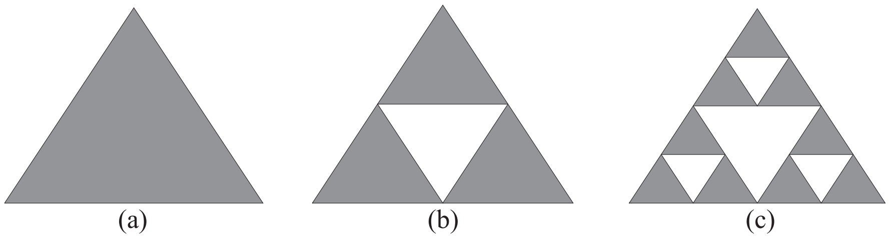


**FIGURE 7.13:** Generation of the Sierpinski gasket of Example 7.6: (a) the zeroth generation (equilateral triangle), (b) first generation, (c) second generation. The generations continue on forever to form the actual set.

Next, on each of the three (equilateral) triangles that make up this first generation, we again perform the same procedure of deleting the upside-down central subtriangle to obtain the generation-two set (Figure 7.13(c)). This process is to continue on forever and this is how the Sierpinski gasket set is formed. The sixth generation is shown in Figure 7.14.

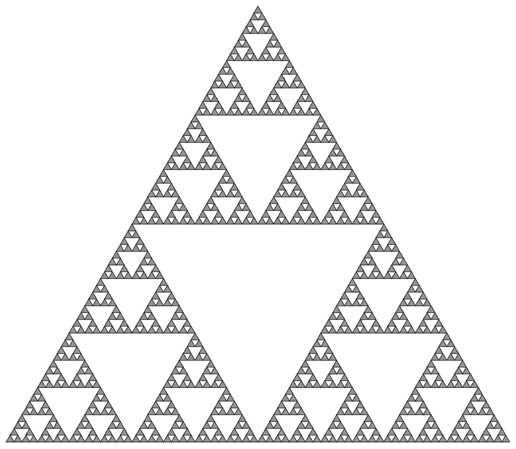


**FIGURE 7.14:** Sixth generation of the Sierpinski gasket fractal of Example 7.6.

Notice that higher generations become indistinguishable to the naked eye, and that if we were to focus on one of the three triangles of the first generation, the Sierpinski gasket looks the same in this triangle as does the complete gasket. The same is true if we were to focus on any one of the nine triangles that make up the second generation, and so on.

EXERCISE FOR THE READER 7.8: (a) Show that the *n*th generation of the Sierpinski triangle is made up of $3^n$ equilateral triangles. Find the area of each of these *n*th-generation triangles, assuming that the initial sidelengths are one.
(b) Show that the area of the Sierpinski gasket is zero.
NOTE: It can be shown that the Sierpinski gasket has dimension $\log 4/\log 3$ $=1.2618...$, where the *dimension* of a set is a rigorously defined measure of its

true size. For example, any countable union of line segments or smooth arcs is of dimension one and the inside of any polygon is two-dimensional. Fractals have dimensions that are nonintegers. Thus a fractal in the plane will have dimension somewhere (strictly) between 1 and 2 and a fractal in three-dimensional space will have dimension somewhere strictly between 2 and 3. None of the standard sets in two and three dimensions have this property. This noninteger dimensional property is often used as a definition for fractals. The underlying theory is quite advanced; see [Fal-85] or [Mat-95] for more details on these matters.

In order to better understand the self-similarity property of fractals, we first recall from high-school geometry that two triangles are similar if they have the same angles, and consequently their corresponding sides have a fixed ratio. A **similarity transformation** (or **similitude** for short) on $\mathbf{R}^2$ is an affine transformation made up of one or more of the following special transformations: scaling (with both $x$- and $y$-factors equal), a reflection, a rotation, and/or a shift. In homogeneous coordinates, it thus follows that a similitude can be expressed in matrix form as follows:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = T\left(\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}\right) = \begin{bmatrix} s\cos\theta & -s\sin\theta & x_0 \\ \pm s\sin\theta & \pm s\cos\theta & y_0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = H\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{16}$$

where $s$ can be any nonzero real number and the signs in the second row of $H$ must be the same. A scaling with both $x$- and $y$-factors being equal is customarily called a **dilation**.

EXERCISE FOR THE READER 7.9: (a) Using Theorem 7.2 (and its proof), justify the correctness of (16).
(b) Show that for any two similar triangles in the plane there is a similitude that transforms one into the other.
(c) Show that if any particular feature (e.g., reflection) is removed from the definition of a similitude, then two similar triangles in the plane can be found, such that one cannot be transformed to the other by this weaker type of transformation.

The self-similarity of a fractal means, roughly, that for the whole fractal (or at least a critical piece of it), a set of similitudes $S_1, S_2, \cdots, S_K$ can be found (the number $K$ of them will depend on the fractal) with the following property: All $S_j$'s have the same scaling factor $s < 1$ so that F can be expressed as the union of the transformed images $F_i = S_i(F)$ and these similar (and smaller) images are **essentially disjoint** in that different ones can have only vertex points or boundary edges in common. Many important methods for the computer generation of fractals will hinge on the discovery of these similitudes $S_1, S_2, \cdots, S_K$. Finding them also has other uses in both the theory and application of fractals. These concepts will be important in Methods 1 and 2 in our solution of the following example.

**EXAMPLE 7.7:** Write a MATLAB function M-file that will produce graphics for the Sierpinski gasket fractal.

SOLUTION: We deliberately left the precise syntax of the M-file open since we will actually give three different approaches to this problem and produce three different M-files. The first method is a general one that will nicely take advantage of the self-similarity of the Sierpinski gasket and will use homogeneous coordinate transform methods. It was, in fact, used to produce high-quality graphic of Figure 7.14. Our second method will illustrate a different approach, called the **Monte Carlo method**, that will involve an iteration of a random selection process to obtain points on the fractal, and will plot each of the points that get chosen. Because of the randomness of selection, enough iterations produce a reasonably representative sample of points on the fractal and the resulting plot will give a decent depiction of it. Monte Carlo is a city on the French Riviera known for its casinos (it is the European version of Las Vegas). The method gets its name from the random (chance) selection processes it uses. Our third method works similarly to the first but the ideas used to create the M-file are motivated by the special structure of the geometry, in this case of the triangles.

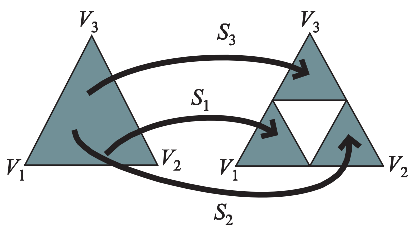


**FIGURE 7.15:** The three natural similitudes $S_1, S_2, S_3$ for the Sierpinski gasket with vertices $V_1, V_2, V_3$ shown on the zeroth and first generations. Since the zeroth generation is an equilateral triangle, so must be the three triangles of the first generation.

*Method 1:* The Sierpinski gasket has three obvious similitudes, each of which transforms it into one of the three smaller "carbon copies" of it that lie in the three triangles of the first generation (see Figure 7.15). These similitudes have very simple form, involving only a dilation (with factor 0.5) and shifts. The first transformation $S_1$ involves no shift. Referring to the figure, it is clear that $S_2$ must shift $V_1$ to the midpoint of the line segment $V_1V_2$ that is given by (as a vector) $(V_1+V_2)/2.$. The shift vector needed to do this, and hence the shift vector for $S_2$ is $(V_2-V_1)/2$. (Proof: If we shift $V_1$ by this vector we get $V_1+(V_2-V_1)/2=(V_2+V_1)/2$.) Similarly the shift vector for $S_3$ is $(V_3-V_1)/2$. It follows that the corresponding matrices for these three similitudes are as given below:

$$S_1 \sim \begin{bmatrix} .5 & 0 & 0 \\ 0 & .5 & 0 \\ 0 & 0 & 1 \end{bmatrix},\ S_2 \sim \begin{bmatrix} .5 & 0 & (V_2(1)-V_1(1))/2 \\ 0 & .5 & (V_2(2)-V_1(2))/2 \\ 0 & 0 & 1 \end{bmatrix},\ S_3 \sim \begin{bmatrix} .5 & 0 & (V_3(1)-V_1(1))/2 \\ 0 & .5 & (V_3(2)-V_1(2))/2 \\ 0 & 0 & 1 \end{bmatrix}.$$

Program 7.1, `sgasket1 (V1,V2,V3,ngen)`, has four input variables: `V1`, `V2`, `V3` should be the row vectors representing the vertices $(0,0)$, $(1,\sqrt{3}),(2,0)$ of a particular equilateral triangle, and `ngen` is the generation number of the Sierpinski gasket to be drawn. The program has no output variables, but will produce a graphic of this generation `ngen` of the Sierpinski gasket. The idea behind the algorithm is the following. The three triangles making up the generation-one gasket can be obtained by applying each of the three special similitudes $S_1,S_2,S_3$ to the single generation-zero Gasket. By the same token, each of the nine triangles that comprise the generation-two gasket can be obtained by applying one of the similitudes of $S_1,S_2,S_3$ to one of the generation-one triangles. In general, the triangles that make up any generation gasket can be obtained as the union of the triangles that result from applying each of the similitudes $S_1,S_2,S_3$ to each of the previous generation triangles. It works with the equilateral triangle having vertices $(0,0),(1,\sqrt{3}),(2,0)$. The program makes excellent use of recursion.

**PROGRAM 7.1:** Function M-file for producing a graphic of any generation of the Sierpinski gasket on the special equilateral triangle with vertices $(0,0),(1,\sqrt{3}),(2,0)$ (written with comments in a way to make it easily modified to work for other fractals).

```
function sgasket1(V1,V2,V3,ngen)
%input variables: V1,V2,V3 should be the vertices [0 0], [1,sqrt(3)],
%and [2,0] of a particular equilateral triangle in the plane taken as
%row vectors, ngen is the number of iterations to perform in
%Sierpinski gasket generation.
%The gasket will be drawn in medium gray color.

%first form matrices for similitudes
   S1=[.5 0 0;0 .5 0;0 0 1];
   S2=[.5 0 1; 0 .5 0;0 0 1];
   S3=[.5 0 .5; 0 .5 sqrt(3)/2;0 0 1];

if ngen == 0
   %Fill triangle
 fill([V1(1) V2(1) V3(1) V1(1)], [V1(2) V2(2) V3(2) V1(2)], [.5 .5
.5])
 hold on
else
%recursively invoke the same function on three outer subtriangles
%form homogeneous coordinate matrices for three vertices of triangle
   A=[V1; V2; V3]'; A(3,:)=[1 1 1];
   %next apply the similitudes to this matrix of coordinates
   A1=S1*A; A2=S2*A; A3=S3*A;
%finally, reapply sgasket1 to the corresponding three triangles with
%ngen bumped down by 1.  Note, vertex vectors have to be made into
%row vectors using '(transpose).
   sgasket1(A1([1 2],1)', A1([1 2],2)', A1([1 2],3)', ngen-1)
   sgasket1(A2([1 2],1)', A2([1 2],2)', A2([1 2],3)', ngen-1)
   sgasket1(A3([1 2],1)', A3([1 2],2)', A3([1 2],3)', ngen-1)
end
```

To use this program to produce, for example, the generation-one graphic of Figure 7.13(b), one need only enter:

```
>> sgasket1([0 0], [1 sqrt(3)], [2 0], 1)
```

If we wanted to produce a graphic of the more interesting generation-six Sierpinski gasket of Figure 7.15, we would have only to change the last input argument from 1 to 6. Note, however, that this function left the graphics window with a `hold on`. So before doing anything else with the graphics window after having used it, we would need to first enter `hold off`. Alternatively, we could also use the following command:

| | |
|---|---|
| `clf` → | Clears the graphics window. |

In addition to recursion, the above program makes good use of MATLAB's elaborate matrix manipulation features. It is important that the reader fully understands how each part of the program works. To this end the following exercise should be useful.

EXERCISE FOR THE READER 7.10: (a) Suppose the above program is invoked with these input variables: $V1 = [0\ 0]$, $V2 = [1\ \sqrt{3}\ ]$, $V3 = [2\ 0]$, $\text{ngen} = 1$. On the first run/iteration, what are the numerical values of each of the following variables: `A, A1, A2, A3, A1([1 2],2), A3([1 2],3)`?

(b) Is it possible to modify the above program so that after the graphic is drawn, the screen will be left with `hold off`? If yes, show how to do it; if not, explain.

(c) In the above program, the first three input variables $V1, V2, V3$ seem a bit redundant since we are forced to input them as the vertices of a certain triangle (which gave rise to the special similitudes $S1$, $S2$, and $S3$). Is it possible to rewrite the program so that it has only one input variable `ngen`? If yes, show how to do it; if not, explain.

*Method 2:* The Monte Carlo method also will use the special similitudes, but its philosophy is very different from that of the first method. Instead of working on a particular generation of the Sierpinki gasket fractal, it goes all out and tries to produce a decent graphic of the actual fractal. This gets done by plotting a representative set of points on the fractal, a random sample of such. Since so much gets deleted from the original triangle, a good question is What points exactly are left in the Sierpinski gasket? Certainly the vertices of any triangle of any generation will always remain. Such points will be the ones from which the Monte Carlo method samples. Actually there are a lot more points in the fractal than these vertices, although such points are difficult to write down. See one of the books on fractals mentioned earlier for more details.

Here is an outline of how the program will work. We start off with a point we call "Float" that is a vertex of the original (generation-zero) triangle, say *V1*. We then randomly choose one of the similitudes from $S_1, S_2, S_3$, and apply this to

"Float" to get a new point "New," that will be the corresponding vertex of the generation-one triangle associated with the similitude that was used (lower left for $S_1$, upper middle for $S_3$, and lower right for $S_2$). We plot "New," redefine "Float" = "New," and repeat this process, again randomly selecting one of the similitudes to apply to "Float" to get a new point "New" of the fractal that will be plotted. At the *N*th iteration, "New" will be a vertex of one of the *N*th-generation triangles (recall there are $3^N$ such triangles) that will also lie in one of the three generation-one triangles, depending on which of $S_1, S_2, S_3$ had been randomly chosen. Because of the randomness of choices at each iteration, the points that are plotted usually give a decent rendition of the fractal, as long as a large enough random sample is used (i.e., a large number of iterations).

**PROGRAM 7.2:** Function M-file for producing a Monte Carlo approximation graphic of Sierpinski gasket, starting with the vertices *V1, V2,* and *V3* of any equilateral triangle (written with comments in a way that will make it easily modified to work for other fractals).

```
function [ ] = sgasket2(V1,V2,V3,niter)
%input variables: V1,V2,V3 are vertices of an equilateral triangle in
%the plane taken as row vectors, niter is the number of iterations
%used to obtain points in the fractal.  The output will be a plot of
%all of the points.  If niter is not specified, the default value
%of 5000 is used.
%if only 3 input arguments are given (nargin==3), set niter to
%default
if nargin == 3, niter = 5000; end

%Similitude matrices for Sierpinski gasket.
S1=[.5 0 0;0 .5 0;0 0 1];
S2=[.5 0 (V2(1)-V1(1))/2; 0 .5 (V2(2)-V1(2))/2;0 0 1];
S3=[.5 0 (V3(1)-V1(1))/2; 0 .5 (V3(2)-V1(2))/2;0 0 1];

%Probability vector for Sierpinski gasket has equal probabilities
%(1/3)for choosing one of the three similitudes.
P = [1/3 2/3];

%prepare graphics window for repeated plots of points
clf, axis('equal'); hold on;

%introduce "floating point" (can be any vertex) in homogeneous
%coordinates
Float=[V1(1);V1(2);1];
i = 1; %initialize iteration counter

%Begin iteration for creating new floating points and plotting each
%one that arises.
while i <= niter
   choice = rand;
   if choice < P(1);
      New = S1 * Float;
      plot (New(1), New(2));
   elseif choice < P(2);
      New = S2 * Float;
      plot (New(1), New(2));
   else       New = S3 * Float;
```

```
      plot (New(1), New(2));
   end;
   Float=New;    i = i + 1;
end
hold off
```

Unlike the last program, this one allows us to input the vertices of any equilateral triangle for the generation-zero triangle. The following two commands will invoke the program first with the default 5000 iterations and then with 20,000 (the latter computation took several seconds).

```
>> sgasket2([0 0], [1 sqrt(3)], [2 0])
>> sgasket2([0 0], [1 sqrt(3)], [2 0], 20000)
```

The results are shown in Figure 7.16. The following exercise should help the reader better undertstand how the above algorithm works.

EXERCISE FOR THE READER 7.11: Suppose that we have generated the following random numbers (between zero and one): .5672, .3215, .9543, .4434, .8289, .5661 (written to 4 decimals).
(a) What would be the corresponding sequence of similitudes chosen in the above program from these random numbers?
(b) If we used the vertices [0 0], [1 sqrt(3)], [2 0] in the above program, find the sequence of different "Float" points of the fractal that would arise if the above sequence of random numbers were to come up.
(c) What happens if the vertices entered in the program `sgasket2` are those of a nonequilateral triangle? Will the output ever look anything like a Sierpinski gasket? Explain.

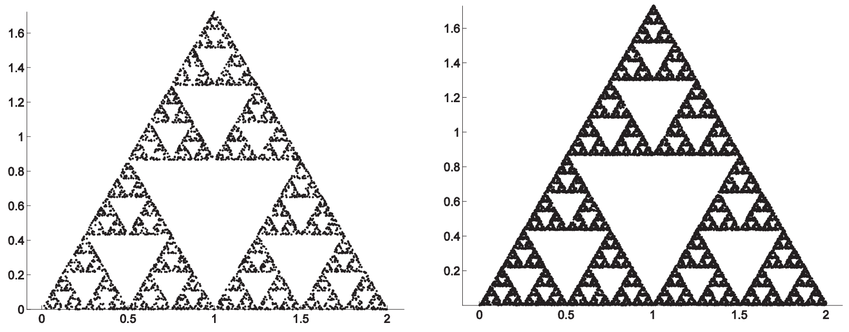


**FIGURE 7.16:** Monte Carlo renditions of the Sierpinski gasket via the program `sgasket2`. The left one (a) used 5000 iterations while the right one (b) used 20,000 and took noticeably more time.

*Method 3:* The last program we write here will actually be the shortest and most versatile of the three. Its drawback is that, unlike the other two, which made use of the specific similitudes associated with the fractal, this program uses the special geometry of the triangle and thus will be more difficult to modify to work for other fractals. The type of geometric/mathematical ideas present in this program, however, are useful in writing other graphics programs. The program `sgasket3(V1,V2,V3,ngen)` takes as input three vertices `V1`, `V2`, `V3` of a triangle (written as row vectors), and a positive integer `ngen`. It will produce a graphic of the `ngen`-generation Sierpinski gasket, as did the first program. It is again based on the fact that each triangle from a positive generation gasket comes in a very natural way from the triangle of the previous generation in which it lies. Instead of using similitudes and homogeneous coordinates, the program simply uses explicit formulas for the vertices of the ($N$ + 1)st-generation triangles that lie within a certain $N$th-generation triangle. Indeed, for any triangle from any generation of the Sierpinski gasket with vertices $V_1, V_2, V_3$, three subtriangles of this one form the next generation (see Figure 7.15), each has one vertex from this set, and the other two are the midpoints from this vertex to the other two. For example (again referring to Figure 7.15) the lower-right triangle will have vertices $V_2$, $(V_1 + V_2)/2$ = the midpoint of $V_2V_1$, and $(V_2 + V_3)/2$ = the midpoint of $V_2V_3$. This simple fact, plus recursion, is the idea behind the following program.

**PROGRAM 7.3:** Function M-file for producing a graphic of any generation of the Sierpinski gasket for an equilateral triangle with vertices *V1, V2*, and *V3*.

```
function sgasket3(V1,V2,V3,ngen)
%input variables: V1,V2,V3 are vertices of a triangle in the plane,
%written as row vectors, ngen is the generation of Sierpinski gasket
%that will be drawn in medium gray color.
if ngen == 0
%Fill triangle
 fill([V1(1) V2(1) V3(1) V1(1)],...
 [V1(2) V2(2) V3(2) V1(2)], [.5 .5 .5])
   hold on
   else
%recursively invoke the same function on three outer subtriangles
   sgasket3(V1, (V1+V2)/2, (V1+V3)/2, ngen-1)
   sgasket3(V2, (V2+V1)/2, (V2+V3)/2, ngen-1)
   sgasket3(V3, (V3+V1)/2, (V3+V2)/2, ngen-1)
end
```

EXERCISE FOR THE READER 7.12: (a) What happens if the vertices entered in the program `sgasket3` are those of a nonequilateral triangle? Will the output ever look anything like a Sierpinski gasket? Explain.
(b) The program `sgasket3` is more elegant than `sgasket1` and it is also more versatile in that the latter program applies only to a special equilateral triangle. Furthermore, it also runs quicker since each iteration involves less computing. Justify this claim by obtaining some hard evidence by running both programs (on the standard equilateral triangle of `sgasket1`) and comparing `tic/toc` and flop counts (if available) for each program with the following values for `ngen`: 1, 3, 6, 8, 10.

Since programs like the one in Method 3 of the above example are usually the most difficult to generalize, we close this section with yet another exercise for the reader that will ask for such a program to draw an interesting and beautiful fractal known as the **von Koch[4] snowflake,** which is illustrated in Figure 7.17. The iteration scheme for this fractal is shown in Figure 7.18.

EXERCISE FOR THE READER 7.13: Create a MATLAB function, call it `snow(n)`, that will input a positive integer `n` and will produce the *n*th generation of the so-called von Koch snowflake fractal. Note that we start off (generation 0) with an equilateral triangle with sidelength 2. To get from one generation to the next, we do the following: For each line segment on the boundary, we put up (in the middle of the segment) an equilateral triangle of 1/3 the sidelength. This construction is illustrated in Figure 7.18, which contains the first few generations of the von Koch snowflake. Run your program (and include the graphical printout) for the values: $n = 1$, $n = 2$, and $n = 6$.

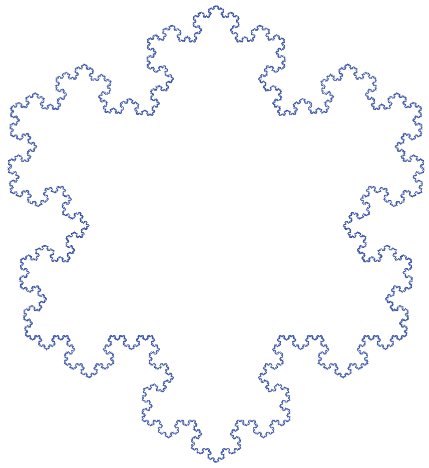


**FIGURE 7.17:** The von Koch snowflake fractal. This illustration was produced by the MATLAB program `snow(n)` of Exercise for the Reader 7.13, with an input value of 6 (generations).

**Suggestions:** Each generation can be obtained by plotting its set of vertices (using the plot command). You will need to set up a for loop that will be able to produce the next generation's vertices from those of a given generation. It is helpful to think in terms of vectors.

[4] The von Koch snowflake was introduced by Swedish mathematician Niels F. H. von Koch (1870–1924) in a 1906 paper *Une méthode géométrique élémentaire pour l'étude de certaines questions de la théorie des courbes planes*. In it he showed that the parametric equations for the curve ($x(t)$, $y(t)$) give an example of functions that are everywhere continuous but nowhere differentiable. Nowhere differentiable, everywhere continuous functions had been first discovered in 1860 by German mathematician Karl Weierstrass (1815–1897), but the constructions known at this time all involved very complicated formulas. Von Koch's example thus gives a curve (of infinite arclength) that is continuous everywhere (no breaks), but that does not have a tangent line at any of its points. The von Koch snowflake has been used in many areas of analysis as a source of examples.

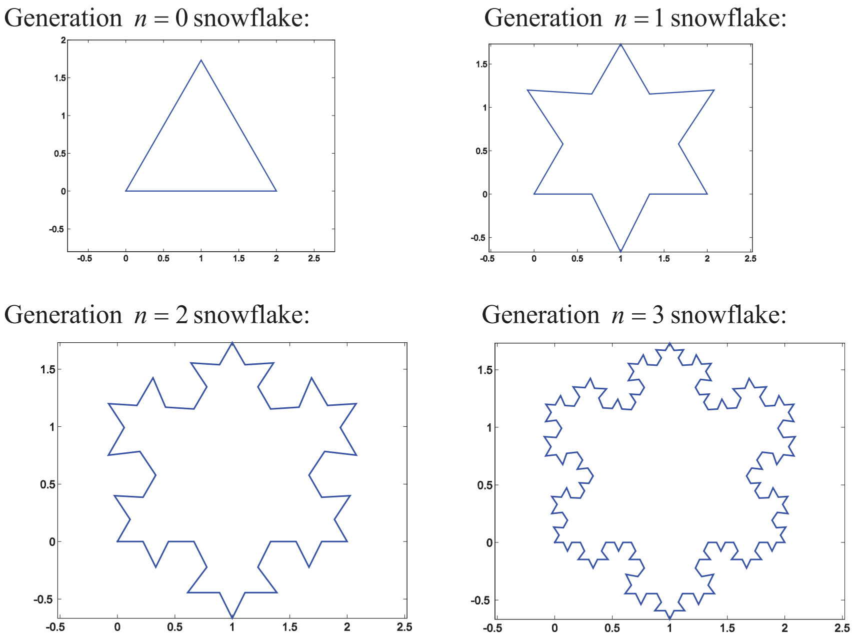


**FIGURE 7.18:** Some initial generations of the von Koch snowflake. Generation zero is an equilateral triangle (with sidelength 2). To get from any generation to the next, each line segment on the boundary gets replaced by four line segments each having 1/3 of the length of the original segment. The first and fourth segments are at the ends of the original segment and the middle two segments form two sides of an equilateral triangle that protrudes outward.

## EXERCISES 7.2:

NOTE: In the problems of this section, the "CAT" refers to the graphic of Example 7.2 (Figure 7.3(a)), the "CAT with eyes" refers to the enhanced version graphic of Example 7.5 (Figure 7.10), and the "full CAT" refers to the further enhanced CAT of Exercise for the Reader 7.7(b) (Figure 7.11). When asked to print a certain transformation of any particular graphic (like the CAT) along with the original, make sure to print the original graphic in one plotting style/color along with the transformed graphic in a different plotting style/color. Also, in printing any graphic, use the `axis(equal)` setting to prevent any distortions and set the axis range to accommodate all of the graphics nicely inside the bounding box

1. Working in homogeneous coordinates, what is the transformation matrix $M$ that will scale the CAT horizontally by a factor of 2 (to make a "fat CAT") and then shift the cat vertically down a distance 2 and horizontally 1 unit to the left? Create a before and after graphic of the CAT.

2. Working in homogeneous coordinates, what is the transformation matrix $M$ that will double the size of the horizontal and vertical dimensions of the CAT and then rotate the new CAT by an angle of $45°$ about the tip of its left ear (the double-sized cat's left ear, that is)? Include a before-and-after graphic of the CAT.

3. Working in homogeneous coordinates, what is the transformation matrix $M$ that will shift the left eye and pupil of the "CAT with eyes" by 0.5 units and then expand them both by a factor of

2 (away from the centers)? Apply this transformation just to the left eye. Next, perform the analogous transformation to the CAT's right eye and then plot these new eyes along with the outline of the CAT, to get a cat with big eyes.

4. Working in homogeneous coordinates, what is the transformation matrix $M$ that will shrink the "CAT with eyes"'s left eye and left pupil by a factor of 0.5 in the horizontal direction (toward the center of the eye) and then rotate them by an angle of $25°$? Apply this transformation just to the left eye, reflect to get the right eye, and then plot these two along with the outline of the CAT, to get a cat with thinner, slanted eyes.

5. (a) Create a MATLAB function M-file, called `reflx(Ah, x0)` that has inputs, `Ah`, a matrix of homogeneous coordinates of some graphic, and a real number `x0`. The output will be the homogeneous coordinate vertex matrix obtained from `Ah` by reflecting the graphic over the line $x = x0$. Apply this to the CAT graphic using $x0 = 2$, and give a before-and-after plot.
(b) Create a similar function M-file `refly(Ah, y0)` for vertical reflections (about the horizontal line $y = y0$) and apply to the CAT using $y0 = 4$ to create a before and after plot.

6. (a) Create a MATLAB function M-file, called `shift(Ah,x0,y0)` that has as inputs `Ah`, a matrix of homogeneous coordinates of some graphic, and a pair of real numbers `x0,y0`. The output will be the homogeneous coordinate vertex matrix obtained from `Ah` by shifting the graphic using the shift vector (`x0,y0`). Apply this to the CAT graphic using `x0` = 2 and `y0` = −1 and give a before-and-after plot.
(b) Create a MATLAB function M-file, called `scale(Ah,a,b,x0,y0)` that has inputs `Ah`, matrix of homogeneous coordinates of some graphic, positive numbers: `a` and `b` that represent the horizontal and vertical scaling factors, and a pair of real numbers $x0$, $y0$ that represent the coordinates about which the scaling is to be done. The output will be the homogeneous coordinate vertex matrix obtained from `Ah` by scaling the graphic as indicated. Apply this to the CAT graphic using $a = .25$, $b = 5$ once each with the following sets for $(x0, y0)$: (0,0), (3,0), (0,3), (2.5,4) and create a single plot containing the original CAT along with all four of these smaller, thin cats (use five different colors/plot styles).

7. Working in homogeneous coordinates, what is the transformation matrix $M$ that will reflect an image about the line $y = x$? Create a before-and-after graphic of the CAT.
**Suggestion:** Rotate first, reflect, and then rotate back again.

8. Working in homogeneous coordinates, what is the transformation matrix $M$ that will shift the left eye and left pupil of the "CAT with eyes'' to the left by .0.5 units and then expand them by a factor of 2 (away from the centers)? Apply this transformation just to the left eye, reflect to get the right eye, and then plot these two along with the outline of the "CAT with eyes," to get a cat with big eyes.

9. The **shearing** on $\mathbf{R}^2$ that shears by $b$ in the $x$-direction and $d$ in the $y$-direction is the linear transformation whose matrix is $\begin{bmatrix} 1 & b \\ c & 1 \end{bmatrix}$. Apply the shearing to the CAT using several different values of $b$ when $c = 0$, then set $b = 0$ and use several different values of $c$, and finally apply some shearings using several sets of nonzero values for $b$ and $c$.

10. (a) Show that the $2 \times 2$ matrix $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$, which represents the linear transformation for rotations by angle $\theta$, is invertible, with inverse being the corresponding matrix for rotations by angle $-\theta$.
(b) Does the same relationship hold true for the corresponding $3 \times 3$ homogeneous coordinate transform matrices? Justify your answer.

11. (a) Show that the $3 \times 3$ matrix $\begin{bmatrix} 1 & 0 & x_0 \\ 0 & 1 & y_0 \\ 0 & 0 & 1 \end{bmatrix}$, which represents the shift with shift vector $\begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$, is invertible, with its inverse being the corresponding matrix for the shift using the opposite shift vector.

12. Show that the $2 \times 2$ matrix $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$ indeed represents the linear transformation for rotations by angle $\theta$ around the origin (0,0).
**Suggestion:** Let $(x, y)$ have polar coordinates $(r, \alpha)$; then $(x', y')$ has polar coordinates $(r, \alpha + \theta)$. Convert the latter polar coordinates to rectangular coordinates.

13. (*Graphic Art: Rotating Shrinking Squares*) (a) By starting off with a square, and repeatedly shrinking it and rotating it, get MATLAB to create a graphic similar to the one shown in Figure 7.19(a).
(b) Next modify your construction to create a graph similar to the one in Figure 7.19(b) but that uses alternating colors.
**Note:** This object is not a fractal.

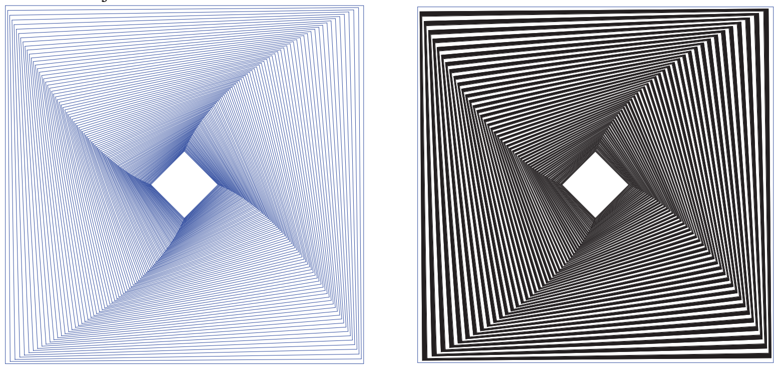


**FIGURE 7.19:** A rotating and shrinking square of Exercise 13: (a) (left) with no fills; (b) (right) with alternate black-and-white fills.

14. (*Graphic Art: Cat with Eyes Mosaic*) The cat mosaic of Figure 7.20 has been created by taking the original CAT, and creating new pairs of cats (left and right) for each step up. This construction was done with a for loop using 10 iterations (so there are 10 pairs of cats above the original), and could easily have been changed to any number of iterations. Each level upward of cats got scaled to 79% of the preceding level. Also, for symmetry, the left and right cats were shifted upward and to the left and right by the same amounts, but these amounts got smaller (since the cat size did) as we moved upward.
(a) Use MATLAB to create a picture that is similar to that of Figure 7.20, but replace the "CAT with eyes" with the ordinary CAT.
(b) Use MATLAB to create a picture that is similar to that of Figure 7.20.
(c) Use MATLAB to create a picture that is similar to that of Figure 7.20, but replace the "CAT with eyes" with the "full CAT" of Figure 7.11.
**Suggestion:** You should definitely use a for loop. Experiment a bit with different schemes for horizontal and vertical shifting to get your picture to look like this one.

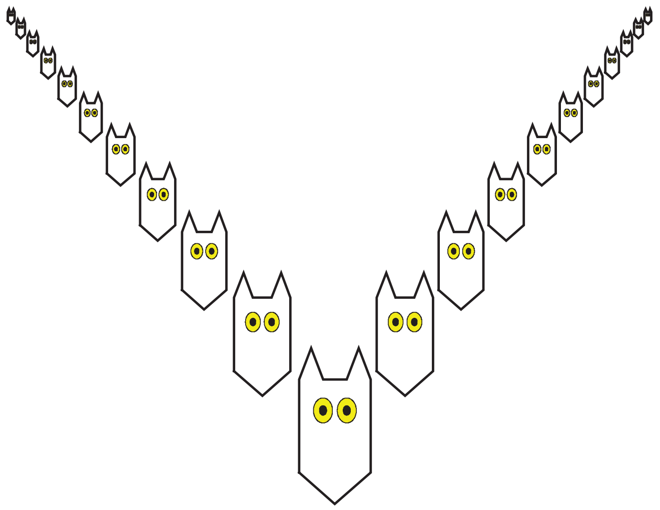


**FIGURE 7.20:** CAT with eyes mosaic for Exercise 14(b). The original cat (center) has been repeatedly shifted to the left and right, and up, as well as scaled by a factor of 79% each time we go up.

15. (*Movie: "Sudden Impact"*) (a) Create a movie that stars the CAT and proceeds as follows: The cat starts off at the left end of the screen. It then "runs" horizontally towards the right end of the screen. Just as its right side reaches the right side of the screen, it begins to shrink horizontally (but not vertically) until it degenerates into a vertical line segment on the right side of the screen.
(b) Make a movie similar to the one in part (a) except that this one stars the "CAT with eyes" and before it begins to run to the right, its pupils move to the right of the eyes and stay there.
(c) Make a film similar to the one in part (b) except that this one should star the "full CAT" (Figure 7.11) and upon impact with the right wall, the cat's smile changes to a frown.

16. (*Movie: "The Chase"*) (a) Create a movie that stars the "CAT with eyes" and co-stars another smaller version of the same cat (scaled by factors of 0.5 in both the *x*- and *y*-directions). The movie starts off with the big cat in the upper left of the screen and the small cat to its right side (very close). Their pupils move directly toward one another to the end of the eyes, and at this point both cats begin moving at constant speed toward the right. When the smaller cat reaches the right side of the screen, it starts moving down while the big cat also starts moving down. Finally, cats stay put in the lower-right corner as their pupils move back to center.
(b) Make the same movie except starring the "full CAT" and costarring a smaller counterpart.

17. (*Movie: "Close Encounter"*) (a) Create a movie that stars the "full CAT" (Figure 7.11) and with the following plot: The cat starts off smiling and then its eyes begin to shift all the way to the lower left. It spots a solid black rock moving horizontally directly toward its mouth level, at constant speed. As the cat spots this rock, its smile changes to a frown. It jumps upward as its pupils move back to center and just misses the rock as it brushes just past the cat's chin. The cat then begins to smile and falls back down to its original position.
(b) Make a film similar to the one in part (a) except that it has the additional feature that the rock is rotating clockwise as it is moving horizontally.
(c) Make a film similar to the one in part (b) except that it has the additional feature that the cat's pupils, after having spotted the rock on the left, slowly roll (along the bottom of the eyes) to the lower-right postion, exactly following the rock. Then, after the rock leaves the viewing window, have the cat's pupils move back to center postion.

18. (*Fractal Geometry*: *The Cantor Square*) The Cantor square is a fractal that starts with the *unit square* in the plane: $C_0 = \{(x, y) : 0 \le x \le 1 \text{ and } 0 \le y \le 1\}$ (generation zero). To move to the next generation, we delete from this square all points such that at least one of the coordinates is inside the middle 1/3 of the original spread. Thus, to get $C_1$ from $C_0$, we delete all the points $(x, y)$ having either $1/3 < x < 2/3$ or $1/3 < y < 2/3$. So $C_1$ will consist of four smaller squares each having sidelength equal to 1/3 (that of $C_0$) and sharing one corner vertex with $C_0$. Future generations are obtained in the same way. For example, to get from $C_1$ (first generation) to $C_2$ (second generation) we delete, from each of the four squares of $C_1$, all points $(x, y)$ that have one of the coordinates lying in the middle 1/3 of the original range (for a certain square of $C_1$). What will be left is four squares for each of the squares of $C_1$, leaving a total of 16 squares each having sidelength equal to 1/3 that of the squares of $C_1$, and thus equal to 1/9. In general, letting this process continue forever, it can be shown by induction that the *n*th-generation Cantor square consists of $4^n$ squares each having sidelength $1/3^n$. The Cantor square is the set of points that remains after this process has been continued indefinitely.
(a) Identify the four similitudes $S_1, S_2, S_3, S_4$ associated with the Cantor square (an illustration as in Figure 7.16 would be fine) and then, working in homogeneous coordinates, find the matrices of each. Next, following the approach of Method 1 of the solution of Example 7.7, write a function M-file `cantorsq1(V1,V2,V3,V4, ngen)`, that takes as input the vertices `V1 = [0 0]`, `V2 = [1 0]`, `V3 = [1 1]`, and `V4 = [0 1]` of the unit square and a nonnegative integer `ngen` and will produce a graphic of the generation `ngen` Cantor square.
(b) Write a function M-file `cantorsq2(V1,V2,V3,V4, niter)` that takes as input the vertices `V1`, `V2`, `V3` `V4` of any square and a positive integer `niter` and will produce a Monte Carlo generated graphic for the Cantor square as in Method 2 of the solution of Example 7.7. Run your program for the square having sidelength 1 and lower-left vertex $(-1, 2)$ using `niter` = 2000 and `niter` = 12,000.
(c) Write a function M-file `cantorsq3(V1,V2,V3,V4, ngen)` that takes as input the vertices `V1`, `V2`, `V3` `V4` of any square and a positive integer `ngen` and will produce a graphic for the `ngen` generation Cantor square as did `cantorsq1` (but now the square can be any square). Run your program for the square mentioned in part (b) first with `ngen` = 1 then with `ngen` = 3. Can this program be written so that it produces a reasonable generalization of the Cantor square when the vertices are those of any rectangle?

19. (*Fractal Geometry*: *The Sierpinski Carpet*) The Sierpinski carpet is the fractal that starts with the unit square $\{(x, y) : 0 \le x \le 1 \text{ and } 0 \le y \le 1\}$ with the central square of 1/3 the sidelength removed (generation zero). To get to the next generation, we punch eight smaller squares out of each of the remaining eight squares of sidelength 1/3 (generation one), as shown in Figure 7.21. Write a function M-file, `scarpet2(niter)`, based on the Monte Carlo method that will take only a single input variable `niter` and will produce a Monte Carlo approximation of the Sierpinski carpet. You will, of course, need to find the eight similitudes associated with this fractal and get their matrices in homogeneous coordinates. Run your program with inputs `niter` = 1000, 2000, 5000, and 10,000.

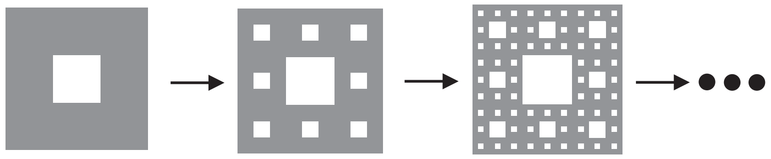


**FIGURE 7.21:** Illustration of generations zero (left), one (middle), and two (right) of the Sierpinski gasket fractal of Exercises 19, 20, and 21. The fractal consists of the points that remain (shaded) after this process has continued on indefinitely.

20. (*Fractal Geometry*: *The Sierpinski Carpet*) Read first Exercise 19 (and see Figure 7.21), and if you have not done so yet, identify the eight similitudes $S_1, S_2, \cdots, S_8$ associated with the Sierpinski carpet along with the homogeneous coordinate matrices of each. Next, following the approach of Method 1 of the solution of Example 7.7, write a function M-file `scarpet1(V1,V2,V3,V4, ngen)` that takes as input the vertices `V1 = [0 0]`, `V2 = [1 0]`, `V3 = [1 1]`, and `V4 = [0 1]` of the unit square and a nonnegative integer `ngen` and will produce a graphic of the generation `ngen` Cantor square.
**Suggestions:** Fill in each outer square in gray, then to get the white central square "punched out," use the `hold on` and then `fill` in the smaller square in the color white (rgb vector [1 1 1]). When MATLAB fills a polygon, by default it draws the edges in black. To suppress the edges from being drawn, use the following extra option in the fill commands: `fill(xvec, yvec, rgbvec, 'EdgeColor', 'none')`. Of course, another nice way to edit a graphic plot from MATLAB is to import the file into a drawing software (such as Adobe Illustrator or Corel Draw) and modify the graphic using the software.

21. (*Fractal Geometry*: *The Sierpinski Carpet*) (a) Write a function M-file called `scarpet3(V1,V2,V3,V4, ngen)` that works just like the program `scarpet1` of the previous exercise, except that the vertices can be those of any square. Also, base the code not on similitudes, but rather on mathematical formulas for next-generation parameters in terms of present-generation parameters. The approach should be somewhat analogous to that of Method 3 of the solution to Example 7.7.
(b) Is it possible to modify the `sgasket1` program so that it is able to take as input the vertices of any equilateral triangle? If yes, indicate how. If no, explain why not.

22. (*Fractal Geometry*: *The Fern Leaf*) There are more general ways to construct fractals than those that came up in the text. One generalization of the self similarity approach given in the text allows for transformations that are not invertible (similitudes always are). In this exercise you are to create a function M-file, called `fracfern(n)`, which will input a positive integer $n$ and will produce a graphic for the fern fractal pictured in Figure 7.22, using the Monte Carlo method. For this fractal the four transformations to use are (given by their homogeneous coordinate matrices)

$$S1=\begin{bmatrix}0&0&0\\0&.16&0\\0&0&1\end{bmatrix},\quad S2=\begin{bmatrix}.85&.04&0\\-.04&.85&1.6\\0&0&0\end{bmatrix},$$

$$S3=\begin{bmatrix}.2&-.26&0\\.23&.22&1.6\\0&0&1\end{bmatrix},\quad S4=\begin{bmatrix}-.15&.28&0\\.26&.24&.44\\0&0&1\end{bmatrix},$$

and the associated probability vector is [ .01 .86 .93] (i.e., in the Monte Carlo process, 1% of the time we choose $S1$, 85% of the time we choose $S2$, 7% of the time we choose $S3$, and the remaining 7% of the time we choose $S4$).
**Suggestion:** Simply modify the program `sgasket2` accordingly.

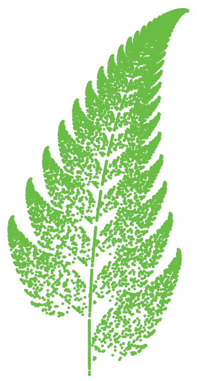


**FIGURE 7.22:** The fern leaf fractal.

23. (*Fractal Geometry*: *The Gosper Island*) (a) Write a function M-file `gosper(n)` that will input a positive integer `n` and will produce a graphic of the $n$th generation of the **Gosper island** fractal, which is defined as follows: Generation zero is a regular hexagon (with, say, unit side lengths). To get from this to generation one, we replace each of the six sides on the boundary of generation zero with three new segments as shown in Figure 7.23. The first few generations of the Gosper island are shown in Figure 7.24.

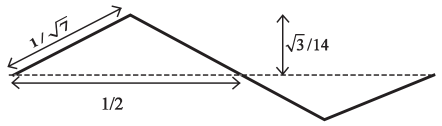


**FIGURE 7.23:** Iteration scheme for the definition of the Gosper island fractal of Exercise 23. The dotted segment represents a segment of a certain generation of the Gosper island, and the three solid segments represent the corresponding part of the next generation.

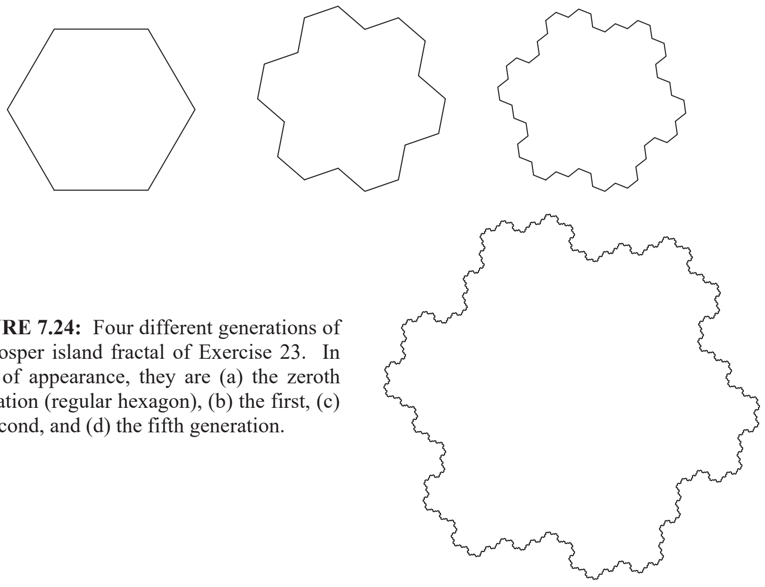


**FIGURE 7.24:** Four different generations of the Gosper island fractal of Exercise 23. In order of appearance, they are (a) the zeroth generation (regular hexagon), (b) the first, (c) the second, and (d) the fifth generation.

(b) (*Tessellations of the Plane*) It is well known that the only regular polygons that can tessellate (or tile) the plane are the equilateral triangle, the square, and the regular hexagon (honeybees have figured this out). It is an interesting fact that any generation of the Gosper island can also be used to tessellate the plane, as shown in Figure 7.25. Get MATLAB to reproduce each of tessellations that are shown in Figure 7.25.

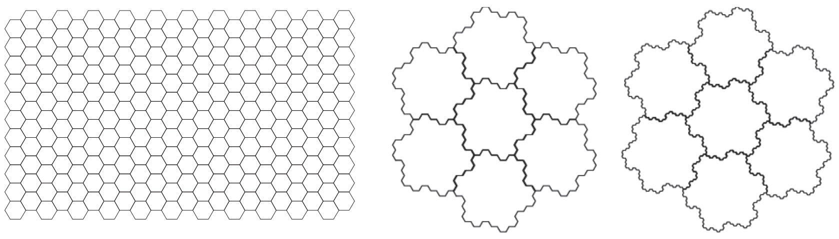


**FIGURE 7.25:** Tessellations with generations of Gosper islands. The top one (with regular hexagons) is the familiar honeycomb structure.

## 7.3: NOTATIONS AND CONCEPTS OF LINEAR SYSTEMS

The general linear system in $n$ variables $x_1, x_2, \cdots, x_n$ and $n$ equations can be written as

**EFR 7.3:** Using the `fill` command as was done in the text to get the gray cat of Figure 7.3(b), you can get those other-colored cats by simply replacing the RGB vector for gray by the following: Orange → RGB = [1 .5 0], Brown → RGB = [.5 .25 0], Purple → RGB = [.5 0 .5]. Since each of these colors can have varying shades, your answers may vary. Also, the naked eye may not be able to distinguish between colors arising from small perturbations of these vectors (say by .001 or even .005). The RGB vector representing MATLAB's cyan is RGB = [0 1 1].

**EFR 7.4:** By property (10) (of linear transformations): $L(\alpha P_1) = \alpha L(P_1)$; if we put $\alpha = 0$, we get that $L(\vec{0}) = \vec{0}$ (where $\vec{0}$ is the zero vector). But a shift transformation $T_{V_0}(x,y) = (x,y) + V_0$ satisfies $T_{V_0}(\vec{0}) = \vec{0} + V_0 = V_0$. So the shift transformation $T_{V_0}$ being linear would force $V_0 = \vec{0}$, which is not allowed in the definition of a shift transformation (since then $T_{V_0}$ would then just be the identity transformation).

**EFR 7.5:** (a) As in the solution of Example 7.4, we individually multiply out the homogeneous coordinate transformation matrices (as per the instructions in the proof of Theorem 7.2) from right to left. The first transformation is the shift with vector (1,0) with matrix: $T_{(1,0)} \sim \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = H_1$. After this we apply a scaling $S$ whose matrix is given by $S \sim \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = H_2$. The homogeneous cooordinate matrix for the composition of these two transformations is: $M = H_2H_1 = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 2 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$. We assume (as in the text) that we have left in the graphics window the first (white) cat of Figure 7.3(a) and that the CAT matrix $A$ is still in our workspace. The following commands will now produce the new "fat CAT":

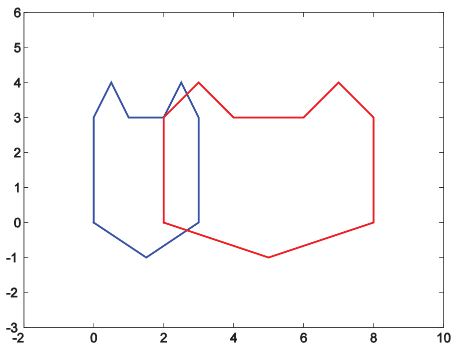
```
>> H1=[1 0 1;0 1 0; 0 0 1]; H2=[2 0 0;0 1 0;0 0 1]; M=H2*H1
>> AH=A; AH(3,:)=ones(1,10);  %homogenize the CAT matrix
>> AH1=M*AH; % homogenized "fat CAT" matrix
>> hold on
>> plot(AH1(1,:), AH1(2,:), 'r')
>> axis([-2 10 -3 6]) % set wider axes to accommodate "fat CAT"
>> axis('equal')
```

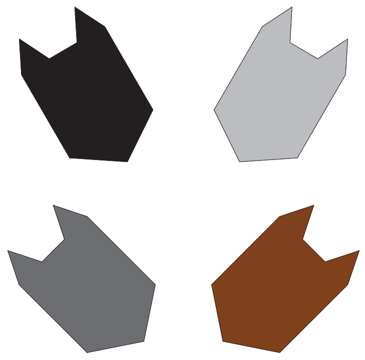
The resulting plot is shown in the left-hand figure that follows.
(b) Each of the four cats needs to first get rotated by its specified angle about the same point (1.5, 1.5). As in the solution to Example 7.4, these rotations can be accomplished by first shifting this point to (0, 0) with the shift $T_{(-1.5,-1.5)}$, then performing the rotation, and finally shifting back with the inverse shift $T_{(1.5,1.5)}$. In homogeneous coordinates, the matrix representing this composition is (just like in the solution to Example 7.4):

$$M = \begin{bmatrix} 1 & 0 & 1.5 \\ 0 & 1 & 1.5 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} 1 & 0 & -1.5 \\ 0 & 1 & -1.5 \\ 0 & 0 & 1 \end{bmatrix}.$$

After this rotation, each cat gets shifted in the specified direction with $T_{(\pm 1,\pm 1)}$. For the colors of our cats let's use the following: black (rgb = [0 0 0]), light gray (rgb = [.7 .7 .7]), dark gray (rgb = [.3 .3. .3]), and brown (rgb = [.5 .25 0]). The following commands will then plot those cats:

```
>> clf, hold on  %prepare graphic window
>> %upper left cat,  theta = pi/6 (30 deg), shift vector = (-3, 3)
>> c = cos(pi/6); s = sin(pi/6);
>> M=[1 0 1.5;0 1 1.5;0 0 1]*[c -s 0;s c 0;0 0 1]*[1 0 -1.5;0 1 -
1.5;0 0 1];
>> AUL=[1 0 -3;0 1 3;0 0 1]*M*AH;
>> fill(AUL(1,:), AUL(2,:), [0 0 0])
>> %upper right cat,  theta = -pi/6 (-30 deg), shift vector = (3, 1)
>> c = cos(-pi/6); s = sin(-pi/6);
>> M=[1 0 1.5;0 1 1.5;0 0 1]*[c -s 0;s c 0;0 0 1]*[1 0 -1.5;0 1 -
1.5;0 0 1];
>> AUR=[1 0 1;0 1 1;0 0 1]*M*AH;
>> fill(AUR(1,:), AUR(2,:), [.7 .7 .7])
>> %lower left cat,  theta = pi/4 (45 deg), shift vector = (-3, -3)
>> c = cos(pi/4); s = sin(pi/4);
>> M=[1 0 1.5;0 1 1.5;0 0 1]*[c -s 0;s c 0;0 0 1]*[1 0 -1.5;0 1 -
1.5;0 0 1];
>> ALL=[1 0 -3;0 1 -3;0 0 1]*M*AH;
>> fill(ALL(1,:), ALL(2,:), [.3 .3 .3])
>> %lower right cat,  theta = -pi/4 (-45 deg), shift vector = (3, -3)
>> c = cos(-pi/4); s = sin(-pi/4);
>> M=[1 0 1.5;0 1 1.5;0 0 1]*[c -s 0;s c 0;0 0 1]*[1 0 -1.5;0 1 -
1.5;0 0 1];
>> ALR=[1 0 3;0 1 -3;0 0 1]*M*AH;
>> fill(ALR(1,:), ALR(2,:), [.5 .25 0])
>> axis('equal'), axis off %see graphic w/out distraction of axes.
```





**EFR 7.6:** (a) This first M-file is quite straightforward and is boxed below.

```
function B=mkhom(A)
B=A;
[n m]=size(A);
B(3,:)=ones(1,m);
```

(b) This M-file is boxed below.

```
function Rh=rot(Ah,x0,y0,theta)
%viz. EFR 7.6; theta should be in radians
%inputs a 3 by n matrix of homogeneous vertex coordinates, xy
%coordinates of a point and an angle theta.  Output is corresponding
%matrix of vertices rotated by angle theta about (x0,y0).

%first construct homogeneous coordinate matrix for shifting (x0,y0)
to (0,0)
SZ=[1 0 -x0;0 1 -y0; 0 0 1];
%next the rotation matrix at (0,0)
R=[cos(theta) -sin(theta) 0; sin(theta) cos(theta) 0;0 0 1];
%finally the shift back to (x0,y0)
SB=[1 0 x0;0 1 y0;0 0 1];
%now we can obtain the desired rotated vertices:
Rh=SB*R*SZ*Ah;
```

**EFR 7.7:** (a) The main transformation that we need in this movie is vertical scaling. To help make the code for this exercise more modular, we first create, as in part (b) of the last EFR, a separate M-file for vertical scaling:

```
function Rh=vertscale(Ah,b,y0)
%inputs a 3 by n matrix of homogeneous vertex coordinates, a (pos.)
%numbers a for  y- scales, and an optional  arguments y0
%for center of scaling.  Output is homogeneous coor. matrix of scaled
%vertices.  default value of y0 is 0.

if nargin <3
    y0=0;
end
%first construct homogeneous coordinate matrix for shifting y=y0 to
%y=0
SZ=[1 0 0;0 1 -y0; 0 0 1];
%next the scaling matrix at (0,0)
S=[1 0 0; 0 b 0;0 0 1];
%finally the shift back to y=0
SB=[1 0 x0;0 1 y0;0 0 1];
%now we can obtain the desired scaled vertices:
Rh=SB*S*SZ*Ah;
```

Making use of the above M-file, the following script recreates the CAT movie of Example 7.4 using homogeneous coordinates:

```
%script for EFR 7.6(a):  catmovieNo1.m  cat movie creation
%Basic CAT movie, where cat closes and reopens its eyes.
clf, counter=1;

A=[0  0  .5  1  2  2.5  3  3  1.5  0; ...
     0  3  4   3  3   4   3  0  -1   0]; %Basic CAT matrix
Ah = mkhom(A); %use the M-file from EFR 7.6

t=0:.02:2*pi;  %creates time vector for parametric equations for eyes
xL=1+.4*cos(t); y=2+.4*sin(t); %creates circle for left eye
LE=mkhom([xL; y]); %homogeneous coordinates for left eye
xR=2+.4*cos(t); y=2+.4*sin(t); %creates circle for right eye
RE=mkhom([xR; y]); %homogeneous coordinates for right eye
xL=1+.15*cos(t); y=2+.15*sin(t); %creates circle for left pupil
LP=mkhom([xL; y]); %homogeneous coordinates for left pupil
xR=2+.15*cos(t); y=2+.15*sin(t); %creates circle for right pupil
RP=mkhom([xR; y]); %homogeneous coordinates for right pupil

for s=0:.2:2*pi
 factor = (cos(s)+1)/2;
 plot(A(1,:), A(2,:), 'k'), hold on
 axis([-2 5 -3 6]), axis('equal')
 LEtemp=vertscale(LE,factor,2); LPtemp=vertscale(LP,factor,2);
 REtemp=vertscale(RE,factor,2); RPtemp=vertscale(RP,factor,2);
 hold on
 fill(LEtemp(1,:), LEtemp(2,:),'y'), fill(REtemp(1,:),
REtemp(2,:),'y')
 fill(LPtemp(1,:), LPtemp(2,:),'k'), fill(RPtemp(1,:),
RPtemp(2,:),'k')
 M(:, counter) = getframe;
 hold off
 counter=counter+1;
end
```

(b) As in part (a), the following script M-file will make use of two supplementary M-files, `AhR=reflx(Ah, x0)` and, `AhS=shift(Ah, x0, y0)`, that perform horizontal reflections and shifts in homogeneous coordinates, respectively. The syntaxes of these M-files are explained in

Exercises 5 and 6 of this section. Their codes can be written in a fashion similar to the code `vertscale` but for completeness are can be downloaded from the ftp site for this text (see the beginning of this appendix). They can be avoided by simply performing the homogeneous coordinate transformations directly, but at a cost of increasing the size of the M-file that we give:

```
%coolcatmovie.m: script for making coolcat movie matrix M of EFR 7.7

%act one:  eyes shifting left/right
t=0:.02:2*pi; counter=1;
A=[0  0  .5  1  2  2.5  3  3  1.5  0; ...
     0  3  4   3  3   4   3  0  -1   0];
x=1+.4*cos(t); y=2+.4*sin(t);xp=1+.15*cos(t); yp=2+.15*sin(t);
LE=[x;y]; LEh=mkhom(LE); LP=[xp;yp]; LPh=mkhom(LP);
REh=reflx(LEh, 1.5); RPh=reflx(LPh, 1.5);
LW=[.3 -1; .2 -.8]; LW2=[.25 -1.1;.25 -.6]; %left whiskers
LWh=mkhom(LW); LW2h=mkhom(LW2);
RWh=reflx(LWh, 1.5); RW2h=reflx(LW2h, 1.5); %reflect left whiskers
                                             %to get right ones
M=[1 1.5 2;.25 -.25 .25]; Mh=mkhom(M);  %matrix & homogenization of
                                        %cats mouth
Mhrefl=refly(Mh,-.25); %homogeneous coordinates for frown
for n=0:(2*pi)/20:2*pi
plot(A(1,:), A(2,:),'k')
axis([-2 5 -3 6]), axis('equal')
hold on
plot(LW(1,:), LW(2,:),'k'), plot(LW2(1,:), LW2(2,:),'k')
plot(RWh(1,:), RWh(2,:),'k')
plot(RW2h(1,:), RW2h(2,:),'k')
plot(Mhrefl(1,:), Mhrefl(2,:),'k')
fill(LE(1,:), LE(2,:),'y'), fill(REh(1,:), REh(2,:),'y')
LPshft=shift(LPh,-.25*sin(n),0); RPshft=shift(RPh,-.25*sin(n),0);
fill(LPshft(1,:), LPshft(2,:),'k'), fill(RPshft(1,:),
RPshft(2,:),'k')
Mov(:, counter)=getframe;
hold off
counter = counter +1;
end

%act two:  eyes shifting up/down
for n=0:(2*pi)/20:2*pi
plot(A(1,:), A(2,:),'k')
axis([-2 5 -3 6]), axis('equal')
hold on
plot(LW(1,:), LW(2,:),'k'), plot(LW2(1,:), LW2(2,:),'k')
plot(RWh(1,:), RWh(2,:),'k')
plot(RW2h(1,:), RW2h(2,:),'k')
plot(Mhrefl(1,:), Mhrefl(2,:),'k')
fill(LE(1,:), LE(2,:),'y'), fill(REh(1,:), REh(2,:),'y')
LPshft=shift(LPh,0,.25*sin(n)); RPshft=shift(RPh,0,.25*sin(n));
fill(LPshft(1,:), LPshft(2,:),'k'), fill(RPshft(1,:),
RPshft(2,:),'k')
Mov(:, counter)=getframe;
hold off
counter = counter +1;
end

%act three:  whisker rotating up/down then smiling
for n=0:(2*pi)/10:2*pi
plot(A(1,:), A(2,:),'k')
axis([-2 5 -3 6]), axis('equal')
```

```
hold on
fill(LE(1,:), LE(2,:),'y'),fill(LP(1,:), LP(2,:),'k')
fill(REh(1,:), REh(2,:),'y'),fill(RPh(1,:), RPh(2,:),'k')
LWrot=rot(LWh,.3,.2,-pi/6*sin(n)); LW2rot=rot(LW2h, .25,.25,-
pi/6*sin(n));
RWrot=reflx(LWrot, 1.5); RW2rot=reflx(LW2rot, 1.5);
plot(LWrot(1,:), LWrot(2,:),'k'), plot(LW2rot(1,:), LW2rot(2,:),'k')
plot(RWrot(1,:), RWrot(2,:),'k'),plot(RW2rot(1,:), RW2rot(2,:),'k')
if n == 2*pi
   plot(Mh(1,:), Mh(2,:),'k')
   for n=1:10, L(:,n)=getframe; end
   Mov(:, counter:(counter+9))=L;
   break
else
   plot(Mhrefl(1,:), Mhrefl(2,:),'k')

end
Mov(:, counter)=getframe;

hold off
counter = counter +1;
end

%THE END
```

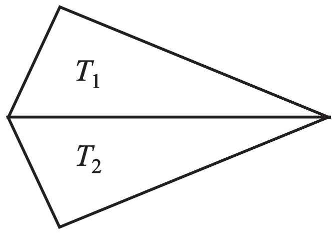
**<u>EFR 7.8</u>:** (a) Certainly the zeroth generation consists of $1 = 3^0$ triangles. Since the sidelength is one, and the triangle has each of its angles being $\pi/3$, its altitude must be $\sin(\pi/3) = \sqrt{3}/2$. Thus, the area of the single zeroth generation triangle is $\sqrt{3}/4$. Now, each time we pass to a new generation, each triangle splits into three (equilateral) triangles of half the length of the triangles of the current generation. Thus, by induction, the $n$th generation will have $3^n$ equilateral triangles of sidelength $1/2^n$ and hence each of these has area $(1/2)\cdot 1/2^n \cdot [\sqrt{3}/2]/2^n = \sqrt{3}/4^{n+1}$.

(b) From part (a), the $n$th generation of the Sierpinski carpet consists of $3^n$ equilateral triangles each having area $\sqrt{3}/4^{n+1}$. Hence the total area of this $n$th generation is $\sqrt{3}(3/4)^n/4$. Since this expression goes to zero as $n \to \infty$, and since the Sierpinski carpet is contained in each of the generation sets, it follows that the area of the Sierpinski carpet must be zero.

**<u>EFR 7.9</u>:** (a) The $2\times 2$ matrices representing dilations $\begin{bmatrix} s & 0 \\ 0 & s \end{bmatrix}$ (s > 0), and reflections with respect to the $x$-axis: $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ or $y$-axis: $\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$ are both diagonal matrices and thus commute with any other $2\times 2$ matrices; i.e., if $D$ is any diagonal matrix and $A$ is any other $2\times 2$ matrix, then $AD = DA$. In particular, these matrices commute with each other and with the matrix representing a rotation through the angle $\theta$: $\begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$. By composing rotations and reflections, we can obtain transformations that will reflect about any line passing through (0,0). Once we throw in translations, we can reflect about any line in the plane and (as we have already seen) rotate with any angle about any point in the plane. By the definition of similitudes, we now see that compositions of these general transformations can produce the most general similitudes. Translating into homogeneous coordinates (using the proof of Theorem 7.2) we see that the matrix for such a composition can be expressed as

$\begin{bmatrix} s\cos\theta & -s\sin\theta & x_0 \\ \pm s\sin\theta & \pm s\cos\theta & y_0 \\ 0 & 0 & 1 \end{bmatrix}$ where $s$ now is allowed to be any nonzero number. If the sign in the second row is negative, we have a reflection: If $s > 0$, it is a $y$-axis reflection; if $s < 0$, it is an $x$-axis reflection.
(b) Let $T_1$ and $T_2$ be two similar triangles in the plane. Apply a dilation, if necessary, to $T_1$ so that it has the same sidelengths as $T_2$. Next, apply a shift transformation to $T_1$ so that a vertex gets shifted to a corresponding vertex of $T_2$, and then apply a rotation to $T_1$ about this vertex so that a side of $T_1$ transforms into a corresponding side of $T_2$.

At this point, either $T_1$ and $T_2$ are now the same triangle, or they are reflections of one another across the common side. A final reflection about this line, if necessary, will thus complete the transformation of $T_1$ into $T_2$ by a similitude.



(c) It is clear that dilations, rotations, and shifts are essential. For an example to see why reflection is needed, simply take $T_1$ to be any triangle with three different angles and $T_2$ to be its reflection about one of the edges (see figure). It is clearly not possible to transform one of these two triangles into the other using any combination of dilations, rotations, and shifts.

**EFR 7.10:** (a) There will be only one generation; here are the outputs that were asked for (in format short):

```
A→         0    1.0000   2.0000           A1 →    0        0.5000   1.0000
           0    1.7321       0                    0         0.8660       0
      1.0000    1.0000   1.0000                   1.0000   1.0000  1.0000

A2 →    1.0000    1.5000    2.0000        A3→     0.5000   1.0000    1.5000
             0    0.8660        0                 0.8660   1.7321    0.8660
        1.0000    1.0000    1.0000                1.0000   1.0000    1.0000

A1([1 2],2) → 0.5000            A3([1 2],2) → 1.5000
              0.8660                          0.8660
```

(b) Since the program calls on itself and does so more than once (as long as `niter` is greater than zero), placing a `hold off` anywhere in the program will cause graphics created on previous runs to be lost, so such a feature could not be incorporated into the program.
(c) Since we want the program to call on itself iteratively with different vertex sets, we really need to allow vertex sets to be inputted. Different vertex inputs are possible, but in order for the program to function effectively, they should be vertices of a triangle to which the similitudes in the program correspond. (e.g., any of the triangles in any generation of the Sierpinski gasket).

**EFR 7.11:** (a) S2, S1, S3, S2, S3, S2
(b) We list the sequence of float points in nonhomogeneous coordinates and in `format short`:
[0.5000 0.8660], [0.2500 0.4330], [1.1250 0.2165], [1.0625 0.9743], [1.5313 0.4871], [1.2656 1.1096].
(c) The program is designed to work for <u>any</u> triangle in the plane. The reader can check that the three similitudes are constructed in a way that uses midpoints of the triangle and the resulting diagram will look like that of Figure 7.15.

**EFR 7.12:** (a) As with `sgasket2`, the program `sgasket3` contructs future-generation triangles simply from the vertices and (computed) midpoints of the current-generation triangles. Thus, it can deal effectively with any triangle and produce Sierpinski-type fractal generations.

(b) For illustration purposes, the following trials were run on MATLAB's Version 5, so as to illustrate the flop count differences. The code is easily modified to work on newer versions of MATLAB by simply deleting the "`flops`" commands.

```
V1=[0 0]; V2=[1 sqrt(3)]; V3=[2 0];   %vertices of an equilateral
triangle
 test = [1 3 6 8 10];
```

```
>> for i=1:5
flops(0), tic,
sgasket1(V1,V2,V3,test(i)), toc,
flops
end
→  (ngen =1) elapsed_time = 0.0600,
ans =191
      (ngen =3) elapsed_time = 0.2500,
 ans =2243
      (ngen =6) elapsed_time = 0.8510,
 ans =62264
      (ngen =8) elapsed_time = 7.2310,
 ans =560900
      (ngen =10) elapsed_time = 65.4640,
ans =5048624
```

```
>> for i=1:5
flops(0), tic,
sgasket3(V1,V2,V3,test(i)), toc,
flops
end
→  (ngen =1) elapsed_time = 0.1400,
 ans = 45
      (ngen =3) elapsed_time = 0.1310,
 ans =369
      (ngen =6) elapsed_time = 0.7210,
ans =9846
      (ngen =8) elapsed_time = 6.2990,
ans =88578
      (ngen =10) elapsed_time = 46.7260,
ans =797166
```

We remind the reader that the times will vary, depending on the machine being used and other processes being run. The above tests were run on a rather slow machine, so the resulting times are longer than typical.

**EFR 7.13:** The M-file is boxed below:

```
function []=snow(n)
S=[0 1 2 0;0 sqrt(3) 0 0];
index=1;
while index <=n
   len=length(S(1,:));
   for i=1:(len-1)
delta=S(:,i+1)-S(:,i);
perp=[0 -1;1 0]*delta;
T(:,4*(i-1)+1)=S(:,i);
T(:,4*(i-1)+2)=S(:,i)+(1/3)*delta;
T(:,4*(i-1)+3)=S(:,i)+(1/2)*delta+(1/3)*perp;
T(:,4*(i-1)+4)=S(:,i)+(2/3)*delta;
T(:,4*(i-1)+5)=S(:,i+1);
end
index=index+1;
S=T;
end
plot(S(1,:),S(2,:)), axis('equal')
```

The outputs of `snow(1)`, `snow(2)`, and `snow(6)` are illustrated in Figures 7.17 and 7.18.

**EFR 7.14:** For any pair of nonparallel lines represented by a two-dimensional linear system: $\begin{bmatrix} a & b \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} e \\ f \end{bmatrix}$, the coefficient matrix will have nonzero determinant $\alpha = ad - bc$. The lines are also represented by the equivalent system $\begin{bmatrix} a/\alpha & b/\alpha \\ c & d \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix}=\begin{bmatrix} e/\alpha \\ f \end{bmatrix}$, where now the coefficient matrix has determinant $(a/\alpha)d-(b/\alpha)c=1$. This change simply amounts to dividing the first equation by $\alpha$.

**EFR 7.15:** (a) As in the solution of Example 7.7, the interpolation equations $p(-2) = 4$, $p(1) = 3$, $p(2) = 5$, and $p(5) = -22$ (where $p(x)=ax^3+bx^2+cx+d$)