Chapter 1

Turtle Graphics

The turtle and the crane and the swallow observe the time of their coming

– Jeremiah 8:7

1.1 Turtle Graphics

Motion generates geometry. The turtle is a handy paradigm for investigating curves generated by motion. Imagine a turtle that after each step leaves a trail connecting her previous location to her new location. As this turtle crawls around on a flat surface, the turtle may traverse paths that form intriguing geometric patterns (see Figure 1.1).

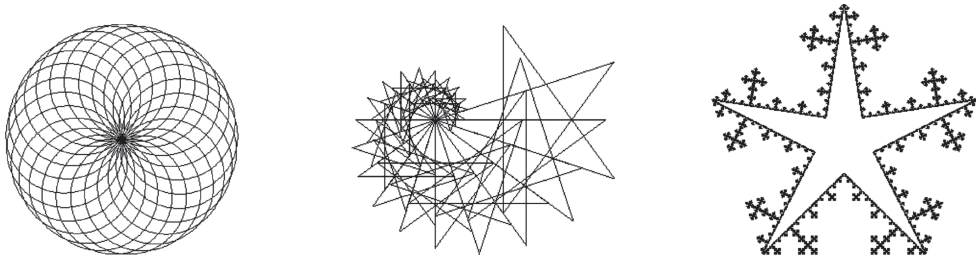

FIGURE 1.1: Some interesting geometric patterns generated by turtle paths.

Some of you may have encountered this mythical turtle before in the programming language LOGO, which has been used in primary schools to introduce young children to programming. But the turtle is more than just a toy for young children. In their book *Turtle Geometry*, Abelson and DiSessa use turtles to investigate many advanced topics, ranging from elementary differential geometry and topology to Einstein's General Theory of Relativity. The study of geometry using programming is the branch of Computer Science called *Computational Geometry*.

We shall begin our investigation of Computer Graphics with Turtle Graphics. After developing a simple variant of LOGO, we will see that Turtle Graphics can be used to generate many diverse shapes, ranging from simple polygons to complex fractals.

1.2 Turtle Commands

To help understand the ideas behind Turtle Graphics, we introduce a virtual turtle. The virtual turtle is a simple creature: she knows only where she is, in which direction she is facing, and her step size. The virtual turtle obeys solely simple commands to change either her location, or her heading, or her notion of scale.

Consider such a virtual turtle living on a virtual plane. The turtle's location can be represented by a point P (a dot) given by a pair of coordinates (x,y); similarly the turtle's heading can be represented by a vector w (an arrow) given by another pair of coordinates (u,v) (see Figure 1.2). The step size of the turtle is simply the length of the vector w. Thus, the step size of the turtle is $|w| = \sqrt{u^2 + v^2}$.

Location is a point; direction is a vector. Points are represented by dots; vectors are represented by arrows. Points have position, but no direction or length; vectors have direction and length, but no fixed position. In many branches of science and engineering, the distinction between points and vectors is often overlooked, but this distinction is very important in Computer Graphics. We shall see shortly that computationally points and vectors are treated quite differently in LOGO.

The pair (P,w) is called the *turtle's state*. Although internally the computer stores the coordinates (x,y) and (u,v), the turtle (and the turtle programmer) has no access to these global coordinates; the turtle knows only the local information (P,w), not the global information (x,y) and (u,v). That is, the turtle knows only that she is here at P facing there in the direction w; she does not know how P and w are related to some global origin or coordinate axes, to other heres and theres.

This state model for the turtle is similar to the model of a billiard ball in classical mechanics, where physicists keep track of a ball's position and momentum. Turtle location is analogous to the position of the billiard ball, and turtle direction is analogous to the momentum of the billiard ball. The main difference between these two models is that for billiard balls, the laws of physics (differential equations) govern the position and momentum of the ball; in contrast, we shall write our own programs to change the location and direction of the turtle.

The turtle responds to four basic commands: FORWARD, MOVE, TURN, and RESIZE. These commands affect the turtle in the following ways:

- FORWARD D: The turtle moves forward D steps along a straight line from her current position in the direction of her current heading, and draws a straight line from her initial position to her final position.
- MOVE D: Same as FORWARD D without drawing a line.
- TURN A: The turtle changes her heading by rotating her direction vector in the plane counterclockwise from her current heading by the angle A.
- RESIZE S: The turtle changes the length of her step size (direction vector) by the factor S.

FIGURE 1.2: The virtual turtle is represented by a point P (dot) and a vector w (arrow).

These four turtle commands are implemented internally in the following fashion:

- FORWARD D and MOVE D—*Translation* (see Figure 1.3)

$$x_{new} = x + Du$$
$$y_{new} = y + Dv$$

- TURN A—Rotation (see Figure 1.4)

$$\begin{matrix} u_{new} = u\cos(A) - v\sin(A) \\ v_{new} = u\sin(A) + v\cos(A) \end{matrix} \Leftrightarrow (u_{new}\ v_{new}) = (u\ v)\begin{pmatrix} \cos(A) & \sin(A) \\ -\sin(A) & \cos(A) \end{pmatrix}.$$

- RESIZE S—Scaling (see Figure 1.5)

$$\begin{matrix} u_{new} = Su \\ v_{new} = Sv \end{matrix} \Leftrightarrow (u_{new}\ v_{new}) = (u\ v)\begin{pmatrix} S & 0 \\ 0 & S \end{pmatrix}.$$

We shall also adopt the following conventions:

- $D < 0 \Rightarrow$ FORWARD D moves the turtle backward.
- $A > 0 \Rightarrow$ TURN A rotates the turtle counterclockwise.
- $A < 0 \Rightarrow$ TURN A rotates the turtle clockwise.
- $S < 0 \Rightarrow$ RESIZE S rotates the turtle by 180° and then scales the direction vector by $|S|$.

Notice that the TURN and RESIZE commands can be implemented using matrix multiplication, but that the FORWARD and MOVE commands cannot be implemented in this manner. This distinction arises because rotation and scaling are linear transformations on vectors, but translation is an affine, not a linear, transformation on points. (A point is not a vector, so transformations on points are inherently different from transformations on vectors. We shall discuss linear and affine transformations—their precise meanings as well as their similarities and differences—in detail in Chapter 4.)

The formulas for executing the four turtle commands can be derived easily from simple geometric arguments. We illustrate the effect of each of these turtle commands in Figures 1.3 through 1.5.

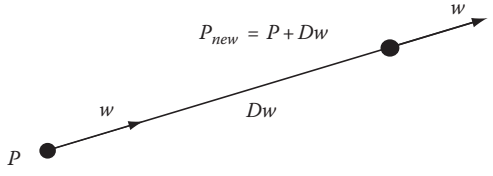

FIGURE 1.3: The command FORWARD D changes the turtle's location, but leaves her direction and step size unchanged. The new turtle location is:

$$P_{new} = P + Dw.$$

Thus, in terms of coordinates,

$$x_{new} = x + Du,$$
$$y_{new} = y + Dv.$$

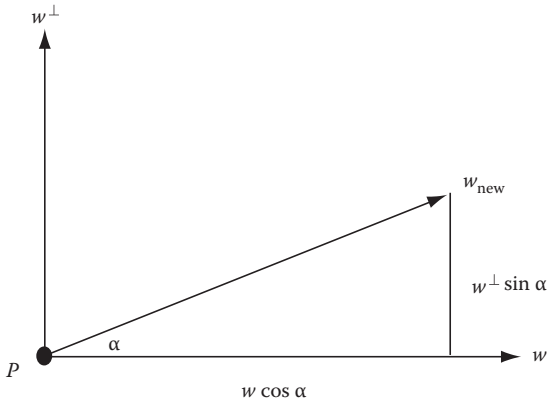

FIGURE 1.4: The command TURN α changes the turtle's heading, but leaves her position and step size unchanged. To derive the turtle's new heading, we work in the turtle's local coordinate system.

Let (P,w) be the turtle's current state, and let $w^\perp$ denote the vector perpendicular to w of the same length as w. Then,

$$w_{new} = w\ \cos(\alpha) + w^\perp\ \sin(\alpha).$$

But if $w = (u,v)$, then $w^\perp = (-v,u)$ (see Exercise 1.11). Therefore, in terms of coordinates,

$$u_{new} = u\cos(\alpha) - v\sin(\alpha),$$
$$v_{new} = v\cos(\alpha) + u\sin(\alpha),$$

or in matrix notation,

$$(u_{new}\ v_{new}) = (u\ v)\begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}.$$

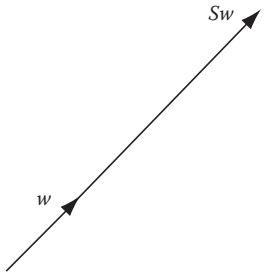

FIGURE 1.5: The command RESIZE S changes the turtle's step size by a factor of S, but leaves her position and heading unchanged. The turtle's new direction vector is given by

$$w_{new} = Sw.$$

Thus, in terms of coordinates,

$$u_{new} = Su,$$
$$v_{new} = Sv,$$

or in matrix notation,

$$(u_{new}\ v_{new}) = (u\ v)\begin{pmatrix} S & 0 \\ 0 & S \end{pmatrix}.$$

The turtle state (P,w) is a complete description of what the turtle knows, and the four turtle commands FORWARD, MOVE, TURN, and RESIZE are the only way that a programmer can communicate with the turtle. Yet, with this simple setup, we can use the turtle to draw an amazing variety of interesting patterns in the plane.

1.3 Turtle Programs

Once the four turtle commands are implemented, we can start to write turtle programs to generate a wide assortment of shapes. The simplest programs just iterate various combinations of the FORWARD, TURN, and RESIZE commands. For example, by iterating the FORWARD and TURN commands, we can create polygons and stars (see Table 1.1). Notice that the angle in the TURN command is the exterior angle, not the interior angle, of the polygon. Circles can be generated by building polygons with lots of sides. Iterating FORWARD and RESIZE, the turtle walks along a straight line, and iterating TURN and RESIZE, the turtle simply spins in place. But by iterating FORWARD, TURN, and RESIZE, the turtle can generate spiral curves (see Figure 1.6).

With a bit more ingenuity (and with some help from the Law of Cosines), we can program the turtle to generate more complicated shapes such as the wheel and the rosette (see Figure 1.7).

The turtle commands FORWARD, TURN, and RESIZE are used to translate, rotate, and scale the turtle. In LOGO, we can also develop turtle programs SHIFT, SPIN, and SCALE to translate, rotate, and scale other turtle programs (see Table 1.2). Examples of SHIFT, SPIN, and SCALE applied to other turtle programs as well as to each other are illustrated in Figures 1.8 through 1.10.

Shapes built by iteration are cute to visualize and fun to build, but by far the most interesting and exciting shapes that can be created using Turtle Graphics are generated by recursion. In Chapter 2, we shall show how to design fractal curves using recursive turtle programs.

TABLE 1.1: Simple turtle programs for generating polygons, stars, and spirals by iterating the basic turtle commands. For the spiral program, A is a fixed angle and S is a fixed scalar.

POLYGON *N*	STAR *N*	SPIRAL *N*, *A*, *S*
REPEAT *N* TIMES	REPEAT *N* TIMES	REPEAT *N* TIMES
FORWARD 1	FORWARD 1	FORWARD 1
TURN $2\pi/N$	TURN $4\pi/N$	TURN *A*
		RESIZE *S*

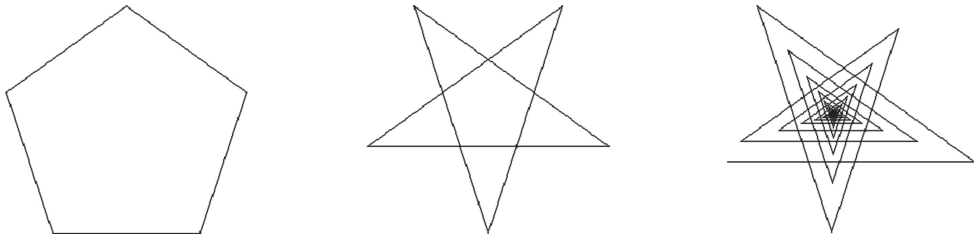

FIGURE 1.6: A pentagon, a five-pointed star, and a spiral generated by the programs in Table 1.1. Here the spiral angle $A = 4\pi/5$ and the scale factor for the spiral is $S = 9/10$.

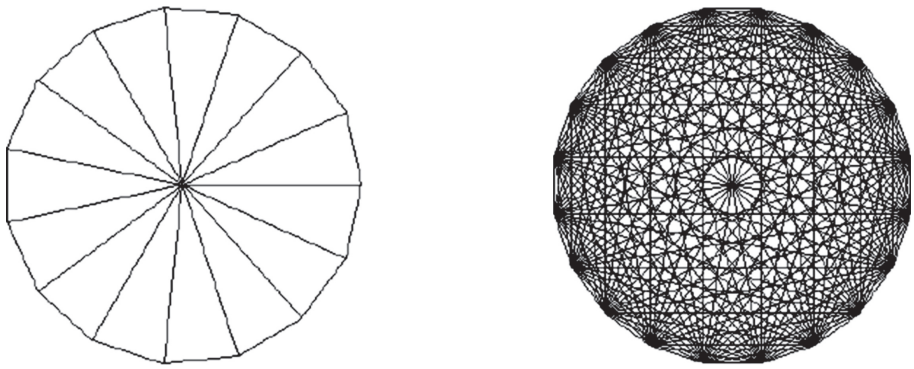

FIGURE 1.7: The wheel and the rosette. The wheel is simply a regular polygon together with the lines connecting its vertices to its center. The rosette is a regular polygon, together with all its diagonals, that is, with the lines joining every vertex to every other vertex. The wheel displayed here has 15 sides and the rosette has 20 sides. We leave it as a challenge to the reader to develop simple turtle programs that generate these shapes, using the Law of Cosines to precalculate the lengths of the radii and the diagonals (see Exercise 1.7).

TABLE 1.2: Turtle programs that translate, rotate, and scale other turtle programs.

SHIFT *Turtle Program*	SPIN *Turtle Program*	SCALE *Turtle Program*
REPEAT *N* TIMES	REPEAT *N* TIMES	REPEAT *N* TIMES
Turtle Program	*Turtle Program*	*Turtle Program*
MOVE *D*	TURN *A*	RESIZE *S*

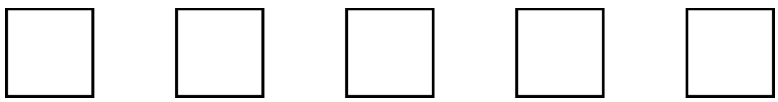

FIGURE 1.8: The effect of applying the SHIFT program to the turtle program for generating a square.

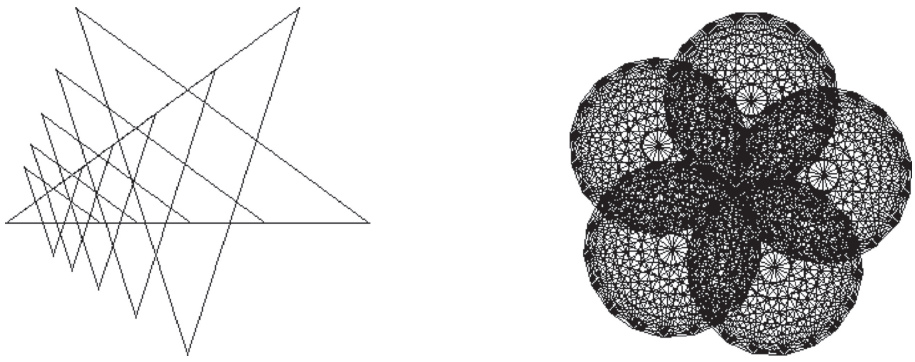

FIGURE 1.9: The effect of SCALE and SPIN. On the left, the SCALE program is applied to the turtle program for generating a star; on the right, the SPIN program is applied to the turtle program for generating a rosette.

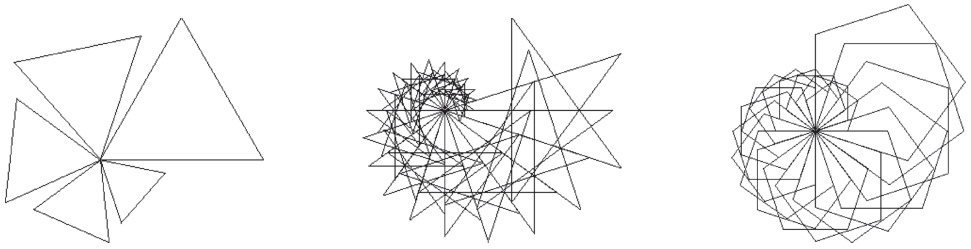

FIGURE 1.10: The effect of composing SPIN with SCALE: starting with a triangle (left), a star (center), and a pentagon (right).

1.4 Summary

Turtle programming is the main theme of this chapter, but this chapter also includes several important leitmotifs. In addition to gaining some facility with turtle programming, you should also pick up on the following distinctions that will be featured throughout this book.

First, there is an important difference between points and vectors. Although in a plane both points and vectors are typically represented by pairs of rectangular coordinates, conceptually points and vectors are different types of objects with different kinds of behaviors. Points are affected by translation, vectors are not. Vectors can be rotated and scaled, points cannot. Thus, if the turtle is in state (P,w), then the FORWARD command translates the point P, but leaves the vector w unchanged. Similarly, the TURN and RESIZE commands rotate and scale the vector w, but leave the point P unchanged.

Second, there are two ways of approaching geometry: conceptual and computational. On the conceptual level, when we think about Euclidean geometry, we think about lengths and angles. Thus, when we program the turtle, we use the FORWARD, TURN, and RESIZE commands to affect distances and directions. Only when we need to communicate with a computer—only when we need to perform actual computations—do we need to descend to the level of coordinates. *Coordinates are a low-level computational tool, not a high-level conceptual device*. Coordinates are akin to an assembly language for geometry. Assembly languages are effective tools for communicating efficiently with a computer, but generally we do not want to think or write programs in an assembly language; rather we want to work in a high-level language. We shall try to keep these two modes—concepts and computations—distinct throughout this book by developing novel ways to think about and to represent geometry. The turtle is only the first of many high-level devices we shall employ for this purpose.

Exercises

1.1. In classical LOGO, the programmer can control the state of the pen by two commands: PENUP and PENDOWN. When the pen is up, the FORWARD command changes the position of the turtle, but no line is drawn. The default state is pen down. Explain why the MOVE D command is equivalent to the following sequence of turtle commands:

```
PENUP
FORWARD D
PENDOWN
```

1.2. In classical LOGO, the turtle's state is represented by a pair (P, A), where P is the turtle's position and A is the angle between the turtle's heading and a fixed direction in the plane. The position P is stored by the xy coordinates of the point P relative to a global coordinate system, and the angle A is the angle between the turtle's heading and the x-axis of this global coordinate system.

   a. Show how to implement the FORWARD and TURN commands using this representation for the turtle's state.

   Suppose we add to the turtle's state a scalar S representing the turtle's step size.

   b. Show how to implement the FORWARD, TURN, and RESIZE commands using the representation (P,A,S)—position, angle, and step size—for the turtle's state.

   c. What are the advantages and disadvantages of the representation (P,A,S) for the turtle's state compared to the representation (P,w) given in the text?

1.3. Write a turtle program that draws a circle circumscribed around:

   a. A polygon with an arbitrary number of sides.

   b. A star with an arbitrary number of vertices.

1.4. Consider the program STAR in Table 1.1:

   a. For what integer values of $N > 5$ does the program STAR fail to draw a star?

   b. What happens if the command TURN $4\pi/N$ is replaced by TURN $8\pi/N$?

1.5. Consider the following program:

```
NEWSTAR N
  REPEAT N TIMES
    FORWARD 1
    TURN π − π/N
```

   a. How do the stars drawn by this NEWSTAR program differ from the stars drawn by the STAR program in Table 1.1?

   b. How do the stars drawn by this NEWSTAR program differ for even and odd values of N? Explain the reason for this curious behavior.

1.6. Consider the following program:

```
TRISTAR N
  REPEAT N TIMES
    FORWARD 1
    TURN 2π/3
    FORWARD 1
    TURN 2π/N − 2π/3
```

   a. How do the stars drawn by this TRISTAR program differ from the stars drawn by the STAR program in Table 1.1?

   b. Suppose that the command TURN $2\pi/3$ is replaced by the command TURN α for an arbitrary angle α, and that the command TURN $2\pi/N - 2\pi/3$ is replaced by the command TURN β. Show that the TRISTAR program still generates a star provided that $\alpha + \beta = 2\pi/N$ and $N\alpha > 2\pi$. What happens when $N\alpha = 2\pi$?

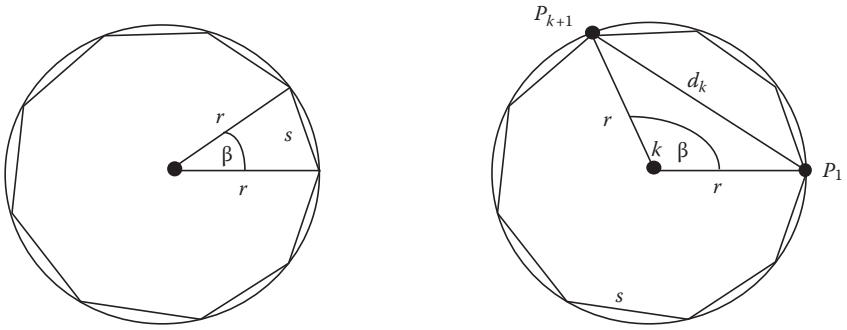

FIGURE 1.11: The Law of Cosines provides a relationship between the length of a side and the distance from a vertex to the center of a regular polygon (left). Similarly, the Law of Cosines provides a relationship between the distance from a vertex to the center of a regular polygon and the length of each diagonal of the polygon. Notice that if the polygon has n sides, then $\beta = 2\pi/n$.

1.7. Write a turtle program that for a regular polygon with an arbitrary number of sides draws:

   a. A wheel

   b. A rosette
   (Hint: See Figure 1.11.)

1.8. Apply the SPIN program to the turtle program for a circle to generate the pattern in Figure 1.1, left.

1.9. Write a turtle program that draws an ellipse.

1.10. Prove that the turtle commands TURN and RESIZE commute by showing that if the turtle starts in the state (P,w), then after executing consecutively the commands TURN A and RESIZE S the turtle will arrive at the same state as when executing consecutively the commands RESIZE S and TURN A. Explain intuitively why this phenomenon occurs.

1.11. Let $w^{\perp}$ denote the vector perpendicular to the vector w of the same length as w. Show that if $w = (u,v)$, then $w^{\perp} = (-v,u)$. (Hint: Use the distance formula and the Pythagorean theorem to show that ΔAOB in Figure 1.12 is a right triangle.)

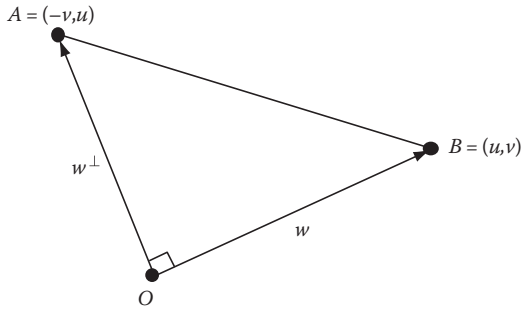

FIGURE 1.12: The vectors $w = (u,v)$ and $w^{\perp} = (-v,u)$ form a right angle at the origin O.