2 From Real Coordinates to Pixels

Throughout the two-dimensional and three-dimensional sections of this book we shall divide the television screen into a text window (of two lines at the top of the screen) and a graphics window (the rest of the screen). As already discussed, on the Model B the graphics window (or *graphics frame*) consists of a rectangular matrix of coloured pixels; the size and number of the pixels and the number of possible colours depend on the MODE setting of the computer (0 to 7). Unless otherwise stated, two-dimensional and three-dimensional geometrical programs will be run in MODE 1. This means that access to these pixels is MODE-dependent and hence we shall consider the graphics frame to be made up of the rectangular matrix of *addressable points* (points for short) that are stacked in NXPIX (= 1280) vertical columns and NYPIX (= 960) horizontal rows. Each pixel on the screen corresponds to a number of different points; although the exact correspondence is naturally MODE-dependent, this need not concern us since the operating system deals with this problem of relationship. Unfortunately the word ‘pixel’ has several different meanings (as do ‘point’ and ‘dot’), so it must be remembered that in this book ‘pixel’ refers to a MODE-dependent group of addressable points on the television screen (see the last column of table 1.1).

Individual points from the set of NXPIX by NYPIX points can be uniquely identified by a bracketed pair of integers, sometimes called a point vector, (I, J), where $0 \leq I \leq NXPIX - 1$ and $0 \leq J \leq NYPIX - 1$; the vector specifies the position of the addressable point in the I^{th} column and J^{th} row, so that the vector (0, 0) identifies the bottom left-hand corner point of the frame. The Model B has its own set of BASIC instructions which enables users to operate on the matrix of addressable points (and hence pixels), so converting them to dots of light on the screen which can be switched off or on in various colours. This enables the operator to produce approximate points, lines, polygons or other special types of area, with a series of colour dots.

The reader will now be taken some way towards generating a two-dimensional and three-dimensional graphics package for the Model B: the programs are given in BASIC and rely (with a few exceptions) on the small number of *primitive* procedures that are given in this chapter.

Primitives that Map Continuous Space on to the Graphics Frame

In general, computer graphics deals with points, lines, areas and volumes in continuous two-dimensional and three-dimensional Euclidean space. The use of addressable point coordinates by themselves for graphics is rather limiting. The definition of objects by means of only discrete pairs of integers is very rare in practical applications. We therefore need to consider ways of plotting views of objects on a graphics screen where positions are measured in real units such as inches, miles or even light-years(!). Therefore we must consider the relationship between two-dimensional real space and the screen pixels (via addressable points). Before we can even attempt this step, however, we must first discuss ways of representing two-dimensional space by means of Cartesian coordinate geometry.

We may imagine two-dimensional space as the plane of this page, but extending, to infinity in all directions. Our description of the coordinate geometry starts by arbitrarily choosing a fixed point in this space, which is called the *coordinate origin*. Through the origin a line is drawn that extends to infinity in both directions - the *x*-axis. The normal convention is to place the *x*-axis from left to right on the page (the horizontal). Another two-way infinite line, the *y*-axis, is drawn through the origin perpendicular to the *x*-axis, hence conventionally this is plotted from the top to the bottom of the page (the vertical). We now draw a scale along each axis; unit distances need not be the same on both axes, but this is normally the case (see figure 2.1). We assume that values on the *x*-axis are positive to the right of the origin and negative to the left: values on the y-axis are positive above the origin and negative below.

We call now uniquely fix the position of point ***p*** in space by specifying its *coordinates* (figure 2.1). The *x-coordinate*, X say, is that distance along the *x*-axis (positive to the right of the axis and negative to the left) at which a line perpendicular to the *x*-axis that passes through ***p*** cuts the *x*-axis. The *y-coordinate*, Y say, is correspondingly defined by using the *y*-axis. These two values, called a coordinate pair or two-dimensional vector, are normally written in brackets thus: (X, Y), the x-coordinate coming before the y-coordinate. We shall usually refer to the pair as a vector - the dimension (in this case dimension two) will be understood from the context in which we use the term. A vector, as well as defining a point (X, Y) in two-dimensional space, may also be used to specify a direction, namely the direction that is parallel to the line that joins the origin to the point (X, Y) - but more of this (and other objects such as lines, curves and polygonal areas) in chapter 3.

We are now in a position to devise means (the above-mentioned primitive procedures for mapping such geometrical concepts on to the two-dimensional discrete rectangular matrix of pixels that forms the graphics frame.

For the present we shall concentrate on two-dimensional space: an extension into three-dimensional space is dealt with starting at chapter 7. In both cases we require a method of mapping a rectangular area of two-dimensional Cartesian

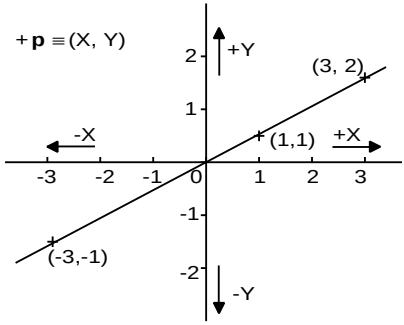


Figure 2.1

space on to the graphics frame. For simplicity we start by insisting that this area has its edges parallel to the *x*-axis and *y*-axis of Cartesian space. Initially we shall assume that this rectangular area of space has its *bottom left-hand corner* identified with the coordinate origin (0.0, 0.0), and that the lengths of the horizontal and vertical edges are HORIZ and VERT respectively. We first identify the origin with the pixel that contains the addressable point (0, 0), and then scale the rectangular area so that it fits into the frame; naturally the area exactly fits the frame only if the ratios HORIZ : VERT and NXPIX : NYPIX are equal (that is for 1280 : 960 = 4 : 3). This is rarely the case, so a scaling factor, XYSCALE, is chosen that maps the point (HORIZ, VERT) in two-dimensional space on to a pixel that is located either on the upper or the right-hand edge of the frame. We can consider this rectangle as a *window* on to Cartesian space; no longer anchored to the coordinate origin, it may wander about space and view rectangular areas that are the same size as the original, although the edges of such areas must be parallel to the original coordinate axes. As a general rule we make VERT roughly three-quarters of HORIZ. If you wish to select a differently shaped window, it is necessary simply to change the values of HORIZ and VERT.

At any time during the execution of the program we may move the coordinate origin from its original position at the bottom left-hand corner of the Hame. Its position relative to the first origin will be stored as XORIG and YORIG; and x-component and y-component respectively. Initially (XORIG, YORIG) is identified with (0.0, 0.0). Hence any point in Cartesian space with coordinates (XPT, YPT), a pair of reals, maps into a pixel that contains an addressable point

with a horizontal component INT((XORIG + XPT) * XYSCALE + 0.5) and a wrtical component INT((YORIG + YPT) * XYSCALE + 0.5). Here INT is the BASIC function that truncates the fractional part of a decimal number and retums an integer. These two components are stored as functions FNX and FNY (see listing 2.2). During the construction of a picture we must consider a *plot pen*, a pair of integers that moves about the graphics aame; initially it is placed at the pixel that is equivalent to the coordinate origin. The constants NXPIX and NYPIX, and the variables XYSCALE, XGRIG and YORIG, must be available at all times to the plotting procedures that follow, so these names must not be used for any other purpose. The procedures were written specifically for the Model B but the general principles of constucting similar procedures fur other graphical devices will also be discussed. To start with, it would be necessary to change the values of NXPIX and NYPIX if a different machine was to be used.

Gur first procedure 'start' defines the graphics and text windows, initialises the required variables and prepares the screen for plotting. Listing 2.1 is an example 'start' procedure for the Model B.

Listing 2.1

```
9700 REM start
9710 DEF PROCstart(B,F)
9720 XORIG=0 : YORIG=0
9730 NXPIX=1280 : NYPIX=960
9740 XYSCALE=NXPIX/HORIZ : YSCALE=NYPIX/VERT
9750 IF XYSCALE>YSCALE THEN XYSCALE=YSCALE
9760 VDU28,0,1,39,0 : VDU24,0;0;1279;959;
9770 GCOL 0,B+128 : CLG : GCOL 0,F
9780 ENDPROC
```

This procedure has two parameters B and F (integers) that represent the background and foreground logical colours respectively. The correct values of these variabes for any choice of colours will depend on the MODE in use, so readers must refer to the user manual. For most purposes we shall use MODE 1 and default logical colours, with foreground black (F = 0) and background white (B = 3).

If we need to write this procedure for a different micro then all that is necessary is to replace the statements that contain the BASIC graphics instructions of the BBC micro by the equivalent commands for the new micro. Most of the other procedures in this book are independent of the structure of the Model B. If at any time during the execution of the program you wish to change the logical colour in which you are drawing, you must use the GCOL option and you can use the VDU 19 option to choose new colours from the list of actual colours.

In 'start' and in many other procedures that follow, it is necessary to transform

the *x/y*-coordinates of a point into their pixel equivalents, so we introduce the two functions FNX and FNY in listing 2.2.

Listing 2.2

```
9800 REM real-to-pixel functions
9810 DEF FNX(Z)=INT((XORIG+Z)*XYSCALE+0.5)
9820 DEF FNY(Z)=INT((YORIG+Z)*XYSCALE+0.5)
```

The next primitive procedure (listing 2.3) is ‘setorigin' ; this enables us to move the coordinate origin by an amount XMOVE horizontally and YMOVE vertically (distances in the scale of the coordinate system), so adjusting the (XORIG, YORIG) values. After such a move the plot pen moves to the pixel that contains the addressable point that is equivalent to the new origin.

Listing 2.3

```
9600 REM setorigin
9610 DEF PROCsetorigin(XMOVE,YMOVE)
9620 XORIG=XORIG+XMOVE : YORIG=YORIG+YMOVE
9630 PROCmoveto(0,0)
9640 ENDPROC
```

We shall be in a position to draw straight lines after we have produced two further procedures: ‘moveto’ which moves the plot pen to a pixel equivalent of the point in coordinate space at one end of the line, and ‘lineto’ which draws the line by moving the plot pen from its present position (set by a previous call to ‘setorigin' , ‘moveto’ or ‘lineto’) to the pixel equivalent ofthe point on the other end of the line. Listings 2.4 and 2.5 show ‘moveto’ and ‘lineto’ procedures that are designed specifically for the Model B. The latter two procedures include statements that initiate the machine-dependent BASIC instructions that MOVE and DRAW between addressable points. Hence if you wish to implement these procedures on a different micro you must introduce the equivalent instructions.

Listing 2.4

```
9500REM moveto
9510DEF PROCmoveto(XPT,YPT)
9520MOVE FNX(XPT),FNY(YPT)
9530ENDPROC
```

Listing 2.5

```
9400 REM lineto
9410 DEF PROClineto(XPT,YPT)
9420 DRAW FNX(XPT),FNY(YPT)
9430 ENDPROC
```

In all but the most elementary machines, it is possible to set up these plotting procedures or their equivalents (and many others as our knowledge increases) in a library file or backing store. Then there is no need to explicitly retype them into each new program. On the Model B we can *SPOOL them as files on audio cassettes or disks, and merge them into other programs by using the *EXEC option. On the companion cassette to this section of the book you will find these procedures as part of the 'lib1' library.

Example 2.1

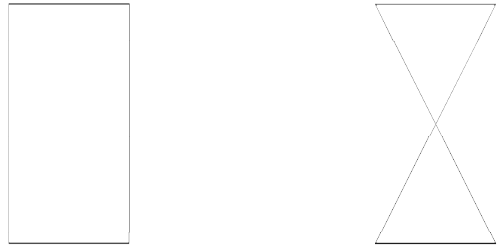
Identify a rectangle in Cartesian space, 40 units by 30 units, with the graphics frame of the Model B. Then draw a square (see listing 2.6) of side 20 units, centred in the rectangle (figure 2.2a). We centre the square on the screen by moving the origin to (20.0, 15.0) and thus definc the corners of the square to be ($\pm$ 10.0, $\pm$ 10.0).

Listing 2.6

```
100 REM drawing a square
110 MODE 1
120 HORIZ=40 : VERT=30
130 PROCstart(3,0)
140 PROCsetorigin(HORIZ/2,VERT/2)
150 PROCmoveto(10,10)
160 PROClineto(-10,10) : PROClineto(-10,-10)
170 PROClineto(10,-10) : PROClineto(10,10)
180 STOP
```

It is as well to note at this juncture that the order in which the points are joined is critical. For example, if the coordinates of the second and third corners of the square are interchanged then figure 2.2b will be drawn.

Next we write a primitive procedure 'triangle' (listing 2.7) which uses the MOVE and PLOT 85 instructions of the Model B for filling in a triangular area bounded by vertices (X1, Y1), (X2, Y2) and (X3, Y3) in logical colour FACECOL (>=0), and draws lines around the perimeter in logical colour EDGECOL (> 0). If one of these values is negative then its corresponding section of the procedure is not entered. Extra lines have been added to cope with degenerate triangles: this is to stop a 'bug' in the early versions of the Model B operating system which produced spurious horizontal lines. If you have operating system OS 1.0 or later then you should simplify 'triangle' .



(a) (b)

Figure 2.2

Listing 2.7

```
10000 REM triangle
10010 DEF PROCtriangle(X1,Y1,X2,Y2,X3,Y3,FACECOL,EDGECOL)
10020 X1=FNX(X1) : X2=FNX(X2) : X3=FNX(X3)
10030 Y1=FNY(Y1) : Y2=FNY(Y2) : Y3=FNY(Y3)
10040 IF FACECOL<0 THEN 10090 ELSE GCOL 0,FACECOL
10050 IF Y1 DIV 4=Y2 DIV 4 AND Y2 DIV 4=Y3 DIV 4 THEN 10080
10060 MOVE X1,Y1 : MOVE X2,Y2 : PLOT 85,X3,Y3
10070 GOTO 10090
10080 MOVE X1,Y1 : DRAW X2,Y2 : DRAW X3,Y3
10090 IF EDGECOL>0 THEN GCOL 0,EDGECOL : MOVE X1,Y1
      : DRAW X2,Y2 : DRAW X3,Y3 : DRAW X1,Y1
10100 ENDPROC
```

Exercise 2.1

If we are using the Model B then it is possible to draw pictures in a variety of colours. We saw in ‘triangle’ that it is necessary to choose the logical colour by means of the GCOL command and we may even wish to define the actual colour by means of the VDU 19 command. Both tasks can be done explicitly in the program or you can write a procedure ‘setcolour' , which uses two integer parameters LOGCOL and ACTCOL to achieve this. Include this in a program that draws solid (as opposed to line) polygons from information read from the keyboard.

Exercise 2.2

In all the plotting procedures above, the scale of the mapping (XYSCALE) is fixed once and for all, and thc horizontal and vertical scaling factors are identical.

There is no need to heed this convention: write a procedure ‘factor’ that alters the horizontal scale by FX and the vertical scale by FY. Naturally this implies that it is now necessary to define two separate scales (XSCALE and YSCALE say); and then of course the functions FNX and FNY must be altered (also see chapter 6).

Exercise 2.3
Furthermore, there is no reason for the x-axis and*y*-axis to be identified with the horizontal and vertical respectively. In fact they need not even be mutually perpendicular. Experiment with these ideas, which necessarily involves changing all the plotting procedures ‘start' , ‘moveto’ etc.

To demonstrate the use of these plotting procedures we shall draw some simple patterns. There are those who think that the construction of patterns is a frivolous waste of time. Nevertheless, we consider it to be a very useful first stage in understanding the techniques of computer graphics. Often patterns of an apparently sophisticated design are the result of very simple programs. Quickly to produce such graphical output is an immediate boost to morale, and gives a lot of confidence to the beginner. Furthermore new designs are always in demand: geometrical art is used for the covers of books and pamphlets and in advertising literature. It can do no harm at all to initiate artistic ideas that will be of great use later when we study the pictorial display of data. Patterns are also an ideal way of introducing some of the basic concepts of computer graphics in a very palatable way. Take the next two examples, which use our simple library of procedures, and look at the important role of trigonometric functions (sine and cosine) and of angular measurement in radians. Remember that pi radians is the same angular measure as 180 degrees.

Example 2.2
Figure 2.3, a very popular design, is constructed by joining each vertex of a regular N-sided polygon (an N-gon) to every other. N cannot be greater than 30.

We set the origin at the centre of the design, and all the vertices at a unit distance from the centre: the sizes of HORIZ and VERT (2.8, 2.1) are chosen so that the design fits neatly on to the screen. If one of these vertices lies on the positive *x*-axis (the horizontal), then the N vertices are all of the form (COS(ALPHA), SIN(ALPHA)), where ALPHA is an angle $2\pi I/N$ and I is chosen from 1 , 2, . . . , or N. Here for the first time we see point coordinates being calculated by the program, not explicitly typed in, as in listing 2.6. Furthermore, ‘ince the prograln uses these values over and over again, it is sensible to store them in arrays and access them when required by specifying the correct array index. Note that in listing 2.8, if $1 \leq I \leq J \leq N$ then the J^{th} point is not joined to the I^{th} point as the line will have already been drawn in the opposite direction.

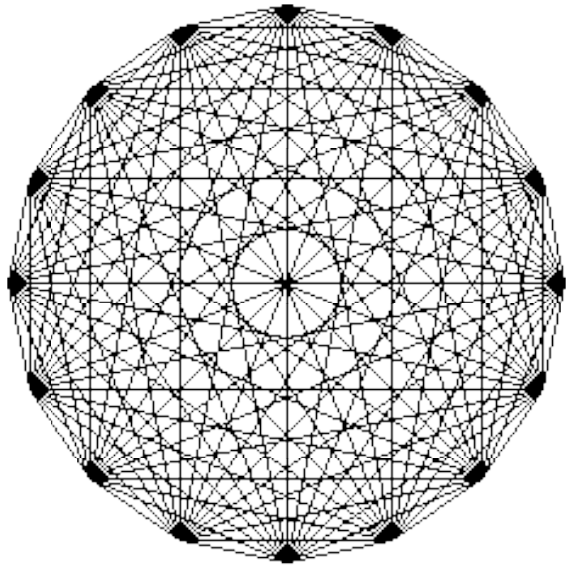


Figure 2.3

Listing 2.8

```
100 REM join points on regular N-gon
110 MODE 1
120 HORIZ=2.8 : VERT=2.1
130 PROCstart(3,0)
140 PROCsetorigin(HORIZ/2,VERT/2)
150 DIM X(30),Y(30)
160 INPUT"Type in N",N
169 REM calculate points in N-gon
170 ALPHA=0 : ADIF=2*PI/N
180 FOR I% = 1 TO N
190 X(I%)=COS(ALPHA) : Y(I%)=SIN(ALPHA)
200 ALPHA=ALPHA+ADIF
210 NEXT I%
219 REM join point I to POINT J , I<J
220 FOR I% = 1 TO N-1
230 FOR J% = I%+1 TO N
240 PROCmoveto(X(I%),Y(I%)) : PROClineto(X(J%),Y(J%))
250 NEXT J% : NEXTI%
260 STOP
```

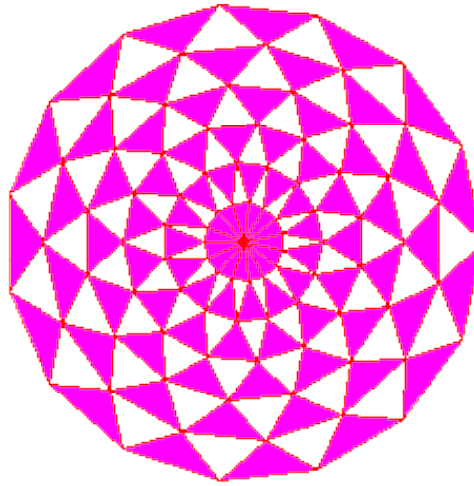


Figure 2.4

Example 2.3

Figure 2.4.is constructed by listing 2.9 in a similar manner. M sets of N points on regular N-gons and one set of N coincident points are given by the following formula. The I^{th} point in the J^{th} set, $I \leq 1 \leq N$ and $0 \leq J \leq M$, is $(R\cos\theta , R\sin\theta)$ where R and θ are given by

$$R = (M - J)/M$$

$$\theta = 2\pi I/N + \alpha$$

where $\alpha = 0$ if MOD(I,2) is zero, and π/N otherwise.

Triangles are then fomied by joining every pair of neighbouring points on all but the inner N-gon to the nearest point inside them.

There are two two immediate observations to be made from these very simple examples. The first concerns *resolution*. Because the graphics frame is a discrete matrix then *straight lines* must be approximated by a sequence of πxels. Unfortunately the resolution of the Model B, like most microcomputer graphics systems, is low (that is, NXPIX and NYPIX are the order of hundreds and the lines appear jagged.

The second observation is that, as N increases in listings 2.8 and 2.9, the outline of the figure (the N-gon) closely approximates to a circle. Therefore we can use this this idea to write a procedure ‘circle’ (listing 2.10) which draws a

Listing 2.9

```
100 REM rose pattern
110 MODE 1
120 DIM X(30),Y(30),XD(30),YD(30)
130 HORIZ=2.8 : VERT=2.1
140 PROCstart(0,3)
150 PROCsetorigin(HORIZ/2,VERT/2)
160 INPUT"N,M",N%,M%
170 AD=PI/N% : AD2=2*AD : RD=1/M%
180 R=1 : AS=0 : A=AS
189 REM setup an outer regular N-gon of unit radius
190 FOR I%=1 TO N%
200 X(I%)=COS(A) : Y(I%)=SIN(A)
210 A=A+AD2 : NEXT I%
219 REM loop through M inner N-gons
220 FOR J%=1 TO M%
230 R=R-RD : AS=AS+AD : A=AS
239 REM setup inner N-gon of radius 1/M smaller than outer N-gon
240 FOR I%=1 TO N%
250 XD(I%)=R*COS(A) : YD(I%)=R*SIN(A)
260 A=A+AD2 : NEXT I%
269 REM form triangles with points from inner and outer N-gons
270 FOR I%=1 TO N%
280 NI%=(I% MOD N%)+1
290 PROCtriangle(X(I%),Y(I%),XD(I%),YD(I%),X(NI%),Y(NI%),1,2)
300 NEXT I%
309 REM reset outer to inner N-gon
310 FOR I%=1 TO N%
320 X(I%)=XD(I%) : Y(I%)=YD(I%)
330 NEXT I%
340 NEXT J%
350 END
```

solid circle with radius R about the centre (XCENT, YCENT) in logical colour FACECOL and circumference in EDGECOL to give a picture siniilar to figure 2.5. If either of these colour parameters is negative then the corresponding section of the procedure is not entered. Note that we are using angles that are measured in radians (that is we are incrementing by 10/(R*XYSCALE) each time through the 1oop), a value that depends on the size of the radius and produces a reasonable circle without waste of effort. Note that since the vertices of the N-gon are needed only once, we do not store their values but calculate them as required. Again the limitation in resolution of the screen is apparent on the perimeter of the circle. Also be wary of rounding errors as these could result in a slice being left out of the circle.

There is another way, other than PLOT 85, for filling in an area. PLOT 77, X, Y moves sideways left and right from the point (X, Y) changing the colour of the pixels until it meets pixels that are not in the background bolour. As an example, see routine ‘circle2’ (listing 2.10). This method does however have obvious limitations when filling in non-convex areas.

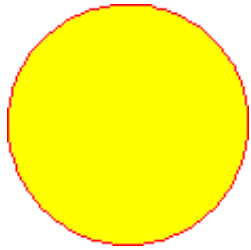


Figure 2.5

Whenever we use such procedures we must be aware of any side effects; for example, the origin or plot head might be modified by the procedure. Thus listing 2.10 changes the position of both the origin and the plot pen. It would theiefore be sensible to add the following line to the 'circle' procedure:

```
480 PRGCsetorigin(-XCENT, -YCENT) : ENDPROC
```

Exercise 2.4

Write a procedure to draw an ellipse of major axis A units (horizontal) and minor axis B units (vertical). Note that a typical point on this ellipse has coordinates (Acos α, Bsin α) where $0 \leq \alpha \leq 2\pi$. However it must be remembered that, unlike the circle, α is not the angle made by the radius through a point with the positive x.-axis: it is simply a descriptive parameter.

Incorporate this procedure into a program that draws a diagram that is similar to figure 2.6. There are two things to note: (1) there is no need for A to be greater than B; (2) observe the optical illusion of the two apparent white diagonal lines. Another illusion can be seen in figure 2.3 - dark circles that radiate out from the centre of the pattern. The study of optical illusions is fascinating (see Tolansky, 1964) and it is a never-ending fount of ideas for patterns. This exercise was introduced because it leads the way into the general technique for drawing curves - see chapters 3 and 6.

Example 2.4

An extension of this idea, the natural next step, is the construction of a spiral. Again the general form of the curve about the origin is (Rcos α, Rsin alpha) but now α varies between angles β and β + 2Npi where beta (the parameter B) is the initial angle that the normal to the spiral makes with the positive *x*-axis, and N is the munber of turns in the spiral. The radius R is no longer a constant value but

Listing 2.10

```
100 REM circle main program
110 MODE1
120 HORIZ=2.8 : VERT=2.1
130 PROCstart(3,0)
140 PROCsetorigin(HORIZ/2,VERT/2)
150 PROCcircle(0.1,-0.1,0.5,1,2)
160 STOP

300 REM circle
310 DEF PROCcircle(XCENT,YCENT,R,FACECOL,EDGECOL)
320 ADIF=10/(R*XYSCALE)
330 PROCsetorigin(XCENT,YCENT)
340 IF FACECOL<0 THEN 420 ELSE GCOL0,FACECOL
349 REM if required draw solid disc
350 MOVE FNX(R),FNY(0)
360 FOR A = ADIF TO 2*PI STEP ADIF
370 MOVE FNX(0),FNY(0)
380 PLOT85,FNX(R*COS(A)),FNY(R*SIN(A))
390 NEXT A
400 MOVE FNX(0),FNY(0)
410 PLOT85,FNX(R),FNY(0)
420 IF EDGECOL<0 THEN ENDPROC ELSE GCOL0,EDGECOL
429 REM if required draw outer circle
430 MOVE FNX(R),FNY(0)
440 FOR A = ADIF TO 2*PI STEP ADIF
450 DRAW FNX(R*COS(A)),FNY(R*SIN(A))
460 NEXT A
470 DRAW FNX(R),FNY(0)
480 ENDPROC

500 REM circle2
510 DEF PROCcircle2(XCENT,YCENT,R,FACECOL,EDGECOL)
520 ADIF=10/(R*XYSCALE)
530 PROCsetorigin(XCENT,YCENT)
540 GCOL0,EDGECOL
549 REM draw outer circle
550 MOVE FNX(R),FNY(0)
560 FOR A = ADIF TO 2*PI STEP ADIF
570 DRAW FNX(R*COS(A)),FNY(R*SIN(A))
580 NEXT A
590 DRAW FNX(R),FNY(0)
599 REM fill in circle
600 GCOL0,FACECOL
610 FOR Y%=FNY(-R) TO FNY(R)
620 PLOT 77,FNX(0),Y%
630 NEXT Y%
640 ENDPROC
```

varies with the value α: if RMAX is the outer radius of the spiral then R is given by the formula

$$R = RMAX(\alpha - \beta)/2N\pi$$

Note that procedure ‘spiral’ (listing 2.11), which centres the spiral at (XCENT, YCENT), causes no side effects because we reset the origin back to its original position before leaving the procedure.

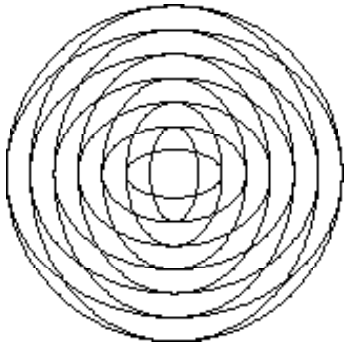


Figure 2.6

The complete program oflisting 2.11 (run in MODE 2) shows another optical illusion. Fifteen spirals in logical colours 1 to 15 are drawn on the screen. Then all the logical colours are turned black and, one at a time, in order, they are turned to white and back to black. This has the effect of producing a rotating spiral: we have added a pause to reduce the speed of rotation. After 30 seconds the screen is cleared and a square is drawn. If you stare at the spirals for this short period and then look at the square, it will appear to expand!

Exercise 2.5
Procedure 'spiral' (with parameters XCENT = 0, YCENT = 0, N = 4, B = 1 and RMAX = 1) produces a diagram similar to figure 2.7a. What happens if you set RMAX to – 1? Use the procedure in a program that generates figure 2.7b. Again note the optical illusion when the observer' s head is moved in a circle in front of the diagram, keeping the horizontal (and hence also the vertical) direction parallel with the original. The spirals appear to rotate about the centre!

Example 2.5
Write a procedure (listing 2.12) that draws diagrams similar to figure 2.8. Here we introduce the concept of an envelope. Instead of drawing a curve by a sequence of small line seginents (as in the circle of listing 2.9), we devise sequence of lines that are tangential to the curve. For example, the figure shows four rectangular hyperbolae that are placed in the *quaters* of the plane.

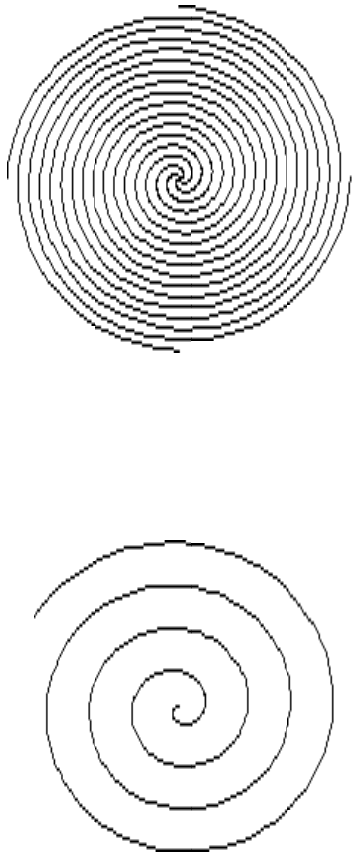


Figure 2.7

Listing 2.11

```
100 REM spiral example
110 MODE 2 : HORIZ=2.8 : VERT=2.1
120 PROCstart(0,3)
130 PROCsetorigin(HORIZ/2,VERT/2)
139 REM setup series of 15 spirals each of a different colour
140 FOR I%=1 TO 15
150 GCOL0,I%
160 PROCspiral(0,0,1.7,4,PI*I%/7.5)
170 NEXTI%
179 REM set each logical colour black
180 FOR I%=1 TO 15
190 VDU19,I%,0,0,0,0
200 NEXT I%
210 N%=1 : O%=15 : T=TIME+3000
220 REPEAT
221 REM set each logical colour to white in turn
      and black out previous 'logical white'
230 VDU19,N%,7,0,0,0,19,O%,0,0,0,0
240 O%=N% : N%= N% MOD 15 +1
249 REM pause
250 FOR I%=1 TO 30 : NEXT I%
260 UNTIL TIME > T
269 REM after 30 seconds draw square and watch the illusion of
      it apparently expanding
270 CLG : VDU20 : GCOL0,1
280 PROCmoveto(.7,.7)
290 PROClineto(.7,-.7) : PROClineto(-.7,-.7)
300 PROClineto(-.7,.7) : PROClineto(.7,.7)
310 END

500 REM spiral
510 DEF PROCspiral(XCENT,YCENT,RMAX,N%,BETA)
520 PROCsetorigin(XCENT,YCENT)
530 ADIF=PI/50 : A=BETA
540 RDIF=RMAX/(N%*100)
550 FOR R=RDIF TO RMAX STEP RDIF
560 PROClineto(R*COS(A),R*SIN(A))
570 A=A+ADIF
580 NEXT R
590 PROClineto(RMAX*COS(BETA),RMAX*SIN(BETA))
600 PROCsetorigin(-XCENT,-YCENT)
610 ENDPROC
```

N points are placed on each of the four arms (of unit length) which divide the plane into the four quarters. The 4N points are therefore (±I/N, 0.0) and (0.0, ±I/N) where I = 1, 2, . . . , N.

Exercise 2.6
Generalise this procedure so that there are a variable number of arms, M, that stretcg out from the origin and divide the plane into equal segrnents.

Exercise 2.7
Draw a diagram similar to figure 2.9. The procedure will have an integer parameter N and should calculate 4N points {I = 1, 2, . . . , 4N)} around the

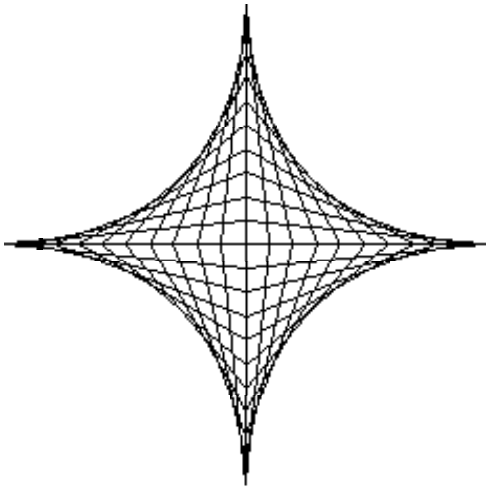


Figure 2.8

Listing 2.12

```
100 REM envelope example
110 MODE 1
120 HORIZ=2.8 : VERT=2.1
130 PROCstart(3,0)
140 PROCsetorigin(HORIZ/2,VERT/2)
150 INPUT"Type N",N%
160 PROCmoveto(1,0) : PROClineto(-1,0)
170 PROCmoveto(0,1) : PROClineto(0,-1)
180 FORI%=1 TO N%
190 X=I%/N% : Y=(N%+1-I%)/N%
200 PROCmoveto(X,0)
210 PROClineto(0,Y) : PROClineto(-X,0)
220 PROClineto(0,-Y) : PROClineto(X,0)
230 NEXT I%
240 STOP
```

edges of a square of unit side, starting at a corner. There is one point at each corner and the points are placed so that the distance between consecutive points is 1/N. Then pairs of points are joined according to the following rule: P(I) is joined to P(J) for all positive values of I and J less than or equal to 4N, such that J – 1 (subtraction modulo 4N) belongs to the sequence 1, 1 + 2, 1 + 2 + 3, ... For example if N is 10 then P(20) is joined to P(21), P(23), P(26), P(30), P(35), P(1), P(8) and P(16).

Figure 2.9

Example 2.6
Emulate a Spirograph, in order to produce diagrams similar to figure 2.10.

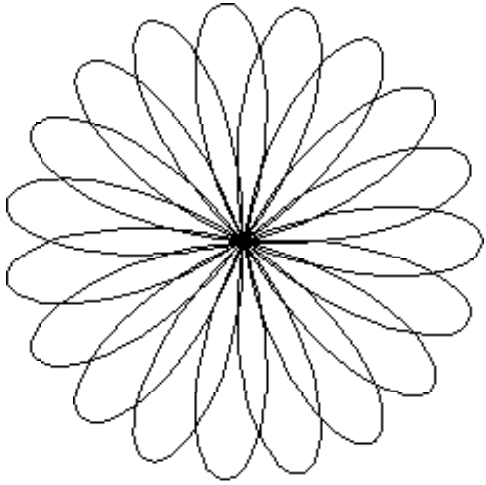


Figure 2.10

A Spirograph consists of a cogged disk inside a cogged circle, which is placed on a piece of paper. Let the outer circle have integer radius A and the disk integer radius B. The disk is always in contact with the circle. There is a anall hole in the disk at a distance D (also an integer) from the centre of the disk, through which is placed a sharp pencil point. The disk is moved around the circle in an anticlockwise manner, but it must always touch the outer circle, the cogs ensure that there is no slipping. The pencil point traces out a pattern, which is complete when the pencil returns to its original position.

Initially we assume that the centres ofthe disk and the circle and also the hole all lie on the positive *x*-axis, the centre of the circle being the coordinate origin. In order to emulate the Spirograph we need to specify a general point of the track of the pencil point. We let α be the angle made with the positive *x*-axis by the line that joins the origin to the point where the circle and disk touch. The point of contact is therefore (Acos α, Asin α) and the centre of the disk is ((A – B)cos α, (A – B)sin α). If β is the angle made by the line that joins the hole to the centre of the disk with the x-direction, then the coordinates of the hole are

((A – B)cos α + Dcos β, (A – B)sin α + Dsin β)

The point of contact between the disk and circle will have moved through a distance Aalpha around the circle, and a distance -Balpha around the disk (the minus sign is because α and β have opposite orientation). Since there is no slipping these distances must be equal and hence we have the equation β = -(A/B)α. The pencil returns to its original position when both α and β are integer multiples of 2pi. When α = 2Npi then β = -N(A/B)2π, and hence the pencil point returns to its original position for the first time when N(A/B) becomes an integer for the first time, that is when N is equal to B divided by the highest common factor of B and A. The function ‘hcf’ (listing 2.13) uses Euclid' s algorithm (see Davenport, 1952) to calculate the highest common factor of two positive integers A and B.

This function is used in the procedure ‘spiro’ (listing 2.13) which calculates tht value of N and then varies α (ALPHA) between 0 and 2Npi in steps of π/100; for each α, the value of β (BETA) is calculated and then the general track is drawn. Figure 2.10 was drawn by a call to ‘spiro’ with A = 12, B = 7 and D = 5. The size of HORIZ and VERT must be chosen so that the figure fits on to the screen; in this case HORIZ = 28 and VERT = 21.

It is evident from this example that drawing patterns is not as straightforward as it appears. Even such a simple picture as figure 2.10 requires the mathematical backup of Euclid. As we progress through computer graphics, we shall discover more and more that it is essential to have at least an elementary knowledge of not only coordinate geometry but also calculus, algebra, Euclidean geometry and number theory. Be prepared to scour your local library (or pester your friendly neighbourhood mathematician) for the necessary information.

Listing 2.13

```
200 REM hcf - Euclid's algorithm
210 DEF FNhcf(A%,B%)
220 LOCAL I%,J%,R%
230 IF A%>B% THEN I%=A% : J%=B% ELSE I%=B% : J%=A%
240 R%=I% MOD J% : IF R%=0 THEN =J% ELSE I%=J%: J%=R% : GOTO 240
300 REM spiro
310 DEF PROCspiro(A%,B%,D%)
320 LOCAL I%,RAB%,ALPHA,BETA,ADIF,AOB,N%,NO%
330 RAB%=A%-B% : ALPHA=0 : ADIF=PI/50 : AOB=A%/B%
340 N%=B%/FNhcf(A%,B%) : NO%=100*N%
350 PROCmoveto(RAB%+D%,0)
360 FOR I%=1 TO NO%
370 ALPHA=ALPHA+ADIF : BETA=ALPHA*AOB
380 PROClineto(RAB%*COS(ALPHA)+D%*COS(BETA),RAB%*SIN(ALPHA)-D%* SIN(BETA))
390 NEXT I%
400 ENDPROC
```

Complete Programs

At this stage we shall group the listings 2.1 (‘start’), 2.2 (two functions FNX and FNY), 2.3 (‘setorigin’), 2.4 (‘moveto’), 2.5 (‘lineto’) and 2.7 (‘triangle’) under the heading ‘lib1' .

I ' lib1’ and listing 2.6 (‘drawing a square’). No INPUT data required.

II ' lib2’ and listing 2.8 (‘joining points of regular N-gon’). Data required: an integer N <= 30.

III ' lib1’ and listing 2.9 (‘rose’). Data required: the values of M and N. Use $N \leq 30$ and for the best results set $M < N \leq 15$

IV ' lib1’ nd listing 2.10 (‘main program’ and ‘circle’). No data required.

V ' lib1’ and listing 2.11 (‘main program’ and ‘spiral’). No data required. When the spirals start to rotate you should stare at the screen for 30 seconds. The square appears to expand.

VI ' lib1’ and listing 2.12 (‘envelope’). Data required: an integer N. For best results titke $2 \leq N \leq 30$.

VII ' lib1' , listing 2.13 (‘Euclid’ and ‘spiro’) and your own main program (use listing 2.10 as a model). Data required: three integers A, B and D, where A > B > D. Choose HORIZ, VERT etc. so that the diagram lits on to the screen; that is, both HORIZ and VERT must be greater than 2 * (A – B + D). Try HORIZ = 28, VERT = 21, A = 12, B = 7, D = 5.