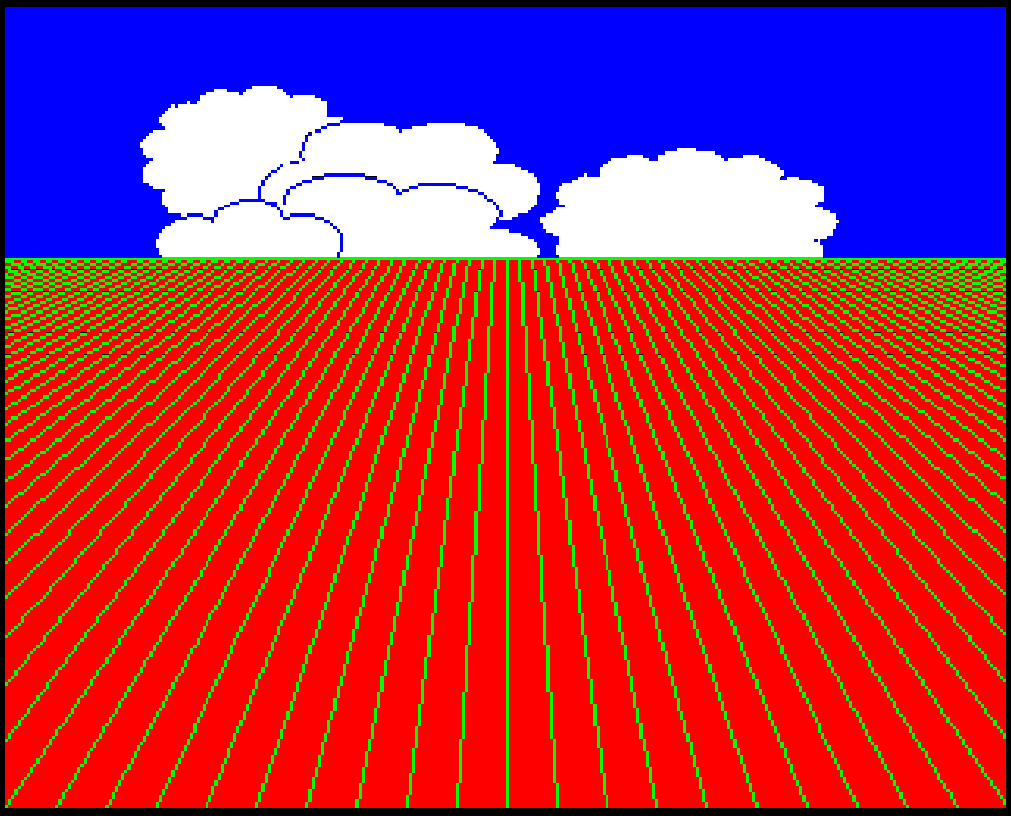
Creative Graphics

on the BBC Microcomputer

JOHN COWNIE



Creative Graphics

on the BBC Microcomputer

JOHN COWNIE

ACORNSOFT

Acknowledgement

Thanks to Tim Dobson for contributing the program entitled 'CIRCLES'.

FIRST EDITION

ISBN 0 907876 03 X

Published by:

Acornsoft Limited
4a Market hill
Cambridge
CB2 3NJ England

DIGITALLY REMASTERED ON ACORN RISC OS COMPUTERS, NOVEMBER 2011.

Contents

Index to programs

FIELD*	Windy field	88
MERRYGO	Merry-go-round	93
RAINBOW	Rainbow	98
ISLAND*	Desert island	101

*Model A users should refer to Appendix A for program modifications. All other programs will run on the Model A if the initial MODE statement is changed to MODE 5 (see Appendix A).

FIELD*	Windy field	88
MERRYGO	Merry-go-round	93
RAINBOW	Rainbow	98
ISLAND*	Desert island	101

*Model A users should refer to Appendix A for program modifications. All other programs will run on the Model A if the initial MODE statement is changed to MODE 5 (see Appendix A).

Introduction

The BBC microcomputer provides high resolution graphics at a remarkably low cost, and the programs in this book arise from a desire to show off this machine's immensely powerful graphics capabilities.

Usually programs like these are consigned to a backroom inhabited by hairy programmers, only to be revealed to the public at exhibitions and demonstrations. Here a wealth of useful techniques are explored to provide some remarkably compact and potent programs. In most cases the program as printed is readily adaptable to produce new and interesting effects.

Programs will run on either Model A or B

As listed all the programs will run directly on the Model B.

Programs can be run on the Model A simply by altering the mode statement at the start of the programs from MODE 1 to MODE 5. The price paid for this alteration will be some loss of resolution or a reduction in the number of colours used, but despite this the visual impact of most of the programs will hardly be affected. In the case of two programs (FIELD and ISLAND) further modifications are necessary, and these are listed in Appendix A at the back of the book.

1 Graphics Commands

At first sight the BBC Microcomputer provides a bewildering range of graphics commands. In most applications it is possible to achieve the same result in a number of different ways, and while this provides great scope for individuality and expressive power, it can also appear very baffling to the novice, Superb effects can be produced with a small subset of the many facilities available. The most useful facilities are explained below.

Modes

The MODE command clears the screen and sets up the mode specified. For graphic programs the most important differences between modes are:

1 The number of colours allowed
2 The resolution, or size, of each pixel

Modes with high resolution and many colours require the most memory.

MODE	Resolution	Colours	Memory
0	640x256	2	20k
1	320x256	4	20k
2	160x256	16	20k
4	120x256	3	10k
5	160x256	4	10k

MODE 0 provides very high resolution but only 2 colours. In MODE 1 the resolution is quite good and the pixels are square. MODE 2 gives rather chunky graphics but lots of colours. MODEs 4 and 5 can be regarded as watered-down versions of 1 and 2, and are the only graphics modes available on the Model A (l6k) machine. All the programs in this book use MODES I or 2, but they can easily be converted to run in MODEs 4 or 5.

Pixel sizes

All graphics modes use the same coordinate system to refer to points on the screen. The screen of the monitor or TV set is divided into a grid of hundreds of tiny squares (or, more accurately, rectangles).

There are 1280 squares from left to right on the screen (graphically speaking the x-axis) and 1024

squares from top to bottom (which by the same analogy can be thought of as the y-axis). The origin (0,0) is at the bottom left-hand corner of the screen.

To refer to these squares individually we use the same method as used to refer to points on a graph: each square is referred to by its (x,y) coordinates.

Every picture drawn on the screen is composed of clusters of these squares called 'pixels': if magnified, even rounded shapes on the screen will be seen to be made up of squares. Small pixels (composed of few squares) will give a smoother curve and a more accurate picture than large pixels, and so small pixels are said to give 'high resolutlon' images, whereas large pixels cause chunky, jagged results.

The only disadvantage of having small pixels is that more pixels are needed to fill the screen and so more memory is required. In all modes the pixels are composed of more than a single square on the coordinate grid.

MODE	Pixel size in coordinate units
0	2x4
1	4x4
2	8x4
4	4x4
5	8x4

One of the advantages of the coordinate system is that programs can be run in different modes since the size and shape of an object drawn will be the same regardless of the mode.

Drawing

All drawing is done by controlling the location of the graphics cursor. The cursor is like the tip of a pen with which you would draw on a piece of paper. You can control where the cursor moves and whether it draws a line as it moves. The location of the graphics cursor is not shown on the screen.

Useful commands

MOVE X,Y
Move the cursor to the point (X,Y). Note that X and Y can be numbers, expressions, variables or functions.

DRAW X,Y
Draw a line from the current position of the cursor to the point (X,Y), It does not matter if (X,Y) is off the screen.

PLOT 69,X,Y
Put a dot at (X,Y).

PLOT 85,X,Y
Fill in the triangle formed by the last two points visited by the cursor and the point (X,Y),

PLOT 0,X,Y
Move the position of the cursor relative to the current position.

PLOT 1,X,Y
Draw a line relative

PLOT 81,X,Y
Fill the triangle formed by the last two points visited and the point obtained by moving (X,Y) relative to the current position.

Examples

(Note: Example programs are not provided on the Creative Graphics cassette,)

1 This sequence will fill a triangle with vertices at (200,200), (800,200) and (200,800):

```
l0 MODE 1
20 MOVE 200,200:MOVE 800,200:PLOT 85,200,800
30 END
```

2 This will scatter white dots at random over the screen:

```
l0 MODE 1:REPEAT
20 PLOT 69,RND(l279),RND(l023)
30 UNTIL FALSE
```

3 A random walk:

```
l0 MODE 1
20 MOVE 640,512
30 REPEAT
40 PLOT 1,RND(3l)-l6,RND(3l)-l6
50 UNTIL FALSE
```

Colours

To understand the way in which colours are controlled it is important to grasp the idea of 'physical' and 'logical' colours. Actually, it is best to discover

how the palette works through trying it out at the keyboard, but here is an explanation of sorts.

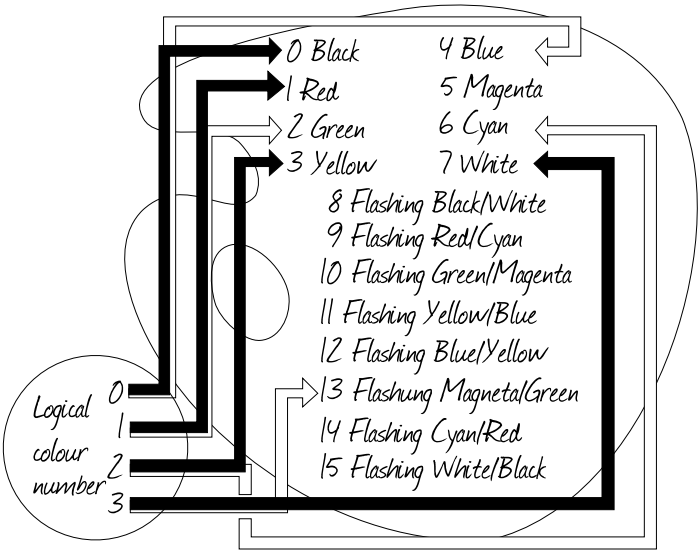
The logical colour is just a number which is assigned to any of l6 physical colours and causes that colour to appear physically on the screen. The logical colours are numbered as follows: 0,1,2, or 3 in the case of 4-colour modes and 0-15 in the case of lti-colour modes. The physical colours are numbered from 0 to 15.

Logical colour produces (Code)	2-colour modes	**Physical colour (Code)**	
0		0	Black
1		7	White
	4-colour modes		
0		0	Black
1		1	Red
2		3	Yellow
3		7	White
	16-colour mode		
0		0	Black
1		1	Red
2		2	Green
3		3	YelIow
4		4	B1ne
5		5	Magenta
6		6	Cyan
7		7	White
8		8	Flashing black-white
9		9	Flashing red-cyan
10		10	Flashing green-magenta
11		11	Flashing yellow-blue
12		12	Flashing blue-yellow
13		13	Flashing magenta-green
14		14	Flashing cyan-green
15		15	Flashing white-black

Note that it is only in the 16-colour mode that all the logical colours are set up to correspond to the physical colour with the same number.

The relationship between physical colours that appear on the screen and the internal logical colours is referred to as the palette. This specifies which physical colour will appear on the screen for a given internal logical colour, as shown diagrammatically

below.



A MODE 1 or MODE 5 statement sets up a palette as described by the black arrows in this diagram - to start with, logical colour 0 is black, logical colour 1 is red, and so on.

To change the palette, you just 'change the arrows' as you wish using a VDU statement reserved specially for this purpose. The white arrows in the diagram show a possible set of changes.

Changing the palette

To change the palette use the VDU 19 statement as follows :

```
VDU 19,L,P;0;
```

where the logical colour number L is made to appear as the physical colour number P on the screen.

Example

Enter MODE 1 or MODE 5, and then enter

```
VDU 19,0,4;0;
```

In MODEs 1 and 5 logical colour 0 is black to begin with, so this statement causes the black on the screen to change colour to physical colour 4 (blue).

Similarly, a further statement

```
VDU 19,3,6;0;
```

causes the logical colour 3 (the white text) to become physical colour 6 (cyan). Now to change the background colour (blue at present) we must refer back to its logical colour - ie 0 (since it was black to begin with). The statement

```
VDU19,0,1;0;
```

will change the background colour to red.

Changing the palette is a powerful facility, and can be used to dramatic effect since the colour on the screen can be changed almost instantly. Here are some of the effects that can be produced:

1 If the logical colour is set to the background colour while an object is being drawn and the palette is changed, 'instant' plotting is produced.

2 In modes with few colours it is possible to select from any of the 8 possible steady colours which will appear on the screen at a given moment.

3 Palette changes can produce moving images.

Selecting a colour

The GCOL command selects which logical colour to draw with and the action that drawing will have on colours already on the screen. After a GCOL command any DRAW or PLOT command will use the colour specified by GCOL. The format of GCOL is as follows:

```
GCOL A,C
```

where A specifies the mode of action and logical colour to use.

There are five possible modes of action:

A=0 plot colour specified regardless of anything already there (paste on top)
A=1 OR logical colour with logical colour already there
A=2 AND logical colour with logical colour already there

A=3 EOR logical colour with logical colour already there

A=4 invert logical colour already there

The most common actions used are 0 and 3. GCOL0,C could be used to paint over all or part of an object in the background — when drawing a cloud in the sky for example. GC0L3,C, the Exclusive-OR action, is important in animated pictures, where an object can move around over the background without affecting it. So, in the arcade game Monsters, your man can run up and down between levels without 'wiping out' the ladders as he goes. Note also that with GCOL3,C plotting the same line twice will erase that line.

Moving the origin

It is possible to move the origin to any point on the screen with the command:

```
VDU 29,X;Y;
```

which moves the origin to the absolute point (x,y). This is useful when drawing objects that are not naturally defined with the origin at the bottom left-hand corner of the screen.

Turning off the text cursor

In graphics programs the blinking text cursor can be turned off in two ways:

```
VDU 5
```

This links the text and graphics cursors, and anything that is printed will appear at the position of the graphics cursor.

```
VDU 23,10,32,0;0;0;
```

This is harder to remember but has no side effects. The text cursor is turned off by loading register l0 of the 6845 screen controller 1C with 32.

Interlace

Interlace causes the screen to judder up and down slightly. This produces the effect of rounding the characters on the screen. Some people find interlace irritating, particularly in graphics programs. It can be turned off like this:

```
VDU 23;8,0;0;0;
```

Sideways scrolling

The region of memory that the screen displays can be altered by reprogramming registers 12 and 13 of the 6845 screen controller 1C. This allows you to scroll the screen sideways. Together these registers bold the l4-bit start address (divided by 8) of the top of the screen; register 13 holds the 8 lower bits, while register 12 holds the top 6 bits.

These registers can be altered like this:

```
VDU 23;register number,value,0;0;0;
```

The FIELD program in Chapter 6 gives more details on using this technique.

Procedures in BASIC

Many of the programs in this book depend for their success on the use of procedures. However small the program, using procedures is good programming practice, and it will be well worth your while getting the hang of how they work. You will inevitably become familiar wi th their use and appreciate their val ne more as you work through this book, but to begin with, here is a brief introduction.

A procedure is like a subroutine in which certain values, called parameters, are set up automatically when the procedure is called. Here is an example that fills the screen with random squares, using the procedure PROCSQ to draw each square:

Example

```
 10 MODE 1
 20 REPEAT
 30 PROCSQ(RND(1300),RND(1000),72)
 40 GCOL 0,RND(3)
 50 UNTIL FALSE
 60 END
 70 DEF PROCSQ(X%,Y%,S%)
 80 VDU 29,X%;Y%;
 90 MOVE 0,0:MOVE S%,0:PLOT 85,S%,S%
100 MOVE 0,S%:PLOT 85,0,0
110 ENDPROC
```

Description of the program

```
10  4-colour mode

20        Each time around this loop a square is
          drawn.
```

30	Call the procedure PROCSQ to draw a square at a random point with a side length 72.
40	Select a random colour.
50	Carry on forever.
60	This is never reached, but helps to show where the body of the program ends and the procedure declarations begin.
70	PROCSQ draws a square at the point (X%,Y%) with a side length S%.
80	Put the origin at (X%,Y%).
90-100	Fill in the square.
110	Return to the point at which PROCSQ was called.

Here, a procedure that draws a square is defined in 70-110. The procedure takes three parameters: and S%, and these specify the position and size of the square.

Each time the procedure is called at line 30 the 'formal parameters' X%, Y% and S% used in the definition take the values of the 'actual parameters' given for that particular call of the procedure; here these are RND(l300), RND(l000) and 72. The procedure is then executed and on completion control is returned to the statement after the call.

Procedures can be called from anywhere within a program, and the order in which procedures are declared does not matter.

A few of the advantages of procedures are listed below.

1 Procedures make programs much easier to understand.

2 Dividing a problem into small chunks makes to write and test programs.

3 Different effects can be produced simply by altering the parameters supplied to a procedure.

4 Modular programs are less likely to suffer from obscure bugs due to unwanted interactions between different parts of the program.

2 Functions and Symmetry

BBC BASIC provides a large number of built-in functions which can greatly simplify many programs. The diversity and power of the functions available often make it possible to produce the same effect in many different ways. This allows the user to adopt the clearest and most natural construction to express his ideas, without being unduly inhibited by the limitations of the language. An analogy can be found in natural languages, which include a vast spectrum of expressive words making it possible to communicate ideas in many subtle and colourful ways.

Three ways of drawing a circle

The next three programs all draw a circle. They demonstrate that even a simple circle offers a wealth of opportunities for selecting the particular technique you feel happiest with.

1 Iterative method

This method can produce a good approximation to a circle very rapidly . I t does not call any trigonometrical functions, which tend to be rather slow. The disadvantages of this technique are that it can produce some unpredictable results, and that it is not immediately obvious how it works:

CIRCLE1

```
  0 REM Iterative method
 20 MODE 1
 30 S%=20
 40 VDU29,640;512;
 50 X=500
 60 Y=90
 70 MOVEX,Y
 80 REPEAT
 90 DRAWX,Y
100 X=X+Y/S%
110 Y=Y-X/S%
120 UNTIL POINT(X,Y)<>0
130 END
```

Description of program

30 Determine the size of the steps around the circle.
50-70 Define (X,Y) as the starting point.
100-110 Calculate next point.
120 Carry on until the line bumps into itself.

2 Polar coordinate method

This method is based on the polar coordinate equation for a circle. A polar coordinate consists of an angle and a length. In this system a circle is described by specifying a fixed length and an angle. To draw a circle each of the polar points must be converted into an (x,y) coordinate. The polar point with angle A and length S Is the same as the point S*COS(A),S*SIN(A).

The method is very simple to program; the size of the circle is easily altered, and it is very clear what is going on. This method can easily be adapted to draw ellipses and forms the basis for many interesting patterns like lissajoux figures.

CIRCLE2

```
 0 REM Polar method
20 MODE1
30 VDU29,640;512;
40 S%=400
50 MOVE0,S%
60 FORA=0 TO 2*PI STEP PI/30
70 DRAWS%*SIN(A),S%*COS(A)
80 NEXT
90 END
```

Description of program

40 Make the radius of the circle S%.
60 Take angle A (measured in radians), and step around the circle.
70 Draw a line to the next point on the circle.

3 Quadratic solution method

This method is based on the fact that the equation for a circle, centred about the origin, is X*X+Y*Y=S*S, where S is the radius of the circle.

By rearranging the equation we get Y=SQR(S*S-x*x), and using this we can substitute many values for X and obtain the corresponding Y-coordinate on the edge of

the circle.

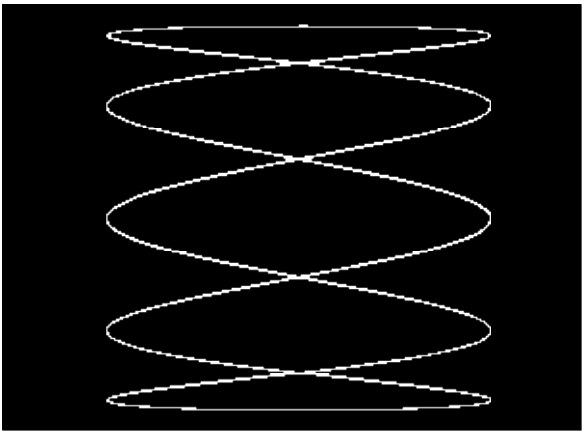
CIRCLE3

```
    0 REM Quadratic method
   20 MODE1
   30 VDU29,640;512;
   40 S%=400
   50 FOR T%=0 TO 1
   60 MOVE -S%,0
   70 FOR X%=-S% TO S% STEP 8
   80 Y%=SQR(S%*S%-X%*X%)
   90 IF T%=1 THEN Y%=-Y%
  100 DRAWX%,Y%
  110 NEXT X%,T%
  120 END
```

Description of program

50	Determine which half of the circle is being drawn.
70	Generate X-coordinates.
80	Calculate Y-coordinates.
90	Pick either positive or negative. square-root depending on T%.
100	Draw the next part of the circle.

Lissajoux figures



Lissajoux figures are fascinating patterns that can form the basis for many weird and wonderful programs. The method for drawing lissajoux figures is similar to

the polar-coordinate method for drawing a circle. For the circle , the angle from which the x- and y-coordinates are derived is the same. Different lissajoux figures are obtained when these angles are out of phase. The following program draws a different lissajoux figure each time the Space Bar is pressed.

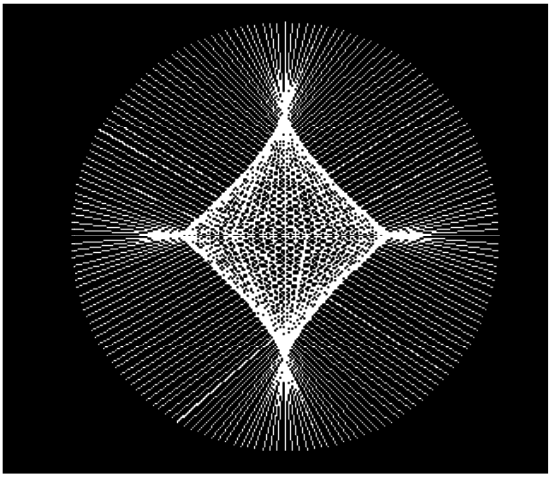
LISS1

```
    0 REM Lissajoux figures
   20 MODE1
   30 VDU23;10,32;0;0;0;
   40 VDU29,640;512;
   50 FORF=0 TO 4 STEP 0.2
   60 MOVE 0,400
   70 A=0
   80 REPEAT
   90 A=A+0.1
  100 DRAW 400*SIN(A),400*COS(A*F)
  110 UNTIL INKEY(0)=32
  120 CLS
  130 NEXT
  140 END
```

Description of program

30	Turn off the cursor.
40	Make (0,0) the centre of the screen.
50	Determine step size.
60-90	Start at the top left-hand corner of the screen, and increment A by 0.1 each time round the loop.
100	Draw the pattern.
110	Wait for the Space Bar to be pressed.
130	Start another pattern.

Lissajoux pattern



There are many ways in which the basic lissajoux patterns can be enhanced. Here is a program that uses straight lines to join the points that trace out two intermeshing figures. The pattern obtained depends on the random numbers chosen at lines 50 and 60. If you press ESCAPE at any stage a new pattern will begin.

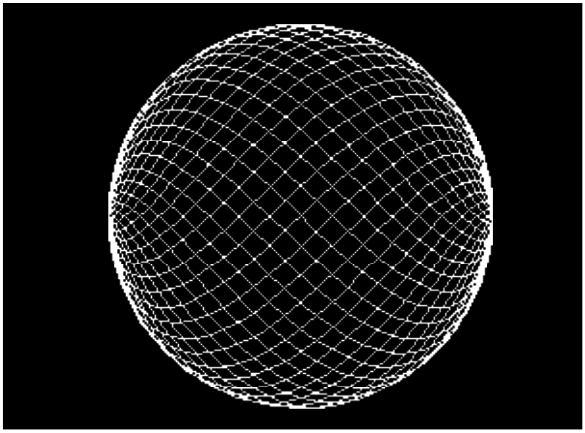
LISS2

```
    0 REM Lissajoux pattern
   20 ON ERROR GOTO 30
   30 MODE1
   40 VDU5
   50 B%=RND(5)
   60 C%=RND(5)
   70 VDU29,640;512;
   80 GCOL0,RND(3)
   90 FORA=0TO 1000 STEP PI/30
  100 X%=250*COS(A)
  110 MOVE X%,Y%
  120 DRAW500*COS(A/B%),500*SIN(A/C%)
  130 NEXT
```

Description of program

20	Jump to line 30 when ESCAPE is pressed.
50-60	B% and C% effect the shape of the outer figure.
100-110	Move the point back and forth along the line x=y
120	Draw a line to the point that runs around the lissajoux figure.

Ball of wool



Here is an example of the kind of pattern that can be created starting from the basic idea of lissajoux figures. A wealth of similar patterns can be explored simply by piling up different combinations of SIN and COS functions and seeing what comes out.

WOOLBAL

```
    0 REM Ball of wool
   20 MODE1
   30 VDU5
   40 GCOL0,RND(3)
   50 S%=400
   60 VDU29,640;512;
   70 MOVE0,0
   80 FORA=0 TO 125.7 STEP 0.1
   90 DRAW S%*SIN(A),S%*COS(A)*SIN(A*0.95)
  100 NEXT
  110 REPEAT UNTIL FALSE
```

Description of program

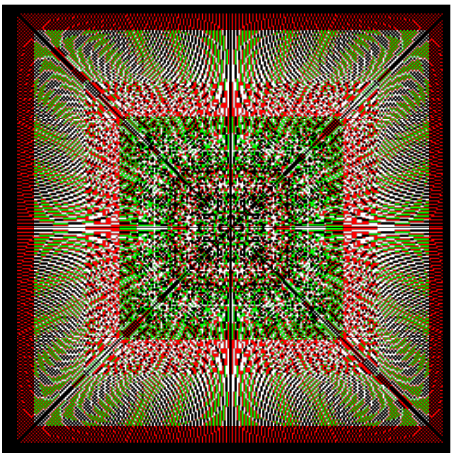
50	S% is the radius of the ball.
80	The value 125.7 is about the angle at which the pattern starts repeating itself.
90	Believe it or not this will trace out a ball.
110	Loop forever on this line once the pattern has been drawn.

Symmetry

By exploiting the symmetry of a pattern it is often possible to simplify and speed up the program used to generate it. In graphic applications much calculation can be avoided by using reflections in the axis to generate the symmetric parts of the object being drawn.

Apart from straightforward reflection like this, there is also the invaluable option on the BBC Microcomputer of moving the origin from one place to another during the program, and this used in con junction with reflection can produce some even more impressive designs.

Persian Carpet



This program produces a pattern similar to that of a Persian carpet. To achieve this effect the program draws a series of radial lines through the centre of a square. The symmetric nature of the carpet is exploited in line 90 where the x— and y-coordinates of the previous line are interchanged.

The GCOL action Exclusive-OR causes lines that overlap either to cancel out or to produce a new colour.

Interference patterns of this kind are known as 'Moiré patterns'. The carpet is produced by superimposing many of these patterns, each of a different size.

CARPET

```
    0 REM Carpet
   20 MODE1
   30 VDU5
   40 VDU29,640;512;
   50 FORS%=20TO500STEP40
   60 GCOL3,RND(3)
   70 VDU19,RND(3),RND(8)-1;0;
   80 FORX%=-S%TOS%STEP 8
   90 MOVE -S%,X%:DRAW S%,-X%
  100 MOVE X%,-S%:DRAW -X%,S%
  110 NEXT X%,S%
  120 GOTO50
```

Description of program

20	4-colour mode.
30	Remove text cursor.
40	Put the origin in the centre of the screen.
50	S% is the size of the pattern being drawn. A new layer of pattern is drawn each time round this loop.
60	Select drawing action EOR and one of the three colours at random.
70	Change the palette so that any of the non-flashing colours can appear.
80-110	Draw the pattern.
120	Start again.

Possible changes

This program offers many opportunities for exploring variations on the original. Some interesting alterations are suggested below.

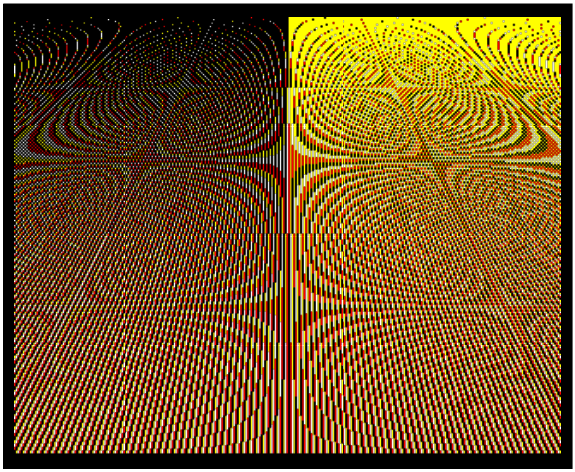
1 Try running the program in MODE 0 or MODE 2.

2 The step size at line 50 can be changed.

3 The step size at line 80 has a critical effect on the type of pattern produced. Interesting effects can be produced with smaller, larger or random step sizes here.

4 By moving the origin for each pattern you could fill the screen with lots of small carpets to produce a patchwork quilt.

Multicoloured pattern



This program draws a pattern that is constructed from a series of diverging lines running down the screen. The program produces a fascinating combination of colours, and shows how the appearance of a colour can be affected by the colours next to it.

PATTERN

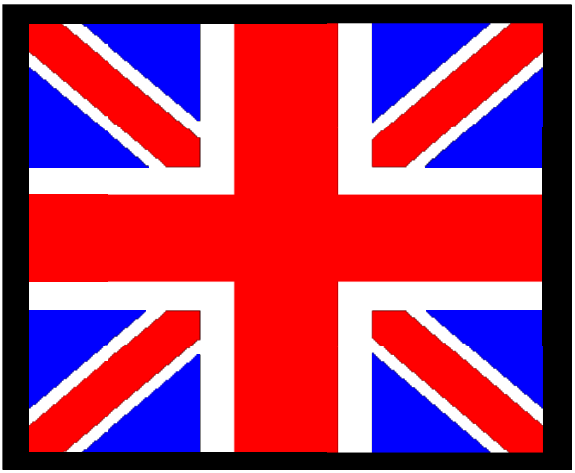
```
  0 REM Multicoloured pattern
 20 MODE1
 30 VDU5
 40 VDU29,640;0;
 50 FORP%=0 TO 640
 60 GCOL0,P%AND3
 70 MOVEP%*4,0:DRAWP%,1024
 80 MOVE-P%*4,0:DRAW-P%,1024
 90 NEXT
100 REPEAT
110 VDU19,RND(3),RND(7);0;
120 A=INKEY(60)
130 UNTIL FALSE
```

Description of program

60 Select a logical colour between 0 and 3.
70-80 Draw symmetric lines to the left and right of the y-axis.

100 Loop around changing the palette forever.
120 Wait for 0.6 of a second.

Flag



This program uses a procedure which will draw rectangular Union Jacks. The procedure allows the flag to be any size or shape of rectangle, and it be can positioned anywhere on the screen. The drawing method exploits many of the symmetrical elements in the flag. (Actually the 'real' flag has some subtle asymmetric bands but this procedure should produce a flag that satisfies all but the most pedantic readers.)

FLAG

```
  0 REM Flag
 20 MODE1
 30 VDU23;10,32;0;0;0;
 40 PROCJACK(640,512,600,500)
 50 REPEAT UNTIL FALSE
```

Description of program

30 Turn off the blinking text cursor.
40 Draw a flag centred about the point (640,512) with sides of length 1200 and 1000.
50 Do nothing forever.

PROCJACK

This procedure draws a flag centred about (X%,Y%) with sides of length 2*SX% and 2*8*1%.

```
   60 DEFPROCJACK(X%,Y%,SX%,SY%)
   70 VDU24,X%-SX%;Y%-SY%;X%+SX%;Y%+SY%;
   80 VDU29,X%;Y%;
   90 VDU19,2,4;0;
  100 GCOL0,130
  110 CLG
  120 GCOL0,2
  130 FORJ%=5 TO 8 STEP 3
  140 Y1%=SY%+SY%/J%:Y2%=SY%-SY%/J%
  150 IF J%=5  THEN GCOL0,3ELSE GCOL0,1
  160 FORI%=0 TO 1
  170 MOVE-SX%,-Y1%:MOVE-SX%,-Y2%:PLOT85,SX%,Y2%
  180 MOVESX%,Y1%:PLOT85,-SX%,-Y2%
  190 Y1%=-Y1%:Y2%=-Y2%
  200 NEXT I%,J%
  210 FORJ%=3 TO 5 STEP 2
  220 IF J%=3 THEN GCOL0,131 ELSE GCOL0,129
  230 VDU24,-SX%/J%;-SY%;SX%/J%;SY%;
  240 CLG
  250 VDU24,-SX%;-SY%/J%;SX%;SY%/J%;
  260 CLG
  270 NEXT
  280 VDU26
  290 ENDPROC
```

Description of PROCJACK

70	Define a graphics window around the whole flag.
80	Put the origin at the centre of the flag.
90	ake logical colour 2 appear as dark blue.
100	Select the graphics background colour to be logical colour 2.
120	Fill the graphics window with the selected graphics background colour. This method of filling a rectangle will not work if the rectangle is partially off the screen.
130-200	Draw the diagonal bands.
210-270	Draw the horizontal bands.
280	Restore the default text and graphics windows.

Sketch Pad

This program allows you to draw lines on the screen, using the keyboard to control the pattern produced.

The cursor starts in the middle of the screen and leaves a trail behind it as you move it about the screen. The control keys are shown below:

```
    0 REM Sketch Pad
   20 MODE1
   30 VDU5
   40 MOVE 640,512
   50 REPEAT
   60 PLOT1, 4*(INKEY(-98)-INKEY(-67)),
4*(INKEY(-105)-INKEY (-73))
   70 UNTIL FALSE
```

Description of program

30 Remove text cursor.

60 INKEY(—98) returns -1 if the 'Z' key is depressed; otherwise it returns zero. The other INKEY functions on this line perform similar functions for the keys 'X', '/' and ':' .PLOT 1,x,y draws a line relative to the current cursor positon.

3 The Third Dimension

To represent a 3-dimensional object on the screen a number of tricks can be used to fool the observer into interpreting the flat image as a 3—dimensional view. Luckily, because the brain is very good at extracting 3—dimensional information from a flat picture, only a few simple depth cues are required to create the illusion of the third dimension.

These simple techniques are very easy to program, and simple programs can produce excellent results. For more general 3-D views and truly lifelike effects much more complicated techniques and considerable computing power are needed.

Colours

The choice of colours for different parts of an object can help give an impression of depth. Cold or dark colours tend to appear to be further away than warm bright colours. This is especially marked on the BBC machine with the colour dark blue, which gives the impression of great distance.

Polo

Here is a program that uses different colours to give the illusion of depth. The object drawn is a hoop composed of a spiral looping around a circle which gives a similar shape to the popular mints of the same name. To draw the hoop we trace around a small circle as its centre moves round the hoop. The impression of depth is obtained by squashing the small circle in the y-direction, and colouring the most distant half of each loop dark-blue.

POLO

```
 0 REM Polo
20 MODE1
30 VDU19,2,4;0;
40 VDU29,640;512;
50 VDU5
60 R%=40
70 MOVE 12*R%,0
80 FORA=0 TO 2*PI STEP 0.01
90 T=A*50
```

```
100 S=SIN(T)
110 IF S>0 THEN GCOL 3,2 ELSE GCOL 0,RND(2)*2-1
120 DRAW R%*(10*COS(A)+2*COS(T)),R%*(10*SIN(A)+S)
130 NEXT
140 GOTO80
```

Description of program

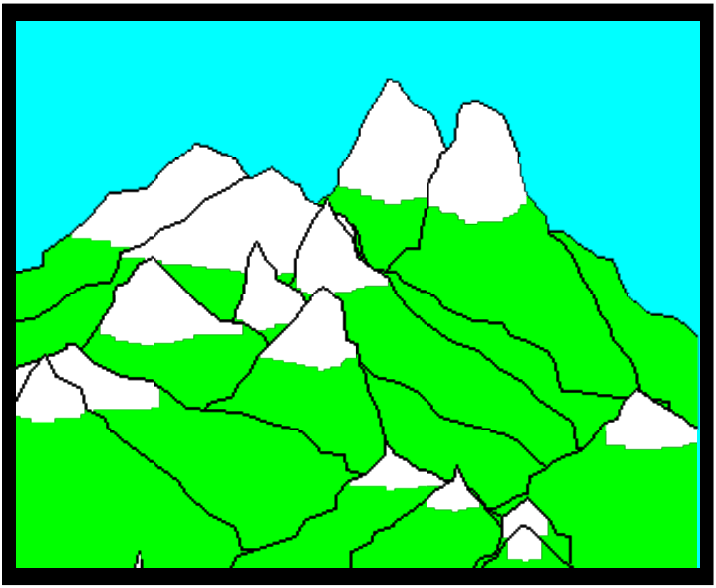
20	Select a 4-colour mode.
30	Change the palette so that logical colour 2 appears as dark blue.
40	Put the origin in the centre of the screen.
60	R% affects the size of the circles.
70	Move to the first point.
80	Step around the large circle.
90	Save A*50 as T to give a marginal increase in speed.
100	This avoids calculating SIN(T) more than once each time around the loop.
110	Select the colour and plotting action. The action EOR ensures that the blue lines will always appear behind the red and white line. RND(2)*2—l selects either red or white.
120	Draw a line to the next point around the spiral. COS(A) and SINUA) control the position in the large circle. COS(T) and S control the position in the small circles.
140	Start again.

Hidden line removal

One of the most obvious observations about a 3.imensional view is that you cannot see an object if it is behind something. While this is very simple to understand in a 3—dimensional world it causes many complications if we want to represent a [s-dimensional view on a flat screen.

To give the impression of depth distant objects must be obscured, or partly obscured, by the objects in the foreground. The main problems are how to tell which parts of a distant object are not visible and how to draw a partly-obscured object, which may be a very different shape from the original. The next program uses a very simple method to achieve the required effect.

Mountains



This program draws a view of randomly-generated snow-capped mountains as shown in the photograph above. The colours are chosen at random, and although this can produce some ridiculous effects, it can also produce colour schemes that are reminiscent of the subtle hues of an alpine landscape.

The largest most distant mountains are drawn first, and then 'closer' mountains are put on top of these obscuring them where they overlap. This technique for eliminating hidden lines, by drawing from the back and then over-plotting, is useful in many applications. Although at first sight it may seem to be rather slow and extravagant, the alternatives are considerably more complicated and probably not much faster. The method to use will obviously be governed by the nature of the object being drawn, but the 'pasting on top' approach, as used here, gives an easy solution for irregular objects.

The technique relies on the ability to fill in areas of colour rapidly. For line drawings a mask around the foreground object is filled in black (or whatever the background colour is) and then the object is drawn on top of the mask.

When the entire picture is complete the program will wait for any key to be pressed before starting again.

Each mountain is drawn as follows:

1 Choose a random point (X_PEAK%,Y_PEAK%) to be the summit.

2 Choose two values X_SLOPE% and Y_SLOPE% to determine the slope of the right-hand side of the mountain.

3 Draw in steps down the mountain-side by adding random x and y values, determined by the slope, to the current position. The space below the line drawn is filled in with the logical colour 2. If we are still near the top of the mountain the area below the edge down to the snow-line is filled with logical colour 3.

4 The side is followed until it goes off the screen.

5 The method above is repeated for the left-hand side.

6 The mountain is now complete.

MOUNTS

```
    0 REM Mountains
   20 MODE1
   30 VDU19,0,6;0;
   40 VDU5
   50 FOR MOUNTAIN%=900 TO 0 STEP -60
   60 X_PEAK%=RND(1200)
   70 Y_PEAK%=MOUNTAIN%+RND(50)
   80 FOR SIDE%=0TO1
   90 X_SLOPE%=RND(40)+20
  100 Y_SLOPE%=RND(20)+30
  110 MOVE X_PEAK%,Y_PEAK%
  120 X%=X_PEAK%:Y%=Y_PEAK%
  130 REPEAT
  140 IF SIDE%=0 THEN X1%=X%+RND(X_SLOPE%) ELSE
X1%=X%-RND(X_SLOPE%)
  150 Y1%=Y%-RND(Y_SLOPE%)
  160 SNOW_LINE%=Y_PEAK%-Y1%/5-50:GCOL0,2
  170 MOVEX1%,Y1%: PLOT85,X1%,0: MOVEX%,
0:PLOT85,X%,Y%
  180 IF SNOW_LINE%<Y1% THEN GCOL0,3:MOVEX1%,Y1%:
PLOT85,X1%,SNOW_LINE%:MOVEX%,SNOW_LINE%:PLOT85,X%,Y%
  190 X%=X1%:Y%=Y1%
  200 GCOL0,1:DRAWX%,Y%
  210 UNTIL POINT(X%,Y%)=-1
  220 NEXT SIDE%
  230 VDU19,RND(3),RND(8)-1;0;
  240 NEXT MOUNTAIN%
  250 A=GET:GOTO50
```

Description of program

30	Have a light-blue sky.
40	Remove the text cursor by linking the text and graphics cursor.
50	Each time around this loop a mountain is drawn. The y-coordinate of the peak is slightly above MOUNTAIN%.
60-70	Select a point for the peak.
80	Each time around this loop a single side of the mountain is drawn.
90-100	Select the slope of the mountain-side.
110	Move to the top of the mountain. '
120	The point (X%,Y%) is the current position on the mountain-side.
130	In this loop we step down the slope until we run off the edge of the screen.
140-150	(Xl%,Yl%) is the next point down the mountain-side.
160	Calculate where the snow-line will be, and select the mountain colour.
170	Fill the region below the line between (X%,Y%) and (Xl%,Yl%).
180	If we are above the snow-line draw some snow.
190	Move (X%,Y%) one step down.
200	Draw a line along the mountain-side, so that it will stand out from more distant mountains.
210	Carry on until we run off the edge of the screen.
230	Change the colour scheme at random.
250	Wait for a key to be pressed then start again.

Perspective

It is not too hard to write a generalised routine that will produce a perspective view of a ii-dimensional object. The routine could take a stored representation of the object and allow you to view it from any point.

This approach requires 3-dimensional information about the object to be stored in the program, and the stored representation is then acted on by the viewing routine to give the required perspective image which is projected onto a flat plane. The perspective view is rather like the shadow cast by the object.

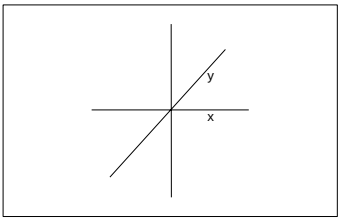
For simple programs this method would be rather

cumbersome and slow, and has many facilities that are not required to give just one view of the object.

A more straightforward method is to draw a single perspective view of the object directly on the screen, the illusion of depth being created by suitably distorting the shape of the flat object. Some of the techniques used to give the impression of ti-dimensional objects are illustrated below:

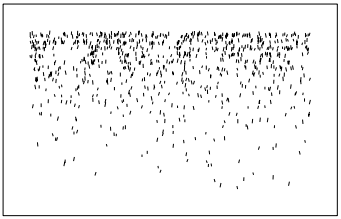
1 Axis

In this view of a 3-D axis it is easy to imagine the y-axis running out of the screen. Distant points on this axis are represented by displacing the point to the right and moving it up slightly.



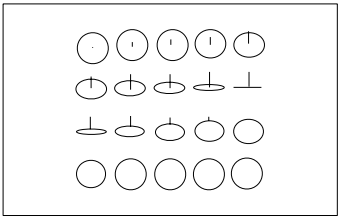
2 Small distant objects; more numerous

he further away an object is the smaller it appears. Also you can see more objects the further away they are.



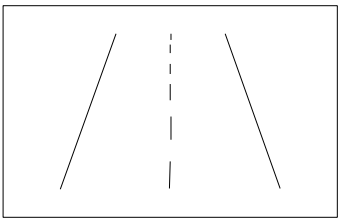
3 Squashed circles

Here, simply squashing a circle gives the impression of viewing a flat disc from an angle.

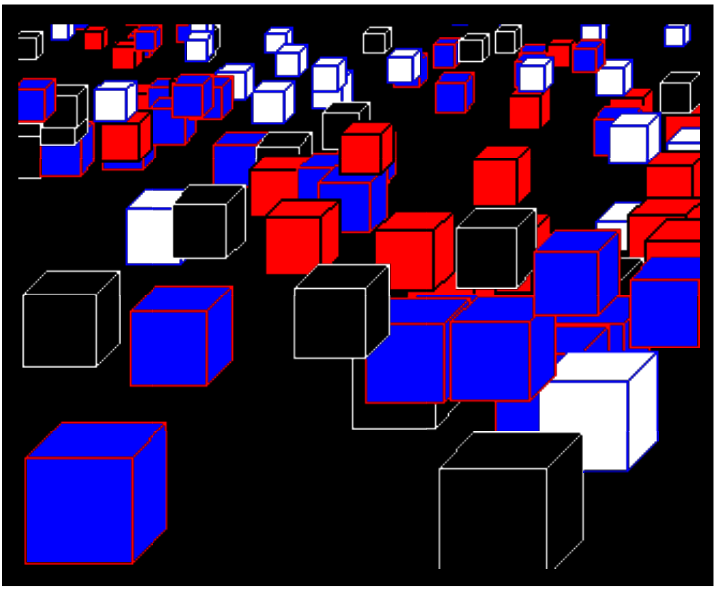


4 Converging lines

These are often used to depict a road stretching into the distance.



Cubes



This program uses a procedure that draws a very simple perspective view of a cube to fill the screen with advancing cubes. The most distant cubes are drawn first and then obscured by the foreground cubes which are 'pasted on top'. The 3-13 effect is enhanced by drawing a large number of small cubes in the distance, drawing fewer in the foreground, and increasing the size of the cubes as they approach the observer.

CUBES

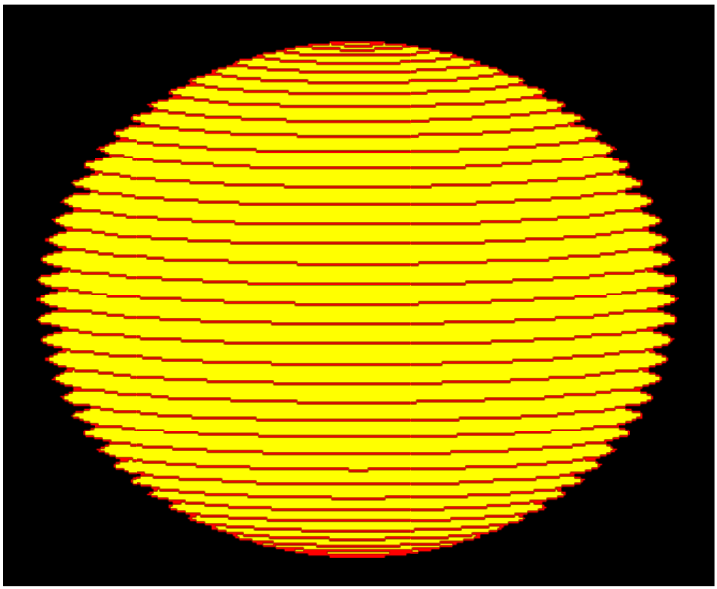
```
    0 REM Cubes
   20 MODE1
   30 VDU5
   40 VDU19,2,4;0;
   50 REPEAT
   60 FORY%=0 TO 1200 STEP 10
   70 H%=1100-RND(Y%)
   80 PROCCUBE(RND(1300)-50,H%,(1200-H%)
/6,RND(4)-1)
   90 NEXT
  100 VDU19,RND(3),RND(7);0;
  110 UNTIL FALSE
  120 DEFPROCCUBE(X%,Y%,S%,C%)
  130 D%=S%/3:E%=S%+D%
```

```
  140 VDU29,X%;Y%;
  150 GCOL0,C%
  160 MOVE0,0:MOVE0,S%:PLOT85,D%,E%
  170 MOVE0,0:PLOT85,E%,E%
  180 MOVE0,0:PLOT85,E%,D%
  190 MOVE0,0:PLOT85,S%,0
  200 GCOL0,C%+3
  210 DRAWS%,S%:DRAW0,S%:DRAW0,0:DRAWS%,
0
  220 MOVE0,S%
  230 DRAWD%,E%:DRAWE%,E%
  240 DRAWE%,D%:DRAWS%,0
  250 MOVES%,S%:DRAWE%,E%
  260 ENDPROC
```

Description of program

20	4-colour mode.
30	Link text and graphics cursors to get rid of the text cursor.
40	Make logical colour 2 appear as dark blue.
50	Each time around this loop the cubes advance to the front of the screen.
60	Move Y% down the screen.
70	Make H% the Y% coordinate at which a cube is drawn. Calculating H% like this ensures that fewer cubes are drawn in the foreground.
80	Draw a cube with random colour and x-coordinate. The size grows as the y-coordinate decreases.
100	Change the palette at random.
120	PROCCUBE(X%,Y%,S%,C%) draws a cube of size S% in logical colour C% at the point X%,Y%.
130	E% and D% are used to draw the far side of the cube.
160-190	Fill in the cube.
210-250	Draw the edges of cube.

Sphere



The following program draws a 3-dimensional view of a sphere. The sphere is represented by a series of flat circular discs. The program reads in the following values:

ST% controls the step size around the loop that draws each disc, and how many discs are drawn. Good results are obtained when ST% is 50.

S% is the radius of the sphere. To fill the screen make 8% about 500.

C1% and C2% specify the colours of the centre and edge of the discs. They must be in the range 0 to 3.

SPHERE

```
  0 REM Sphere
 20 MODE1
 30 VDU29,640;512;
 40 DIM C(300)
 50 INPUT"Step size ? "ST%
 60 INPUT"Size ?"S%
 70 INPUT"Central colour ? "C1%
 80 INPUT"Edge colour ? "C2%
 90 CLS
100 VDU23;10,32,0;0;
```

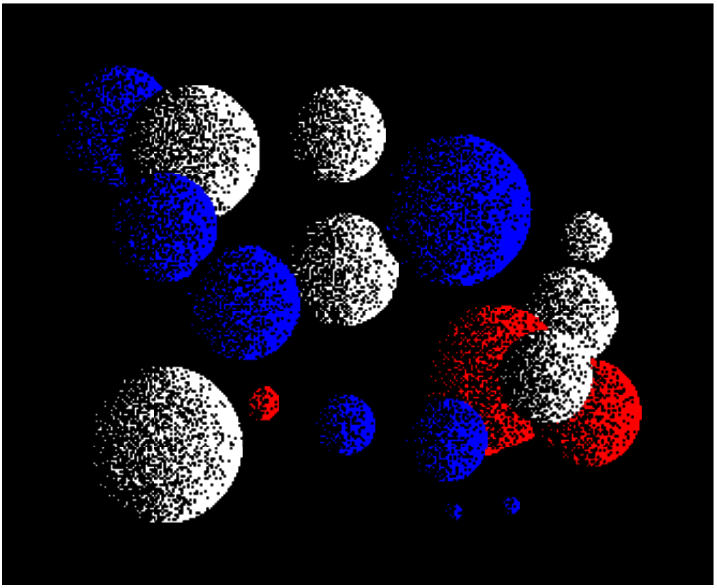
```
110 I%=-2
120 FORA=0TO 2*PI+PI/ST% STEP PI/ST%
130 I%=I%+2
140 C(I%)=S%*COS(A)
150 C(I%+1)=S%*SIN(A)
160 NEXT
170 FORFI=-PI/2 TO PI/2 STEP PI/ST%
180 VDU29,600;500+S%*0.8*SIN(FI);
190 MOVE0,0
200 CS=COS(FI)
210 F=CS*SIN(PI/ST%)
220 GCOL0,C1%
230 FORJ%=0 TO I% STEP 2
240 X%=CS*C(J%):Y%=F*C(J%+1)
250 MOVE0,4:PLOT85,X%,Y%
260 NEXT
270 GCOL0,C2%
280 MOVE CS*C(0),F*CS*C(1)
290 FORJ%=2 TO I% STEP 2
300 DRAW CS*C(J%),F*C(J%+1)
310 NEXT J%,FI
320 GOTO50
```

Description of program

20	4-colour mode.
30	Move the origin to the centre of the screen.
40	The array C will hold the coordinates of points around the edge of each disc that is drawn .
50-80	Read in values for the step size around each loop, the size of the sphere, and the logical colour numbers of the centre and edge of each disc.
100	Remove the text cursor.
110	I% is used to index the array C(200).
120-160	In this loop we store all the coordinates the points around the edge of the largest disc.
170	Draw a disc each time around this loop.
180	Move the origin to the centre of the disc to be drawn.
200-210	CS and F are used to reduce the size of the x- and y-coordinates of the stored disc, so that the discs give the outline of a sphere.
220	Select the colour of the centre of the disc.
230-260	Draw the centre of the disc.
270	Select the colour of the line around the edge of the disc.
280-310	Draw the line around the edge of the disc.

```
320       Go back and start again.
```

Planets



One of the limitations of the graphics on the BBC microcomputer is that there is no control over the brightness and intensity of the colours, making it hard to produce the shading effects that are required to represent lifelike 3-dimensional objects.

This program gets round this problem by using a random distribution of coloured dots to give the impression of different colour intensities.

The intensity of colour in a region depends on the number of pixels that are set - for maximum brightness all the pixels will be on, the dimmest effect is achieved when all the pixels are off, and there is a range of values between these extremes. For each pixel in the region we choose a random number between I and a maximum value (for example 100 to give 100 different shades). If the random number chosen is less than the region's intensity number then we set that pixel; otherwise it is black.

The limitation of this method is that it is rather slow since the random function is called for every dot in the region. It could be speeded up by using a pseudo-random number obtained by incrementing a randomly-chosen pointer into part of the ROM.

PLANET1

```
    0 REM Planets
   20 MODE1
   30 VDU5
   40 REPEAT
   50 VDU29,RND(1000)+100;RND(800)+100;
   60 LC%=RND(3)
   70 SIZE%=RND(150)
   80 SIZES%=SIZE%*SIZE%
   90 FORY%=-SIZE%TOSIZE%STEP4
  100 X%=SQR(SIZES%-Y%*Y%)
  110 X2%=2*X%
  120 FORI%=-X%TOX%STEP4
  130 IF RND(X2%)-X%<I% THEN GCOL0,LC% E
LSE GCOL 0,0
  140 PLOT69,I%,Y%
  150 NEXT I%,Y%
  160 VDU19,LC%,RND(7);0;
  170 UNTIL FALSE
```

Description of program

20 4-colour mode.

30 Remove text cursor.

40 Draw a planet each time round this loop.

50 The centre of the planet will be at the origin, defined to be at a random position not too close to the edge of the screen.

60 LC% is the logical colour in which the planet is drawn.

70 SIZE% is the radius of the planet.

90 The planet is drawn with lines of dots. In MODE 1 each dot is a 4x4 square, hence STEP 4.

100 The point (X%,Y%) is the end of the current line of dots.

120 This loop draws a line of dots.

130 This determines whether a dot is light or dark. For each dot a random number is picked and compared with the position of that dot. This ensures that there is 100% chance of a dot being bright on the far right-hand side and 0% on the left, with a smooth range of

intermediate values across the planet.

140 Draw a dot at the point (I%,Y%).

160 Change the palette so that any of 7 colours can appear.

Character-defined planets

The disadvantage of the previous program is that it runs very slowly. The following program produces a similar effect but is much faster - it also demonstrates how to define your own characters.

PLANET2

```
    0 REM Character Defined Planets
   20 MODE1:VDU5
   30 FORF=0TO1:CLG:GCOL0,3
   40 VDU29,500;500;:C%=2
   50 SIZ%=C%*32:SIZS%=SIZ%*SIZ%
   60 FORY%=-SIZ%TOSIZ%STEP4
   70 X%=SQR(SIZS%-Y%*Y%)
   80 X2%=2*X%
   90 FORI%=-X%TOX%STEP4
  100 R%=RND(X2%):IFF=0THEN R%=0
  110 IFR%<I%+X%THEN PLOT 69,I%,Y%
  120 NEXT,
  130 PROCCH(504-SIZ%,500+SIZ%,SIZ%*2,SI
Z%*2,224+15*F)
  140 NEXT
  150 REPEAT
  160 VDU29,RND(1000);RND(1000);
  170 FORI=0TO1
  180 IF I=0 THEN R%=225:GCOL0,0 ELSE R%
=240:GCOL0,RND(3):VDU19,RND(3),RND(7);0;
  190 FORJ%=0TO 32-SIZ%*2STEP-32
  200 FOR I%=0TOSIZ%*2-32STEP 32
  210 MOVEI%,J%:VDUR%:R%=R%+1:NEXT,:NEXT
  220 UNTIL0
```

Description of program

30 The first time around this loop we draw a solid cirle, then a shaded planet; both are saved as a sequence of characters.

60 STEP 4 determines the number of characters needed to save the planet. Valid values are in the range 1 to 4.

Draw the planet or background mask.

130 Call PROCCH to define characters to represent the shape we have just drawn.

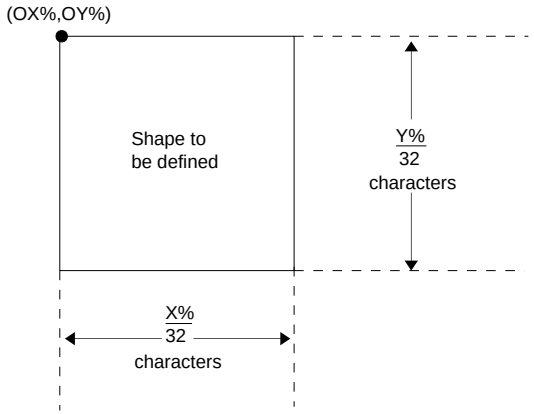
150 We now run all over the screen dumping characters.

170-180 First draw a back mask, and then the planet on top of this in a random colour.

190-210 Put the characters defined in PROCCH in their original positions.

PROCCH

This routine takes a region of the screen and defines characters to reproduce the pattern in that region. The routine will only work in modes in which the pixels are 4x4. The region to be turned into characters is specified as shown below:



The parameters X% and Y% must be multiples of 32 so that the region is exactly covered wi th whole characters. The parameter ST% is the code for the first character to be used. Memory is reserved for defining characters with codes 224 to 255, so when PROCCH is first called it would be sensible to supply the value 224 to ST%.

```
230 DEFPROCCH(OX%,OY%,X%,Y%,ST%)
240 DIM B%8
250 CN%=0
260 FOR J%=0TO 32-Y% STEP -32
270 FOR I%=0TOX%-32 STEP 32
280 VDU29,OX%+I%;OY%+J%;:C%=-1:CN%=CN%+1
290 FOR IY%=1TO-32STEP-4
300 V%=0:C%=C%+1
310 FOR IX%=1TO32STEP4
```

```
  320 IF POINT(IX%,IY%)=3 THEN PIX%=1 ELSE PIX%=0
  330 V%=V%*2+PIX%:NEXT
  340 B%?C%=V%:NEXT
  350 IF ST%+CN%=256:PRINT"TOO BIG":ENDPROC
  360 VDU23,ST%+CN%,B%?1,B%?2,B%?3,B%?4,B%?5,B%?6,
B%?7,B%?8
  370 MOVE0,-4:GCOL0,1:VDU(ST%+CN%)
  380 NEXT,
  390 ENDPROC
```

Description of PROCCH

240 The byte array B% will hold the eight values that specify the bit pattern of each row of the character being defined.

260-270	These loops move the point (I%,J%) to the top left-hand corner of each character that is defined.
280	Put the origin at the top left-hand corner of the current character. C% is used to index the array B%; it counts the rows of pixels within the character. Since we are about to define a new character we add one to CN%.
290	This loop moves the scan down one row.
300	V% will hold a number that describes the bit pattern on a row.
310	This loop scans across the 8 pixels which will form a row within the character.
320	If the pixel at the point (IX%,IY%) is white then make PIX%=l.
330	Alter V% so that it also describes;the last pixel.
340	Save the value of V% describing that row of pixels in the array B%.
350	Make sure that we are not going to try to define an illegal character.
360	Define the character number ST%+CN% using the bit patterns we have saved in B%.
370	Put the character just defined on top of the pattern it is derived from to make sure it is correct.

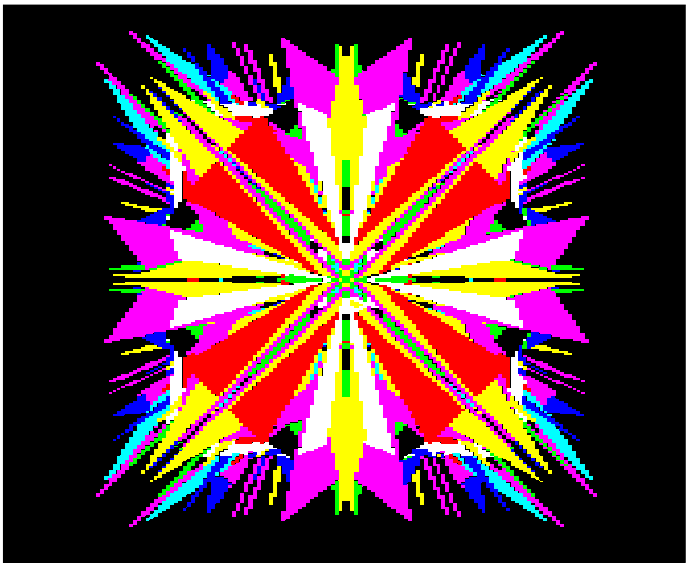
4 Animation

While it is relatively easy to draw complex static objects, animation is considerably harder. The processing power and memory size of the BBC machine limit the quality of animated images that can be produced. To achieve a fully--animated picture, a series of views of the scene must be rapidly displayed in sequence. This requires a large amount of processing to be done. Despite this, it is possible to derive good effects by exploiting the ability to change the palette of logical-physical colour relationships.

Redrawing

One way of producing movement is to continuously alter the picture that is being displayed. The following program uses this technique to simulate a kaleidoscope. This does not produce a smoothly-animated image, but the pattern produced is animated in the sense that it is always in a state of f1ux. The program also demonstrates that simply redrawing a pattern is not fast enough to produce true animation.

Kaleidoscope



In a mechanical kaleidoscope coloured flakes are randomly shaken up, and a symmetrical pattern is produced by reflecting these in two mirrors. This program generates three random points that form the edges of a triangle, and then rotates and reflects the triangle to produce a similar effect to that produced by the mirrors.

KALEIDO

```
    0 REM Kaleidoscope
   20 MODE2
   30 VDU5
   40 VDU29,640;520;
   50 REPEAT
   60 FORL%=12TO500STEP20
   70 GCOL0,RND(8)-1
   80 X%=RND(L%):Y%=RND(X%)
   90 X1%=RND(L%):Y1%=RND(X1%)
  100 X2%=RND(L%):Y2%=RND(X2%)
  110 MOVEX%,Y%:MOVEX1%,Y1%:PLOT85,X2%,Y2%
  120 MOVE-X%,Y%:MOVE-X1%,Y1%:PLOT85,-X2%,Y2%
  130 MOVEX%,-Y%:MOVEX1%,-Y1%:PLOT85,X2%,-Y2%
  140 MOVE-X%,-Y%:MOVE-X1%,-Y1%:PLOT85,-X2%,-Y2%
  150 MOVEY%,X%:MOVEY1%,X1%:PLOT85,Y2%,X2%
  160 MOVE-Y%,X%:MOVE-Y1%,X1%:PLOT85,-Y2%,X2%
  170 MOVEY%,-X%:MOVEY1%,-X1%:PLOT85,Y2%,-X2%
  180 MOVE-Y%,-X%:MOVE-Y1%,-X1%:PLOT85,-Y2%,-X2%
  190 NEXT L%
  200 UNTIL FALSE
```

Description of program

20	16-colour mode (actually, only 8 non-flashing colours).
30	Remove text cursor.
40	Define the centre of the kaleidoscope to be at the origin - here this is just off-centre of the screen.
60	L% controls the size of the kaleidoscope. This loop makes the pattern grow outwards. Select a colour at random, including black.
80-100	Choose the corners of the initial triangle.
110-180	Fill in the initial triangle with 5 other rotations and reflections of it.
190	Start again.

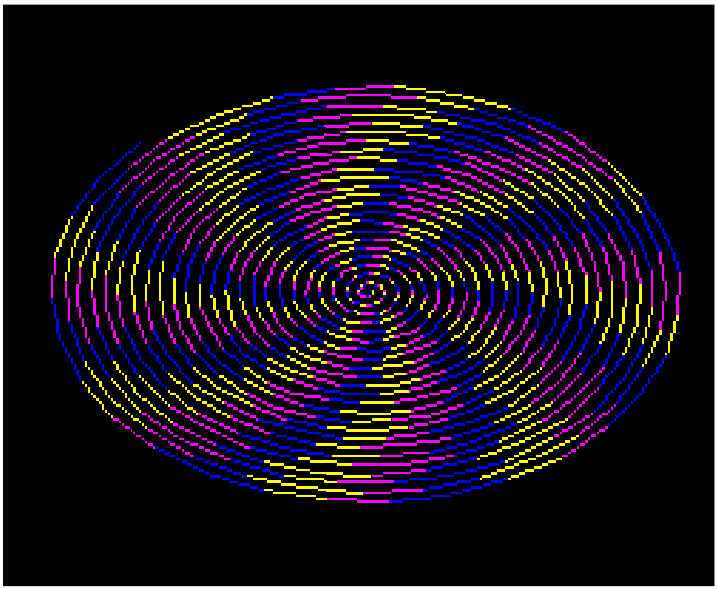
Palette changes

Drawing or redrawing an object is a relatively slow process, but palette changes can alter the appearance of an object very rapidly. If a single object is drawn with a number of different colours you can give the illusion of motion simply by changing the palette, and this avoids the need to redraw each view of the object. A similar technique is commonly used in neon signs that appear to move by changing the coloured lights that are turned on.

Animation by palette changes alone is most easily applied to a rotating object such as a beach ball, where the rotating object does not 'change shape' but stays within the confines of its original outline as it turns.

In programs that use rapid palette changes 'you may notice that the display is subject to dark bands and flicker. This is because the palette changes are not synchronised to the video scan time. To avoid this you could wait for vertical sync before changing the palette. This can be done using the operating system call *FXl9 which exists in release 1.0 of the MOS.

Flat Spiral



Here is a short program that uses palette changes to

rotate a flat spiral as it is being drawn. The spiral grows outwards from the centre of the screen, rotating as it expands. The pattern drawn is in fact an ellipse, and this gives the impression of a 3-dimensional spinning disc. SPIRAL1

SPIRAL1

```
    0 REM Flat Spiral
   20 MODE1
   30 VDU5
   40 C%=1:F%=RND(5)
   50 VDU29,640;512;
   60 MOVE0,0
   70 FORA=0TO300STEP0.2
   80 GCOL0,1+(3.8*A)MOD3
   90 DRAW3*A*SIN(A),2*A*COS(A)
  100 FORI%=1TO3
  110 VDU19,I%,(C%+I%)MOD3+F%;0;
  120 NEXT
  130 C%=(C%+1)MOD3
  140 NEXT
  150 GOTO 20
```

Description of program

40 C% keeps track of the current state of the palette. F% determines which physical colours will appear.

50 Put the origin in the centre of the screen.

70 The variable A determines the angle around the spiral and the size.

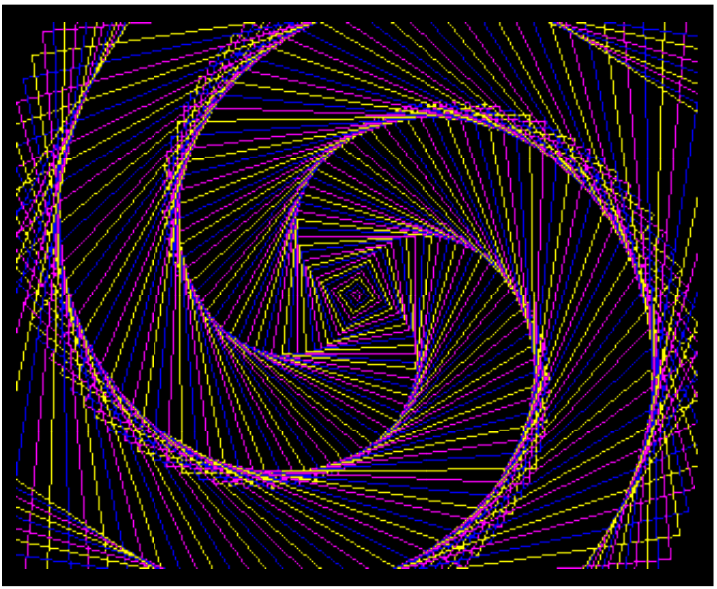
100-120 Change the colours of the arms as we draw around the spiral. The factor 3.8 makes the arms reasonably straight.

100-120 Change the palette to give the impression of movement by moving each of the physical-logical colour relationships on one step.

130 Alter C% so that next time the palette is changed the colours will move on one step.

150 Start again.

Rotating squares



This program produces a curious spiral pattern by rotating squares that gradually increase in size. When the pattern has been completed a moving effect is produced by changing the palette.

ROTSQ

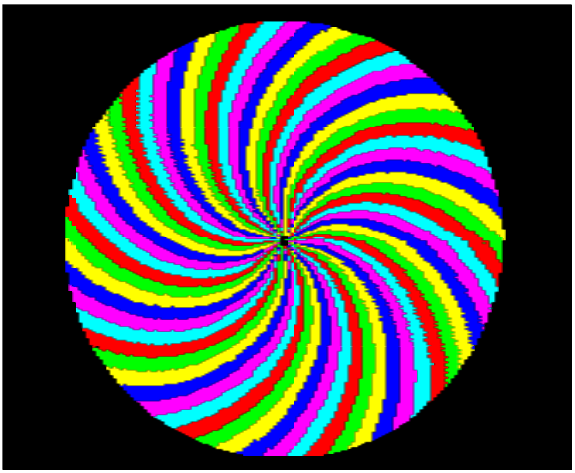
```
    0 REM Rotating Square
   20 MODE1
   30 VDU5
   40 VDU29,640;512;
   50 C%=1
   60 FORI%=0TO900 STEP 12
   70 A=I%/200
   80 X%=I%*SIN(A)
   90 Y%=I%*COS(A)
  100 MOVEX%,Y%
  110 DRAW-Y%,X%:DRAW-X%,-Y%:DRAWY%,-X%:
DRAWX%,Y%
  120 GCOL0,C%
  130 C%=(C%+1)MOD3+1
  140 NEXT
  150 F%=3
  160 REPEAT
  170 FORJ%=1TO3
  180 VDU19,J%,(J%+C%)MOD3+F%;0;
  190 NEXT
```

```
  200 C%=(C%+1)MOD3+1
  210 DELAY=INKEY(10)
  220 UNTIL FALSE
```

Description of program

20 4-colour mode.
30 Remove text cursor.
40 Put the origin in the centre of the screen.
50 Hold the the current logical colour.
60 The size of the squares is determined by I%.
70 A is the angle through which each square is rotated.
80-90 Calculate the coordinates of one corner.
100-110 Draw a rotated square.
120 Select colour.
130 Choose the colour of the next square.
140 On leaving this loop the pattern has been drawn. All that follows is the palette changes which give the impression of animation.
150 F% determines which three physical colours will be displayed.
170-190 Change each colour to the one next to it.
200 Select the next colour.
210 Cause a short delay. (Pressing any key will cancel the delay.)

Multi-coloured spiral



Animation using palette changes becomes smoother as more logical colours are used. Here all the non-flashing colours are used to fill a spiral, which is then rotated when any key is pressed. The spiral is filled using rings of triangles that alternately point inwards and outwards, so that adjacent rings mesh together.

SPIRAL2

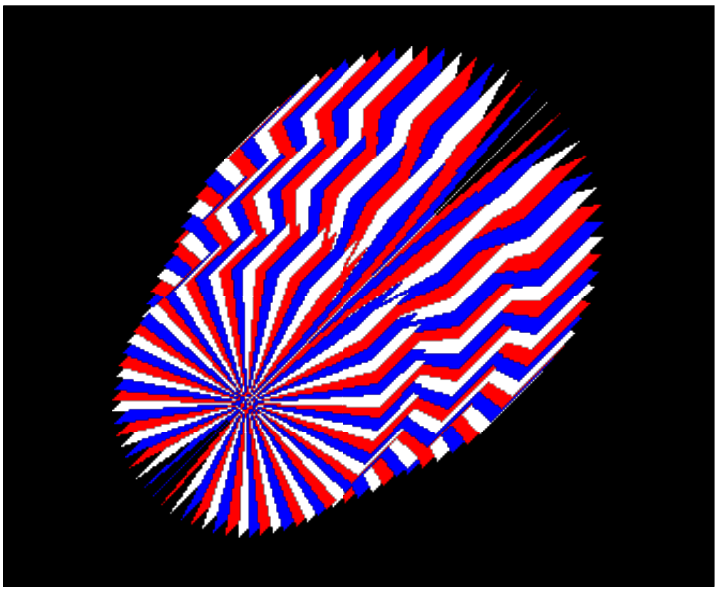
```
    0 REM Spiral
   20 MODE2
   30 VDU29,640;512;23;9;0;0;0;
   40 MOVE0,0
   50 MOVE0,0
   60 Z=512
   70 A%=180
   80 P=8
   90 DIMA(480):FORQ=0TO480:A(Q)=SINRADQ
:NEXT
  100 R%=1
  110 VDU5
  120 REPEAT
  130 R%=R%-1:IFR%=0 R%=6
  140 FORB%=0TO5
  150 FORA%=B%TOB%+359STEP6
  160 GCOL0,R%:R%=R%+1:IFR%>6 R%=1
  170 PLOT&55,A(A%+90)*Z+1,A(A%)*Z+1
  180 MOVEA(A%+93)*(Z-72),A(A%+3)*(Z-72)
  190 NEXT
  200 Z=Z-P
  210 NEXT
  220 UNTILZ<=120
  230 FORI%=0TO7:VDU19,I%,0;0;:NEXT
  240 A%=GET
  250 DIM A$(7)
  260 FORZ=1TO6:FORY=1TO6
  270 A$(Z)=CHR$19+CHR$Y+CHR$((Y+Z-1)MOD
6+1)+CHR$0+CHR$0+CHR$0+A$(Z):NEXT
  280 NEXT
  290 FORQ%=1TO6:N%=TIME+4:REPEATUNTILTI
ME>N%:PRINTA$(Q%);:NEXT
  300 GOTO290
```

Description of program

30	Set up the origin at the centre of the screen and turn interlace off.
90	Compile a table of sine values to save time later on.
110	Turn cursor off.

120	This outer loop, in conjunction with the loop starting at line 140, decrements the radius of the circles drawn by the inner loop (lines 150-190), thus making the spiral grow inwards.
150-190	This inner loop draws the circle of triangles.
200-220	Close outer loop
230	Make the screen go black.
240	Wait for a key to be pressed.
250-280	Set up string array A$ to change the palette.
290-300	Change the palette using predefined array.

Rotating Fan



Here is a program that simulates the rotating fans inside a jet engine. The fan blades are coloured red, white and blue, and are drawn to give a perspective view of the fan. Three fans are drawn, and then rotated. The central fan appears to rotate in the opposite direction to the other two; this is because when this fan is drawn it is coloured in the opposite direction to the outer fans.

ROTFAN

```
    0 REM Rotating fan
   20 MODE1
   30 VDU19,2,4;0;
   40 VDU5
   50 X%=740:Y%=612:S%=400:C%=1:N%=TRUE
   60 FOR FAN=1 TO 3
   70 N%=NOT N%
   80 X%=X%-100:Y%=Y%-100:S%=S%-50
   90 FORA=1.25*PI TO PI/4-PI/35 STEP -P
I/35
  100 C%=(C%+1)MOD3
  110 IF N% THEN CT%=2-C% ELSE CT%=C%
  120 PROCSIDE(X%,Y%,S%,A,CT%+1)
  130 PROCSIDE(X%,Y%,S%,2.5*PI-A,3-CT%)
  140 NEXT A,FAN
  150 C%=1
  160 REPEAT
  170 FORI%=1 TO 3
  180 J%=(C%+I%)MOD3 +1
  190 IFJ%=2 THEN J%=4
  200 IF J%=3 THEN J%=7
  210 VDU19,I%,J%;0;
  220 NEXT
  230 C%=C%+1
  240 DELAY=INKEY(12)
  250 UNTIL FALSE
  260 DEFPROCSIDE(X%,Y%,S%,A,C%)
  270 X1%=S%*COSA:Y1%=S%*SINA
  280 D%=S%/3
  290 VDU29,X%;Y%;
  300 GCOL0,C%
  310 MOVE0,0:PLOT0,X1%,Y1%:PLOT81,D%,D%
  320 PLOT0,-X1%,-Y1%:PLOT85,0,0
  330 ENDPROC
```

Description of program

30 Redefine the palette to make logical colour 2 appear as dark blue.

50 Position the centre of the current fan at the point (X%, Y%) . S% gives the radius, C% controls the colour of the blades, and N% affects the order in which the colours are altered and thus the directions in which the fans will rotate.

60 Draw a fan each time round this loop.

70 Toggle N% so that adjacent fan will rotate in opposite directions.

80 Alter X%,Y% and S% so that the fans appear to approach the observer and get smaller towards the front.

90 Draw a fan with two vanes for each value of A.

100 Select the colour of the next vane.

110 Alter the order in which colours are selected depending on the value of N%,

120-130 Draw two vanes. This ensures that the closer vanes obscure the more distant ones.

On leaving these loops the entire picture has been drawn, and all that follows is concerned with palette changes.

150 C% keeps track of the state of the palette.

160 Move all the logical-physical colour relationships on one step each time around this loop.

170 I% runs through all the logical colours for this mode.

180 J% is the next physical colour to select,

190-200 Alter J% so that the colours dark blue and white appear.

210 Change the palette for each value of I%.

230 Update C% so thegt the colours will move on one step.

240 Wait 12 centi--seconds.

250 Carry on forever.

260 PROCSIDE draws a single vane of the fan, which resembles a page of an open book, and takes the following parameters:

X%,Y% gives position of the centre of the fan.

S% is the radius of the fan.

A is the angle of the vane.

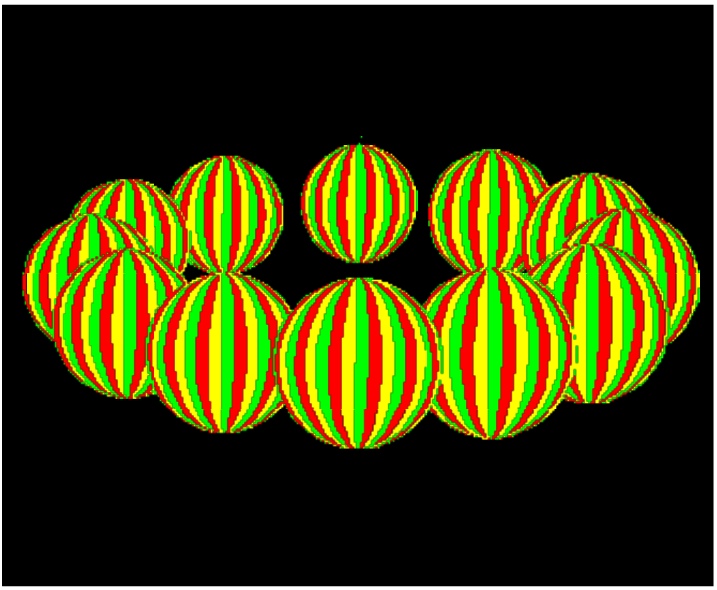
C% is the logical colour of the vane.

280 (Xl%,Yl%) is the point on the front edge of the vane.

290 D% is used to draw to the back the vane.

310-320 Put the origin at (X%,Y%).

Beach balls



This program draws a ring of multicoloured beach balls. These are then rotated by palette changes. The procedure PROCEALL draws a single ball. Since it takes quite a long time to draw the entire ring it might be best to alter the program to draw and rotate a single ball first. This will avoid delays if the palette-changing code does not work the first time you run it.

BEACHBA

```
  0 REM Beach Balls
 20 MODE 1
 30 VDU 5
 40 S%=100
 50 FOR T=0 TO PI-PI/6 STEP PI/6
 60 S%=S%+10
 70 PROCBALL(S%,640+500*SIN(T),512+150*COS(T),1)
 80 NEXT
 90 S%=100
100 FOR T=-PI/6 TO -PI STEP -PI/6
110 S%=S%+10
120 PROCBALL(S%,640+500*SIN(T),512+150*COS(T),1)
130 NEXT
140 REPEAT
150 FOR F=1 TO 6 STEP 0.02
160 FOR I%=1 TO 3
170 VDU 19,I%,(C%+I%)MOD3+F;0;
```

```
180 NEXT
190 C%=(C%+1)MOD3
200 A=INKEY(10)
210 NEXT
220 UNTIL FALSE
230 END
```

Description of program

40	S% gives the radius of the ball.
50-80	Draw the balls on the right-hand side of the ring, increasing the size of the balls as they come towards the viewer.
90-130	Draw the balls on the left-hand side of the ring, starting with the most distant ball as before.
140	Now rotate the balls forever.
150	Determine which physical colours will appear in the ball. This loop makes the balls change colour after rotating for a while.
160-210	Rotate the balls using palette changes.

PROCEALL

The ball is drawn by running round and round an ellipse that gradually gets thinner and thinner. The impression of depth results from the way in which the differently-coloured ellipses are overlayed to resemble the segments of a beach ball. The procedure uses the following parameters:

SIZE% gives the radius of the ball.

(X%,Y%) is the position of the centre of the ball.

C% is the first colour to be used. This affects the direction in which the ball will rotate when the palette changes begin.

```
240 DEFPROCBALL(SIZE%,X%,Y%,C%)
250 VDU 29,X%;Y%;
260 MOVE 0,0
270 MOVE0,SIZE%
280 FOR A=0 TO 60.4 STEP 0.2
290 SA=SIN(A)
300 Q%=1+(1+A/(PI*2))MOD3
310 GCOL 0,Q%
320 IF SA<0 THEN GCOL 0,4-Q%
330 X%=SIZE%*SA*COS(A/40)
340 PLOT 85,X%,SIZE%*COS(A)
```

```
350 PLOT 85,X%,0
360 NEXT
370 ENDPROC
```

Description of PROCBALL

250	Put the origin at the centre of the ball.
260	Move to the first point on the shrinking ellipse path.
280	Sweep round and round the ellipse incrementing the angle A.
290	Save SIN(A) as SA so that this is only calculated once.

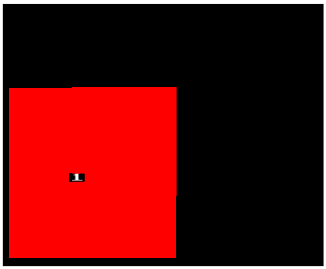
Instant plotting

The idea behind 'instant plotting' is to use' palette changes to obscure the next view while it is being drawn. Then by altering the palette it is possible to swap instantly from one frame to the next.

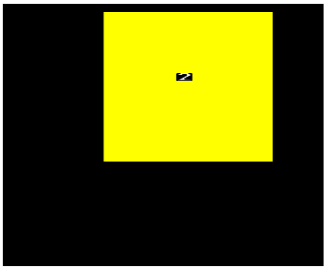
The simple approach of using a different logical colour for the next view is complicated by the fact that in each frame the object being drawn will probably overlap with the previous view. To overcome this the next view can be drawn using the graphics action OR, while the palette changes take care of the added complication of overlapping parts that this introduces.

The various stages in the method are illustrated in the diagrams that follow.

1 Start with the initial view drawn in logical colour l.

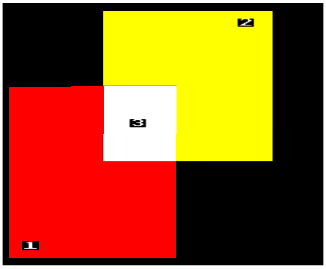


2 Draw the next view in logical colour 2 using graphics action OR.



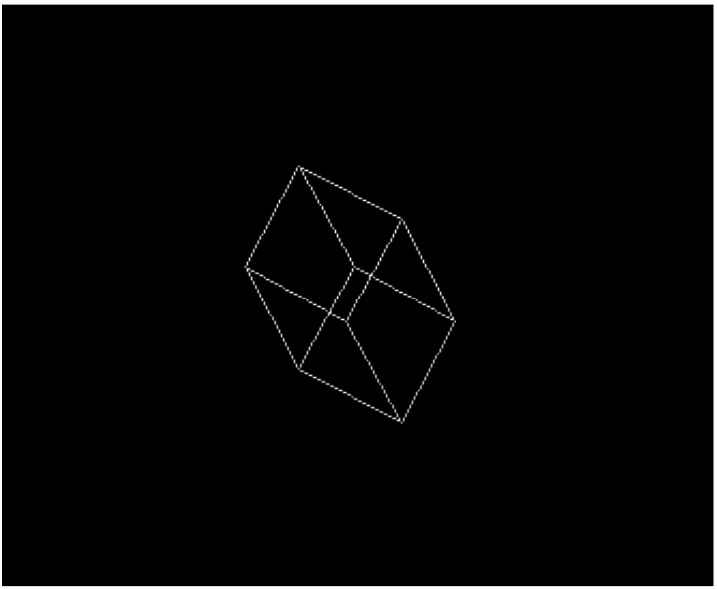
3 Change the palette to show logical colour 2 and obscure logical colour l.

4 Remove the old view by redrawing it in logical colour 2 using graphics action AND.



5 Repeat from step I but now the initial view is in colour 2.

Tumbling box



This program demonstrates how the 'instant plotting' technique can be used to produce a remarkably effective animated view. The program shows a box tumbling towards you.

BOX

```
  0 REM Tumbling box
 20 MODE1
 30 VDU29,640;512;
 40 VDU5
 50 COL%=RND(7)
 60 VDU19,3,COL%;0;19,1,COL%;0;
 70 C%=1:A=4
 80 GCOL0,C%:PROCBOX(A,A*10)
 90 PROCBOX(A,A*10)
100 VDU19,C%,COL%;0;19,3-C%,0;0;
110 REPEATA=A+0.1
120 C%=C%EOR3
130 GCOL1,C%:PROCBOX(A,A*10)
140 VDU19,C%,COL%;0;19,3-C%,0;0;
150 GCOL2,C%:PROCBOX(A-0.1,10*(A-0.1))
160 UNTIL FALSE
```

Description of program

50	Determine the physical colour of the moving box.
60	Make logical colours 3 and I appear as physical colour COL%.
70	C% determines the physical colour to use. A gives the size and orientation of the box.
80-90	Draw the first view of the box.
100	Ensure the next view will not appear while it is being drawn.
110	Each time around this loop a new box is drawn and the previous view is undrawn.
120	Toggle the value of C% between I and 2. OR the new box.
140	Show the new view and obscure the old one.
150	Remove the old box by ANDing with logical colour C%.
160	Carry on forever.

PROCBOX

This draws a wire-frame view of a box. The position and orientation of the box are determined by the value of AN. S% gives the size of the box. The box is constructed by first drawing the 'front' face, and then drawing an identically-shaped 'back' face running up and down the connecting edges as this is done.

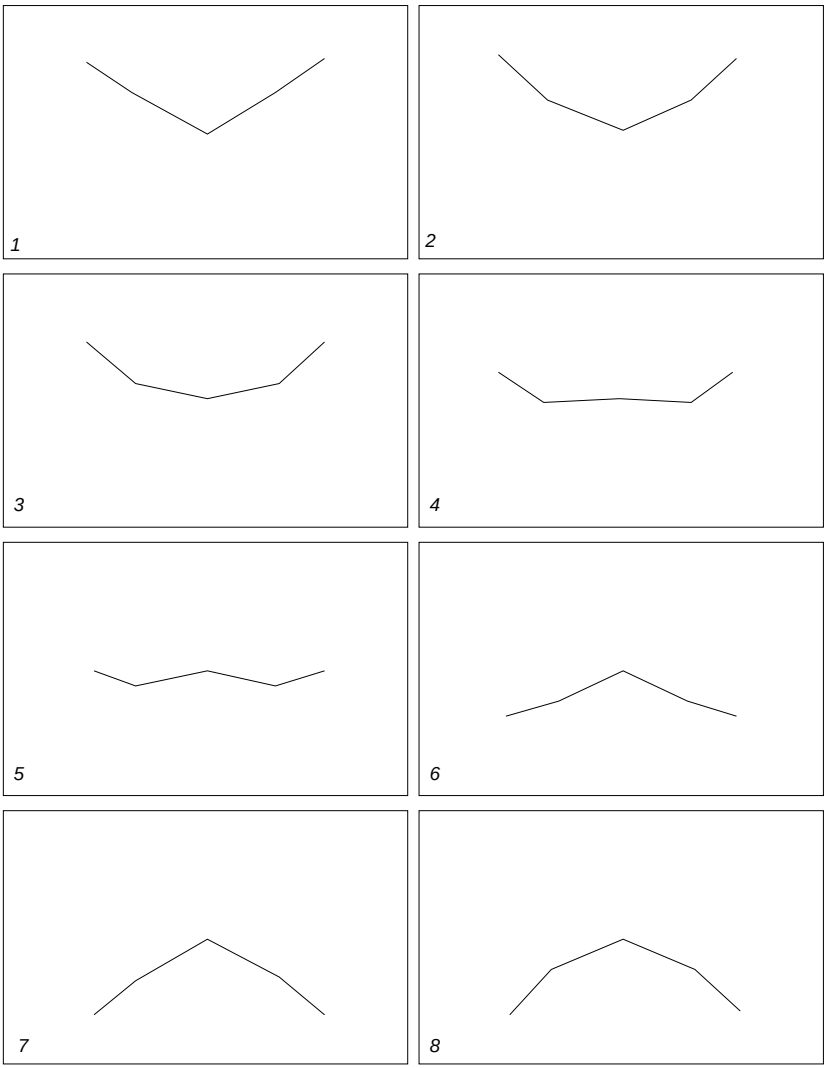
Description of PROCBOX

280	Draw one face.
210	Draw an edge joining the 'front' and 'back' faces.
220-240	Draw three lines of the 'back' face with back-front connecting edges.
250	Draw the final line of the back face.

Condor

Here, four lines are used to represent a majestic condor soaring above the Andes. The image is convincing because of the way in which these simple lines are smoothly-animated to capture the motion of

the bird's wings. The various wing positions used in this animation are shown below:



The background landscape is built out of random lines. These give the impression of dark peaks rising from a seething mass of vegetation.

CONDOR

```
0 REM Condor
```

```
   20 MODE1
   30 VDU5
   40 PROCANDES
   50 C%=1:L%=FALSE
   60 ANGLE=0
   70 REPEAT
   80 ANGLE=ANGLE+0.2
   90 C%=C%EOR3:GCOL1,C%
  100 OX%=640+500*COS(ANGLE/12)
  110 OY%=712+200*SIN(ANGLE/12)
  120 X%=(OY%-1030)/4:Y%=X%*COS(ANGLE)
  130 A%=X%*1.5:B%=A%*COS(ANGLE-0.4)
  140 PROCBIRD(X%,Y%,A%,B%,OX%,OY%)
  150 VDU19,C%,2;0;19,3-C%,0;0;
  160 IF L% THEN GCOL2,C%:PROCBIRD(X1%,Y
1%,A1%,B1%,OX1%,OY1%) ELSE L%=TRUE
  170 X1%=X%:Y1%=Y%
  180 A1%=A%:B1%=B%
  190 OX1%=OX%:OY1%=OY%
  200 UNTIL FALSE
  340 ENDPROC
```

Description of program

50	L% is used to decide whether there is an old view of the bird that has to be undrawn. ANGLE determines both the wing position and the bird's location.
100-110	Calculate the new position of the bird.
120-130	Calculate the new wing positions.
140	Draw the new bird.
150	Flip to view the new bird.
160	Undraw the old bird, unless this is the first time around the loop and there is no old bird to undraw.
170-190	Make the new bird the old bird.

PROCANDES

```
  210 DEFPROCANDES
  220 GCOL0,3
  230 VDU19,3,2;0;
  240 MOVE0,0:DRAW1279,0:DRAW1276,1020:D
RAW0,1020:DRAW0,0
  250 FORJ%=-50 TO 100 STEP 4
  260 MOVE0,J%
  270 FORI%=0TO130
  280 PLOT1,10,RND(25)-12
  290 NEXT I%,J%
  300 ENDPROC
```

Description of PROCANDES

220 Logical colour 3 is not altered by the animation palette changes.

240 Draw a frame around the screen.

250 For each value of J% we draw a random jagged line across the screen.

270-290 Draw a wiggly line from left to right.

PROCBIRD

```
  310 DEFPROCBIRD(X%,Y%,A%,B%,OX%,OY%)
  320 VDU29,OX%;OY%;
  330 MOVE-A%,B%:DRAW-X%,Y%:DRAW0,0:DRAWX%,Y%:DRAW
A%,B%
```

Description of PROCBIRD

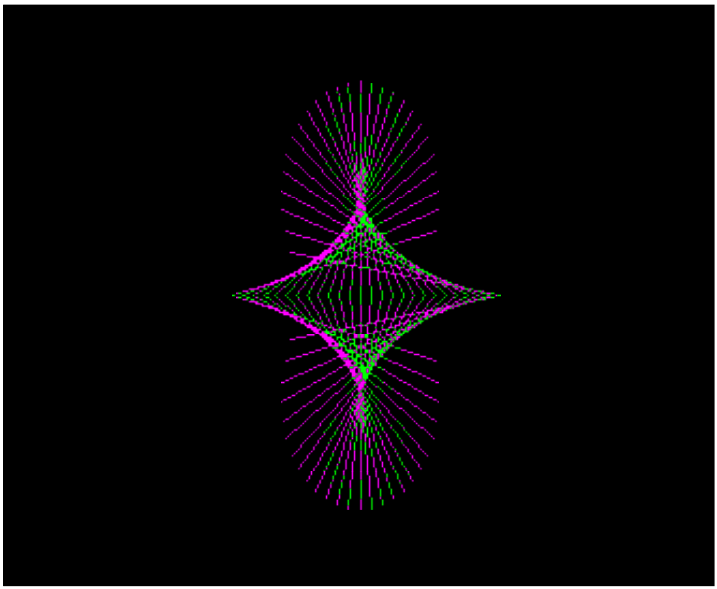
310 PROCEIRD takes the following parameters:

OX% and OY% define the origin for the bird

A% and B% are the ends of the wings

X% and Y% are the elbows, or 'bends' in the wings

Plankton



This program draws a weird pattern that looks a bit like a microscopic bug that pollutes the water supply. The program picks a random physical colour for the parts of successive views of the bug that overlap.

PLANK

```
    0 REM Plankton
   20 MODE1
   30 VDU5
   40 DIM S(121)
   50 I%=-2
   60 FORA=0TO 2*PI STEP PI/30
   70 I%=I%+2
   80 S(I%)=400*COS(A)
   90 S(I%+1)=400*SIN(A)
  100 NEXT
  110 C%=1:L%=FALSE:COL%=RND(7)
  120 VDU19,2,COL%;0;19,3,RND(7);0;
  130 REPEAT
  140 FOR J%=0 TO 118 STEP 2
  150 C%=C%EOR3:GCOL1,C%
  160 PROCPAT(J%)
  170 VDU19,C%,COL%;0;19,3-C%,0;0;
  180 IF L% THEN GCOL2,C%:PROCPAT(J%-2)
ELSE L%=TRUE
  190 NEXT
```

```
  200 UNTIL FALSE
```

Description of program

40	The array S will hold a table of coordinates.
50-100	Save the coordinates that are calculated from the slow trig functions SIN and COS in the array S.
140	J% has the same effect as an angle that rotates once as J% goes from 0 to 118.
160	Draw the new pattern.
180	Remove the old pattern.

PROCPAT

This draws the micro-organism in an orientation determined by J%.

PROCPAT

```
  210 DEFPROCPAT(J%)
  220 FORI%=0TO500 STEP 20
  230 J%=(J%+2)MOD120
  240 X%=S(J%):Y%=S(J%+1)
  250 VDU29,400+I%;512;
  260 MOVE X%,-Y%:DRAW0,0:DRAW X%,Y%
  270 NEXT
  280 ENDPROC
```

Description of PROCPAT

220 I% runs along the backbone.

230	This puts the legs at different angles.
240	Look up the (X%,Y%) position from the value of J%.
250	Move the origin.
260	Draw a pair of 'legs'.

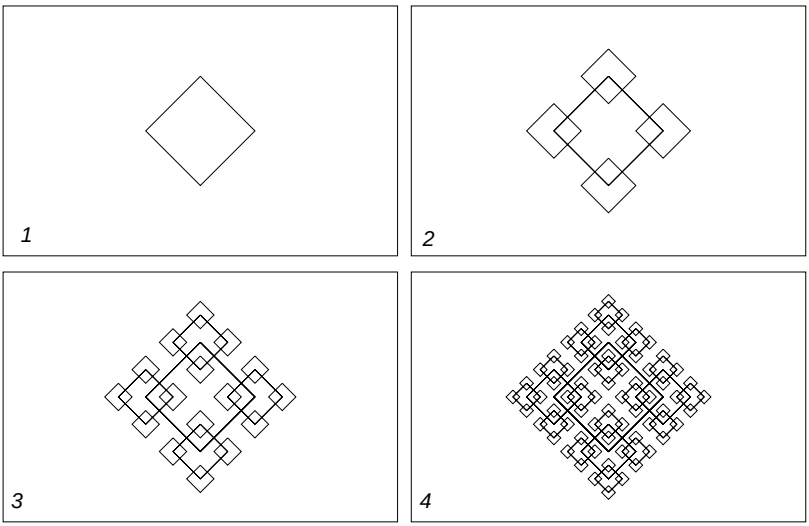
5 Recursion

Recursive patterns

A large number of complicated patterns can be defined very simply using recursion, whereby a simple shape is drawn over and over again until a complete pattern is built up. Recursion means, simply, repetition and the term can be used to describe any repetitive process.

A 'replacement rule' is repeatedly applied to the initial pattern, and this principle may also determine the size and position of successive versions of the shape relative to the original.

Here is a simple example:



In this case the initial pattern is a diamond. The rule applied here is to draw a smaller diamond centred about each of the corners of the original. As you can see, this rule does not have to be applied many times before a complicated pattern emerges.

Recursive procedures

The BASIC interpreter on the BBC machine allows you to define recursive procedures. A recursive procedure has a call to itself within its definition. Here is an example showing how a recursive procedure can be used to mimic the action of a FOR... NEXT loop. (Obviously

in this case the FOR... NEXT loop would be much neater and more efficient.)

Example

```
10 PROCLOOP(0)
20 END
30 DEF PROCLOOP(N%)
40 IF N%=11 THEN ENDROC
50 PRINT N%
60 PROCLOOP(N%+1)
70 ENDPROC
```

Description of program

10	Call PROCLOOP with 0.
20	Stop.
30	PROCLOOP takes one parameter N%. The value of N% determines the number of times we recurse.
40-60	If N% is equal to 11 then return, otherwise print out N% and call PROCLOOP with the value N%+1.
70	This final ENDPROC is vital.

This program will print the numbers from 0 to l0. The recursive procedure PROCLOOP consists of three parts which are:

1 A test for an end condition that will terminate the recursion

2 An action that the procedure printing out N%

3 A recursive call that alters the parameters in some way

In general these parts can be found in some form in every recursive procedure. The order in which these sections appear is important; for example, if the procedure calls itself before it has tested for the end condition it will never be able to stop recursing and will run out of room.

Each time a new call of the procedure is made the 'depth' of recursion increases. The depth of recursion is in some cases similar to the number of times a loop is executed. When a program leaves a procedure it returns to the level it was at just before entering it. The deeper a program recurses the more information it has to store about how to get back to the top

level.

Local variables

Local variables can only be altered inside the procedure in which they appear, and have no effect on any variable outside their own procedure. This is generally useful since it is possible to define a procedure without worrying about name clashes.

In recursive procedures it is often vital to declare variables to be local. If this is not done recursive calls of the procedure will alter the variables in calling the procedure, and the effects of this can be obscure.

In BBC BASIC the parameters of a procedure are automatically treated as local variables; for any other variable to be local it must be explicitly declared to be so. This program, which prints out coloured blobs, demonstrates a typical case in which a local variable is required:

Example

```
  l0 MODE 7
  20 PROCDO(I)
  30 END
  40 DEF PROCDO(L%)
  50 LOCAL I%
  60 IF L%=5 THEN ENDPROC
  70 FOR I%=1 TO 7
  80 PROCDO(L%+1)
  90 VDU (l28+I%),255
 100 NEXT
 110 ENDPROC
```

Each time PROCDO is called it should call itself seven times in the loop starting at line 70. If I% is not declared to be local the recursive calls of PROCDO would affect the value of I% in the first procedure. Try removing line 50 to see how this affects the program.

Advantages and limitations of recursion

The great power of recursion is that it can provide a very neat solution to certain kinds of problem. This power must be judged against some of its limitations.

The main problems are:

1 Recursion can be slow

2 Recursion requires a lot of memory

The first problem can arise because each time a procedure is called there is a delay while the parameters are passed to the procedure and local variables are set up. The speed at which this is done depends on how efficiently this mechanism has been implemented.

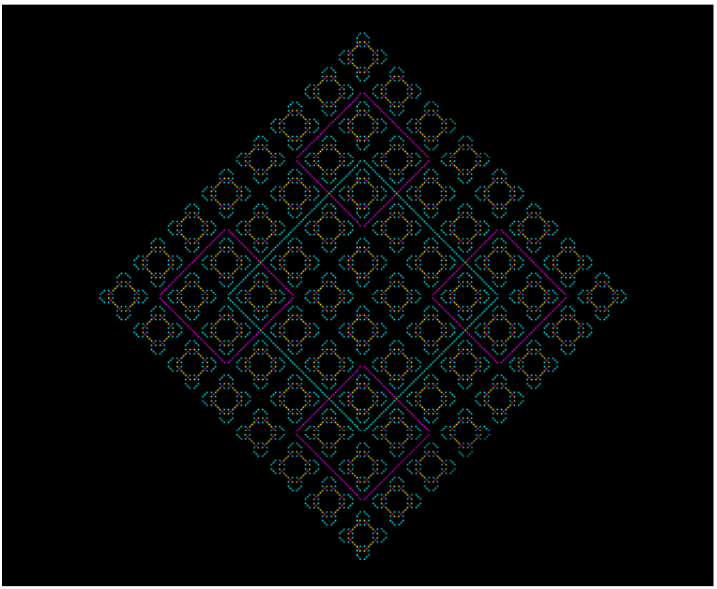
Luckily the interpreter on the BBC machine is quite fast and the overheads incurred each time a procedure is called are not very large. Since recursive procedures require many procedure calls they will run faster if the cost of each call is kept to a minimum. This can be achieved by not passing unnecessary parameters that do not change, and keeping the number of local variables to a minimum.

Each time a procedure is called some more memory is required to hold the local variables and the return address from which the procedure was called. Recursion allows procedures to call themselves, and at each new level some more memory is needed. It is very easy to define a procedure which can gobble up memory at an alarming rate, for example if a large number of local variables are used and if the procedure recurses many times. To avoid this care must be taken to ensure that the end condition stops the recursion before it runs out of room. Also, because all the parameters are saved every time the procedure is called, the number of parameters used should be limited to avoid passing parameters that do not alter, thus avoiding possible wastage of valuable memory.

Replacement patterns

The next two programs are examples of patterns that are derived from repeatedly applying a replacement , rule to an initial pattern. Both start from a very simple pattern. Surprisingly, the outline of the final pattern depends more on where the replacements are made than on the original shape.

Recursive diamonds



This program illustrates how a complicated pattern crew easily be defined and rapidly drawn using a recursive procedure. The pattern is formed by drawing a diamond followed by four diamonds, each half the size of the original, centred about its corners, as shown on page 63. On each of these diamonds a further four smaller diamonds are drawn. This continues until the size of the diamonds has become less than the minimum size spec if ied.

RECDIA

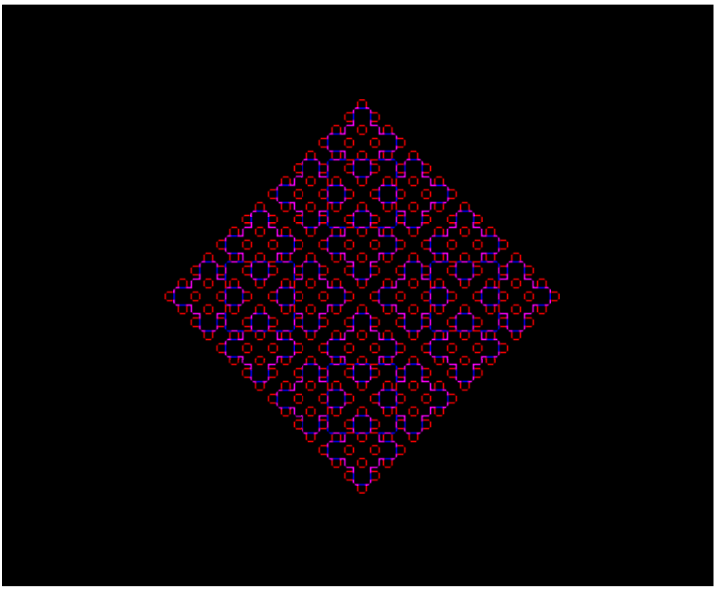
```
  10 MODE 1
  20 VDU 5
  30 M%=400
  40 REPEAT
  50 VDU 19,RND(3),RND(7);0;
  60 GCOL 3,RND(3)
  70 PROCDIA(640,512,512)
  80 M%=M%/2
  90 UNTIL M%<16
 100 GOTO 30
 110 DEFPROCDIA(X%,Y%,S%)
 120 IF S%<M% THEN ENDPROC
 130 S%=S%/2
 140 PROCDIA(X%+S%,Y%,S%)
 150 PROCDIA(X%-S%,Y%,S%)
```

```
  160 PROCDIA(X%,Y%-S%,S%)
  170 PROCDIA(X%,Y%+S%,S%)
  180 VDU 29,X%;Y%;
  190 MOVE 0,S%
  200 DRAW S%,0:DRAW 0,-S%:DRAW -S%,0:DR
AW 0,S%
  210 ENDPROC
```

Description of program

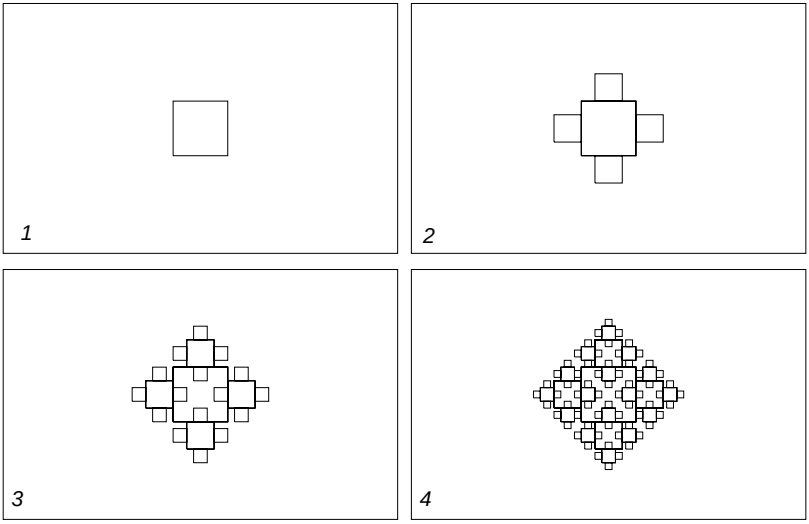
10	4-colour mode
20	Remove text cursor
30	M% gives the minimum size a diamond can be.
40	Call the recursive procedure PROCDIA (X%,Y%,S%) each time round this loop.
50	Redefine one of the three logical colours to appear as any of the possible non-flashing colours.
60	Select a random colour and graphics action EOR.
70	Call PROCDIA in the centre of the screen with an initial size of 512.
80	Halve the maximum line length; this will cause PROCDIA to recurse one level deeper next time it is called.
90	If M% was less than this the recursion would take too long and the lines very small.
100	Start all over again.
110	Each time PROCDIA(X%,Y%,S%) diamond of size S% is drawn the point (X%,Y%), provided drawn would be is called a centred about that S% is bigger than M%; otherwise, the procedure does nothing,
120	Do nothing if the size is less than M%.
130	Halve the size.
140-170	Call PROCDIA to draw half-sized diamonds at each corner.
180	Put the origin at the point (X%,Y%).
190-200	Draw a diamond of size S%.

Recursive squares



This pattern is similar to the recursive diamonds pattern. Here the recursive step is to replace one square by the original square plus four smaller squares lying on its sides. Because the squares are drawn with the graphics action Exclusive-OR, in a random colour, this step can cause parts of the original square to be erased. The procedure is called repeatedly recursing to a deeper level on each call. This method produces a wide range of dynamic patterns.

Pattern construction



RECSQ

```
    0 REM Recursive Squares
   20 MODE1
   30 VDU5
   40 V%=300
   50 REPEAT
   60 GCOL3,RND(3)
   70 VDU19,RND(3),RND(7);0;
   80 PROCC(500,400,256)
   90 V%=V%/2
  100 UNTILV%<8
  110 GOTO40
  120 END
  130 DEFPROCC(X%,Y%,S%)
  140 IF S%<V% THEN ENDPROC
  150 PROCC(X%+S%/4,Y%+S%,S%/2)
  160 PROCC(X%+S%,Y%+S%/4,S%/2)
  170 PROCC(X%-S%/2,Y%+S%/4,S%/2)
  180 PROCC(X%+S%/4,Y%-S%/2,S%/2)
  190 VDU29,X%;Y%;
  200 MOVE0,0
  210 DRAW0,S%:DRAWS%,S%:DRAWS%,0:DRAW0,
0
  220 ENDPROC
```

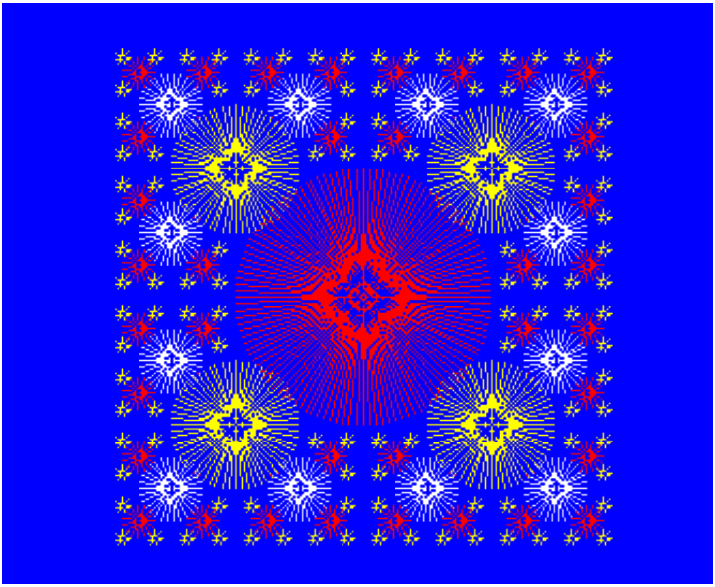
Description of program

40	V% controls how many times PROCC will recurse.
50	Each time around this loop we call PROCC.
60	Select graphics action EOR with a random logical colour.
70	Display any of the 6 colours.
80	Call PROCC in the centre of the screen with initial size 256.
90	Reduce V% causing PROCC to recurse one level deeper next time it is called.
100	Limit V%. Since each pixel is 4x4 this seems a reasonable limit. If V% were much less than this then the time required for PROC would be excessive.
110	Start again.
120	(We never get here.)
130	Draw a square at (X%,Y%) of size S%.
140	Do nothing if S% is less than V%.
150-180	Call PROCC on each side of the square.
190	Move the origin to (X%,Y%).
200	Draw the square.

Intelligent parameters

In the previous examples the number of recursive calls and the way in which the parameters are altered at each call remain the same at every level. This introduces a very strict regularity to the pattern produced. It is possible to have a parameter that controls the nature of the recursive calls. This leads to exciting possibilities in which different actions can be taken at different levels of recursion,

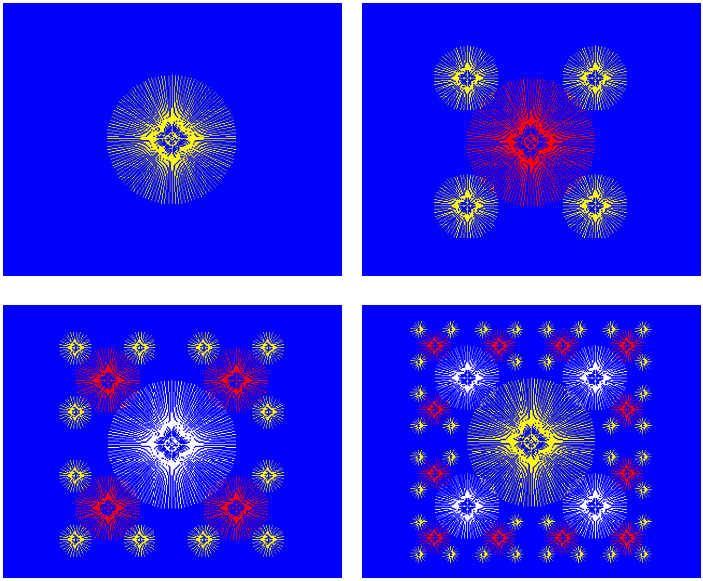
Circles



This program draws a pattern built up from circles of decreasing size growing outwards from the central circle. The program uses recursion and has a very simple way of ensuring that circles are not drawn on top of each other.

This is done by having a parameter in the circle-drawing procedure that indicates where the current circle is joined to its parent. By examining this parameter it is possible to avoid any recursive calls that would cause circles to be overwritten. A result of this is that the final pattern is square and fills the screen in a surprisingly intelligent way.

Effect of the parameter P%



CIRCLES

```
    0 REM Circles
   20 MODE1
   30 VDU5
   40 VDU19,0,4;0;
   50 PROCCIRCLES(640,512,240,0,0)
   60 REPEATUNTIL0
   70 DEF PROCCIRCLES(OX%,OY%,R%,L%,P%)
   80 IFL%=5 THEN ENDPROC
   90 VDU29,OX%;OY%;
  100 GCOL3,L%MOD3+1
  110 FORA=0TO2*PI-PI/R% STEPPI*4/R%
  120 MOVE0,0:DRAWR%*COSA,R%*SINA
  130 NEXT
  140 IFP%<>3 PROCCIRCLES(OX%-R%,OY%-R%,
R%/2,L%+1,1)
  150 IFP%<>4 PROCCIRCLES(OX%+R%,OY%-R%,
R%/2,L%+1,2)
  160 IFP%<>1 PROCCIRCLES(OX%+R%,OY%+R%,
R%/2,L%+1,3)
  170 IFP%<>2 PROCCIRCLES(OX%-R%,OY%+R%,
R%/2,L%+1,4)
  180 ENDPROC
```

Description of program

```
40        Make the background dark blue.
```

50 Call PROCCIRCLES. The centre of the first circle will be at (640,512) and it will have a radius of 240. The next parameter determines the level of recursion and the colour. The last value determines where circles are drawn around the current circle.

60 This stops the prompt from BASIC appearing once the pattern has been drawn.

70 PROCCIRCLES takes the following parameters:

OX%,OY% are the coordinates of the centre of the circle.

R% The radius of the circle.

L% The depth of recursion, also used select a new colour at each level.

P% The position at which the current is joined to its parent circle.

80 Stop after 5 recursions.

90 Move the origin.

100 Select a colour specified by L% with action EOR.

110-130 Draw the radial circle lines.

140-170 Call PROCCIRCLES at a maximum of four positions around the edge of the original. Omit the position that connects the current circle to its parent.

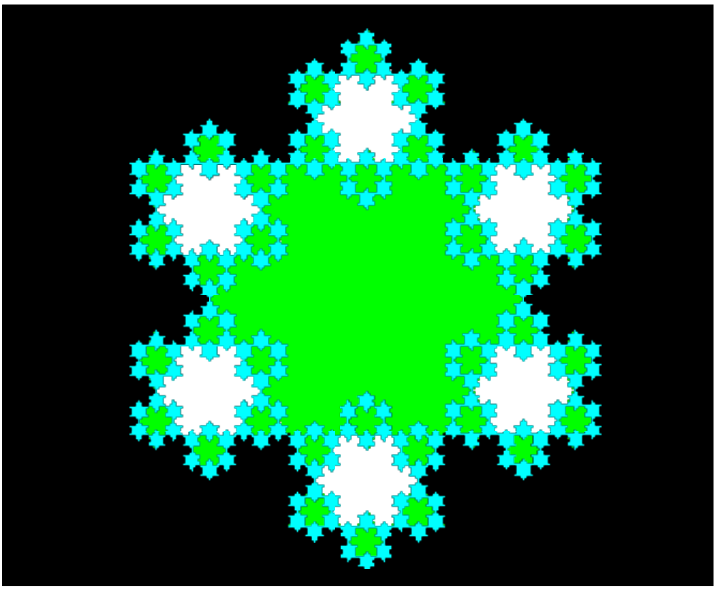
170 Leave the procedure.

Recursive calls within a loop

If a procedure is to call itself a large number of times, it would be very inconvenient to have to write out each call explicitly. By placing the call in a loop the desired effect is produced. However, there are some pitfalls to avoid if this is done.

The value of the control variable of the loop must not after while the procedure in the loop is being executed. If the procedure being called is the same as the calling procedure it is bound to have a loop contr'ol Variable with the same name. To ensure the variable retains its value across the recursive call the loop variable must be declared to be a local variable. The following program uses this method.

Koch flake



The Koch flake is sometimes called the snowflake curve, because its curious shape resembles a snowflake. Here it is constructed by repeatedly drawing smaller stars on the arms of stars that have already been drawn. The resulting pattern has the outline of two koch flakes embedded in it. These may show up more clearly when the entire pattern has been finished and the palette changes begin. At this stage there is a two-second delay between each colour change, unless you hold down any key, in which case this delay is removed and you can rapidly step through the colours until an interesting colour scheme is found.

KOCH

```
 0 REM Koch Flake
20 MODE1
30 VDU5
40 VDU19,1,4;0;:VDU19,2,6;0;
50 CS=COS(PI/6)
60 PROCSTAR(640,512,500,2)
70 REPEAT
80 VDU19,RND(3),RND(8)-1;0;
90 A=INKEY(200)
```

```
  100 UNTIL FALSE
  110 END
  120 DEFPROCSTAR(X%,Y%,S%,C%)
  130 LOCAL I
  140 IF S%<12 THEN ENDPROC
  150 IF C%=0 THEN GCOL0,2 ELSE GCOL0,C%
  160 VDU29,X%;Y%;
  170 XL%=S%*CS:YL%=S%/2
  180 MOVE 0,S%:MOVE XL%,-YL%:PLOT85,-XL
%,-YL%
  190 MOVE 0,-S%:MOVE XL%,YL%:PLOT85,-XL
%,YL%
  200 FORI=0 TO 2*PI-PI/3 STEP PI/3
  210 PROCSTAR(X%+S%*SIN(I)*0.68,Y%+S%*C
OS(I)*0.68,S%/3,(C%+1)AND3)
  220 NEXT
  230 ENDPROC
```

Description of program

40	Make cold colours appear while the flake is being drawn.
50	This value is going to be used often so to speed things up save it as CS.
60	Call PROCSTAR to draw the pattern centered about (640,512) with initial size 500 and logical colour 2.
70	Change the colours on the screen at random.
80	Do a random palette change.
90	Wait 2 seconds if no key is pressed.
100	Carry on until someone presses ESCAPE.
110	We never get here but this helps to show where the procedure declarations start. PROCSTAR(X%,Y%,S%,C%) draws a star in logical colour C% centred about the point (X%,Y%) of size S%, it then calls itself on each of the arms of the star it has just drawn.
130	This will hold an angle that determines which arm the next call of PROCSTAR is for.
140	Do nothing if the size is less than 12.
150	Choose the colour.
160	Put the origin at (X%,Y%).
170	Draw the star using XL% and YL%.
180-190	Draw two overlapping triangles to form a star.
200	This loop calls PROCSTAR for each arm of the star just drawn.
210	Call PROCSTAR to draw a star centred in the middle of the arm pointing at an angle I,

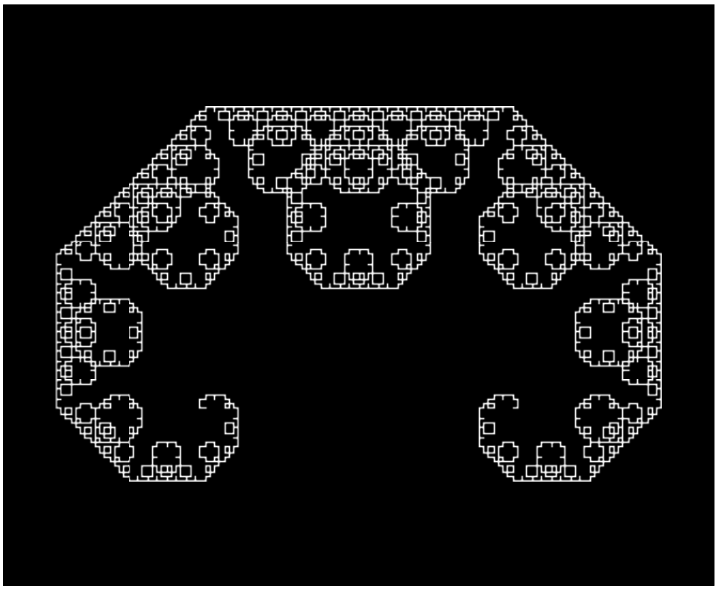
```
            with S% one third its previous value and a
            new colour.
230         Leave the procedure.
```

Line replacement patterns

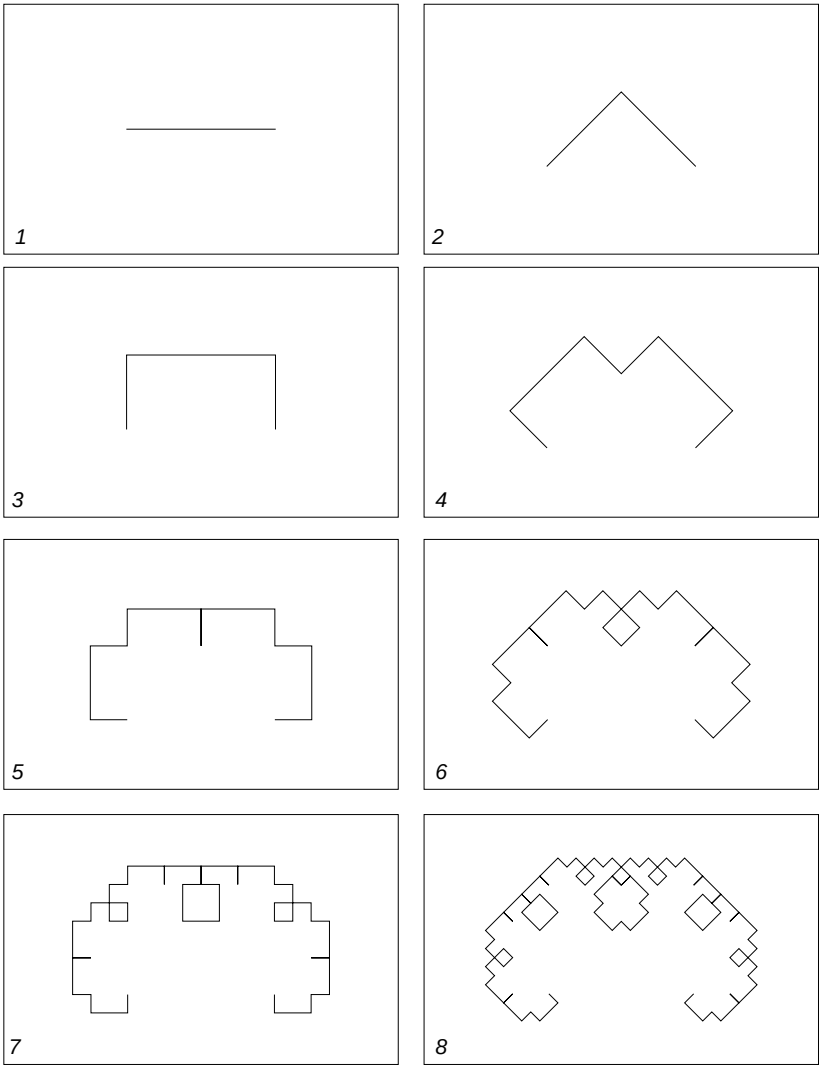
The next two programs produce patterns obtained by repeatedly applying a replacement rule to every straight line in the pattern. In both cases the initial pattern here is simply one straight line.

The two programs produce very different results despite the fact that there is only a slight difference in the two replacement rules. Many other interesting patterns could be produced by modifying the way in which the parameters are altered in the recursive line-drawing procedure.

C-Curve



A C-curve is produced by recursively replacing a straight line between two points by two shorter lines at right-angles to each other that form an elbow connecting the points. The curve is plotted when the length of the lines has become less than a given value. The first few steps in the construction are shown on the next page:



The pattern that results depends on the length 6f the initial line and the depth of recursion. As given, the program is quite slow since it has to calculate a sine and a cosine for each line drawn. One simple way to speed this up would be to save all the possible values of sin and cos that will be required in an array.

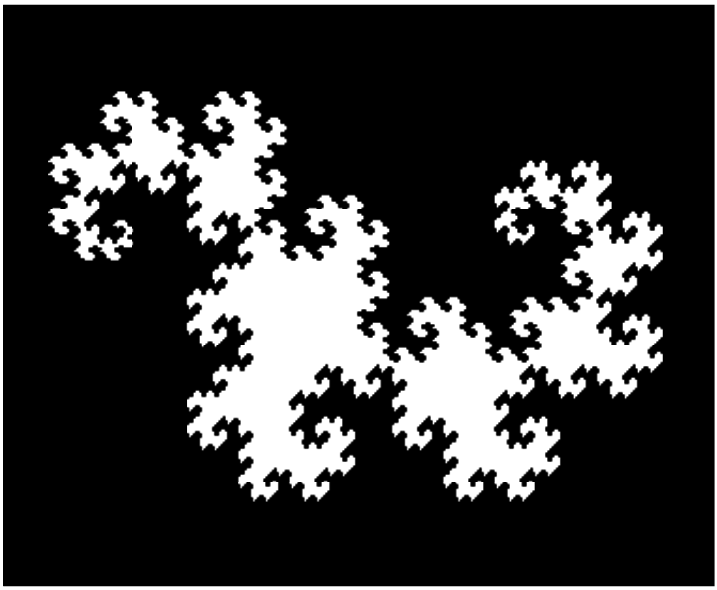
C-CURVE

```
    0 REM C-Curve
   20 MODE1
   30 VDU5
   40 MIN%=15
   50 GCOL0,2
   60 MOVE300,200
   70 PROCC(700,0)
   80 REPEAT UNTIL FALSE
   90 DEFPROCC(L%,ANGLE)
  100 IF L%<MIN% THEN PLOT 1,L%*COS(ANGL
E),L%*SIN(ANGLE):ENDPROC
  110 L%=L%/SQR(2)
  120 PROCC(L%,ANGLE+PI/4)
  130 PROCC(L%,ANGLE-PI/4)
  140 ENDPROC
```

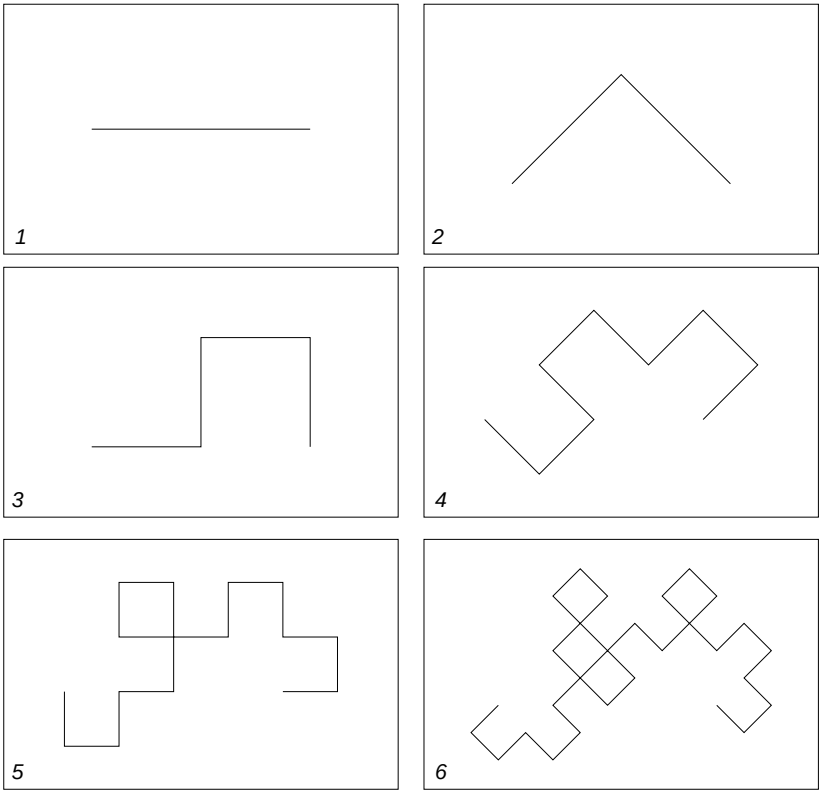
Description of program

40	MIN% gives the maximum length of any line that will be plotted. This is a way of controlling the depth of recursion. If MIN% is large the program will not recurse very far.
60	This is the starting point from which the curve grows.
90	PROCC(L%,ANGLE) will draw a line of length L% at an angle ANGLE starting at the current position if L% is less than MIN%.
100	If the length is short enough, draw the line and then stop.
110	Reduce the length.
120-130	Call PROCC twice for each of the new shorter lines.

Dragon Curve



The dragon curve is so called because of its contorted shape which resembles a Chinese dragon. It is produced by a very similar technique to that used to draw the C-curve, Here the recursive step is to replace a straight line between two points by two shorter lines, forming a hat to one side of the original line. In the C-curve the hat formed is always put on the same side of the line being replaced, whereas to produce a dragon curve the hat is placed alternately on different sides of the line. The first few stages of the construction are shown on the next page.



In this program the drawing process has been speeded up by saving a table of the trig functions so that these do not have to be calculated for each line drawn. The curve is drawn using the triangle fill relative command, PLOT 81, which does not fill the curve exactly but is a reasonable approximation when the lines composing the curve are small.

DRAGON

```
  0 REM Dragon Curve
 20 MODE1
 30 VDU5
 40 DIM S(7),C(7)
 50 A=0
 60 FORI%=0TO7
 70 S(I%)=SIN(A):C(I%)=COS(A)
 80 A=A+PI/4
 90 NEXT
100 M%=12
```

```
  110 GCOL0,RND(3)
  120 MOVE200,700:MOVE200,700
  130 PROCC(1024,0,-1)
  140 REPEATUNTIL FALSE
  150 DEFPROCC(L%,I%,T%)
  160 IF L%<M% THEN PLOT 81,L%*C(I%),L%*
S(I%):ENDPROC
  170 L%=L%*1000/1414
  180 PROCC(L%,(I%+T%)AND7,1)
  190 PROCC(L%,(I%-T%)AND7,-1)
  200 ENDPROC
```

Description of program

40 These will hold SIN(A) and COS(A) for all the values A can take in the construction.

60-90 Store sine and cosine values in arrays S and C. This makes the drawing of the curve much faster because all the trig functions can rapidly be found from the values stored in these arrays.

100 M% gives the maximum length of any line that will be drawn.

120 Select a colour at random. Set the point from which the curve will start. Two moves are required since the curve is to be filled in using PLOT 81.

130 Call PROCC. This will grow out to join a point 1024 units to the right. The first hat is to be placed below the original line.

140 Do nothing until ESCAPE is pressed (this stops the ">" appearing).

150 PROCC takes these parameters:

L% is the length of the line

I% is a value used to index the tables of trig functions, the value of I% corresponds to the angle of the line.

T% is a value that determines on which side of the line the new lines, forming a hat, are to be placed.

160 If the length is short enough draw a line then stop.

170 Reduce the length, prior to the recursive calls of PROCDRAG. This has the same effect as L%=L%/SQR(2] or L%=L%/l.4l4, but is slightly faster than either of these more

obvious methods.

180 Call PROCC. L% is the new length. The new trig table index depends on the value of T%; we either add or subtract PI/4 to the angle. AND 7 ensures that the index wraps around for angles 0 and 2*PI. Draw the next hat 'above'

190 Call PROCC for the other line in the hat. This line is at right angles to the line produced in the call above, and the first hat growing from it is on the opposite side.

200 Leave the procedure.

Tree

Recursive procedures can be used quite effectively to produce irregular patterns; here is one that draws a tree. Each branch of the tree can be thought' of as a separate tree growing out at an angle from the previous branch. The procedure takes parameters that determine the length of the branch, its angle and the number of sub-trees that grow from it. The parameter DEPTH% controls the depth of recursion.

TREE

```
    0 REM Tree
   20 MODE1
   30 MOVE500,200
   40 VDU19,2,2;0;
   50 PROCTREE(PI/2,200,5,4)
   60 END
   70 DEFPROCTREE(ANGLE,HEIGHT%,BRANCHES%,DEPTH%)
   80 LOCAL I%,X%,Y%
   90 IF DEPTH%=0 THEN ENDPROC
  100 X%=HEIGHT%*COS(ANGLE)
  110 Y%=HEIGHT%*SIN(ANGLE)
  120 FOR I%=1TO BRANCHES%
  130 IF DEPTH%=1 THEN GCOL0,2 ELSE GCOL0,1
  140 PLOT1,X%,Y%
  150 PROCTREE(RAD(RND(180)),HEIGHT%/2,BRANCHES%,
DEPTH%-1)
  160 PLOT0,-X%,-Y%
  170 NEXT
  180 ENDPROC
```

Description of program

30 Define the point from which the tree will grow.

40 Set logical colour 2 to appear as green, the colour of the leaves.

50 Call PROCTREE. The effect of these parameter values is as follows:

PI/2 makes the trunk vertical

The trunk will be of length 200.

There will be 5 branches growing from each fork point.

Recurse 4 times.

60 Stop.

70 PROCTREE takes these parameters:

ANGLE (in radians measured anti-clockwis from 'east') determines the angle at which a branch will be drawn.

HEIGHT% gives the length of the branch.

BRANCHES% is the number of subtrees that grow from each fork point.

DEPTH% is a measure of the depth of recursion.

80 Declare 1%, X%, Y% to be local variables, otherwise their values would be altered in recursive calls of PROCTREE at line 150.

90 Have we recursed enough? If so then stop.

100-110 The point (X%,Y%) is the tip of the current branch.

120 PROCTREE is called once each time round this loop.

130 Select the colour.

140 Draw the branch.

150 Draw a subtree from the fork point, with new parameters:

RAD(RND(l80)) is a random angle between 0 and 180 degrees (RAD converts this into radians).

HEIGHT%/?. halves the length of the branches.

BRANCHES% makes the same number of branches grow from each fork in the subtree.

DEPTH%-1 - we have recursed, so reduce DEPTH%.

160 Move back to the fork point.

180 Leave the procedure,

Ideas that have grown from PROCTREE

This tree program is very readily-modified to produce a vast number of different effects. The shape of the tree drawn depends on the values used for the initial parameters. Try altering these to get an idea how each one affects the final result. The way the parameters are altered in the recursive call of PROCTREE at line 150 is very important. See how the structure of the

tree can be changed by altering this line.

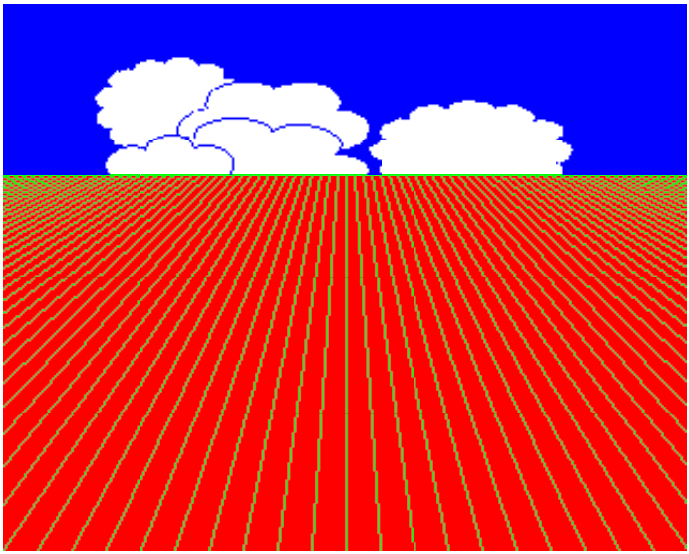
More complicated tree procedures could easily produce very life-like trees. One way to do this would be to have separate procedures to draw different parts of the tree. For example the branch-drawing procedure could call PROCLEAF to draw solid leaves along the shorter branches.

Another possible improvement would be to draw solid branches using the PLOT 85 command, This could be done by introducing a parameter, WIDTH, that would determine the thickness of each branch. A 3-dimensional effect could be produced by shading these solid branches with the random dot technique.

6 Pictures

The programs in this chapter make use of many of the techniques described in the preceding chapters to create four quite complex pictures. Each program is composed of a number of procedure calls that deal with different aspects of the picture. In this way a complicated view is built up from simple parts, each of which can be written and tested separately.

Windy field



Here is a program that uses both horizontal and vertical scrolling to produce an animated picture. The program draws first a view of a windy field above which clouds drift across the sky. Next, the clouds sink below the horizon and the view shifts upwards to the sun.

Scrolling the screen in this way is a neat way of moving from one scene to the next. This technique could also be used in a wider context to clear the screen while avoiding the use of the rather sudden CLS or CLG commands.

The program is built up from a series of procedures, each one drawing a different part of the view or performing a special action. The main program is quite short and is described below. The procedure

declarations that follow could be in any order; these give the details of how to draw a particular object.

The colour scheme chosen for the view is very much a matter of individual taste. You can easily alter the way the palette is set up at lines 30 and 210, or you could write a procedure that would read in logical and physical colour numbers and alter the palette according to the numbers supplied each time the program is run. This way you can draw the picture and {:hen experiment with different colour schemes Interactively.

FIELD

```
    0 REM Windy field
   20 MODE1
   30 VDU23;10,32;0;0;0;
   40 VDU19,0,4;0;19,2,2;0;
   50 WIND%=0
   60 FORY%=860 TO 700 STEP -40
   70 PROCCLOUD(200+RND(870),Y%,100+RND(
100),80+RND(20),TRUE,3,0)
   80 NEXT
   90 REPEAT
  100 WIND%=WIND%-1
  110 VDU23;13,WIND%;0;0;0;
  120 A=INKEY(12)
  130 UNTIL WIND%=0
  140 PROCGROUND(16)
  150 VDU28,0,9,39,0
  160 COLOUR128
  170 A=INKEY(300)
  180 PROCDOWN(10,80)
  190 VDU28,0,31,39,10
  200 PROCDOWN(22,0)
  210 VDU19,2,3;0;19,3,7;0;
  220 PROCSUN(600,700,1000,TRUE,3)
  230 PROCSUN(600,700,200,FALSE,1)
  240 PROCCLOUD(400,600,300,200,FALSE,2,
3)
  250 A=INKEY(300)
  260 REPEATVDU19, RND(4)-1, RND(8)-1;0;
 :A=INKEY(200):UNTIL FALSE
  270 END
```

Description of program

30 Remove the flashing text cursor.

40 Change the palette so that logical colours 0 and 2 appear as blue and green.

50 The variable WIND% will determine how far the screen has been scrolled. It is altered in PROCCLOUD as the clouds are moved.

60-80 Draw some white clouds, scrolling the screen as they are drawn.

90-130 This loop scrolls the screen back to its orignal position.

110 Re-program register 13 of the 6845 video controller chip, with the value of WIND%. The effect of this is to move the portion of memory displayed on the screen.

140 Draw the ground.

150 Define a text window around the sky.

160 Set the text background colour to logical colour 0, the same colour as the sky,

170 Wait 3 seconds, unless a key is pressed.

180 Scroll the sky down so that the clouds sink below the horizon.

190 Define a text window around the ground.

200 Quickly scroll the ground down to leave an empty blue screen. change the palette to make logical colours 2 and 3 appear as yellow and white.

220 Draw the rays in white.

230 Draw the sun in red.

240 Draw a cloud to slightly obscure the sun.

260 Sit there changing the palette at random.

PROCDOWN

This is a procedure that scrolls the current text window downwards. The procedure takes two parameters: N% and D%. N% determines how far to scroll the window and D% specifies the delay between each movement.

```
280 DEFPROCDOWN(N%,D%)
290 FORI%=1 TO N%
300 VDU11
310 A=INKEY(D%)
320 NEXT
330 ENDPROC
```

Description of PROCDOWN

290 Each time around this loop the text window scrolls down one line.

300 This scrolls the screen down one line. We

assume the text cursor is at the top left-hand corner of the window when PROCDOWN is called.

PROCGROUND

This draws the field, which consists of a solid background colour and lines that give the impression of depth as they converge towards the horizon. The background is filled in very rapidly by defining a text window around the region, and then clearing the text window wi th the col our set to be the text background colour.

```
340 DEF PROCGROUND(S%)
350 VDU28,0,31,39,10
360 COLOUR129
370 CLS
380 VDU26
390 GCOL0,2
400 VDU29,640;0;
410 MOVE-640,700:DRAW640,700
420 FORX%=-640 TO 640 STEP S%
430 MOVEX%,700:DRAW4*X%,0
440 NEXT
450 ENDPROC
```

Description of PROCGROUND

350	Define a text window around the ground.
360	Select text background colour to be logical colour 1.
370	Clear the text area, thus filling the region red.
380	Get rid of the text window.
390	This will be the colour of the lines.
400	Move the origin to the centre of the bottom line of the screen.
410	Draw a line along the horizon.
420-440	Draw lines from the horizon to the bottom of the screen, spreading out to give an impression of depth.

PROCCLOUD

This procedure draws a cloud. The cloud outline is obtained by tracing round a small circle as the centre of the circle moves round a larger circle. It takes 7

parameters:

X% is the absolute X-coordinate of the centre of the cloud.

Y% is the absolute Y--coordinate of the centre of the cloud.

SX% is the size of the cloud in the X-direction.

SY% is the size of the cloud in the Y--direction.

SCROL% is a boolean that determines whether to scroll the screen sideways as the cloud is being drawn.

C1% is the logical colour of the solid centre of the cloud.

C2% is, the logical colour of the line around the edge of the cloud.

```
  460 DEFPROCCLOUD(X%,Y%,SX%,SY%,SCROL%,C1%,C2%)
  470 VDU29,X%;Y%;
  480 L%=6+RND(8)
  490 MOVE0,0:MOVESX%+SX%/L%,0
  500 X1%=SX%+SX%/10:Y1%=0
  510 IF WIND%>120 THEN S%=-1 ELSE S%=1
  520 FORI=0TO 6.3 STEP 0.1
  530 IF SCROL% THEN VDU23;13,WIND%,0;0;
0;:WIND%=WIND%+S%
  540 X%=SX%*COS(I)+SX%/L%*COS(I*L%)
  550 Y%=SY%*SIN(I)+SY%/L%*SIN(I*L%)
  560 GCOL0,C1%
  570 MOVE32,12:PLOT85,X%,Y%
  580 MOVEX1%,Y1%
  590 GCOL0,C2%:DRAWX%,Y%
  600 X1%=X%:Y1%=Y%
  610 NEXT
  620 VDU29,0;0;
  630 ENDPROC
```

Description of PROCCLOUD

470	Move the origin to the centre of the cloud.
480	L% controls the number of lumps the cloud will have.
490	Move to the centre and then to the first point on the edge.
500	(Xl%,Yl%) is the first point on the edge of

the cloud; as we trace round the cloud this will always be one step behind the point (X%,Y%).

510 Scroll the screen to the left or the right, as required (this prevents the screen from getting too far out of place).

520 I is the angle that measures how far round the cloud we are. 6.3 is approximately PI*2.

530 Scroll the screen if required.

540-550 Calculate the next point along the edge.

560 Select the interior colour (silver not allowed!).

570 Move to a point inside the cloud and fill a triangle to the edge.

580 Move back to the previous edge point.

590 Select the edge line colour and draw along the cloud edge.

600 Move (Xl%,Yl%) along the edge.

620 Return the origin to the usual place.

PROCSUN

This procedure will either draw a solid circular sun or a circle of rays reaching out from the central point. The boolean variable RAYS% determines which to draw.

The parameters are described below: (X%,Y%) is the absolute position of the centre of circle or rays.

S% is the radius of the circle.

RAYS% is either TRUE or FALSE; draw rays or draw a solid circle.

C% is the logical colour to be used.

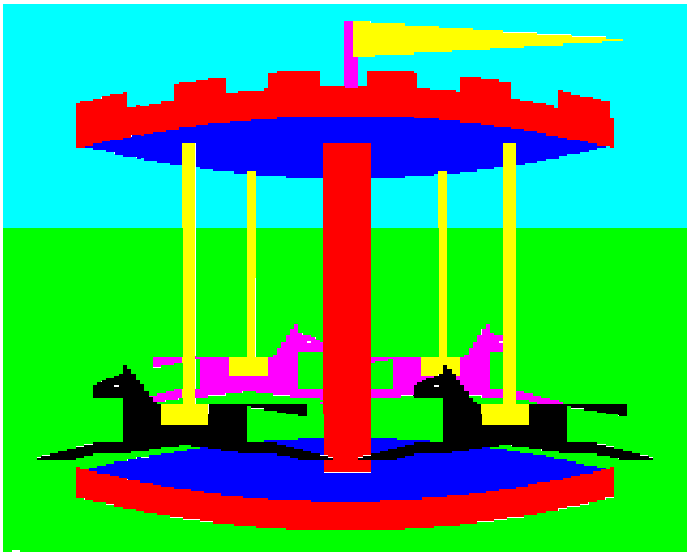
```
  640 DEFPROCSUN(X%,Y%,S%,RAYS%,C%)
  650 VDU29,X%;Y%;
  660 GCOL0,C%
  670 MOVE4,12
  680 T%=TRUE
  690 FORA=0 TO 6.3 STEP 0.1
  700 MOVE0,0
  710 X%=S%*SIN(A)
  720 Y%=S%*COS(A)
  730 IF RAYS% AND T% THEN MOVE X%,Y% EL
SE PLOT 85,X%,Y%
  740 T%=NOT T%
  750 NEXT
```

```
760 ENDPROC
```

Description of PROCSUN

```
650       Put the origin at the centre of the sun.
660       Select the colour specified.
670       Define a point inside the sun.
680       T% toggles on and off as the rays are drawn.
690       Go round a circle. (X%,Y%) is a point on the
          edge of the circle .
730       Either fill a triangle to the edge, or leave
          a gap between rays.
740       Toggle T%.
```

Merry-go-round



This program draws a multi-coloured merry-go-round. The merry-go-round has a base, a roof with a flag on the top and four horses. These are drawn by the procedures PROCEASE, PROCTOP, PROCELAG and PROCHORSE.

The routine PROCRECT fills in a rectangle in a given colour . It does this by defining a graphics window around the region and then clearing this region with the colour required as the graphics background colour. This method must be used with care since if any point

of the window is off the screen the entire screen will be filled. Despite this, the rectangular fill using a window is a useful complement to the more common triangle fill technique. A similar method can be used with text windows; these give much faster filling but the window cannot be positioned to a single pixel resolution.

MERRYGO

```
    0 REM Merry-Go-Round
   20 MODE2
   30 VDU5
   40 PROCRECT(0,600,1272,1020,6)
   50 PROCRECT(0,0,1272,600,2)
   60 PROCBASE(640,150)
   70 PROCTOP(640,760)
   80 PROCFLAG(640,868)
   90 FORX%=460 TO 820 STEP 360
  100 PROCHORSE(X%,300,90,5,FALSE)
  110 PROCRECT(X%,340,X%+8,710,3)
  120 NEXT
  130 PROCRECT(600,150,680,760,1)
  140 FORX%=340 TO 940 STEP 600
  150 PROCHORSE(X%,200,110,0,TRUE)
  160 PROCRECT(X%,250,X%+16,760,3)
  170 NEXT
  180 END
```

Description of program

40 Draw the sky.
50 Draw the ground.
60 Draw the base of the merry-go-round.
70 Draw the top.
80 Draw a flag.
90-120 This loop draws two horses, facing right, on the far side of the platform.
130 Draw the central pillar.
140-170 Draw the front two horses facing left.

PROCRECT

This procedure fills in a rectangle in a given colour. It does this by defining a graphics window around the area to be filled, and then clearing the graphics screen with the colour specified.

```
  190 DEFPROCRECT(X%,Y%,SX%,SY%,C%)
```

```
200 VDU24,X%;Y%;SX%;SY%;
210 GCOL0,128+C%
220 CLG
230 VDU26
240 ENDPROC
```

Description of PROCRECT

200 Define graphics window.
210 Select graphics background colour.
220 Fill the window with that colour.
230 Get rid of the window.

PROCBASE

The base consists of two parts; the floor drawn in blue, and the side.

This procedure draws a perspective view of the base, which can be thought of as a rather flat drum, or a very large coin.

```
250 DEF PROCBASE(X%,Y%)
260 VDU29,X%;Y%;
270 FORX%=-500 TO 500 STEP 4
280 Y%=60-(X%*X%)/4000
290 MOVE X%,Y%
300 GCOL0,4:PLOT1,0,-2*Y%
310 GCOL0,1:PLOT1,0,-50
320 NEXT
330 ENDPROC
```

Description of PROCBASE

260 Move the origin to the centre of the base.
270 The base is 1000 units across - it is filled in by drawing lots of lines next to each other.
280 Calculate the Y-coordinate of the next line.
290 Move to the top of the next line.
300 Draw the floor in blue.
310 Draw the edge.

PROCTOP

The top of the merry-go-round is similar to the base. The main difference is that the roof has turrets along

its side.

```
  340 DEFPROCTOP(X%,Y%)
  350 VDU29,X%;Y%;
  360 FORX%=-500 TO 500 STEP 4
  370 Y%=60-(X%*X%)/4000
  380 MOVEX%,-Y%
  390 GCOL0,4
  400 PLOT 1,0,2*Y%
  410 GCOL0,1
  420 IF((500+X%)DIV90)MOD2=1 THEN 440
  430 PLOT 1,0,80:PLOT0,0,-80:GOTO450
  440 PLOT1,0,50:PLOT0,0,-50
  450 NEXT
  460 VDU26
  470 ENDPROC
```

Description of PROCTOP

350-400	As for the first few lines of PROCEASE.
420	Decide whether to draw a line in a turret.
430	Draw a turret line.
440	Draw a line in the gap between turrets.
460	The VDU 26 statement causes a return to the default text and graphics windows; one side effect of this, utilised here, is to return the origin to its normal place.

PROCFLAG

This draws a flag with its base at the point (X%,Y%).

```
  480 DEFPROCFLAG(X%,Y%)
  490 PROCRECT(X%,Y%,X%+16,Y%+120,5)
  500 MOVEX%+16,Y%+120
  510 GCOL0,3:PLOT0,500,-30:PLOT81,-500,-30
  520 ENDPROC
```

Description of PROCFLAG

490	Draw the flag-pole.
500	Move to the top of the pole.
510	Move to the tip of the flag and fill the flag in.

PROCHORSE

This draws a horse. The procedure requires the following parameters:

(X%,Y%) is the location of the horse.

S% controls how large the horse will be.

C% is the logical colour of the body of the horse.

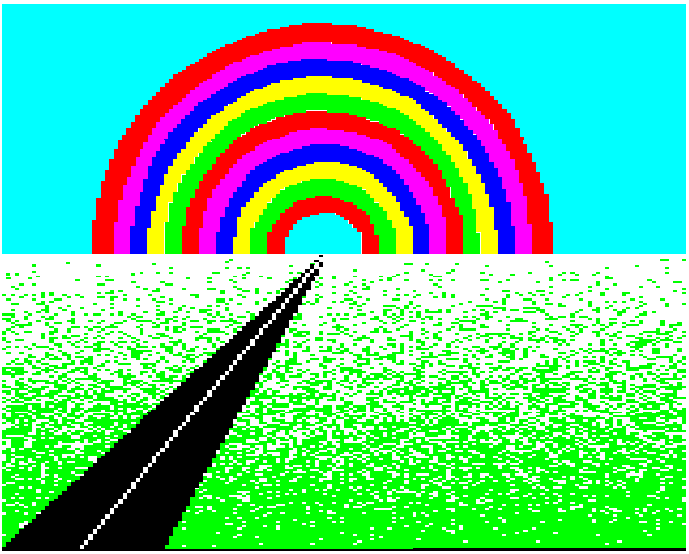
RIGHT% is a boolean value that determines the direction the horse is facing.

```
  530 DEF PROCHORSE(X%,Y%,S%,C%,RIGHT%)
  540 PROCRECT(X%-S%,Y%,X%+S%,Y%+S%/1.5,C%)
  550 VDU29,X%;Y%;
  560 GCOL0,C%
  570 MOVE0,0
  580 FORI%=-S% TO S% STEP 2*S%
  590 MOVEI%,-S%/8:PLOT85,1.5*I%,-S%/8
  600 MOVE2*I%,-S%/4:PLOT85,2.5*I%,-S%/4
  610 MOVE1.5*I%,0:PLOT85,0,0
  620 NEXT
  630 IF RIGHT% THEN D%=-S% ELSE D%=S%
  640 MOVE D%/2,S%/1.5
  650 MOVE D%,S%*2/1.5
  660 PLOT 85,D%,S%/1.5
  670 MOVED%*1.5,S%*1.2/1.5
  680 MOVED%,S%*1.2/1.5
  690 PLOT85,D%*1.5,S%
  700 PLOT85,D%,S%*1.2
  710 MOVE -D%,S%/1.5
  720 MOVE-D%*2,S%/1.5
  730 PLOT85,-D%*2,S%/2
  740 GCOL0,7:PLOT69,D%*1.2,S%
  750 VDU29,0;0;
  760 PROCRECT(X%-S%/3,Y%+S%/3,X%+S%/3,Y
%+S%/1.5,3)
  770 ENDPROC
```

Description of PROCHORSE

540	Draw the body.
550-520	Draw the legs.
630	Is the horse to be facing right or left?
640-700	Draw the head.
710-730	Draw the tail.
740	Draw the eye.
760	Draw the saddle.

Rainbow



This program draws a view of a rainbow spanning a road that shoots through a snow-covered grassland. There is plenty of scope for adding more features to this rural scene. For example clouds, hills, houses or trees might enhance the bleak landscape.

Even the simplest picture can be dramatically improved by introducing some animation. Here clouds could be made to blow across the sky and their shadows could drift across the grass. The large number of logical colours make animated effects possible by drawing several different picture frames. Palette changes can then be used to reveal each of these in turn.

RAINBOW

```
  0 REM Rainbow
 20 MODE2
 30 VDU5
 40 PROCSKY(550)
 50 PROCBOW(600,550,80,32)
 60 PROCGRASS(550)
 70 PROCROAD(600,550)
 80 END
```

All that remains to do now is to declare the procedures that are to be called above. The order in

which these procedures are declared has no effect on the operation of the program.

PROCBOW

This draws a rainbow whose centre is at the point (X%,Y%). D% is the radius of the innermost band of colour in the rainbow. S% is the thickness of the bands. The rainbow is drawn by filling in small triangles formed between the two concentric circles on the edges of each band of colour.

```
   90 DEFPROCBOW(X%,Y%,D%,S%)
  100 VDU29,X%;Y%;
  110 FORI%=0TO10
  120 MOVE-(I%*S%+D%),0:MOVE-((I%+1)*S%+
D%),0
  130 GCOL0,(I% MOD 5)+1
  140 FORA=-PI/2-0.1 TO PI/2+0.2STEP 0.2
  150 PLOT85,(D%+I%*S%)*SIN(A),(D%+I%*S%
)*COS(A)
  160 PLOT85,((I%+1)*S%+D%)*SIN(A),((I%+
1)*S%+D%)*COS(A)
  170 NEXT A,I%
  180 ENDPROC
```

Description of PROCBOW

100	Move the origin to the centre of the rainbow.
110	Each time round this loop we draw a band of colour.
120	Move to the left-hand side of the rainbow. Two moves are needed since PLOT 85 fills up to the last two points visited.
130	Select the colour for that band. The order of colours is not the same as for a real-world rainbow.
140	Move round the circle by increasing the angle A.
150	Fill in the triangle whose tip is on the inner circle.
160	Do the same moving to the outer circle.
170	carry on until all the bands have been drawn.

PROCGRASS

This procedure draws a perspective view of snow covered grass up to the line Y=H%. First the grass is drawn by simply filling the area green, next the snow covering is produced using a dot shading method.

```
190 DEFPROCGRASS(H%)
200 VDU29,0;0;
210 GCOL0,2
220 MOVE0,0:MOVE0,H%:PLOT85,1300,H%
230 MOVE1300,0:PLOT85,0,0
240 GCOL0,7
250 FORX%=0TO1300STEP8
260 FORY%=4TOH%STEP4
270 IF RND(H%)<Y% THEN PLOT 69,X%,Y%
280 NEXT Y%,X%
290 ENDPROC
```

Description of PROCGRASS

200	Make sure the origin is in the right place.
210-230	Fill the grass area green.
250	Move along the x-axis drawing lines of dots. In MODE 2 pixels are 8x4 coordinate units.
260	Draw a line of dots to the horizon.
270	Make the dots become denser as we approach the horizon.

PROCROAD

This draws a straight road from the origin to a point (X%,Y%). The road consists of a dark background (the tarmac surface) and a central white line (no overtaking).

```
300 DEFPROCROAD(X%,Y%)
310 VDU29,0;0;
320 GCOL0,0
330 MOVEX%,Y%
340 MOVE300,0:PLOT85,0,0
350 GCOL0,7:MOVE150,0:DRAWX%,Y%
360 ENDPROC
```

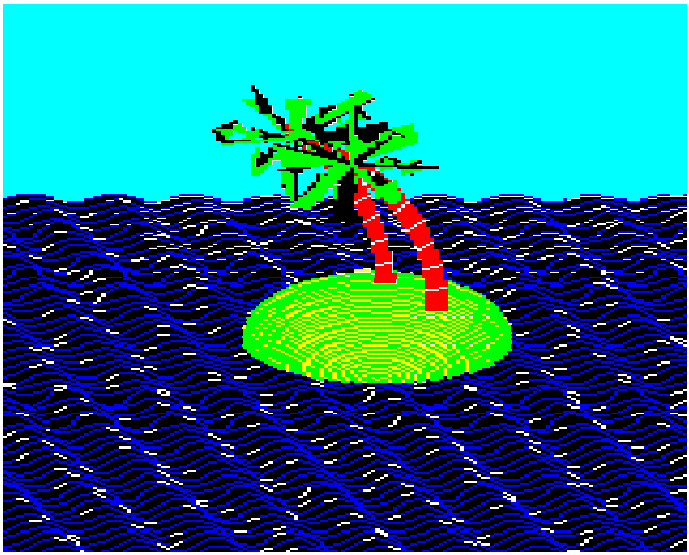
PROCSKY

This simply fills the region above the line y=Y% in the logical colour 6. This is pale blue with the default palette settings. The sky is drawn using two

triangle fill commands.

```
370 DEFPROCSKY(Y%)
380 GCOL0,6
390 MOVE0,Y%:MOVE0,1200:PLOT85,1300,1200
400 MOVE1300,Y%:PLOT85,0,Y%
410 ENDPROC
```

Desert Island



This program draws a view of a desert island with palm trees on it. The program is quite long but is divided up into fairly simple procedures that each draw a part of the picture. Once the entire picture has been drawn the sea springs to life as waves go dancing across it. Because of the modular nature of the program it should not be too hard to get each section to work before joining them together to draw the entire picture.

ISLAND

```
  0 REM Island
 20 MODE2
 30 VDU5
 40 PROCSEA(650)
```

```
  50 PROCISLE(700,400,250)
  60 PROCPALM(700,500,300)
  70 PROCPALM(800,450,300)
  80 REPEAT
  90 PROCSURF
 100 UNTIL FALSE
 110 END
```

The main program simply calls the various procedures one by one. All that follows are the procedure declarations that go into the details of how to draw these objects.

PROCSEA

The sea is composed of a series of sine waves drawn next to each other. To speed up the drawing a table of sine values is stored in the array S. The waves are drawn using 8 different logical colours, and this is exploited later in PROCSURF to give the impression of motion. The horizon is at the line y=650; above this PROCSEA fills in the sky which grows out of the sea.

```
 120 DEFPROCSEA(YL%)
 130 DIM S(100)
 140 FORI%=8TO15:VDU19,I%,4;0;:NEXT
 150 FORI%=0TO100:S(I%)=SIN(I%):NEXT
 160 FORY%=0 TO YL% STEP 12
 170 VDU29,0;Y%;
 180 S%=8+RND(10):MOVE0,S%*S(Y% MOD 100)
 190 C%=7+RND(8)
 200 FORX%=0 TO 1279 STEP 24
 210 GCOL0,C%:C%=C%+1:IF C%=16 THEN C%=8
 220 DRAW X%,S%*S((X%+Y%) MOD 100)
 230 NEXT,
 240 YL%=Y%:S%=8
 250 VDU29,0;YL%;
 260 MOVE0,S%*S(YL% MOD 100)
 270 FORX%=0 TO 1279 STEP 24
 280 NY%=S%*S((X%+24+YL%) MOD 100)
 290 GCOL0,6:Y%=S%*S((X%+YL%)MOD 100)
 300 MOVE X%,1200:PLOT 85,X%+24,1200:
MOVE X%+24,NY%:PLOT85,X%,Y%
 310 GCOL0,C%:C%=C%+1:IF C%=16 THEN C%=8
 320 DRAWX%+24,NY%
 330 NEXT
 340 ENDPROC
```

Description of PROCSEA

150 Define logical colours 8-15 to appear as

dark blue.

160 Save sine values in the array S.

170 Draw lines of waves up to the horizon.

180 Put the origin at the start of the line.

190 S% is a slightly random amplitude. Move to the start of the wave.

200 Select a random logical colour to start.

210 Move across drawing the wave.

220 Select a new colour for each part of the wave

230 Draw the wave.

240 On leaving these loops all that remains is to fill in the sky, and draw one more wave along the horizon.

250 Fudge the horizon to be above the last wave drawn. Make the amplitude 8.

260 Put the origin at the start of the horizon wave .

270 Move to the start of the horizon wave.

280 Now move along the horizon.

290 NY% is the next Y value on the horizon wave.

300 Select logical colour 6 for the sky, and calculate the current Y value,

310 Fill in the rectangle of sky above the next step along the horizon wave. The aim of all this business is to get the sky and sea to knit together.

320 Select the colour of the next segment of the horizon wave.

330 Draw that part of the wave.

PROCISLE

The island has the outline of a slightly squashed oval. The first two parameters give the position of the island, and the third specifies its size. There are two stages in drawing the island: first, the outline is drawn and the whole next, curved lines are drawn give an impression of depth.

```
350 DEFPROCISLE(XO%,YO%,S%)
360 GCOL0,3
370 VDU29,XO%;YO%;:MOVE0,0
380 FORA=0 TO 2*PI+0.2 STEP 0.2
390 X%=S%*SIN(A):Y%=COS(A)*S%
400 IF Y%>0 THEN Y%=Y%/2 ELSE Y%=Y%/3
```

```
410 MOVE0,0:PLOT85,X%,Y%
420 NEXT
430 GCOL0,2
440 FORA=PI/2 TO PI STEP 0.05
450 S1%=S%*SIN(A)
460 VDU29,XO%;YO%+COS(A)*S%/3;
470 MOVE-S1%,0
480 FORB=-PI/2 TO PI/2+0.2 STEP 0.2
490 X%=S1%*SIN(B):Y%=COS(B)*S1%/2
500 DRAWX%,Y%
510 NEXT,
520 ENDPROC
```

Description of PROCISLE

370	The colour of the sand.
390	This loop traces the outline of the island in colour.
400	The point (X%,Y%) moves around a circle.
410	Squash this circle.
420	Fill in a segment.
440	The colour of the grass
450	We draw a curved line each time around this loop.
460	S1% is the X-coordinate of a point on the edge of the island.
470	This moves the origin and specifies the vertical displacement of each curved line.
480-510	Draw a curved line.

PROCPALM

This draws a palm tree of size S% with its base at (X%,Y%). The palm is composed of two parts: the trunk and the leaves. The curved trunk of the tree is formed by filling the region between portions of two circles, one slightly larger than the other. The leaves are filled using a random walk along radial lines of a circle whose centre is at the top of the trunk.

```
530 DEFPROCPALM(X%,Y%,S%)
540 VDU29,X%-S%;Y%;:MOVES%,0
550 S1%=1.1*S%
560 ST=40/S%
570 FORA=0TO PI/3 STEP ST
580 GCOL0,1
590 SI=SIN(A):CO=COS(A)
600 MOVES%*CO,S%*SI:MOVES1%*CO,S1%*SI
610 PLOT85,S1%*COS(A+2*ST),S1%*SIN(A+2*ST)
620 MOVES%*COS(A+ST),S%*SIN(A+ST):PLOT85,S%*CO,S%*SI
630 GCOL0,7:DRAWS1%*CO,S1%*SI
```

```
  640 NEXT
  650 VDU29,X%+S%*(COS(PI/3)-1);Y%+S%*SIN
(PI/3)+0.05*S%;
  660 GCOL0,2:MOVE0,0:MOVE0,4
  670 FORA=0 TO 2*PI STEP 0.8
  680 FORI%=0TOS% STEP 16
  690 IF RND(2)-1 THEN GCOL0,0 ELSE GCOL0,2
  700 X%=I%/2*SIN(A)+RND(I%/4)-I%/8
  710 Y%=I%/3*COS(A)+RND(I%/4)-I%/8
  720 PLOT85,X%,Y%
  730 NEXT I%,A
  740 ENDPROC
```

Description of PROCPALM

550	S% gives the radius of the inner circle used to draw the trunk.
560	Sl% will be the radius of the outer circle.
570	ST% is the angular step size; this determines the number of segments in the trunk.
580-650	Move round the circle filling the segments in logical colour I (red) and drawing lines between them in logical colour 7 (white).
660	Put the origin at the top of the trunk.
670	Move roughly to the centre of the leaf circle.
680	Go all the way round the circle.
690	I% moves out to give radial lines.
700	Either draw in green or black.
710-720	Introduce some randomness.
730	Fill that part of the leaf.

PROCSURF

This procedure redefines the physical colours that correspond to the logical colours numbered 8-15. Two effects are produced: first, an impression of ripples moving across the waves, and secondly, the impression of more substantial waves.

In the first case only one of the 8 logical colours is changed at one time. To produce more surf the next section introduces a gap between the two 'points' at which the palette changes occur. The groundwork for this procedure has been done in PROCSEA where the multicoloured sea is drawn. There is enormous scope

for further effects to be produced.

```
  750 DEF PROCSURF
  760 C%=8
  770 FORI%=0TO100
  780 VDU19,C%,7;0;
  790 A=INKEY(12)
  800 VDU19,C%,4;0;
  810 C%=C%+1:IF C%=16 THEN C%=8
  820 NEXT
  830 FORJ%=1TO100
  840 R%=RND(5)
  850 FORK%=0TO20
  860 C%=C%+1:IF C%=16 THEN C%=8
  870 VDU19,C%,4;0;
  880 VDU19,(C%+R%)MOD 8 +8;0;
  890 A=INKEY(12)
  900 NEXT K%,J%
  910 ENDPROC
```

Description of PROCSURF

770 C% gives the logical colour to be changed.

780 While in this loop simple ripples are produced.

790 Make logical colour C% appear white.

800 Wait 12 centi-seconds.

810 Make log1cal colour C% appear dark blue again.

820 Increase C%; this causes the ripple to move to the right.

840 While in this loop more complicated waves appear.

850 The value of R% determines how large the area of white surf will become.

860 Do 20 surf movements before changing R%.

880 Make logical colour C% appear as dark blue.

890 The effect of this is to make a gap between where the wave becomes white and the point at which it reverts to being blue. This means that instead of a small dash, a portion of the wave appears in white.

Appendix A

Running programs on the Model A

The programs in this book and on the cassette are supplied as versions that take advantage of the high-resolution graphics capabilities of the BBC Microcomputer Model B. Although on the Model A there will be lower resolution and fewer colours, Model A users will lose very little in terms of overall effect. The following instructions show how to alter the programs so they will run on the Model A.

All but two of the programs in this book will run on the Model A if the MODE statement at the beginning of the program we, changed to MODE 5. The procedure is as follows:

1 To load programs from cassette use the LOAD command instead of CHAIN. So to load the program MOUNTS, for example, type

  LOAD "MOUNTS"

2 When the program has loaded, type

  LIST 10,40

  which wall list lines l0 to 40 inclusive when you press RETURN.

3 Find the MODE statement at line 20, or line 30 of the program. It will probably read

  20 MODE 1

  All you have to do change it is enter (next to the prompt)

  20 MODE and press RETURN.

  Now when you LIST the program you will see that line 20 reads 'MODES'.

4 To run the program, type RUN.

Modifications for FIELD and ISLAND

FIELD and ISLAND require further modification as listed below.

FIELD

Change line 20 to read MODE 5
Change line 150 to read VDU 28,0,9,19,0
Change line 190 to read VDU 28,0,9,19,0
Change line 350 to read VDU 28,0,31,19,10

ISLAND

Change line 20 to read MODE 5.

Delete lines go and 150 from the program. The way to do this is to type at the end of the listing:

```
90 (RETURN)
150 (RETURN)
```

When you LIST the program you will see that lines 90 and 150 have been removed.

PROCSURF, the procedure giving the effect of white surf on the waves, will not operate on the Model A.

Index