DK

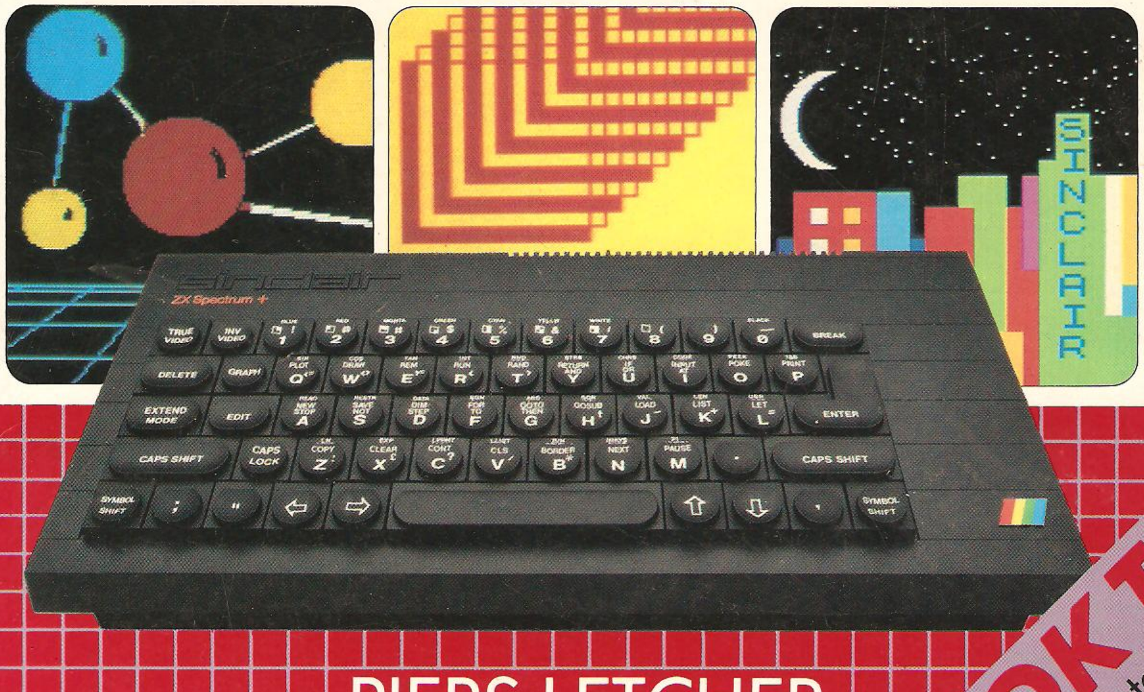
Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

ZX SPECTRUM AND ZX SPECTRUM+

GRAPHICS



PIERS LETCHER

BOOK THREE

A complete BASIC plus machine-code system for fast high-resolution graphics

DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

ZX SPECTRUM ZX SPECTRUM +

GRAPHICS

THE DK SCREEN-SHOT PROGRAMMING SERIES

Books One and Two in the DK Screen-Shot Programming Series brought to home computer users a new and exciting way of learning how to program in BASIC. Following the success of this completely new concept in teach-yourself computing, the series now carries on to explore the speed and potential of machine-code graphics. Fully illustrated in the unique Screen-Shot style, the series continues to set new standards in the world of computer books.

BOOKS ABOUT THE ZX SPECTRUM+

This is Book Three in a series of guides to programming the ZX Spectrum+. It contains a complete BASIC-and-machine-code graphics language for the Spectrum+, and features its own graphics editor which enables you to use all these facilities directly from the keyboard. Together with its companion volumes, it builds up into a complete programming and graphics system.

ALSO AVAILABLE IN THE SERIES

Step-by-Step Programming for the **Commodore 64**

Step-by-Step Programming for the **BBC Micro**

Step-by-Step Programming for the **Acorn Electron**

Step-by-Step Programming for the **Apple IIe**

Step-by-Step Programming for the **Apple IIc**

PIERS LETCHER

After graduating with a degree in Computer Systems, Piers Letcher has worked in many areas of the computer industry, from programming and selling mainframes to designing and marketing educational software. He was Peripherals Editor of *Personal Computer News* until May 1984 and has since written a guide to peripherals and a number of other books for popular home micros.

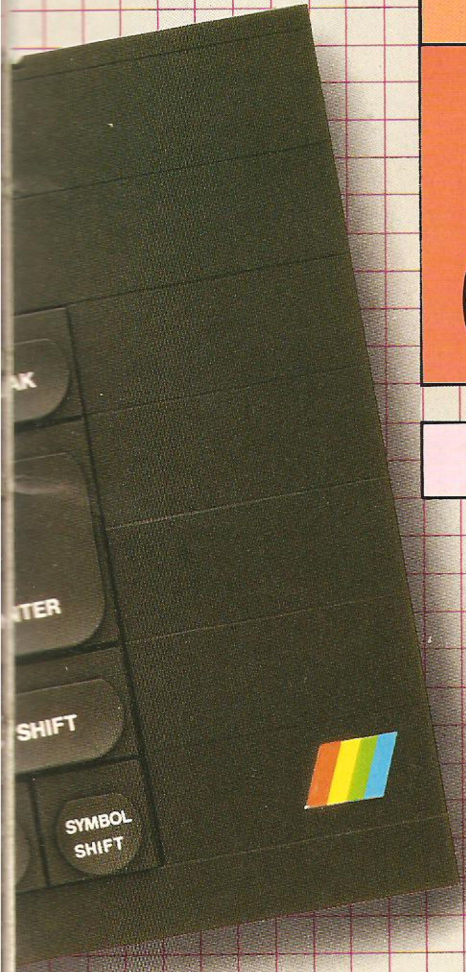
BOOK THREE


sinclair
ZX Spectrum +
TRUE VIDEO
INV VIDEO
BLUE DEF FN ! 1
RED FN @ 2
MGNTA LINE # 3
GREEN OPEN # $ 4
CYAN CLOSE # % 5
YELLW MOVE & 6
WHITE ERASE ' 7
POINT (8
CAT) 9
BLACK FORMAT _ 0
DELETE
GRAPH
SIN ASN PLOT Q <=
COS ACS DRAW W <>
TAN ATN REM E >=
INT VERIFY RUN R <
RND MERGE RAND T >
STR$ [RETURN AND Y
CHR$] IF OR U
CODE IN INPUT AT I
PEEK OUT POKE O
TAB © PRINT P
EXTEND MODE
EDIT
READ ~ NEW STOP A
RESTR | SAVE NOT S
DATA \ DIM STEP D
SGN { FOR TO F
ABS } GOTO THEN G
SQR CIRCLE GOSUB H ↑
VAL VAL$ LOAD J -
LEN SCRN$ LIST K +
USR ATTR LET L =
CAPS SHIFT
CAPS LOCK
LN BEEP COPY Z :
EXP INK CLEAR X £
LPRNT PAPER CONT C ?
LLIST FLASH CLS V /
BIN BRIGHT BORDER B *
INKY$ OVER NEXT N
PI INVERS PAUSE M
SYMBOL SHIFT
;
"

DK

Screen Shot
PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING
ZX SPECTRUM ZX SPECTRUM +
GRAPHICS

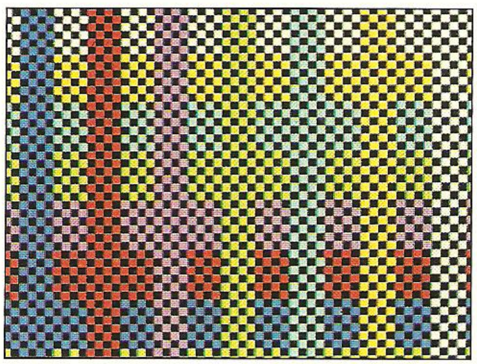


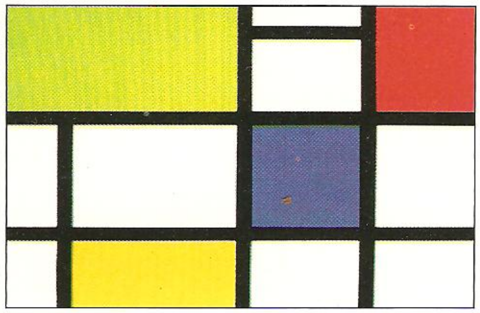
PIERS LETCHER

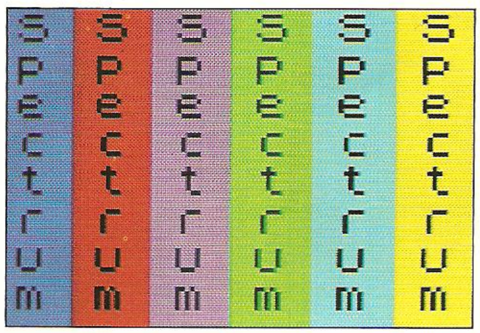
DORLING KINDERSLEY · LONDON

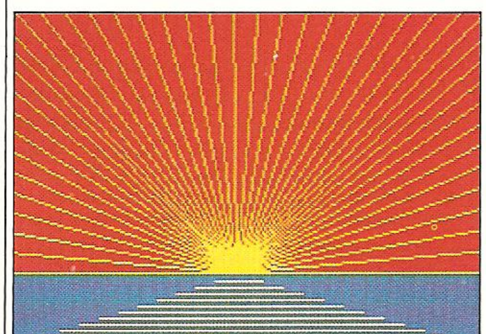
BOOK THREE

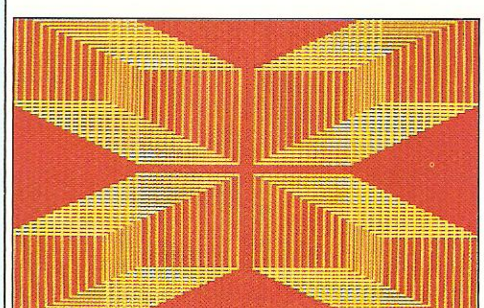
CONTENTS











The DK Screen-Shot Programming Series was conceived, edited and designed by Dorling Kindersley Limited, 9 Henrietta Street, Covent Garden, London WC2E 8PS.

Editor Michael Upshall
Designer Steve Wilson
Photographer Vincent Oliver
Series Editor David Burnie
Series Art Editor Peter Luff
Managing Editor Alan Buckingham

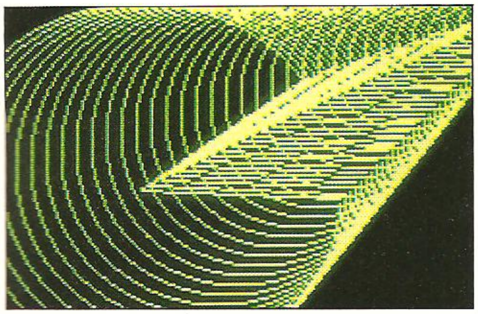
First published in Great Britain in 1985 by Dorling Kindersley Limited, 9 Henrietta Street, Covent Garden, London WC2E 8PS.
Second impression 1985

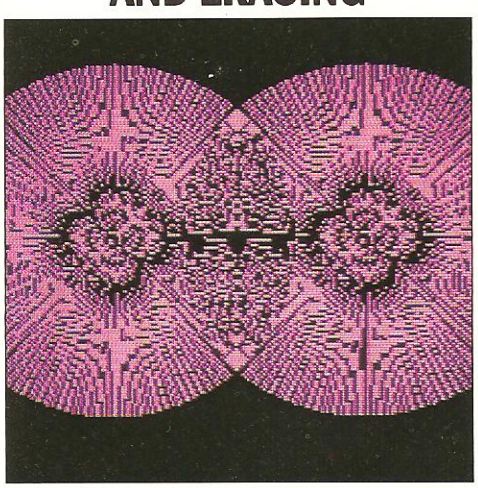
British Library Cataloguing in Publication Data

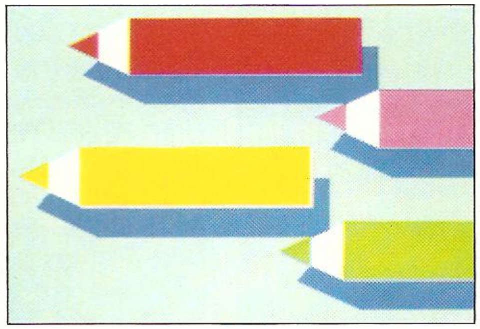
Letcher, Piers
Step-by-step programming
ZX Spectrum and ZX Spectrum+
Graphics.
- (DK screen shot programming series) Bk. 3
1. Sinclair ZX Spectrum (Computer)
- Programming
I. Title
001.64'2 QA76.8.S625

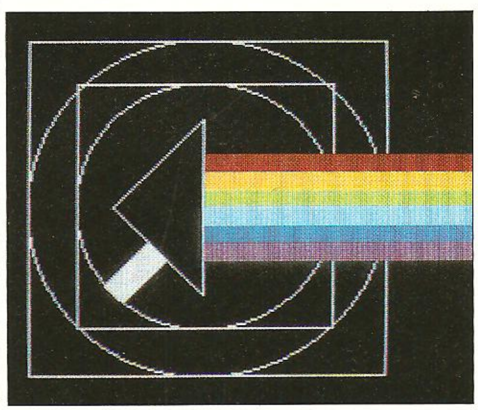
ISBN 0-86318-103-1

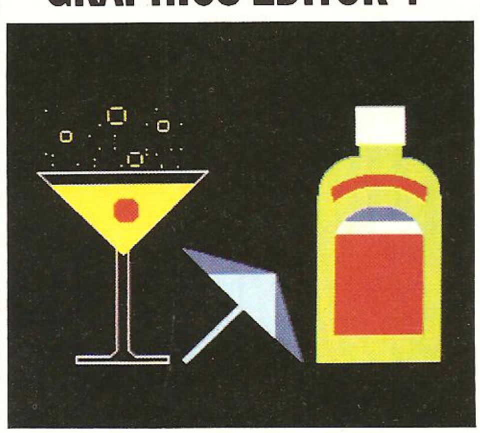
Typesetting by Gedset Limited, Cheltenham, England
Reproduction by Reprocolor Llovet S.A., Barcelona, Spain
Printed and bound in Italy by A. Mondadori, Verona

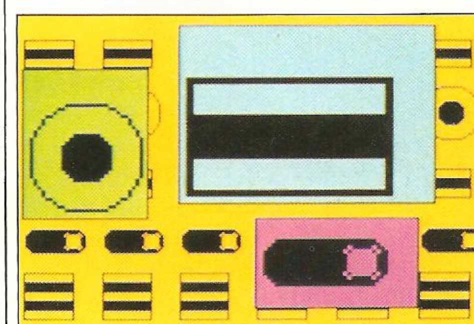












ABOUT THIS BOOK

The Sinclair Spectrum is one of the most popular microcomputers ever produced. One reason for its success has been its remarkable ability to produce graphic displays rivalling those produced by much larger computers designed only ten or fifteen years ago. However, graphics programming in BASIC underutilizes the Spectrum. To produce the kind of displays seen in commercially available games, you need to use machine code as well as BASIC.

What is machine code?

The heart of the Spectrum, the Z80 central processor, cannot understand BASIC. A BASIC program must first be translated into a simpler language that the machine can understand (hence the term "machine code"). This code is in the form of binary 1s and 0s. Before the processor can execute a BASIC program line, all keywords and variables are first converted to machine-code instructions.

BASIC is an example of what is known as an "interpreted", as opposed to a "compiled", language — that is, it is executed by the central processor line by line rather than as a complete program. While an interpreted language is easier to use, it is also slower in execution. By writing programs in machine code, you can miss out the BASIC interpreter altogether. In addition, machine code allows you to utilize many features of your Spectrum which cannot be reached from BASIC, so that you can therefore achieve far more impressive results than would ever be possible from the simpler, but more restricted, BASIC. You can get an idea of how much faster machine code is by seeing the time taken for the programs in this book to run.

Disadvantages of machine code

Given all the advantages of machine code in both speed and flexibility, why not ignore BASIC and use machine code all the time? The answer is simply convenience. Using machine code is time-consuming, difficult and frustrating, and attempting to write your own code is only for the expert. When you see machine-code listings, they are usually in a "disassembled" form, that is, with some of the numbers translated into mnemonics such as LD for LOAD, and JP for JUMP. But a special disassembler program is required simply to give you a machine-code listing in this form, and these mnemonics are themselves far from simple. Using machine code even the simplest operations in BASIC, such as drawing a line on the screen, require many lines of machine code. In addition, machine code has no error-trapping routines such as those in BASIC. If a mistake is made when keying in a BASIC program, the program will not be lost (although the program may refuse to RUN at some point); in machine code, without error-trapping routines, a mistake will probably cause the Spectrum to crash, erasing both the program and its DATA.

The solution

This book combines the advantages of machine code with the convenience and simplicity of BASIC. This is done by giving the machine code in the form of ready-made and tested routines, which you can then use in your BASIC programs. The machine code is shown as DATA statements in BASIC, which means it isn't necessary for you to understand anything about machine code to be able to use the routines. The DATA is given in the form of decimal numbers, rather than in binary or hexadecimal (to base 16), so that the machine code is in the form most convenient for you to read and key in.

The machine-code routines

Here is an example of a machine code routine (the point-plot routine, FNf, from page 17):

ROUTINE LISTING

```
7350 LET b=61500: LET l=60: LET
z=0: RESTORE 7360
7351 FOR i=0 TO l-1: READ a
7352 POKE (b+i),a: LET z=z+a
7353 NEXT i
7354 LET z=INT (((z/l)-INT (z/l)
)*l)
7355 READ a: IF a<>z THEN PRINT
"??": STOP

7360 DATA 42,11,92,1,4
7361 DATA 0,9,86,14,8
7362 DATA 9,94,62,175,147
7363 DATA 216,95,167,31,55
7364 DATA 31,167,31,171,230
7365 DATA 248,171,103,122,7
7366 DATA 7,7,171,230,199
7367 DATA 171,7,7,111,122
7368 DATA 230,7,71,4,62
7369 DATA 254,15,16,253,6
7370 DATA 255,168,71,126,176
7371 DATA 119,201,0,0,0
7372 DATA 24,0,0,0,0
```

Each routine in the book is shown like this, in the form of a BASIC program. The machine code is contained as a series of DATA statements in lines 7360-7372. At the beginning of the routine in lines 7350-7355, there are a few lines of BASIC. This is a loader program; variable b tells the computer where in memory to begin loading the routine, and variable l the number of bytes in the routine. When the loader routine is RUN, this routine is placed in memory from address 61500 onwards, and has a length of 60 bytes.

As shown here, of course, the routine is simply a list of numbers, and has no visible meaning. These numbers are the ready tested and assembled machine code which has then been converted to a sequence of decimal numbers. Each number corresponds to a single instruction or item of DATA required by the routine;

hence, all the numbers have values between 0 and 255, the maximum value of a byte. All you need to know about the routine is what it does and what information it requires so that you can call it correctly from your BASIC program.

All the routines in the book are defined as functions. Each function is individually coded by the letters a to t; a complete list of functions is given on pages 60-61. Demonstration BASIC programs can be found on the same page as the routine, which give an indication of the kind of displays possible using the machine code.

How to use the routines

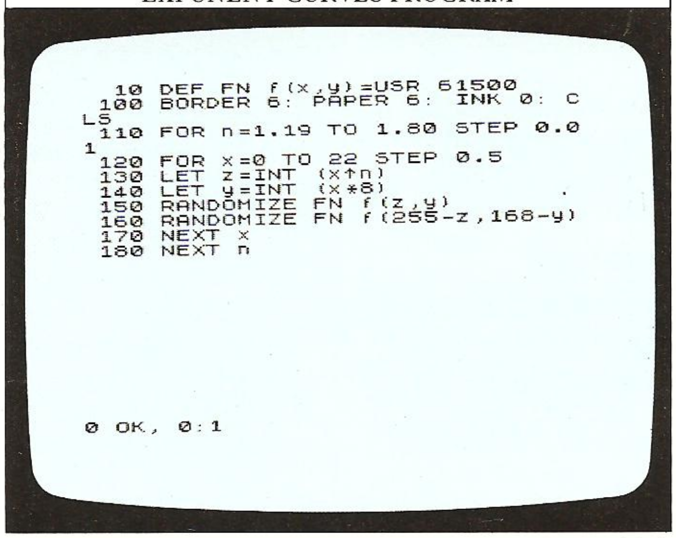
To use any program in this book, simply key in a machine-code routine together with a BASIC program which demonstrates its use. You will find full details of how to do this on pages 8-9. When you RUN the program, you will immediately begin to see the true power of your Spectrum.

As you progress through the book and the range of routines grows, the BASIC programs grow too by calling several routines to produce increasingly complex graphics. By keying in each routine just once, and then SAVEing it onto cassette or Microdrive, you will have a sophisticated graphics capability at your fingertips.

The programs in use

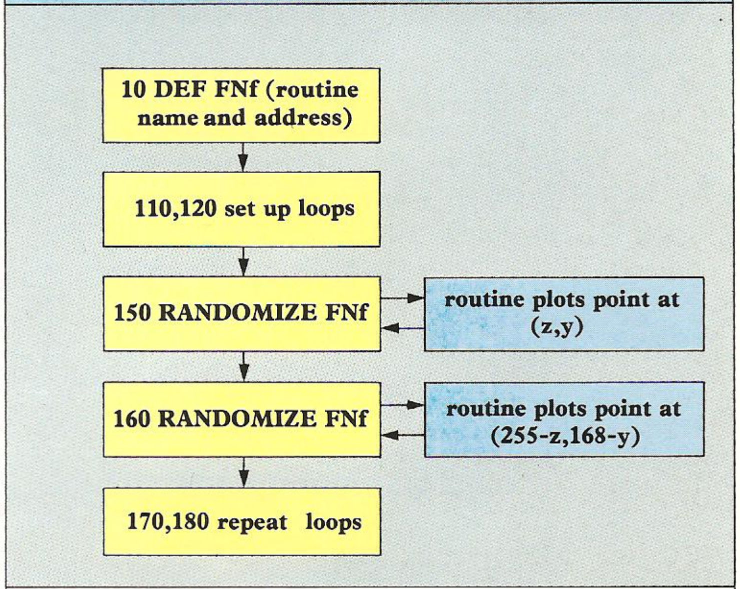
A typical program from this book (the exponent curves program on page 17) contains two details which will be unfamiliar to BASIC programmers who have not used machine code before:

EXPONENT CURVES PROGRAM



First, you will see in line 10 a DEF FN statement, which is used to instruct the computer that a machine-code routine with two parameters (x and y) is located at address 61500 in memory. You will also notice two RANDOMIZE FN commands (lines 150 and 160). These are the calls to the point-plot routine, and the numbers in brackets which follow them are the parameter values to be passed to the machine-code routine (in this case, the co-ordinates of the point to be plotted). When RUNning, the program is carried out by the computer in this way:

POINT-PLOT PROGRAM FLOWCHART



On the left side of this diagram is the main BASIC program, and on the right you can see the machine-code routines, called twice using a RANDOMIZE FN statement. You will see from the diagram that the machine-code is used here very much as a subroutine would be used in BASIC, with variables passed to the routines each time they are called.

What the routines do

Much of this book gives you machine-code versions of the graphics commands you are familiar with in BASIC. You will find that the machine code is often many times faster, and offers you alternative ways of producing graphics which will often be preferable to the BASIC method.

In addition, several routines are included in this book which would simply not be possible in Sinclair BASIC, such as magnification and reduction, and filling in irregular shapes with the current INK colour.

The graphics editor

To make the machine-code routines in this book even easier to use, all the routines contained on pages 10 to 41 have been combined in a single program to form a complete package of routines, which you can use as a graphics editor (pages 42 to 51). No knowledge of BASIC is required to use the graphics editor; even someone with no knowledge of programming, and who has never used a Spectrum before will quickly learn how to produce sophisticated displays using this graphics editor.

USING THE MACHINE CODE

The machine-code routines in this book can easily be incorporated into your BASIC programs without you having to understand the intricacies of how they work. Simply choose a program from this book, and follow the four steps given here.

1: CLEAR memory

As soon as you switch on the Spectrum, type CLEAR 55500. This command resets RAMTOP, the top of the area in memory free for BASIC programs, and ensures that BASIC programs cannot overlap with the machine code (stored in memory from 55500 upwards). Now you can safely use NEW to delete BASIC programs without deleting any of the machine code in memory.

Remember to use CLEAR before loading machine code, since this command erases whatever is in memory above the specified address.

2: Load the machine code

Now type in whatever machine-code routines are required by the BASIC program. After keying in the routine, RUN the short BASIC program which accompanies it: this loads the code into memory. If you keyed in the DATA correctly, you will see an "OK" message on the screen; if not, you will see a couple of question marks. In this case, look again at what you have typed in to trace the mistake.

3: SAVE the routine

When you are sure you have keyed in the routine correctly, SAVE it onto cassette or Microdrive. Always SAVE machine code before using it, to minimize the risk of losing everything you have keyed in. When BASIC errors occur, an error message is usually produced but the program is not lost. Machine-code routines, however, do not generally have error-trapping facilities, and a fault in the code will as often as not cause the Spectrum to crash — deleting everything in memory.

The machine code can be SAVEd in two ways: either in the form of DATA statements like any other BASIC

EXPLANATION OF A MACHINE-CODE BOX

FNi

TRIANGLE DRAW ROUTINE

Start address 60300 **Length** 80 bytes
Other routines called Line draw routine (FNg).
What it does Draws a triangle given the pixel co-ordinates of three points.

Using the routine The routine uses absolute co-ordinates. Specifying off-screen co-ordinates produces an error message; values more than 255 pixels off the screen will probably cause the Spectrum to crash. Colours are set by the current screen INK attributes.

ROUTINE PARAMETERS

DEF FNi(x,y,p,q,r,s)

x,y	specify first corner of triangle (x<256,y<176)
p,q	specify second corner of triangle (p<256,q<176)
r,s	specify third corner of triangle (r<256,s<176)

ROUTINE LISTING

```
7600 LET b=60300: LET l=75: LET
z=0: RESTORE 7610
7601 FOR i=0 TO l-1: READ a
7602 POKE (b+i),a: LET z=z+a
7603 NEXT i
7604 LET z=INT (((z/l)-INT (z/l)
)*l)
7605 READ a: IF a<>z THEN PRINT
"??": STOP

7610 DATA 42,11,92,1,4
7611 DATA 0,9,86,14,8
7612 DATA 9,94,237,83,208
7613 DATA 235,9,86,9,94
7614 DATA 237,83,210,235,9
```

Routine title (a single letter)

Name of routine

Address in memory at which routine is located

Number of bytes in memory taken up by routine

Other machine-code routines which must be present in memory for this routine to work

Purpose of routine

Points to note when using the routine

List of parameters used by the routine, and letters used to describe these parameters

What the parameters do

Maximum and minimum values of parameter to ensure the routine does not plot off-screen points

BASIC loading routine for machine-code DATA

Number of bytes for machine-code (without check digit)

Start address for POKEing DATA

Calculates check digit z

POKEs byte value a into location (b+i)

READs next DATA item, the routine check digit; if this is not the same as z, two question marks are PRINTed to show a mistake has been made

Start of machine-code DATA

listing, or, after you have loaded it into memory, as a block of code. To save machine code, type:

```
SAVE "routine name" CODE start address, length in bytes
```

The start address and length are given at the top of each machine-code box. The diagram on the facing page shows how this information is displayed.

4: LOAD a BASIC program

With the machine-code routine in memory, you can now use it in a BASIC program. DEF FN statements are used to tell the Spectrum the whereabouts of the routine in memory, and what information the routine requires.

Using functions

A machine-code routine can be called simply by specifying its start location, like this:

```
10 RANDOMIZE USR 63000
```

A line like this in a BASIC program, however, is not very informative. It tells you neither what the routine does, nor how many parameters the routine may require when called. This information could be POKEd into the appropriate memory locations — but the consequences of a mistake would be disastrous. Much more reliable is to pass information to the routines using a BASIC function. Functions on the Spectrum are identified by a single letter, and are followed by parameters in brackets. When you define the name and location of the function in your program, you must also specify the parameters, if any, which are to be passed to the routine. For example, the screen clear routine, FNa, requires four parameters:

```
10 DEF FN a(x,y,h,v) = USR 63000
```

Which letters are used after DEF FN is not important; their function is only to tell the computer the number of parameters which will follow the routine call in a BASIC program.

A machine-code function can be called from BASIC in two main ways, both of which require you to combine the keywords FN or USR with a BASIC keyword.

The method used generally in this book is with the keyword RANDOMIZE. Thus,

```
RANDOMIZE FN a(10,10,10,10)
```

would clear a rectangular area of 10x10 characters with the top corner at point 10,10. Note that using RANDOMIZE also resets the random number generator with a new seed; this may cause problems if you are also using a random function in your program.

The second word you can use to call machine code is RESTORE. However, RESTORE also resets the pointer to DATA statements when you use it — which is of course the purpose of the RESTORE statement. If you opt to use RESTORE instead of RANDOMIZE then be especially careful if there are any READ or DATA statements in your program.

QUESTIONS AND ANSWERS

What if I make a mistake in keying in?

Don't panic! Nobody keys in long lists of numbers without making any mistakes. A check routine is included with each machine-code routine to warn you if you made any mistakes in keying in the DATA. This routine compares the DATA you have entered with a check number, which is placed by itself on the last DATA line of each routine.

After the loading program has POKEd the DATA numbers into memory, it looks to see if the check digit is the same as the one currently calculated. If the two numbers are different, the program prints two question marks to show an error has been made. If this happens look through the numbers you have typed in to find the mistake. Having corrected the error, you may still find that the routine fails to load correctly; look to see if you have made more than one error.

Can I start anywhere in the book?

Yes, you can start on any page, but obviously when you key in a program it will not RUN unless the machine-code routine it calls is present in memory. Check before you begin if the program you want to RUN calls any machine-code routines you haven't already keyed in.

Can I use more than one machine-code routine in my programs?

Yes — you can use any combination of routines from this book together. The complete graphics editor program (pages 42-51) provides a convenient way of using machine-code routines together in a single program.

Can I adapt the BASIC programs?

Yes. You can edit the BASIC programs in any way you want to produce different displays, and you will find suggestions for variations throughout the book. One suggestion, though, if you are going to experiment with unusual or off-screen values for the machine-code parameters, is to SAVE what you have keyed in before experimenting. This will prevent you from losing hours of work at the keyboard!

Can I adapt the machine-code routines?

Yes, but at your own risk! Without a good understanding of machine code, it is highly unlikely that you will be able to alter any of the routines successfully. Much more probable is that the Spectrum would crash, with the result that both program and code are wiped from memory.

SCREEN COLOURS 1

The Spectrum CLS command is used to wipe off any ink from the screen, leaving the PAPER and INK attributes for the screen unchanged.

However, there are often occasions when you may want to clear a portion of the screen without disturbing the rest of the display. The partial screen clear routine, FNa, enables you to do this. It clears any rectangular portion of the screen, leaving the PAPER and INK attributes at their current setting. It is used in a program by first defining the function (as in line 10 of the program below), and then calling it with a RANDOMIZE FNa statement (line 160).

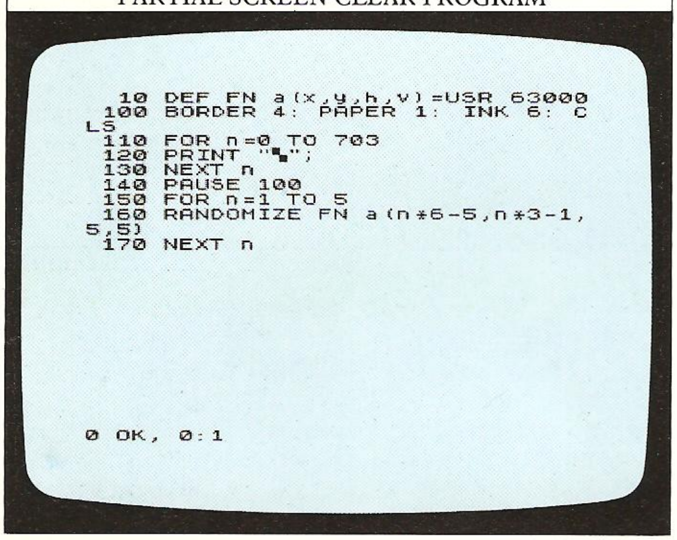
Colours on the Spectrum

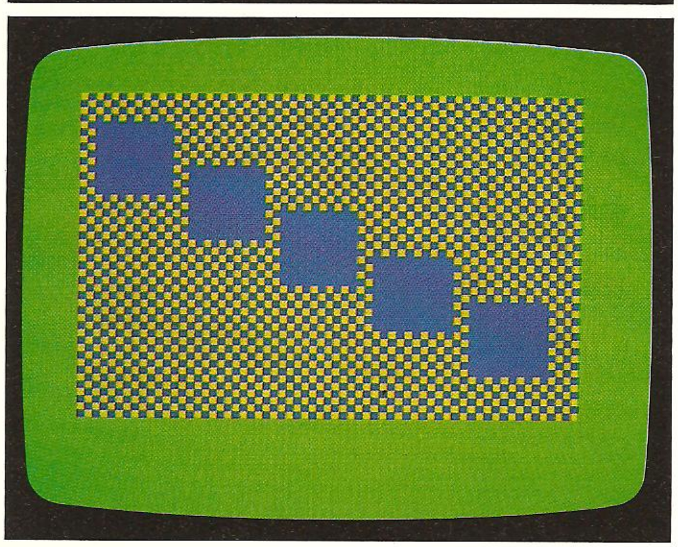
The Spectrum screen colours are set by the familiar INK and PAPER commands. However, although each character square on the Spectrum can have one INK and one PAPER colour, a command such as INK 2 affects the whole screen, even though you may only want the command to affect a small area. How can this be done? It is possible to change the INK attribute of a single character square on the screen by PRINTing spaces in graphics mode and using OVER, or by finding the relevant memory location of the INK attribute, and POKEing the new value, but these would be very slow methods of changing the INK colour on more than a few squares.

Changing colours with machine code

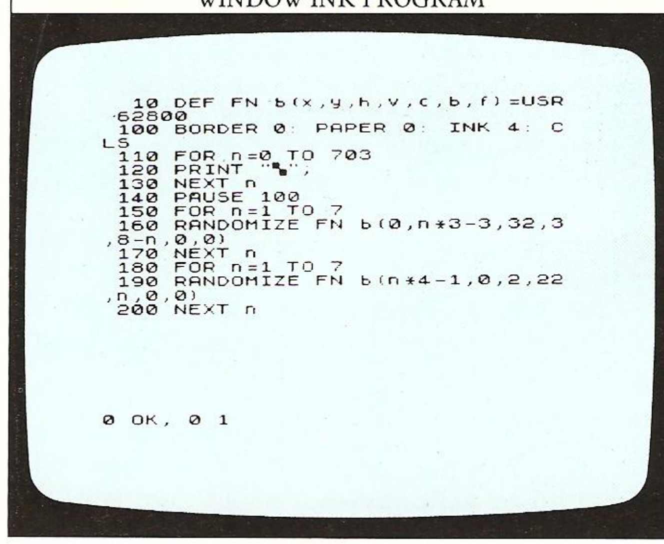
The window ink routine allows you to change the INK colour of any rectangular area of the screen. Whatever you draw on this area of the screen will now be in the INK colour specified by the routine, while outside this area, colours remain in the current Spectrum INK colour. The routine also allows you to set the BRIGHT and FLASH attributes of the area you are specifying.

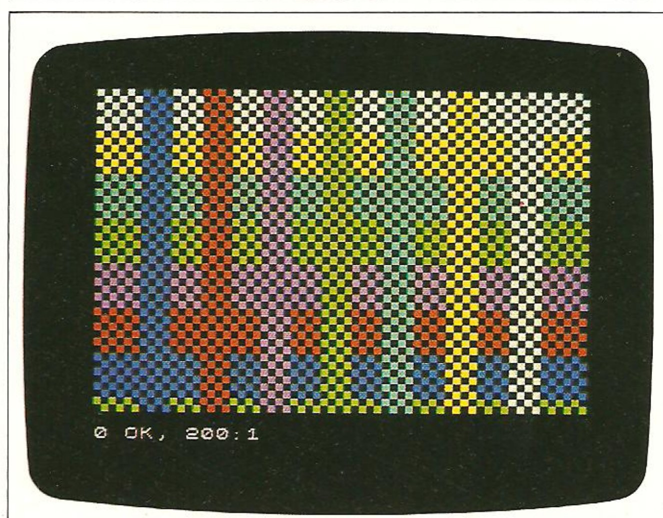
PARTIAL SCREEN CLEAR PROGRAM





WINDOW INK PROGRAM





FNa

PARTIAL SCREEN CLEAR ROUTINE

Start address 63000 **Length** 100 bytes
What it does Similar to BASIC CLS command, but clears only a specified rectangular portion of the screen.

Using the routine Parameter values in this routine represent character positions rather than pixel positions. As with all the machine-code routines in this book, parameter values must be whole numbers.
If the result of adding x and h is greater than 31, or if the sum of y and v is greater than 23, the area to be cleared will run off the screen, and the routine may crash as a result.

ROUTINE PARAMETERS

DEF FNa(x,y,h,v)	
x,y	specify top left-hand corner of area to be cleared (x<32, y<24)
h,v	horizontal and vertical size (x+h<32, y+v<24)

ROUTINE LISTING

```
7000 LET b=63000: LET l=95: LET
z=0
7001 FOR i=0 TO l-1: READ a
7002 POKE (b+i),a: LET z=z+a
7003 NEXT i
7004 LET z=INT (((z/l)-INT (z/l)
)*l)
7005 READ a: IF a<>z THEN PRINT
"??": STOP

7010 DATA 42,11,92,1,4
7011 DATA 0,9,86,1,8
7012 DATA 0,9,94,237,83
7013 DATA 116,246,9,86,9
7014 DATA 94,237,83,118,246
7015 DATA 237,91,116,246,123
7016 DATA 254,23,240,237,83
7017 DATA 116,246,123,230,24
7018 DATA 246,64,103,123,230
7019 DATA 7,183,31,31,31

7020 DATA 31,130,111,58,118
7021 DATA 246,71,197,229,6
7022 DATA 8,197,229,58,119
7023 DATA 246,71,175,119,35
7024 DATA 16,252,225,193,36
7025 DATA 16,240,225,193,62
7026 DATA 32,133,111,48,4
7027 DATA 62,8,132,103,16
7028 DATA 222,201,0,0,0
7029 DATA 82,0,0,0,0
```

While the standard Spectrum screen graphics area is 22 characters deep, with two lines reserved below this for text (lines 23 and 24), both the routines on this page can be used anywhere within the whole screen area, 32 characters wide by 24 deep.

The two programs on the facing page demonstrate the machine-code routines in action. Both programs begin by PRINTing a graphics character over the whole screen (lines 110-130). The partial screen clear program then clears five rectangular areas on the screen. The result is that the INK is deleted in these areas, while the paper colour remains unchanged. The window ink program calls the ink routine in two loops eight times across and down the screen, producing a grid effect.

FNb

WINDOW INK ROUTINE

Start address 62800 **Length** 135 bytes
What it does Sets the INK colour of any specified part of the screen.

Using the routine Be careful that you do not try to set the ink colours of points off the screen. Since parameters h and v are added to x and y respectively, this means that x+h should not be greater than 31, and y+v should not exceed 23. If they do, you may crash the Spectrum and lose the program you were working on.
If, when using the routine, it appears that nothing has happened, then either you have set the INK colour to what it was already, or the area you have altered contained no INK attributes. Try the routine again after printing something in the specified area.
Note that the routine can set the ink colour over the whole Spectrum 32x24 character screen, not just over the normal 32x22 graphics area.

ROUTINE PARAMETERS

DEF FNb(x,y,h,v,c,b,f)	
x,y	specify top left-hand corner of box area (x<32, y<24)
h,v	specify horizontal and vertical sizes of area (x+h<32, y+v<24)
c	specifies ink colour (0<=c<=7)
b	specifies bright (1=bright, 0=off)
f	specifies flash (1=flash, 0=off)

ROUTINE LISTING

```
7050 LET b=62800: LET l=130: LET
 z=0: RESTORE 7060
7051 FOR i=0 TO l-1: READ a
7052 POKE (b+i),a: LET z=z+a
7053 NEXT i
7054 LET z=INT (((z/l)-INT (z/l)
)*l)
7055 READ a: IF a<>z THEN PRINT
"??": STOP

7060 DATA 42,11,92,1,4
7061 DATA 0,9,86,1,8
7062 DATA 0,9,94,237,83
7063 DATA 210,245,9,86,9
7064 DATA 94,237,83,208,245
7065 DATA 9,126,230,7,50
7066 DATA 207,245,9,126,230
7067 DATA 1,40,8,58,207
7068 DATA 245,246,64,50,207
7069 DATA 245,9,126,230,1

7070 DATA 40,8,58,207,245
7071 DATA 246,128,50,207,245
7072 DATA 237,91,210,245,58
7073 DATA 208,245,254,0,200
7074 DATA 237,83,210,245,123
7075 DATA 230,24,203,63,203
7076 DATA 63,203,63,246,88
7077 DATA 103,123,230,7,183
7078 DATA 31,31,31,31,130
7079 DATA 111,58,208,245,71

7080 DATA 197,229,58,209,245
7081 DATA 71,126,230,56,79
7082 DATA 58,207,245,177,119
7083 DATA 35,16,244,225,1
7084 DATA 32,0,9,193,16
7085 DATA 230,201,0,2,5
7086 DATA 53,0,0,0,0
```

SCREEN COLOURS 2

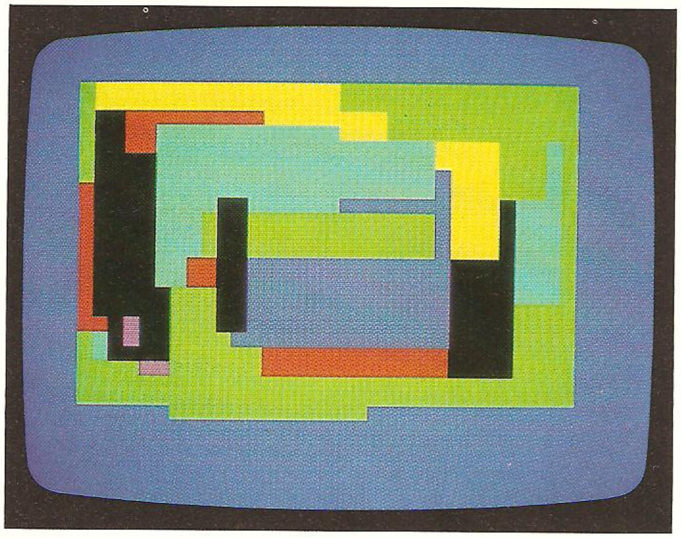
The Spectrum PAPER command sets the background colour of the whole screen. The window paper routine on this page, FNc, allows you to set the paper colour of only a part of the screen, in the same way that you can use the window ink routine to change the INK colours on a part of the screen.

As for the ink routine, the paper routine requires you to specify the top left-hand co-ordinates and height and width of a rectangular area within which the colour is to be changed. Unlike the PAPER command in BASIC, you will see any colour change without having to clear the screen with CLS. Again, like the previous routine, you can use the routine to set the BRIGHT and FLASH attributes of the area. By calling the routine several times you can create a layered effect, with colours apparently superimposed on one another.

A layered effect forms the basis of the random boxes program on this page. Random values are chosen for the start co-ordinates (x1,y1) and horizontal and vertical increments (h1,v1) of the area, and a random colour value is chosen, before the routine is called, inside a loop. Note that the machine-code routine is called using RESTORE rather than RANDOMIZE. Using RANDOMIZE would reset the seed of the random number generator within the loop, so that the same random number sequence would begin again and again.

RANDOM BOXES PROGRAM

```
  10 DEF FN c(x,y,h,v,c,b,f)=USR
 62600
 100 BORDER 1: PAPER 4: CLS
 110 FOR i=1 TO 120
 130 LET x1=INT (RND*17)
 140 LET y1=INT (RND*10)
 150 LET h1=INT (RND*16)
 160 LET v1=INT (RND*15)
 170 LET c1=INT (RND*7)
 180 RESTORE FN c(x1,y1,h1,v1,c1
,0,0)
 190 NEXT i
 200 PAUSE 0

0 OK, 0:1
```

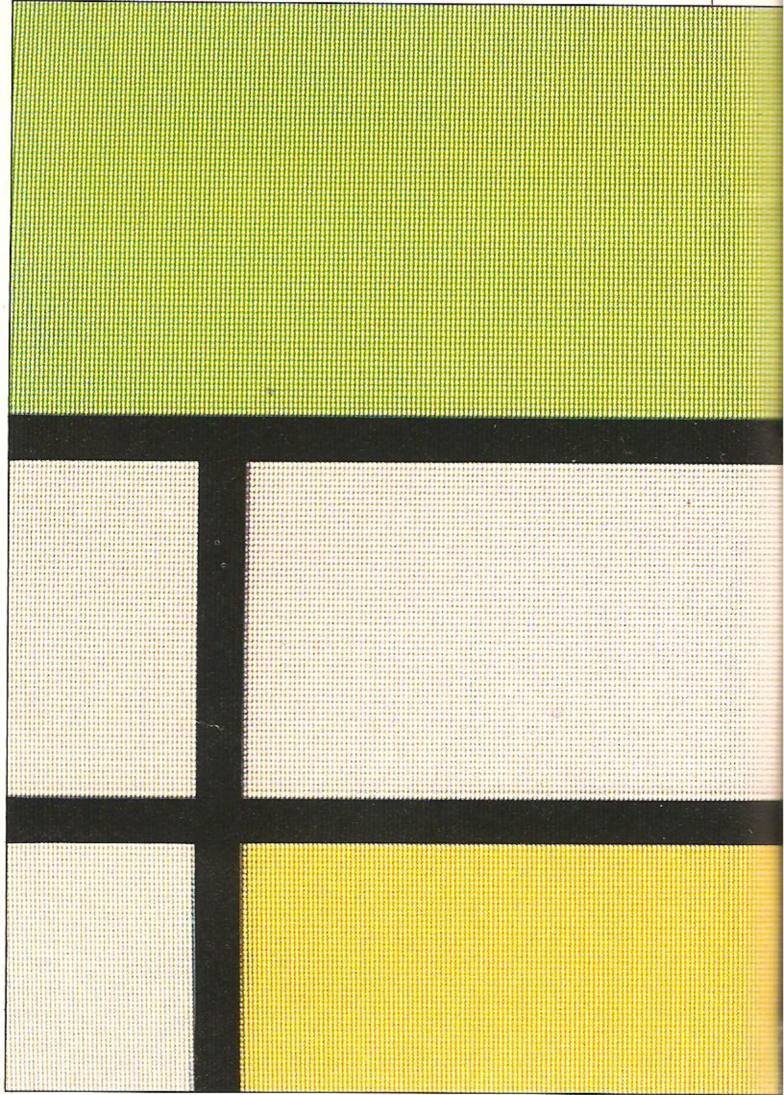


"MONDRIAN" PAINTING PROGRAM

00:01 **second**

How the program works
The window paper routine draws black "lines" (single character-width boxes), and then fills areas of the screen with colour.

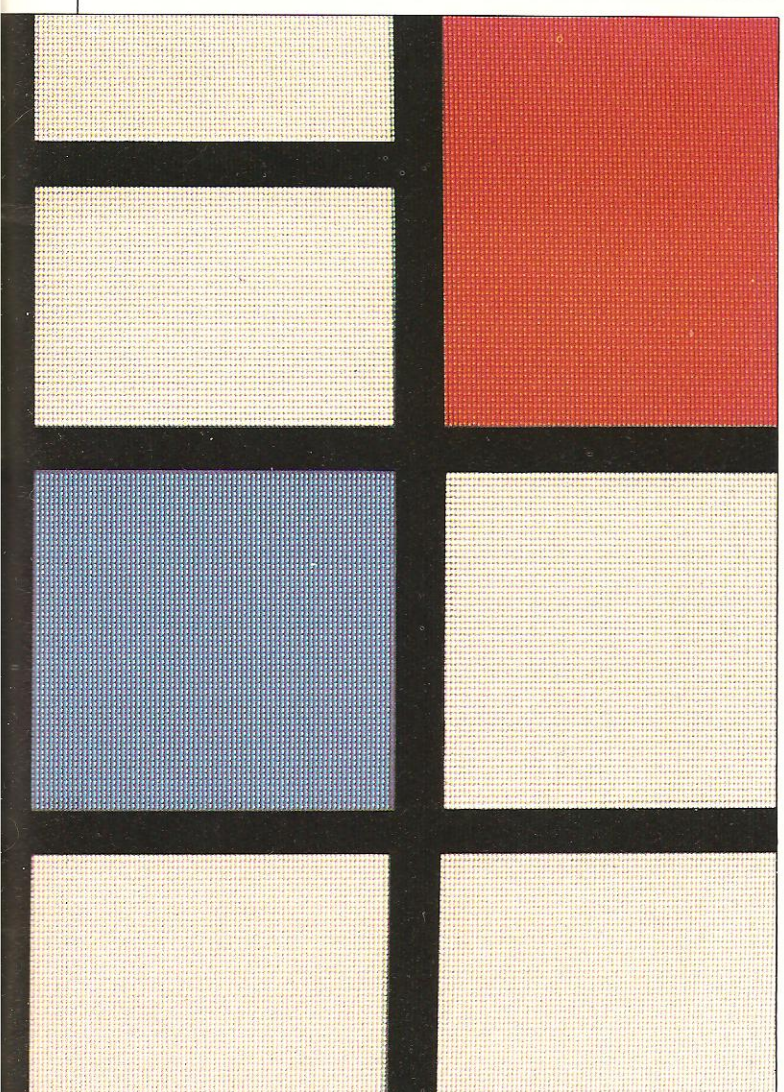
Line 10 defines the window paper routine.
Lines 100-150 draw the black "lines".
Lines 160-190 draw the coloured areas.
Line 190 also stops the program continuing until a key is pressed, so that the bottom two lines of the display are not lost.



The "Mondrian" painting program demonstrates how by using only a single routine, you can produce quite an effective display.

"MONDRIAN" PAINTING PROGRAM

```
  10 DEF FN c(x,y,h,v,c,b,f)=USR
 62600: BORDER 7: PAPER 7: CLS
 100 RANDOMIZE FN c(4,10,1,14,0,
0,0)
 110 RANDOMIZE FN c(16,0,1,24,0,
0,0)
 120 RANDOMIZE FN c(24,0,1,24,0,
0,0)
 130 RANDOMIZE FN c(0,9,32,1,0,0
,0)
 140 RANDOMIZE FN c(0,17,32,1,0,
0,0)
 150 RANDOMIZE FN c(14,3,18,1,0,
0,0)
 160 RANDOMIZE FN c(0,0,16,9,4,1
,0)
 170 RANDOMIZE FN c(5,18,11,6,6,
1,0)
 180 RANDOMIZE FN c(25,0,7,9,2,1
,0)
 190 RANDOMIZE FN c(17,10,7,7,1,
1,0): PAUSE 0

0 OK, 0:1
```



FNc

WINDOW PAPER ROUTINE

Start address 62600 **Length** 150 bytes
What it does Changes the PAPER colour of a specified rectangular area of the screen.

Using the routine The routine works in the same way as the window ink routine, except that here the PAPER attributes are changed within the area specified. As before, it could be dangerous to go beyond the limits set for the parameters, so the sum of x and h should always be less than 32, and y and v together should be less than 24. This is because h and v are relative, not absolute, parameters, which means they are added to x and y respectively to produce the values actually plotted. Thus, if x is 15 and h is 20, then the right-hand edge of the paper area is column 35, which is off the screen.
As before, the routine operates over the whole Spectrum 32x24 character screen, not just over the normal 32x22 graphics area.

ROUTINE PARAMETERS

DEF FNc(x,y,h,v,c,b,f)	
x,y	specify top left-hand corner of box area (x<32, y<24)
h,v	specify bottom right-hand corner of area (x+h<32, y+v<24)
c	specifies paper colour (0<=c<=7)
b	specifies bright (1=bright, 0=off)
f	specifies flash (1=flash, 0=off)

ROUTINE LISTING

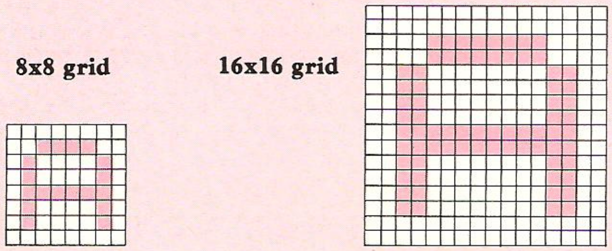
```
7100 LET b=62600: LET l=145: LET
 z=0: RESTORE 7110
7101 FOR i=0 TO l-1: READ a
7102 POKE (b+i),a: LET z=z+a
7103 NEXT i
7104 LET z=INT (((z/l)-INT (z/l)
)*l)
7105 READ a: IF a<>z THEN PRINT
"??": STOP

7110 DATA 42,11,92,1,4
7111 DATA 0,9,86,1,8
7112 DATA 0,9,94,237,83
7113 DATA 22,245,9,86,9
7114 DATA 94,237,83,20,245
7115 DATA 9,126,230,7,203
7116 DATA 39,203,39,203,39
7117 DATA 50,19,245,9,126
7118 DATA 230,1,40,8,58
7119 DATA 19,245,246,64,50

7120 DATA 19,245,9,126,230
7121 DATA 1,40,8,58,19
7122 DATA 245,246,128,50,19
7123 DATA 245,237,91,22,245
7124 DATA 58,20,245,254,0
7125 DATA 200,58,21,245,254
7126 DATA 0,200,237,83,22
7127 DATA 245,123,230,24,203
7128 DATA 63,203,63,203,63
7129 DATA 246,88,103,123,230

7130 DATA 7,183,31,31,31
7131 DATA 31,130,111,58,20
7132 DATA 245,71,197,229,58
7133 DATA 21,245,71,126,230
7134 DATA 7,79,58,19,245
7135 DATA 177,119,35,16,244
7136 DATA 225,1,32,0,9
7137 DATA 193,16,230,201,0
7138 DATA 2,5,0,0,0
7139 DATA 19,0,0,0,0
```

ENLARGED TEXT

Doubling the size of Spectrum characters is quite straightforward in principle. Spectrum characters are drawn on a grid eight pixels by eight; they can be enlarged onto a 16x16 grid by the routine looking at each pixel of the 8x8 grid in turn. If a pixel is filled, then two pixels across and two pixels down are filled on the 16x16 grid. The diagram below gives an example of a character and its enlarged version.

HOW A CHARACTER IS ENLARGED

8x8 grid **16x16 grid**

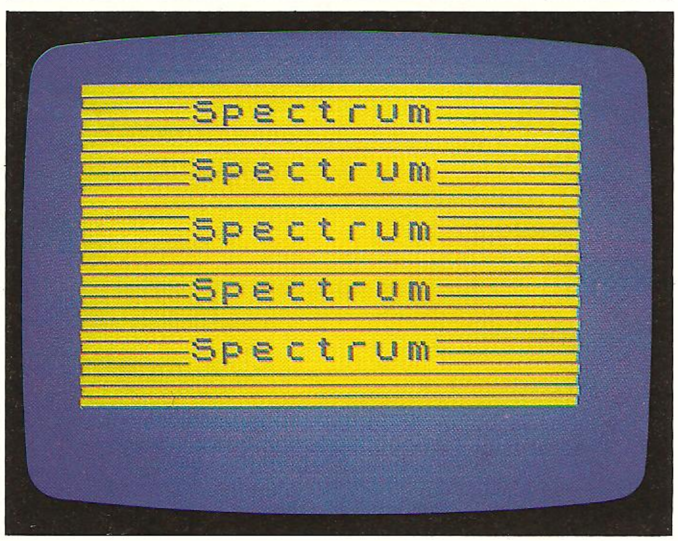


Both routines use this method to enlarge a string of characters (text or graphics) and then print them on the screen at twice their normal size. The horizontal text routine, FNd, prints enlarged characters across the screen; the vertical text routine, FNe, prints the enlarged characters downwards.

The two demonstration programs below show how the routines are used. Both programs begin by defining the word "Spectrum" as the string (n$) to be enlarged by the routine, and both then POKE these characters into memory using a subroutine beginning at line 500. The string must always end with 13, the code for RETURN, to signal to the computer there are no more characters to be enlarged. The horizontal text program prints the string against a background of horizontal lines; the second program displays the vertical string six times, each time with a different coloured background, using the window paper routine.

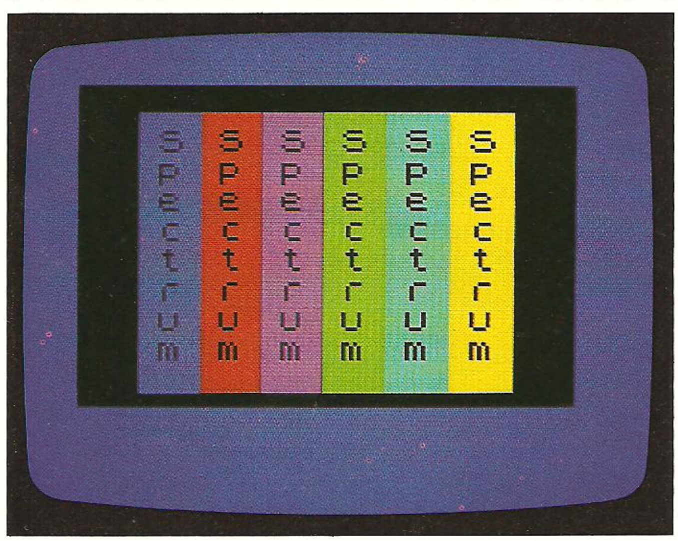
HORIZONTAL TEXT PROGRAM

```
  10 DEF FN d(x,y)=USR 62200
 100 BORDER 1: PAPER 6: INK 1: C
LS
 110 FOR i=0 TO 13
 120 DRAW 255,0: DRAW 0,6
 130 DRAW -255,0: DRAW 0,6
 140 NEXT i
 150 DRAW 255,0
 160 LET n$="Spectrum"
 170 GO SUB 500
 180 FOR m=1 TO 22 STEP 4
 190 RANDOMIZE FN d(7,m)
 200 NEXT m
 499 PAUSE 0
 500 LET l=LEN (n$)
 505 LET k=62499
 510 FOR i=1 TO l
 520 LET n=CODE n$(i)
 530 POKE k+i,n
 540 NEXT i
 550 POKE k+i,13
 560 RETURN

0 OK, 0:1
```



VERTICAL TEXT PROGRAM

```
  10 DEF FN c(x,y,h,v,c,b,f)=USR
 62600
  20 DEF FN e(x,y)=USR 61900
 100 BORDER 1: PAPER 0: INK 0: C
LS
 110 LET n$="Spectrum"
 120 GO SUB 500
 130 LET cl=1
 140 FOR x=5 TO 25 STEP 4
 150 RANDOMIZE FN c(x-1,2,4,19,c
l,0,0)
 160 RANDOMIZE FN e(x,3)
 170 LET cl=cl+1
 180 NEXT x
 499 STOP
 500 LET l=LEN (n$): LET k=62499
 510 FOR i=1 TO l
 520 LET n=CODE n$(i)
 530 POKE k+i,n
 540 NEXT i
 550 POKE k+i,13
 560 RETURN

0 OK, 0:1
```



FNd

ENLARGED HORIZONTAL TEXT ROUTINE

Start address 62200 **Length** 220 bytes
What it does Displays a double-sized version of specified characters horizontally on the screen.

Using the routine Before using this routine, you must first use some BASIC lines to store in memory the text (n$) which you want to display. Lines 500-560 of the programs on the facing page provide an example.

The text is stored as a string in 100 bytes of memory from address 62500 to 62600. The routine continues printing characters from this location onwards until it reaches a RETURN message.

Note that with double-sized characters you are now restricted to 16 characters across the screen; longer strings are continued on the line below. To print a space in the text string, use the graphics blank character (above the 8 key) rather than the space key.

ROUTINE PARAMETERS

DEF FNd(x,y)	
x,y	specify position on screen from which text is to be printed (x<32, y<24)

ROUTINE LISTING

```
7150 LET b=62200: LET l=215: LET
 z=0: RESTORE 7160
7151 FOR i=0 TO l-1: READ a
7152 POKE (b+i),a: LET z=z+a
7153 NEXT i
7154 LET z=INT (((z/l)-INT (z/l)
)*l)
7155 READ a: IF a<>z THEN PRINT
"??": STOP

7160 DATA 42,11,92,1,4
7161 DATA 0,9,86,1,8
7162 DATA 0,9,94,237,83
7163 DATA 240,243,62,99,71
7164 DATA 33,36,244,34,244
7165 DATA 243,197,237,91,240
7166 DATA 243,62,30,186,242
7167 DATA 37,243,22,0,28
7168 DATA 28,237,83,240,243
7169 DATA 62,20,187,250,111

7170 DATA 243,42,244,243,126
7171 DATA 35,34,244,243,254
7172 DATA 31,250,111,243,254
7173 DATA 144,242,111,243,214
7174 DATA 32,1,8,0,42
7175 DATA 54,92,36,9,61
7176 DATA 32,252,34,242,243
7177 DATA 123,230,24,246,64
7178 DATA 103,123,230,7,183
7179 DATA 31,31,31,31,130

7180 DATA 111,34,238,243,205
7181 DATA 113,243,58,241,243
7182 DATA 60,60,50,241,243
7183 DATA 193,16,164,201,193
7184 DATA 201,17,206,243,6
7185 DATA 32,62,0,18,19
7186 DATA 16,252,237,91,242
7187 DATA 243,33,206,243,6
7188 DATA 8,197,26,1,2
7189 DATA 4,197,23,245,203

7190 DATA 22,241,203,22,16
7191 DATA 247,35,193,13,32
7192 DATA 241,43,126,245,43
7193 DATA 126,35,35,119,35
7194 DATA 241,119,35,19,193
7195 DATA 16,220,42,238,243
7196 DATA 17,206,243,14,2
7197 DATA 229,6,8,26,119
7198 DATA 35,19,26,119,19
7199 DATA 43,36,16,245,225
7200 DATA 62,32,133,111,48
7201 DATA 4,62,8,132,103
7202 DATA 13,32,228,201,0
7203 DATA 0,0,0,0,0
```

FNe

ENLARGED VERTICAL TEXT ROUTINE

Start address 61900 **Length** 215 bytes
What it does Displays a double-sized version of specified characters vertically on the screen.

Using the routine This routine works in the same way as the horizontal text routine, but prints text down the screen instead of across it. The same BASIC subroutine is needed to store the text string (lines 500-560 of the demonstration programs on the facing page). Remember to put the string into memory before calling the routine.

Since each character is twice its normal size, only 12 characters down are shown in a column. The routine displays only one vertical line of text, and does not continue a message across to the next column. To display a message longer than 12 characters, call the routine again for each new column of text. To obtain a space in the text use the graphics blank character (above key 8).

ROUTINE PARAMETERS

DEF FNe(x,y)	
x,y	specify position on screen from which text is to be printed (x<32, y<24)

ROUTINE LISTING

```
7250 LET b=61900: LET l=210: LET
 z=0: RESTORE 7260
7251 FOR i=0 TO l-1: READ a
7252 POKE (b+i),a: LET z=z+a
7253 NEXT i
7254 LET z=INT (((z/l)-INT (z/l)
)*l)
7255 READ a: IF a<>z THEN PRINT
"??": STOP

7260 DATA 42,11,92,1,4
7261 DATA 0,9,86,1,8
7262 DATA 0,9,94,237,83
7263 DATA 191,242,62,99,71
7264 DATA 33,36,244,34,195
7265 DATA 242,197,237,91,191
7266 DATA 242,62,30,186,242
7267 DATA 244,241,195,62,242
7268 DATA 62,20,187,250,62
7269 DATA 242,42,195,242,126

7270 DATA 35,34,195,242,254
7271 DATA 31,250,62,242,254
7272 DATA 144,242,62,242,214
7273 DATA 32,1,8,0,42
7274 DATA 54,92,36,9,61
7275 DATA 32,252,34,193,242
7276 DATA 123,230,24,246,64
7277 DATA 103,123,230,7,183
7278 DATA 31,31,31,31,130
7279 DATA 111,34,189,242,205

7280 DATA 64,242,58,191,242
7281 DATA 60,60,50,191,242
7282 DATA 193,16,169,201,193
7283 DATA 201,17,157,242,6
7284 DATA 32,62,0,18,19
7285 DATA 16,252,237,91,193
7286 DATA 242,33,157,242,6
7287 DATA 8,197,26,1,2
7288 DATA 4,197,23,245,203
7289 DATA 22,241,203,22,16

7290 DATA 247,35,193,13,32
7291 DATA 241,43,126,245,43
7292 DATA 126,35,35,119,35
7293 DATA 241,119,35,19,193
7294 DATA 16,220,42,189,242
7295 DATA 17,157,242,14,2
7296 DATA 229,6,8,26,119
7297 DATA 35,19,26,119,19
7298 DATA 43,36,16,245,225
7299 DATA 62,32,133,111,48
7300 DATA 4,62,8,132,103
7301 DATA 13,32,228,201,0
7302 DATA 125,0,0,0,0
```

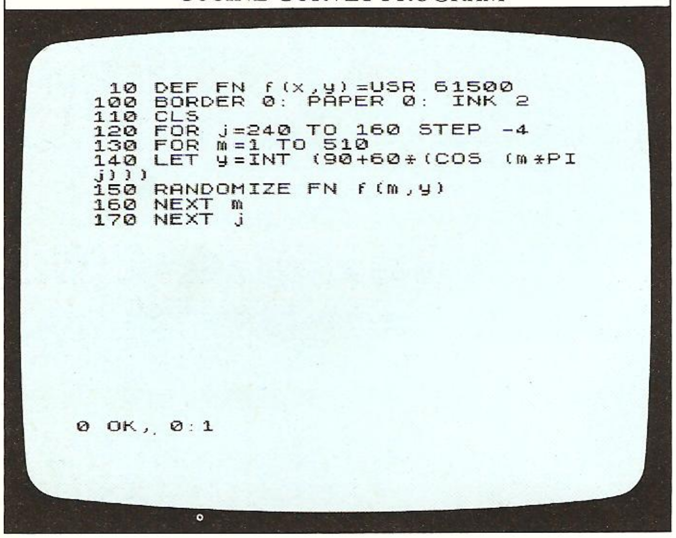
PICTURES WITH POINTS 1

The Spectrum ROM routine which is called by the BASIC command PLOT to draw single points is also used by the BASIC DRAW and CIRCLE commands. The point plot routine given here, FNg, is used in the same way, both to plot points on the screen, and to provide the basis for the other drawing routines in this book, including routines for lines, boxes and circles.

The demonstration programs on this page may seem slower than you would expect. This is not due to the speed of the routine, but because the BASIC program is switching to machine code for each single point and then returning to BASIC. Later drawing routines, which call the point-plot routine from machine code, give a better indication of the routine's true speed. The programs here show only the difference in speed between the BASIC commands RANDOMIZE and PLOT.

The planet program plots random points on horizontal lines which begin and end on the circumference of a circle. There is an increasing probability of a point being plotted towards the right of each line (line 540). The series of exponent curves are produced by varying the horizontal co-ordinate, x.

COSINE CURVES PROGRAM

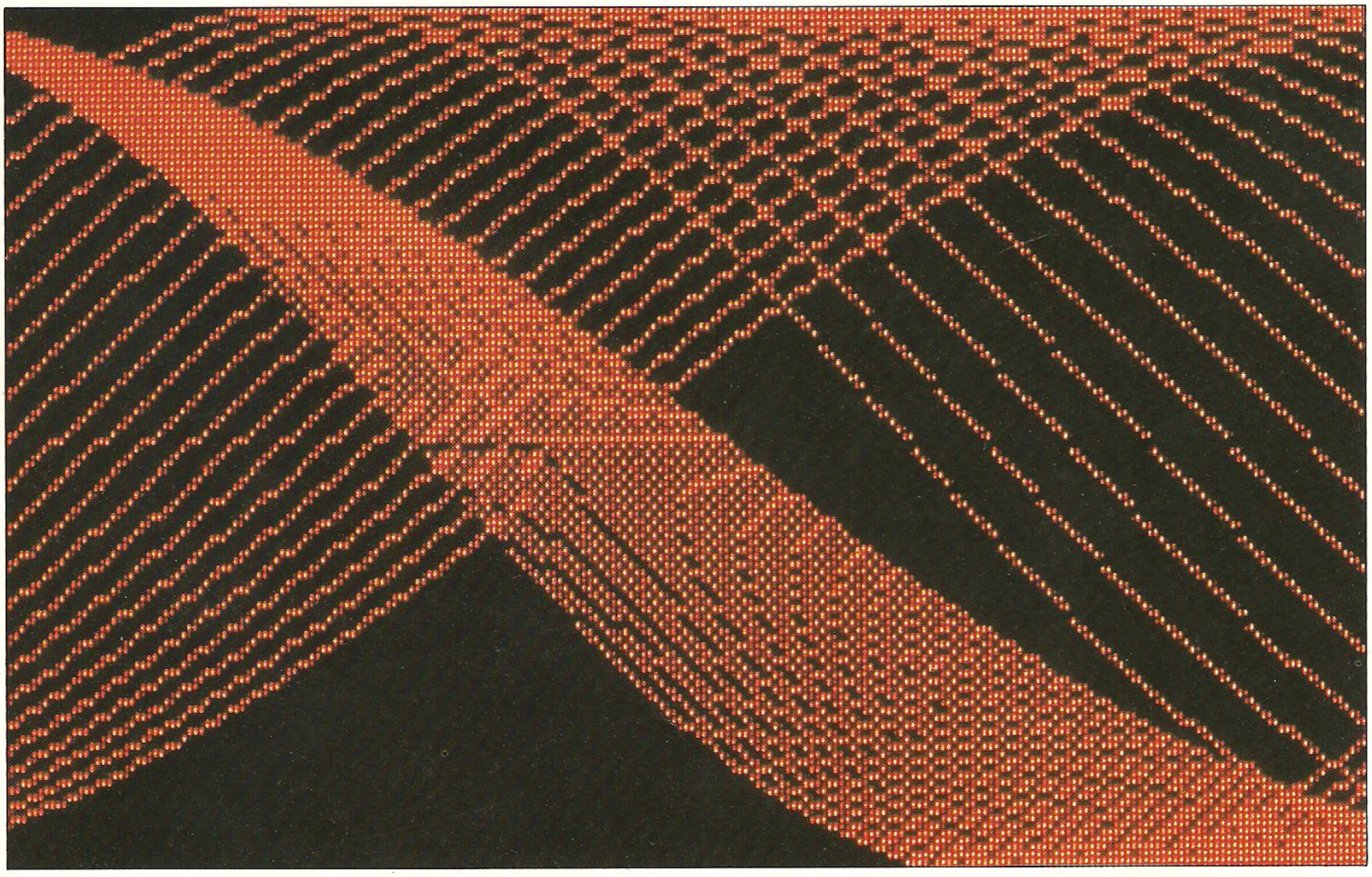


COSINE CURVES PROGRAM

12:30 **minutes**

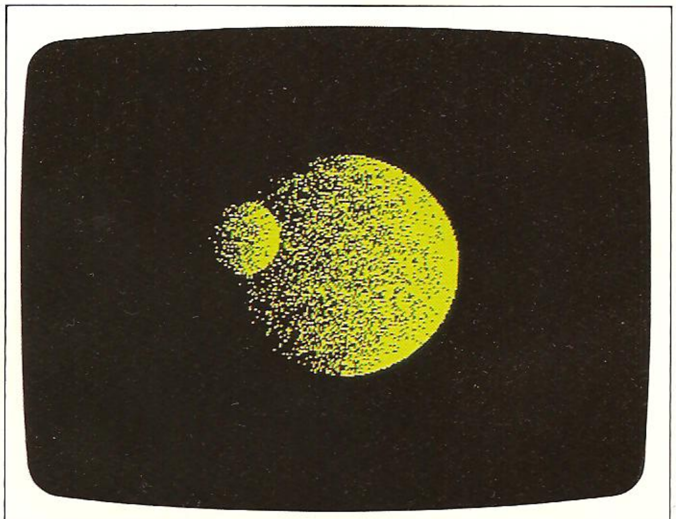
How the program works
Over 10,000 points are plotted in a series of cosine waves.

Line 10 defines the function.
Line 130 sets the horizontal start co-ordinate of each curve.
Line 140 calculates the y co-ordinate (each curve is a slightly different function, since j varies for each curve).
Line 150 plots a point at m,y.



PLANET PROGRAM

```
  10 DEF FN f(x,y)=USR 61500
 100 BORDER 0: PAPER 0: INK 4: C
LS
 110 LET r=60: LET xc=127: LET y.
c=88
 120 GO SUB 500
 130 LET r=20: LET xc=75: LET yc
=100
 140 GO SUB 500
 499 STOP
 500 FOR y=-r TO r
 510 LET x1=INT (SQR (r*r-y*y))
 520 FOR x=-x1 TO x1
 530 LET n=INT (RND*(1)*x1*2)+1
 540 IF n<x1+x THEN RESTORE FN f
(x+xc,y+yc)
 550 NEXT x
 560 NEXT y
 570 RETURN

0 OK, 0:1
```



Line 130 raises x to the power n to determine the point for plotting, z. Line 160 plots the curve again, subtracting co-ordinates from an initial value.

EXPONENT CURVES PROGRAM

```
  10 DEF FN f(x,y)=USR 61500
 100 BORDER 6: PAPER 6: INK 0: C
LS
 110 FOR n=1.19 TO 1.80 STEP 0.0
1
 120 FOR x=0 TO 22 STEP 0.5
 130 LET z=INT (x↑n)
 140 LET y=INT (x*8)
 150 RANDOMIZE FN f(z,y)
 160 RANDOMIZE FN f(255-z,168-y)
 170 NEXT x
 180 NEXT n

0 OK, 0:1
```

FNf

POINT-PLOT ROUTINE

Start address 61500 **Length** 65 bytes
What it does Plots a single point on the screen.

Using the routine The point-plot routine uses pixel rather than character co-ordinates. Pixel co-ordinates are calculated from the bottom left-hand corner of the screen, unlike character co-ordinates which start on the Spectrum from the top left-hand corner. Thus, points are calculated on the screen from 0 to 175 vertically upwards, and from 0 to 255 horizontally: point (255,175) is the top right-hand corner of the screen, for example. Note that routines in this book which use pixel points will not go over the text area of the screen (the bottom two lines of the screen) since the point (0,0) is actually above these two lines.

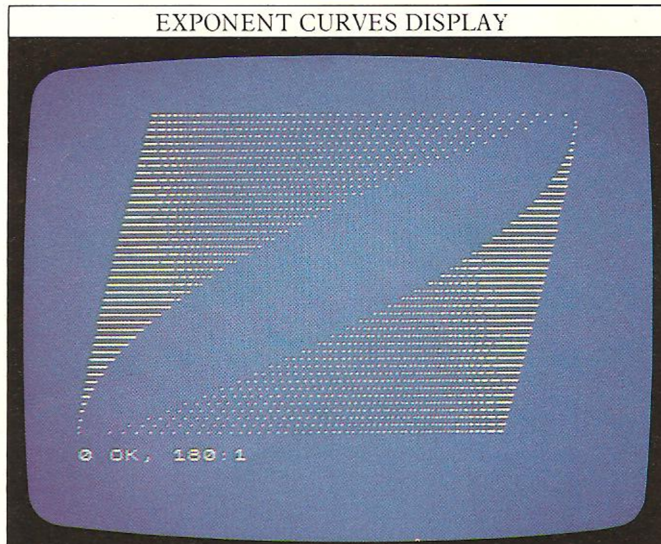
ROUTINE PARAMETERS

DEF FNf(x,y)	
x,y	specify pixel position at which point is to be plotted (x<256, y<176)

ROUTINE LISTING

```
7350 LET b=61500: LET l=60: LET
z=0: RESTORE 7360
7351 FOR i=0 TO l-1: READ a
7352 POKE (b+i),a: LET z=z+a
7353 NEXT i
7354 LET z=INT (((z/l)-INT (z/l)
)*l)
7355 READ a: IF a<>z THEN PRINT
"??": STOP

7360 DATA 42,11,92,1,4
7361 DATA 0,9,86,14,8
7362 DATA 9,94,62,175,147
7363 DATA 216,95,167,31,55
7364 DATA 31,167,31,171,230
7365 DATA 248,171,103,122,7
7366 DATA 7,7,171,230,199
7367 DATA 171,7,7,111,122
7368 DATA 230,7,71,4,62
7369 DATA 254,15,16,253,6
7370 DATA 255,168,71,126,176
7371 DATA 119,201,0,0,0
7372 DATA 24,0,0,0,0
```

EXPONENT CURVES DISPLAY



PICTURES WITH POINTS 2

The displays on the previous page used only a simple BASIC listing and a single routine, the point-plot routine. There is no reason why you should not combine routines together to produce far more complex displays, as demonstrated here.

The cityscape program

The large display on this page is produced by a single program, the cityscape program. The program combines plotted points with three other routines to produce the display.

A total of four routines is used in this program. The skyscrapers are drawn by the window paper routine, FNb; the vertical text routine (FNe) is used to draw the word SINCLAIR, printed in blue by the window ink routine.

The effect of a crowded group of skyscrapers is achieved by drawing coloured windows at random. The effect of random heights but a constant base line is achieved by making all the windows end on the bottom line of the screen. This is done by subtracting the start y co-ordinate (y1) from 25, the total number of vertical text characters on the screen plus one.

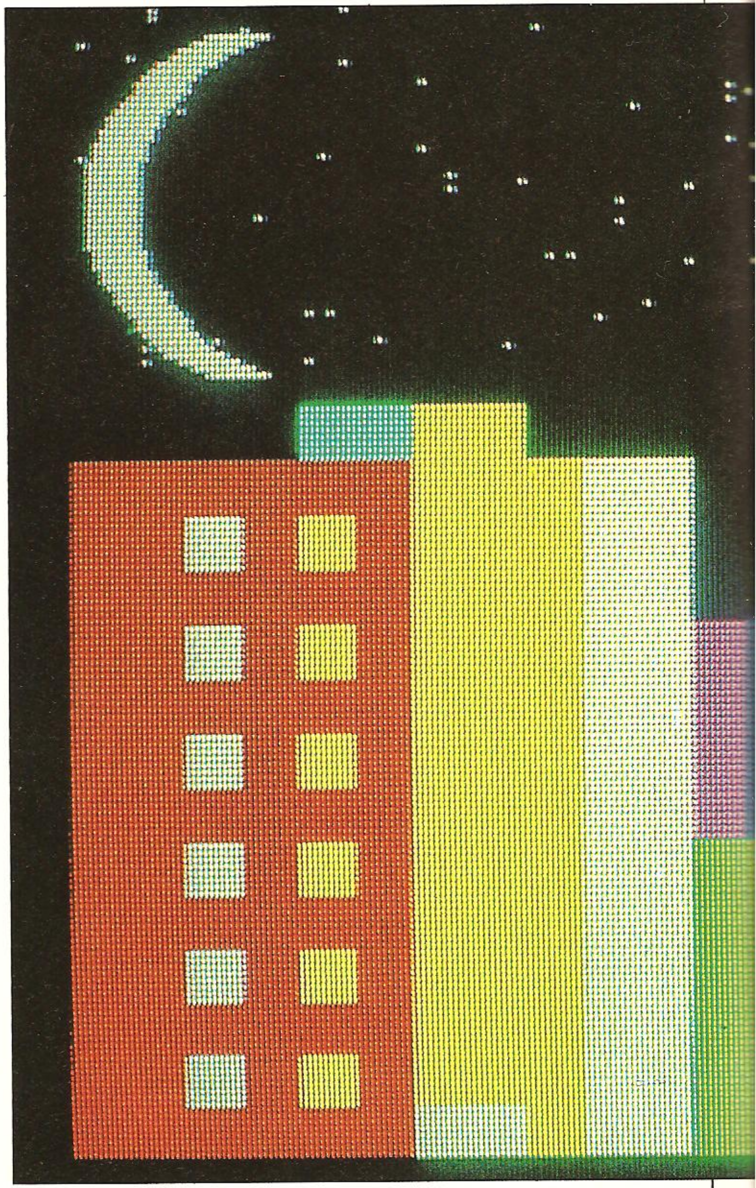
The vertical text routine, which is used to print the word "SINCLAIR", must have the letters which are to be displayed placed in memory before the routine is called. Lines 110-170 take the characters from the word one at a time and POKE them into memory ready to be used by the routine. As before, the final character entered is 13, the ASCII code for carriage return. You will remember that the routine requires the co-ordinates of the top left-hand character (x,y) as well as the stored text string in order to print the text. The window ink routine is used in line 360 to give a blue colour to the area over which the text is to be printed.

CITYSCAPE PROGRAM

```
  10 DEF FN b(x,y,h,v,c,b,f)=USR
62800
  20 DEF FN e(x,y)=USR 61900
  30 DEF FN c(x,y,h,v,c,b,f)=USR
62600
  40 DEF FN f(x,y)=USR 61500
 100 PAPER 0: INK 7: BORDER 0
 110 CLS : LET n$="SINCLAIR"
 120 LET l=LEN n$: LET k=62499
 130 FOR i=1 TO l
 140 LET n=CODE n$(i)
 150 POKE k+i,n
 160 NEXT i
 170 POKE i+k,13
 180 FOR i=0 TO 298 STEP 2
 190 LET x1=(INT (255*RND))
 200 LET y1=174-(INT (74*RND))
 210 RESTORE FN f(x1,y1)
 220 NEXT i
 230 RANDOMIZE FN c(0,14,25,15,1
,0,0)
 240 FOR i =1 TO 50

scroll?
```

Line 210 calls the point-plot routine to plot the stars. Random co-ordinates are chosen in lines 190 and 200 for each star. The moon is drawn in lines 530-560 by using semicircles nested inside each other, using the Spectrum BASIC DRAW command. A later routine in this book will enable you to produce circles using machine code. Finally, the meteor is drawn as a series of straight lines (lines 490-510), again using BASIC.

You will notice from the listing for this program that a convention has been used for all the listings in this book. Lines numbered from 10 to 90 are used for the function definitions, while lines 100 onwards are used for the main listing. You can thus see clearly which machine-code routines have been used for each program, as they are placed at the beginning of the listing.



CITYSCAPE PROGRAM CONTD.

```
 250 RANDOMIZE
 260 LET h1=2+INT (RND*4)
 270 LET y1=10+INT (RND*15)
 280 LET v1=25-y1
 290 LET x1=INT (RND*27)
 300 LET c1=2+INT (RND*6)
 310 RANDOMIZE FN c(x1;y1,h1,v1,
c1,0,0)
 320 NEXT i
 330 RANDOMIZE FN c(16,9,1,15,4,
1,0)
 340 RANDOMIZE FN c(17,6,2,18,4,
1,1)
 350 RANDOMIZE FN c(19,10,1,14,4
,1,0)
 360 RANDOMIZE FN b(17,6,2,18,1,
0,1)
 370 RANDOMIZE FN e(17,6)
 380 RANDOMIZE FN c(1,11,5,14,2,
0,0)
 390 FOR i=12 TO 22 STEP 2
 400 RANDOMIZE FN c(2,i,1,1,7,1,

scroll?
```

CITYSCAPE PROGRAM CONTD.

```
0)
 410 RANDOMIZE FN c(4,i,1,1,6,1,
0)
 420 NEXT i
 430 RANDOMIZE FN c(23,18,9,7,3,
0,0)
 440 FOR i=24 TO 30 STEP 2
 450 RANDOMIZE FN c(i,19,1,1,4,1
,0)
 460 RANDOMIZE FN c(i,21,1,1,4,1
,0)
 470 NEXT i
 480 INK 6: BRIGHT 1
 490 FOR i=-6 TO 6 STEP 2
 500 PLOT 250,165-i
 510 DRAW -70,i-40: NEXT i
 520 INK 7
 530 FOR i=0 TO 7
 540 PLOT 20+i,150
 550 DRAW 0,-50,0.8*PI
 560 NEXT i
 570 PAUSE 0

0 OK, 0:1
```

CITYSCAPE PROGRAM

00:15 **seconds**

This program calls four routines, all of which must be present in memory before RUNning the program:

- window ink routine (FNb) page 11
- window paper routine (FNc) page 13
- enlarged vertical text routine (FNe) page 15
- point-plot routine (FNf) page 17

How the program works

Lines 10-40 define the routines.

Line 110 defines the text string.

Line 120 sets up a loop to POKE characters into memory.

Line 130 POKEs a single character into memory.

Line 170 POKEs ASCII code 13 into memory.

Lines 180-220 print stars at random points in the top 74 pixel rows of the screen.

Lines 240-320 set up values for the window paper boxes and draw them — a total of 50 boxes.

Lines 330-380 draw the "buildings" with windows which appear in front of the paper boxes (lines 340 and 360 print flashing boxes).

Lines 480-510 draw the comet.

Lines 520-570 draw the moon (a series of semicircles).

LINE GRAPHICS 1

Lines are drawn in BASIC on the Spectrum using the command DRAW. This command uses relative co-ordinates, that is, the command is followed by co-ordinates which specify the distance from the current plot position. A line is then drawn from this point to the specified point. It isn't always as simple as it may seem to calculate this horizontal and vertical increment from the current plot position.

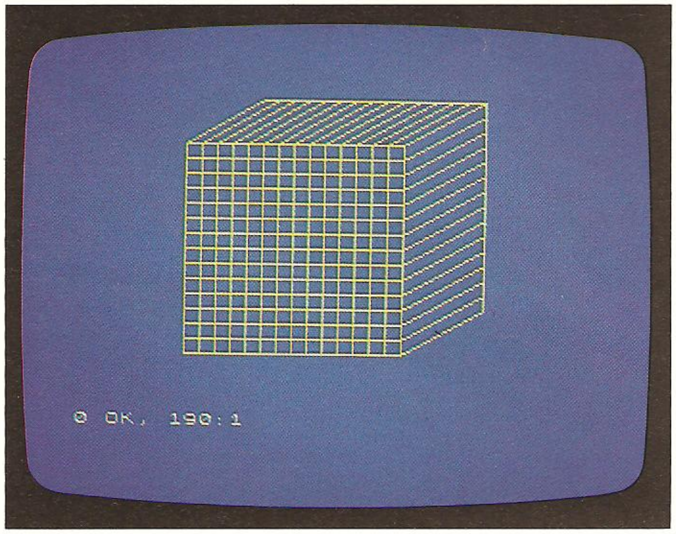
The line-draw routine

The routine on this page, FNg, offers an alternative to Spectrum DRAW for drawing straight lines. This routine is faster than the DRAW command, and uses absolute, rather than relative co-ordinates. Thus, you no longer have to worry about calculating distances from the current plot position.

The line-draw routine contains some error-trapping to prevent the Spectrum crashing if the routine is called

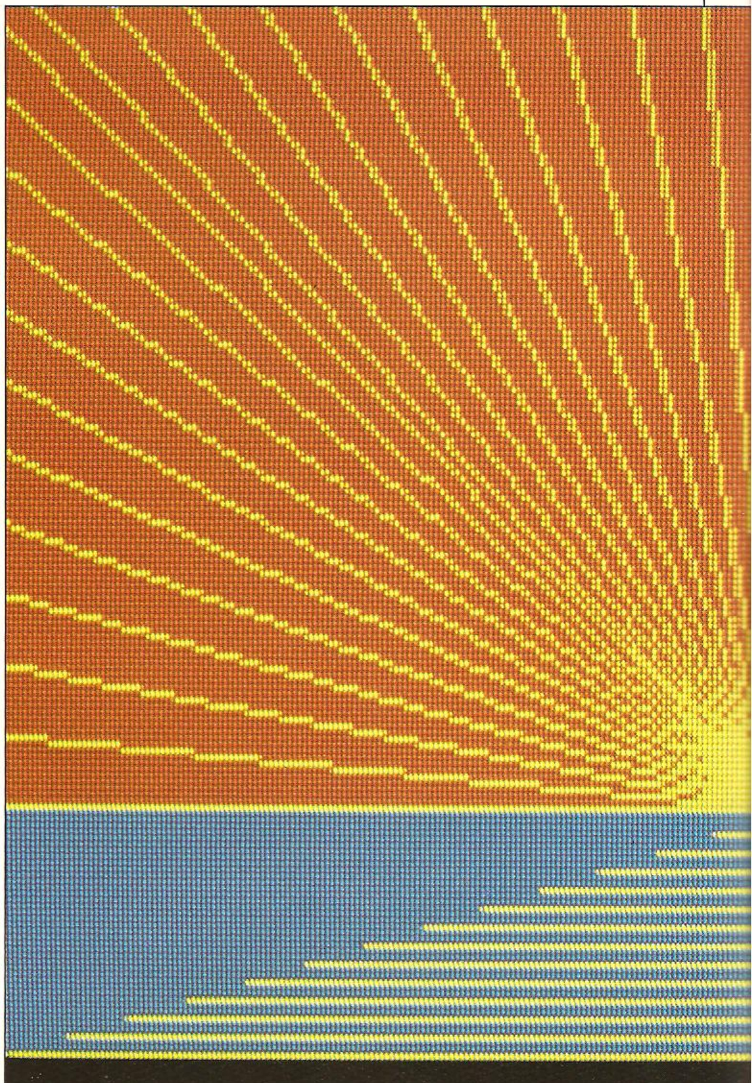
CUBE PROGRAM

```
  10 DEF FN g(x,y,p,q)=USR 60700
 100 BORDER 1: PAPER 1: INK 6
 110 CLS
 120 FOR j=0 TO 116 STEP 8
 130 RANDOMIZE FN g(56,24+j,168,
24+j)
 140 RANDOMIZE FN g(56+j,24,56+j
,136)
 150 RANDOMIZE FN g(56+j,136,96+
j,160)
 160 RANDOMIZE FN g(168,24+j,208
,48+j)
 170 NEXT j
 180 RANDOMIZE FN g(96,160,208,1
60)
 190 RANDOMIZE FN g(208,160,208,
48)

0 OK, 0:1
```



SUNSET PROGRAM

```
  10 DEF FN c(x,y,h,v,c,b,f)=USR
 62600
  20 DEF FN g(x,y,p,q)=USR 60700
 100 BORDER 0: PAPER 1: INK 6: C
LS
 110 RANDOMIZE FN c(0,0,32,17,2,
0,0)
 120 FOR j=40 TO 174 STEP 12
 130 RANDOMIZE FN g(0,j,128,40)
 140 RANDOMIZE FN g(255,j,128,40
)
 150 NEXT j
 160 FOR j=6 TO 255 STEP 12
 170 RANDOMIZE FN g(j,175,128,40
)
 180 NEXT j
 190 FOR j=36 TO 0 STEP -3
 200 RANDOMIZE FN g((10+j*3),j,(
248-j*3),j)
 210 NEXT j

0 OK, 0:1
```



with off-screen co-ordinates. This makes it easier to devise complex graphics displays using a trial and error method, since there is less danger of losing both program and routine if off-screen co-ordinates are entered.

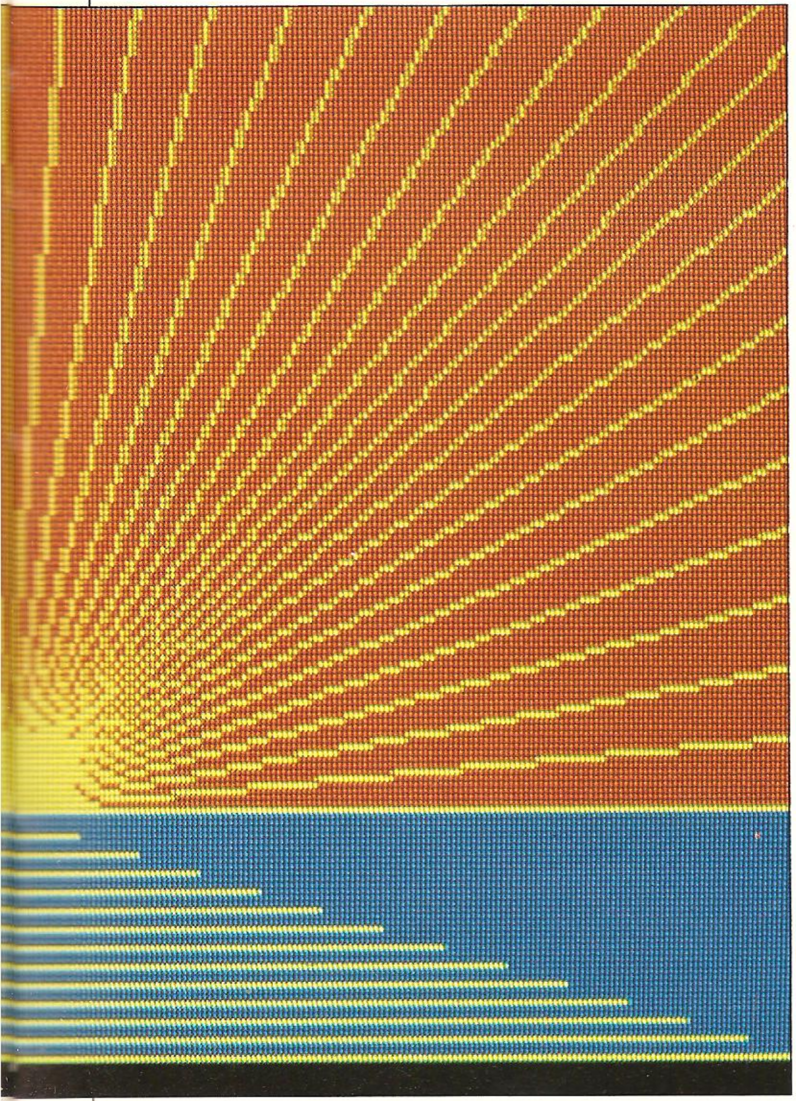
The cube display on page 20 is formed from a series of lines. The program is very simple. The line draw routine draws four lines repeatedly in the loop from lines 120 to 170: two to draw the grid pattern, and two for the perspective effect. Lines 180 and 190 specify the two lines which complete the cube shape.

SUNSET PROGRAM

00:03 seconds

How the program works
Yellow lines are drawn radiating from a point at the centre of the screen to points on the edge. Horizontal lines are then added to create a reflection effect.

Line 110 sets the blue colour in the bottom part of the screen using the window paper routine.
Lines 130 and 140 draw the horizon.
Lines 160-180 draw the yellow lines in the sky.
Lines 190-210 draw the horizontal lines on the lower half of the screen.



FNg

LINE-DRAW ROUTINE

Start address 60700 **Length** 205 bytes
What it does Draws a line between two specified points.

Using the routine This routine draws a line joining any two pixel points on the screen. Although the Spectrum already has a line draw routine available in BASIC, the version given here is much faster, and uses absolute rather than relative co-ordinates. Ths means that co-ordinates p,q represent the position of the end point of the line, not the horizontal and vertical increment from x,y.
The routine will usually work if off-screen points are specified, but for safety some error-trapping has been incorporated. If you attempt to plot lines off the screen, you will see an "Integer out of range" message, unless the value you have entered is more than 255 pixels off the screen; in this case the routine will probably crash.

ROUTINE PARAMETERS	
DEF FNg(x,y,p,q)	
x,y	specify start position of line (x<256,y<176)
p,q	specify end position of line (p<256,q<176)

ROUTINE LISTING

```
7400 LET b=60700: LET l=210: LET
 z=0: RESTORE 7410
7401 FOR i=0 TO l-1: READ a
7402 POKE (b+i),a: LET z=z+a
7403 NEXT i
7404 LET z=INT (((z/l)-INT (z/l)
)*l)
7405 READ a: IF a<>z THEN PRINT
"??": STOP

7410 DATA 42,11,92,1,4
7411 DATA 0,9,86,14,8
7412 DATA 9,94,237,83,26
7413 DATA 237,205,226,237,94
7414 DATA 42,26,237,217,229
7415 DATA 217,237,115,175,237
7416 DATA 1,1,1,122,148
7417 DATA 210,70,237,6,255
7418 DATA 237,68,87,123,149
7419 DATA 210,80,237,14,255

7420 DATA 237,68,95,122,187
7421 DATA 48,10,106,237,67
7422 DATA 177,237,175,71,195
7423 DATA 107,237,178,202,167
7424 DATA 237,107,90,237,67
7425 DATA 177,237,14,0,99
7426 DATA 123,31,133,218,118
7427 DATA 237,188,218,128,237
7428 DATA 148,87,217,237,91
7429 DATA 177,237,195,132,237

7430 DATA 87,197,217,209,42
7431 DATA 26,237,123,133,95
7432 DATA 122,60,132,218,164
7433 DATA 237,202,167,237,61
7434 DATA 87,237,83,26,237
7435 DATA 205,179,237,217,122
7436 DATA 29,32,205,195,167
7437 DATA 237,202,147,237,237
7438 DATA 123,175,237,217,225
7439 DATA 217,201,181,214,1

7440 DATA 1,62,175,147,218
7441 DATA 249,36,95,167,31
7442 DATA 55,31,167,31,171
7443 DATA 230,248,171,103,122
7444 DATA 7,7,7,171,230
7445 DATA 199,171,7,7,111
7446 DATA 122,230,7,71,4
7447 DATA 62,254,15,16,253
7448 DATA 6,255,168,71,126
7449 DATA 176,119,201,229,197
7450 DATA 205,179,237,193,225
7451 DATA 9,86,9,201,0
7452 DATA 192,0,0,0,0
```

LINE GRAPHICS 2

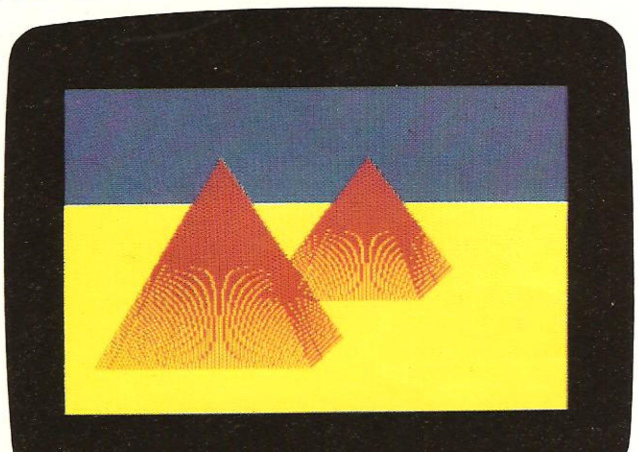
Line-drawing routines are ideal for producing interference patterns. These are produced when a series of lines or points are drawn so close together that what is produced is neither separate lines nor a complete solid, but a pattern.

The pyramid program below shows interference patterns at work. Each pyramid is drawn by a subroutine beginning at line 500, which draws lines from the top of the pyramid (the fixed point tx,ty) to points along a horizontal base line (by). Only the base x co-ordinate (x) is varied within the loop. Interference patterns are seen from near the top of the pyramid (where the lines no longer have the appearance of a solid figure) to a point towards the base of the pyramid (where lines are beginning to be seen distinctly).

The line pattern program demonstrates a related phenomenon. Here the line-draw routine (called in lines 140-170) has the paradoxical effect of producing

PYRAMID PROGRAM

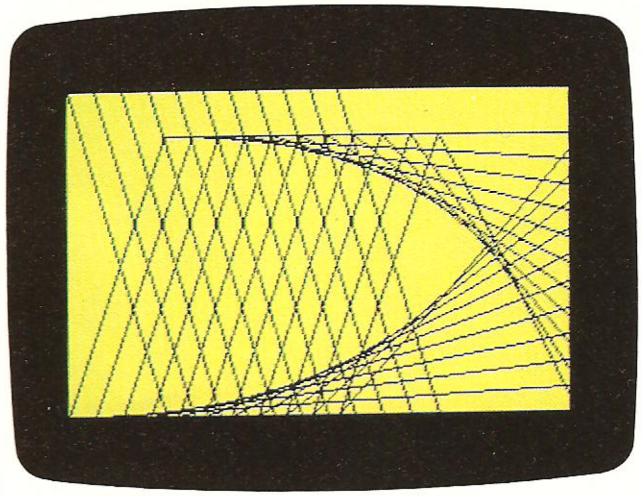
```
  10 DEF FN b(x,y,h,v,c,b,f)=USR
 62800
  20 DEF FN c(x,y,h,v,c,b,f)=USR
 62600
  30 DEF FN g(x,y,p,q)=USR 60700
 100 BORDER 0: PAPER 1: INK 2
 110 CLS
 120 RANDOMIZE FN c(0,8,31,14,6,
0,0)
 130 LET tx=80: LET ty=136: LET
by=24: LET a=16: LET b=128
 140 GO SUB 500
 150 LET tx=156: LET by=60: LET
a=102: LET b=182
 160 GO SUB 500: STOP
 500 FOR x=a TO b STEP 2
 510 RANDOMIZE FN g(tx,ty,x,by)
 520 NEXT x
 530 FOR x=b TO b+16
 540 RANDOMIZE FN g(tx,ty,x,by)
 550 LET by=by+1
 560 NEXT x: RETURN

0 OK, 0:1
```

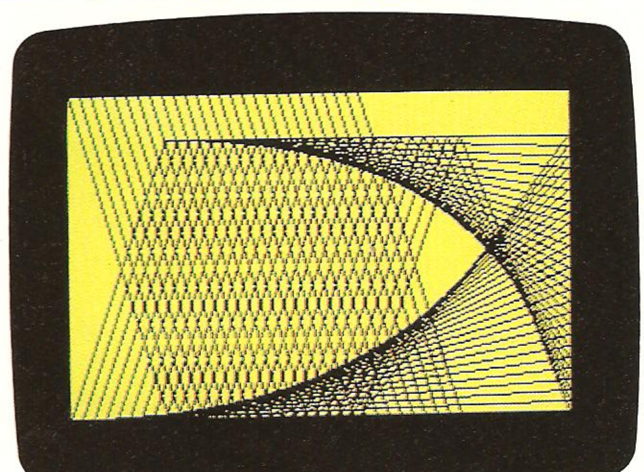


LINE PATTERN PROGRAM

```
  10 DEF FN g(x,y,p,q)=USR 60700
 100 BORDER 0: PAPER 6: INK 0
 110 CLS
 120 FOR i=14 TO 2 STEP -1
 130 FOR j=0 TO 150 STEP i
 140 RANDOMIZE FN g(50+j,0,j,175
)
 150 RANDOMIZE FN g(255,j,j,0)
 160 RANDOMIZE FN g(50+j,150,255
,150-j)
 170 RANDOMIZE FN g(50+j,150,j,0
)
 180 NEXT j
 190 PAUSE 0: CLS
 200 NEXT i

0 OK, 0:1
```

LINE PATTERN DISPLAYS

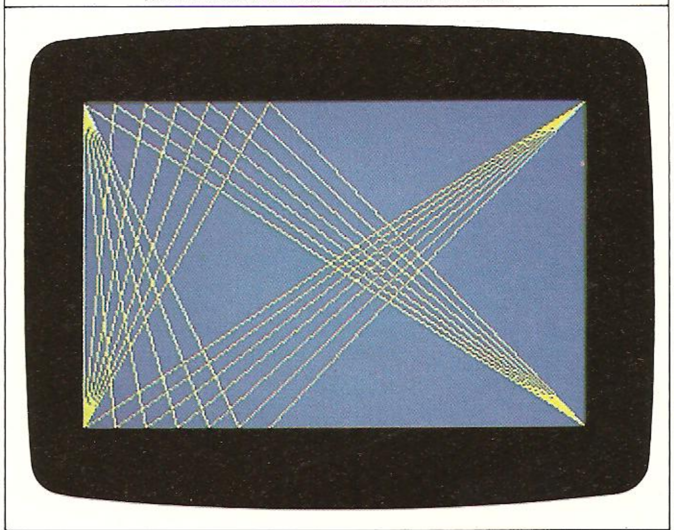


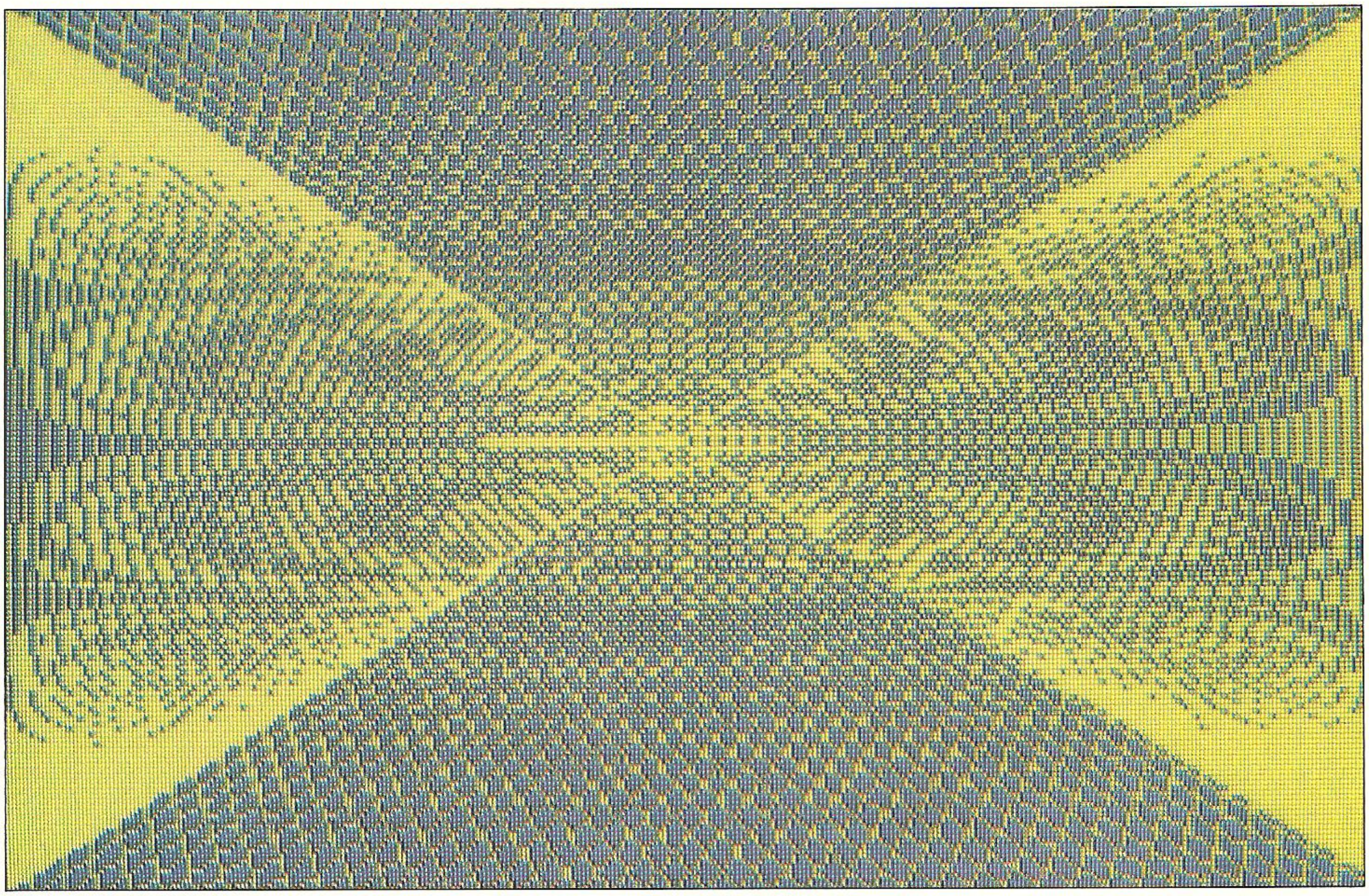


LINE INTERFERENCE PROGRAM

```
 10 DEF FN g(x,y,p,q)=USR 60700
100 BORDER 0: PAPER 1: INK 6
110 CLS
120 FOR i=16 TO 6 STEP -1
130 FOR j=0 TO 255 STEP i
140 RANDOMIZE FN g(0,0,j,175)
150 RANDOMIZE FN g(255,175,j,0)
160 RANDOMIZE FN g(255,0,j,175)
170 RANDOMIZE FN g(0,175,j,0)
180 NEXT j
190 PAUSE 0: CLS
200 NEXT i

0 OK, 0:1
```

LINE INTERFERENCE DISPLAY





curves. The series of horizontal and vertical lines in sequence produces the curve effect. As the lines become closer together, with each successive display, a better effect is obtained.

The line interference program shows how a program similar to the line pattern program can produce interference patterns simply by increasing the number of lines plotted on the screen, from 150 to 255 (the variable i in line 130).

LINE INTERFERENCE PROGRAM

00:06 seconds

How the program works
Patterns are produced by drawing lines from each corner of the screen to the opposite screen edge.
Line 120 sets up the first loop, to draw complete displays.
Line 130 sets up the second, inner loop, which calls the routines 255 times to draw a single display
Lines 140-170 call the line routine to draw four lines.
Line 190 waits for a key to be pressed before clearing the screen and beginning the next display.

TRIANGLES

A triangle shape is useful as the basis of all kinds of graphics displays. Pyramids, mountains, trees and bushes can all be formed from a triangular shape; even the spotlight display on this page is drawn with triangles. However, Spectrum BASIC does not have a single-statement triangle command.

The triangle routine, FNi, enables you to draw triangles quickly and painlessly. Like the line-draw routine (FNg), on which it is based, the triangle routine uses absolute rather than relative co-ordinates; this makes complex graphics displays easier to program.

All the displays shown here make use of the routine within a loop or loops. The repeated triangles program is based on triangles plotted between two parallel lines. Several interesting modifications are possible here: try, for example, changing the first x co-cordinate of the triangle from xc-2⋆y to xc-y. This will produce parallelograms between the parallel screen edges.

The spotlight display is produced by drawing a series of triangles from a single point (5,170). The base of each triangle is a horizontal line, the end points of which lie on the circumference of a shallow ellipse.

The final program, the triangle curves program, is a display of curves produced from sequences of triangles. The outer and inner curves are produced by the routines called in lines 130-140 and 180-190 respectively.

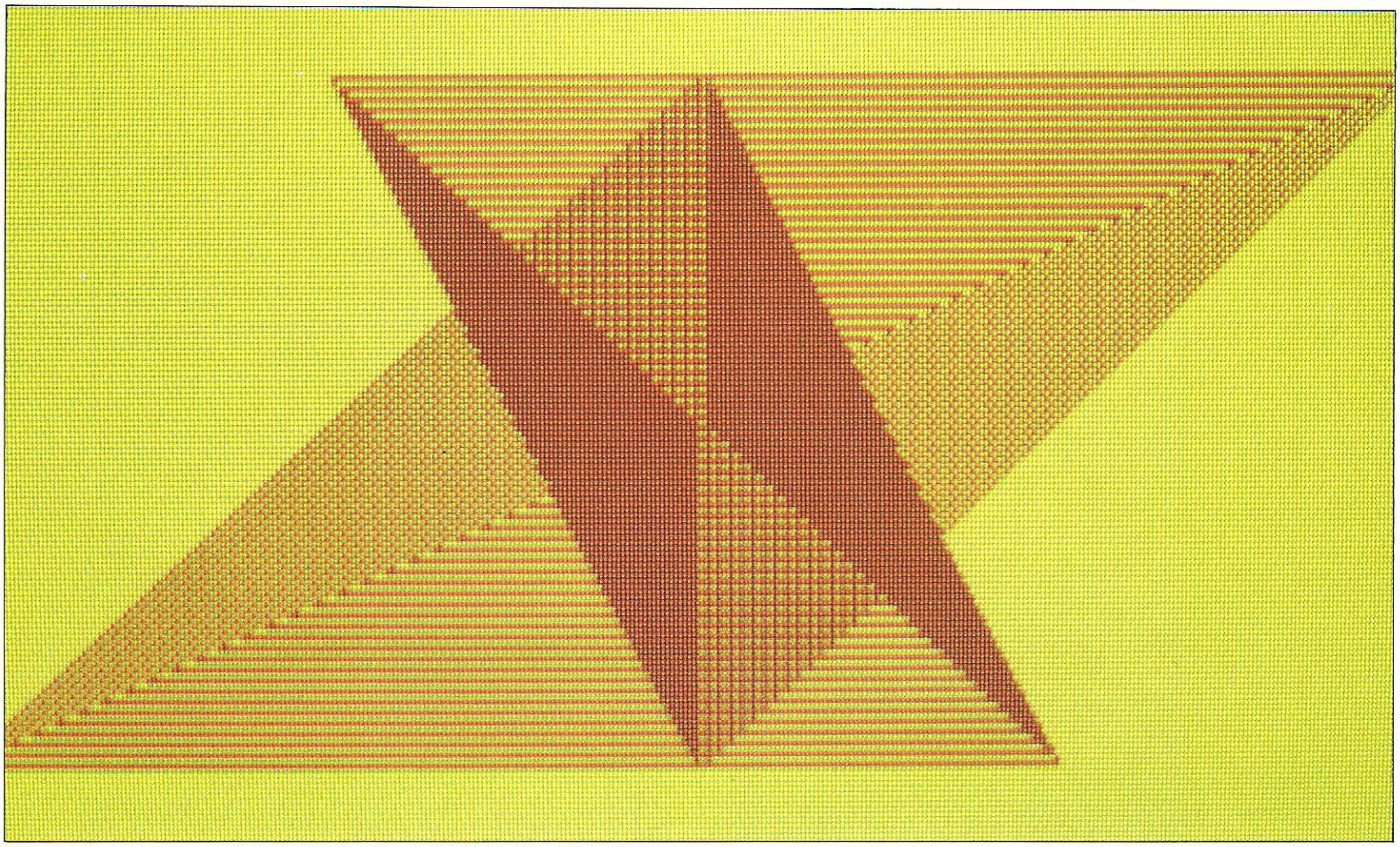
REPEATED TRIANGLES PROGRAM

```
  10 DEF FN i(x,y,p,q,r,s)=USR 6
0300
 100 BORDER 6: PAPER 6: INK 2: C
LS
 110 LET r=63: LET xc=127: LET y
c=90
 120 FOR i=7 TO 2 STEP -1
 130 FOR y=r TO -r STEP -i
 140 RANDOMIZE FN i(xc-2*y,yc-y,
xc,yc+y,xc+y,yc-y)
 150 NEXT y
 160 PAUSE 0: CLS
 170 NEXT i

0 OK, 0:1
```

REPEATED TRIANGLES PROGRAM

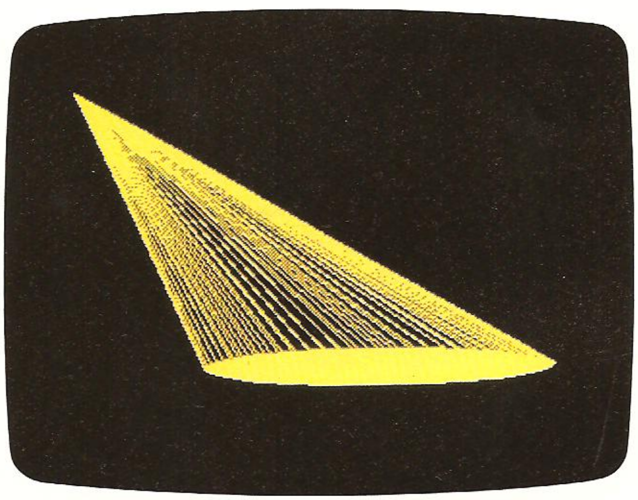
00:03 **seconds**

How the program works
The program draws a series of triangles. Each of the three points of the triangle is moved along a straight line, by changing the variable y for each successive triangle.
Line 120 begins the first loop, which sets the distance between each triangle in the display.
Line 130 sets up the second loop, which draws the pattern.



SPOTLIGHT PROGRAM

```
  10 DEF FN i(x,y,p,q,r,s)=USR 6
0300
 100 BORDER 0: PAPER 0: INK 6: C
LS
 110 LET s=4: LET a=0: LET ad=s*
PI/128
 120 LET x1=162: LET y1=20
 130 FOR i=0 TO 255 STEP s
 140 LET x=x1+INT (90*SIN a)
 150 LET y=y1+INT (10*COS a)
 160 LET x2=x1+INT (90*SIN (a+PI
))
 170 RANDOMIZE FN i(x,y,x2,y,5,1
70)
 180 LET a=a+ad
 190 NEXT i

0 OK, 0:1
```



Line 160 of the curves program sets the central screen area to red using the window ink routine, FNb (which must be present in memory for the program to RUN).

TRIANGLE CURVES PROGRAM

```
  10 DEF FN b(x,y,h,v,b,c,f)=USR
 62800
  20 DEF FN i(x,y,p,q,r,s)=USR 6
0300
 100 BORDER 4: PAPER 4: INK 1
 110 CLS
 120 FOR a=0 TO 175 STEP 4
 130 RANDOMIZE FN i(a,0,0,175-a,
0,0): RANDOMIZE FN i(a,175,0,a,0
,175)
 140 RANDOMIZE FN i(255-a,175,25
5,a,255,175): RANDOMIZE FN i(255
-a,0,255,175-a,255,0)
 150 NEXT a
 160 RANDOMIZE FN b(8,4,16,14,2,
0,0)
 170 FOR a=78 TO 178 STEP 4
 180 RANDOMIZE FN i(a,38,178,a-4
0,256-a,138)
 190 RANDOMIZE FN i(256-a,138,78
,216-a,a,38)
 200 NEXT a

0 OK, 0:1
```

FNi

TRIANGLE DRAW ROUTINE

Start address 60300 **Length** 80 bytes
Other routines called Line draw routine (FNg).
What it does Draws a triangle given the pixel co-ordinates of three points.

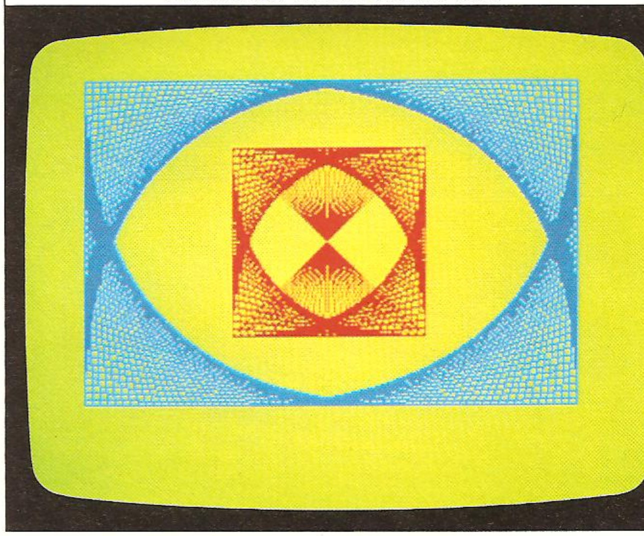
Using the routine The routine uses absolute co-ordinates. Specifying off-screen co-ordinates produces an error message; values more than 255 pixels off the screen will probably cause the Spectrum to crash. Colours are set by the current screen INK attributes.

ROUTINE PARAMETERS	
DEF FNi(x,y,p,q,r,s)	
x,y	specify first corner of triangle (x<256,y<176)
p,q	specify second corner of triangle (p<256,q<176)
r,s	specify third corner of triangle (r<256,s<176)

ROUTINE LISTING

```
7600 LET b=60300: LET l=75: LET
z=0: RESTORE 7610
7601 FOR i=0 TO l-1: READ a
7602 POKE (b+i),a: LET z=z+a
7603 NEXT i
7604 LET z=INT (((z/l)-INT (z/l)
)*l)
7605 READ a: IF a<>z THEN PRINT
"??": STOP

7610 DATA 42,11,92,1,4
7611 DATA 0,9,86,14,8
7612 DATA 9,94,237,83,208
7613 DATA 235,9,86,9,94
7614 DATA 237,83,210,235,9
7615 DATA 86,9,94,237,83
7616 DATA 212,235,42,210,235
7617 DATA 34,26,237,205,51
7618 DATA 237,237,91,208,235
7619 DATA 42,212,235,34,26

7620 DATA 237,205,51,237,237
7621 DATA 91,210,235,42,208
7622 DATA 235,34,26,237,205
7623 DATA 51,237,201,40,6
7624 DATA 40,143,80,123,0
7625 DATA 68,0,0,0,0
```

TRIANGLE CURVES DISPLAY



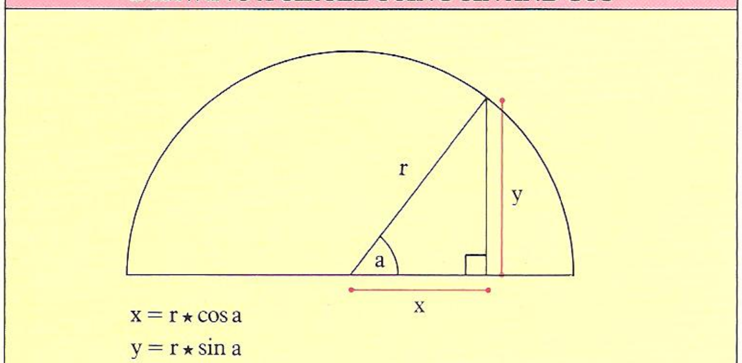
CIRCLES AND ARCS 1

Two methods are commonly used to draw circles on a computer. The first uses a combination of sines and cosines; this is the method used to draw circles in Spectrum BASIC. The sine/cosine method is derived from the fact that, for a right-angled triangle, the length of the horizontal and vertical sides can be calculated from the size of one angle and the length of the third side. If a right-angled triangle is formed between the centre of a circle and any point on the circumference, as shown in the diagram below, then the length of the sides is given by

$$x = r \star \cos(a)$$
$$y = r \star \sin(a)$$

You can see a typical example of circles plotted in Spectrum BASIC in the circle program on this page. The command requires the centre-point and radius to be specified (x,y and r).

DRAWING A CIRCLE USING SIN AND COS

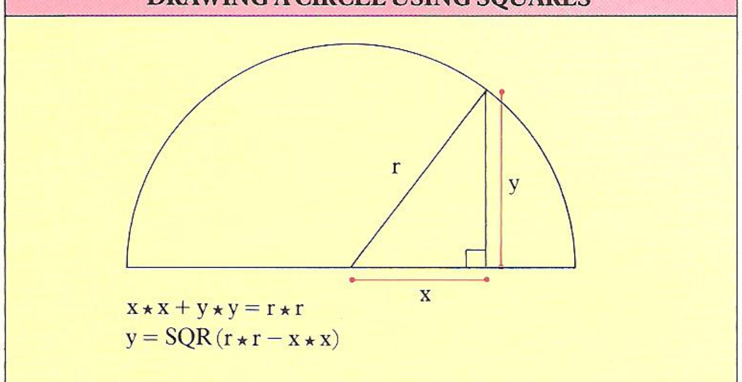


The second method is faster in operation but requires more memory to implement. This method, based on squares, forms the basis of the machine-code routine given here. It is derived from the equation

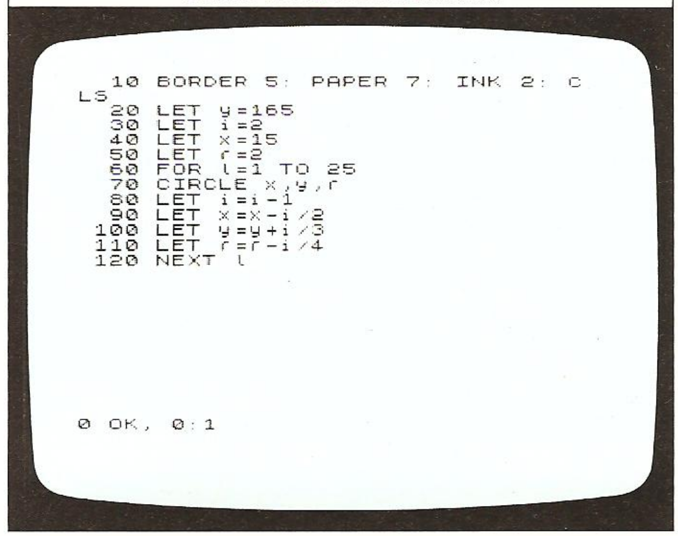
$$x^2 + y^2 = r^2$$

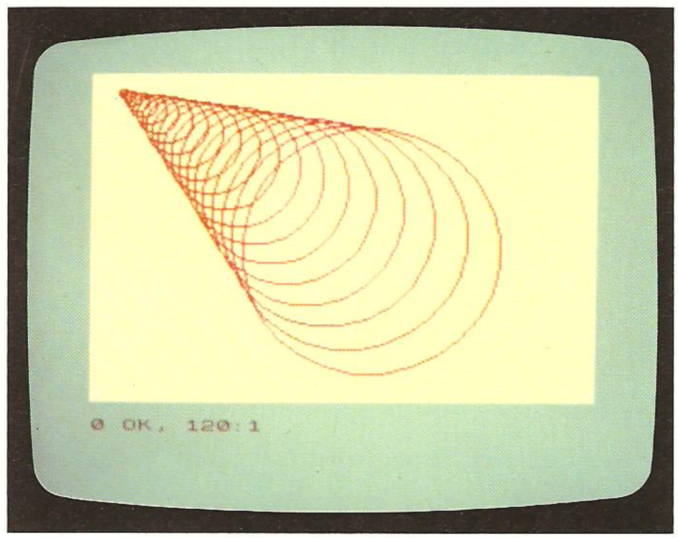
This, of course, is Pythagoras' theorem, which gives the relation between the sides of a right-angled triangle, as shown in the squares method diagram below.

DRAWING A CIRCLE USING SQUARES



BASIC CIRCLES PROGRAM





This method is more complicated than the sine/cosine method, since it must calculate square root values each time a circle is drawn. To use it most effectively, first calculate a list of square roots and then store this list in memory — as done by the routine at address 59600. The list of square roots can then be "consulted" by the main routine. Using stored square roots makes this routine much faster — and more accurate — than using the BASIC CIRCLE command.

Using the routines

Together with the BASIC square loader program, the routines given here do the work of calculating points for circles to be drawn. After keying in these routines you will not yet be able to produce anything on the screen, because the routines do not in themselves draw any curves; for this you must also key in one of the curve routines in the following pages.

MASTER CURVE ROUTINES

Start addresses 59600 and 59000
Length 60 and 525 bytes
Other routines called Point-plot routine (FNf).
What they do Carry out the calculations for the arc, sector and segment routines.

Using the routines The following BASIC program must be keyed in and RUN before using the machine-code routines given here:

```
10 LET j = 59700
20 FOR i = 0 TO 255
30 LET p = i * i : LET h = INT (p / 256)
40 LET l = p—256 * h
50 POKE j,l : POKE j +1,h
60 LET j = j + 2
70 NEXT i
```

This program POKEs the squares of numbers from 0 to 255 into memory. Each square is stored in two bytes, since numbers larger than 16 squared will not fit into a single byte, which has a maximum value of 255. Having keyed in this routine, SAVE the area of memory containing the squares by using the command SAVE "title" CODE 59700,600. These 600 bytes are also used as workspace by the circle routines.

The routine at 59600 calculates square roots and stores them in memory. The longer routine, starting at address 59000, calculates points on the circumference of a circle using these square roots.

ROUTINE LISTING

```
7650 LET b=59600: LET l=55: LET
z=0: RESTORE 7660
7651 FOR i=0 TO l-1: READ a
7652 POKE (b+i),a: LET z=z+a
7653 NEXT i
7654 LET z=INT (((z/l)-INT (z/l)
)*l)
7655 READ a: IF a<>z THEN PRINT
"??": STOP

7660 DATA 62,0,186,32,4
7661 DATA 187,32,1,201,1
7662 DATA 52,233,10,111,3
7663 DATA 10,103,167,237,82
7664 DATA 242,234,232,3,24
7665 DATA 242,17,202,22,96
7666 DATA 105,25,124,167,31
7667 DATA 125,31,201,33,52
7668 DATA 233,22,0,7,48
7669 DATA 1,20,95,25,94
7670 DATA 35,86,201,0,0
7671 DATA 3,0,0,0,0
```

ROUTINE LISTING

```
7700 LET b=59000: LET l=520: LET
 z=0: RESTORE 7710
7701 FOR i=0 TO l-1: READ a
7702 POKE (b+i),a: LET z=z+a
7703 NEXT i
7704 LET z=INT (((z/l)-INT (z/l)
)*l)
7705 READ a: IF a<>z THEN PRINT
"??": STOP

7710 DATA 62,1,50,104,232
7711 DATA 58,112,232,205,246
7712 DATA 232,237,83,117,232
7713 DATA 167,203,26,203,27
7714 DATA 205,208,232,50,119
7715 DATA 232,58,113,232,205
7716 DATA 11,232,50,120,232
7717 DATA 58,114,232,205,11
7718 DATA 232,50,121,232,58
7719 DATA 113,232,205,19,232
7720 DATA 50,115,232,58,114
7721 DATA 232,205,19,232,50
7722 DATA 116,232,71,58,115
7723 DATA 232,144,200,58,115
7724 DATA 232,50,122,232,58
7725 DATA 120,232,71,58,121
7726 DATA 232,144,32,34,60
7727 DATA 50,123,232,58,121
7728 DATA 232,230,1,40,9
7729 DATA 58,116,232,50,124

7730 DATA 232,195,7,231,58
7731 DATA 116,232,71,58,119
7732 DATA 232,144,50,124,232
7733 DATA 195,7,231,60,50
7734 DATA 123,232,58,120,232
7735 DATA 230,1,32,8,62
7736 DATA 0,50,124,232,195
7737 DATA 7,231,58,119,232
7738 DATA 50,124,232,58,120
7739 DATA 232,71,33,105,231

7740 DATA 17,18,0,25,16
7741 DATA 253,34,125,232,58
7742 DATA 120,232,230,1,40
7743 DATA 31,58,122,232,50
7744 DATA 122,232,205,63,232
7745 DATA 42,125,232,233,205
7746 DATA 81,232,58,122,232
7747 DATA 33,124,232,190,40
7748 DATA 43,60,250,98,231
7749 DATA 24,228,58,122,232

7750 DATA 71,58,119,232,167
7751 DATA 144,50,122,232,205
7752 DATA 63,232,42,125,232
7753 DATA 233,205,81,232,58
7754 DATA 122,232,33,124,232
7755 DATA 190,40,6,61,250
7756 DATA 98,231,24,228,58
7757 DATA 120,232,60,50,120
7758 DATA 232,62,0,50,122
7759 DATA 232,58,123,232,61

7760 DATA 200,254,1,202,205
7761 DATA 230,195,239,230,87
7762 DATA 58,122,232,95,58
7763 DATA 111,232,130,87,58
7764 DATA 110,232,131,95,195
7765 DATA 43,231,95,58,122
7766 DATA 232,87,58,111,232
7767 DATA 130,87,58,110,232
7768 DATA 131,95,195,80,231
7769 DATA 95,58,122,232,87

7770 DATA 58,111,232,146,87
7771 DATA 58,110,232,131,95
7772 DATA 195,43,231,87,58
7773 DATA 122,232,95,58,111
7774 DATA 232,146,87,58,110
7775 DATA 232,131,95,195,80
7776 DATA 231,87,58,122,232
7777 DATA 95,58,111,232,146
7778 DATA 87,58,110,232,147
7779 DATA 95,195,43,231,95

7780 DATA 58,122,232,87,58
7781 DATA 111,232,146,87,58
7782 DATA 110,232,147,95,195
7783 DATA 80,231,95,58,122
7784 DATA 232,87,58,111,232
7785 DATA 130,87,58,110,232
7786 DATA 147,95,195,43,231
7787 DATA 87,58,122,232,95
7788 DATA 58,111,232,130,87
7789 DATA 58,110,232,147,95

7790 DATA 195,80,231,6,5
7791 DATA 203,63,16,252,60
7792 DATA 201,230,31,254,0
7793 DATA 200,71,175,79,237
7794 DATA 91,117,232,33,0
7795 DATA 0,25,48,4,12
7796 DATA 32,1,60,16,247
7797 DATA 93,84,111,6,6
7798 DATA 167,203,29,203,25
7799 DATA 203,26,203,27,16

7800 DATA 245,205,208,232,201
7801 DATA 58,122,232,205,246
7802 DATA 232,42,117,232,167
7803 DATA 237,82,93,84,205
7804 DATA 208,232,201,58,104
7805 DATA 232,254,0,40,8
7806 DATA 175,50,104,232,237
7807 DATA 83,106,232,237,83
7808 DATA 108,232,205,72,240
7809 DATA 201,0,0,96,165
7810 DATA 52,130,80,140,30
7811 DATA 20,170,16,11,132
7812 DATA 3,21,7,6,0
7813 DATA 1,10,213,231,0
7914 DATA 234,0,0,0,0
```

CIRCLES AND ARCS 2

With the arc routine, FNj, given on this page, you will be able to draw arcs more quickly and with more flexibility than with the Spectrum DRAW command. The routine can be used to draw either arcs or complete circles by varying the final two parameters.

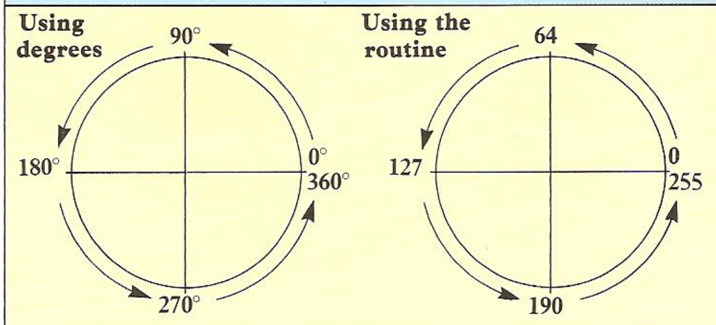
To produce any circle-based program on this page, you must first key in the machine-code routines and the BASIC squares program on page 29, as well as the routine given on this page, since the arc routine calls all these routines.

Starting and finishing the arc

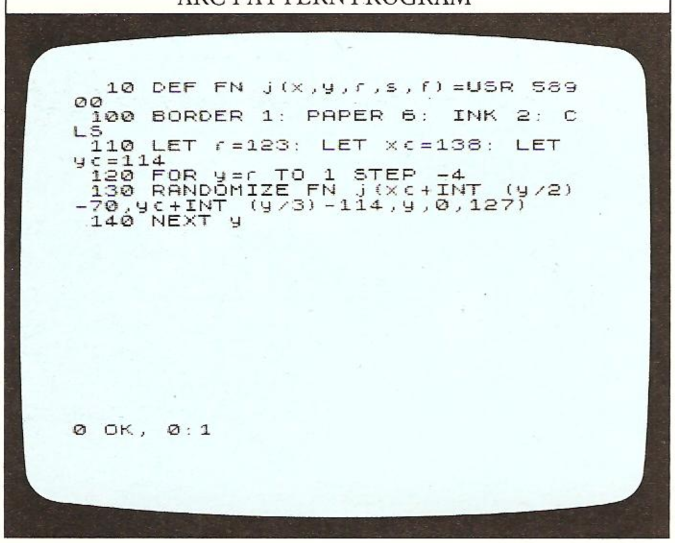
The only complication of the arc routine given here is in specifying how much of the circle is to be drawn. The start and finish parameters can have values from 0 to 255, instead of 0 to 360. This is because the parameters are stored in a single byte in memory, and one byte can have only 256 values (that is, from 0 to 255). This means that s,f values of 0,255 will draw a complete circle, values 0,127 will give a semicircle, and so on.

Since the routine begins drawing from a position horizontally to the right of the selected centre point, s,f values of 0,64 will produce a quarter circle from the right of the centre to a point vertically above it. The diagram below shows how the start and finish parameters correspond to the more usual degrees.

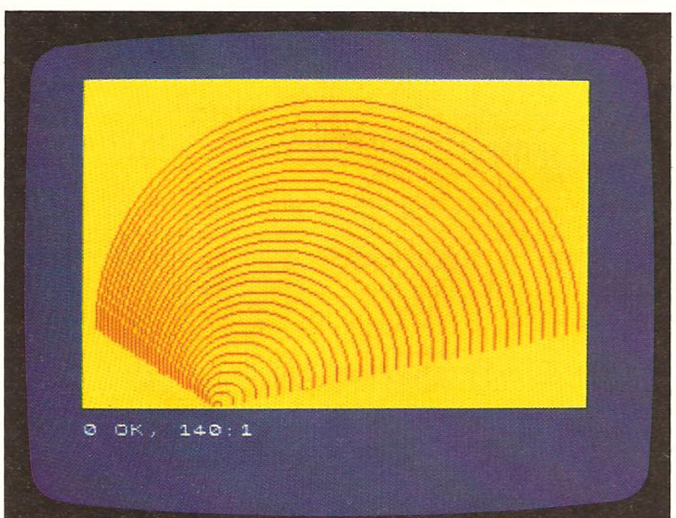
CALCULATING POINTS ON A CIRCLE

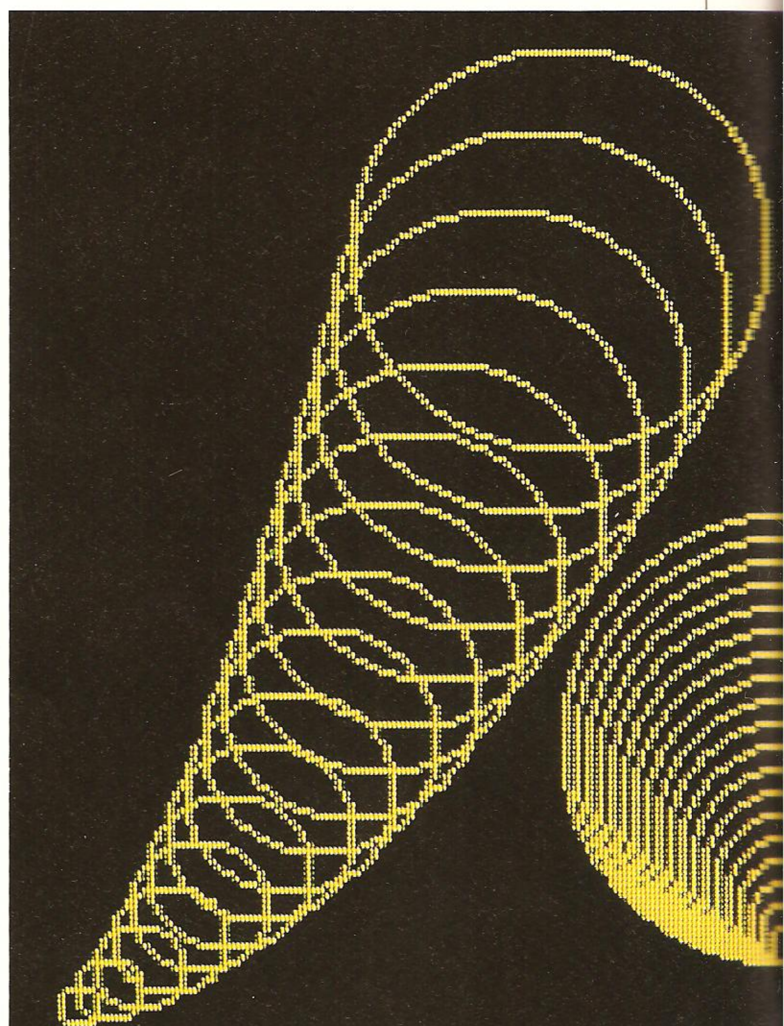


ARC PATTERN PROGRAM



```
  10 DEF FN j(x,y,r,s,f)=USR 589
00
 100 BORDER 1: PAPER 6: INK 2: C
LS
 110 LET r=123: LET xc=138: LET
yc=114
 120 FOR y=r TO 1 STEP -4
 130 RANDOMIZE FN j(xc+INT (y/2)
-70,yc+INT (y/3)-114,y,0,127)
 140 NEXT y



0 OK, 0:1
```





FNj

ARC ROUTINE

Start address 58900 **Length** 45 bytes
Other routines called Master curve routines.
What it does Draws an arc or circle at a specified radius from a centre point.

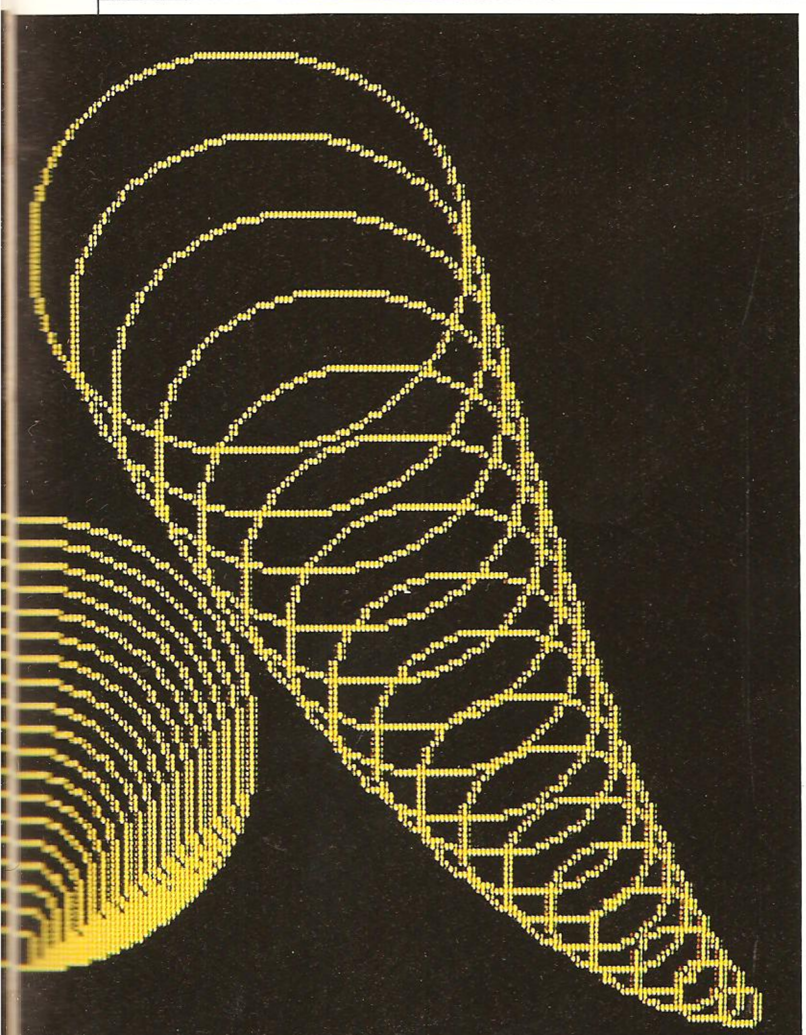
Using the routine The table on the facing page shows how the s and f parameters specify the length of the arc. A difference of 255 will produce a complete circle, 127 a semicircle, and so on. The numbers themselves define the angle from the centre of the circle at which the arc starts and finishes. The arc is drawn from a position due east of the centre of the circle, so that an s or f value of 1 is to the right of the centre point, a value of 128 to the left, and a value of 192 directly beneath the centre of the circle.
Unlike CIRCLE in Spectrum BASIC, you can draw curves with this routine which go some distance off the top or bottom of the screen without an error message appearing.
Because of the way the screen memory works, there are several screen positions where a curve cannot be drawn using this routine. If you find the routine does not work in any position, move the centre point one pixel in any direction.

ROUTINE PARAMETERS

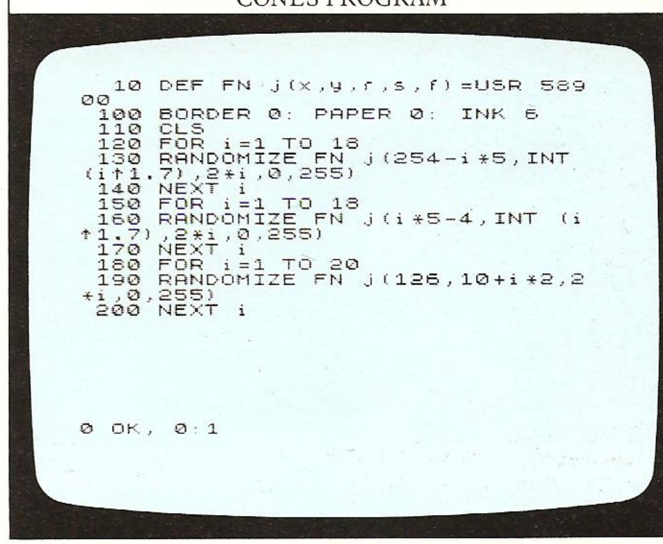
DEF FNj(x,y,r,s,f)	
x,y	specify the centre point from which the arc is drawn (x<256,y<176)
r	specifies the radius of the arc (r<256)
s,f	specify the length of the arc(s<f, s<256,f<256)

ROUTINE LISTING

```
7850 LET b=58900: LET l=40: LET
z=0: RESTORE 7860
7851 FOR i=0 TO l-1: READ a
7852 POKE (b+i),a: LET z=z+a
7853 NEXT i
7854 LET z=INT (((z/l)-INT (z/l)
)*l)
7855 READ a: IF a<>z THEN PRINT
"??": STOP

7860 DATA 42,11,92,1,4
7861 DATA 0,9,86,14,8
7862 DATA 9,94,237,83,110
7863 DATA 232,9,126,50,112
7864 DATA 232,9,126,50,113
7865 DATA 232,9,126,50,114
7866 DATA 232,71,58,113,232
7867 DATA 176,200,195,120,230
7868 DATA 17,0,0,0,0
```



CONES PROGRAM



```
  10 DEF FN j(x,y,r,s,f)=USR 589
00
 100 BORDER 0: PAPER 0: INK 6
 110 CLS
 120 FOR i=1 TO 18
 130 RANDOMIZE FN j(254-i*5,INT
(i↑1.7),2*i,0,255)
 140 NEXT i
 150 FOR i=1 TO 18
 160 RANDOMIZE FN j(i*5-4,INT (i
↑1.7),2*i,0,255)
 170 NEXT i
 180 FOR i=1 TO 20
 190 RANDOMIZE FN j(126,10+i*2,2
*i,0,255)
 200 NEXT i

0 OK, 0:1
```

The cones program produces patterns by varying the x and y co-ordinates of the centre of a circle each time it is drawn. In the left- and right-hand patterns, the x co-ordinate is a function of the variable i, while the y co-ordinate is given by i raised to the power of 1.7. As a result, the circles appear on a curve. In the third loop, only the y co-ordinate is varied, so that the sequence of circles rises vertically.

CONES PROGRAM

00:11 **seconds**

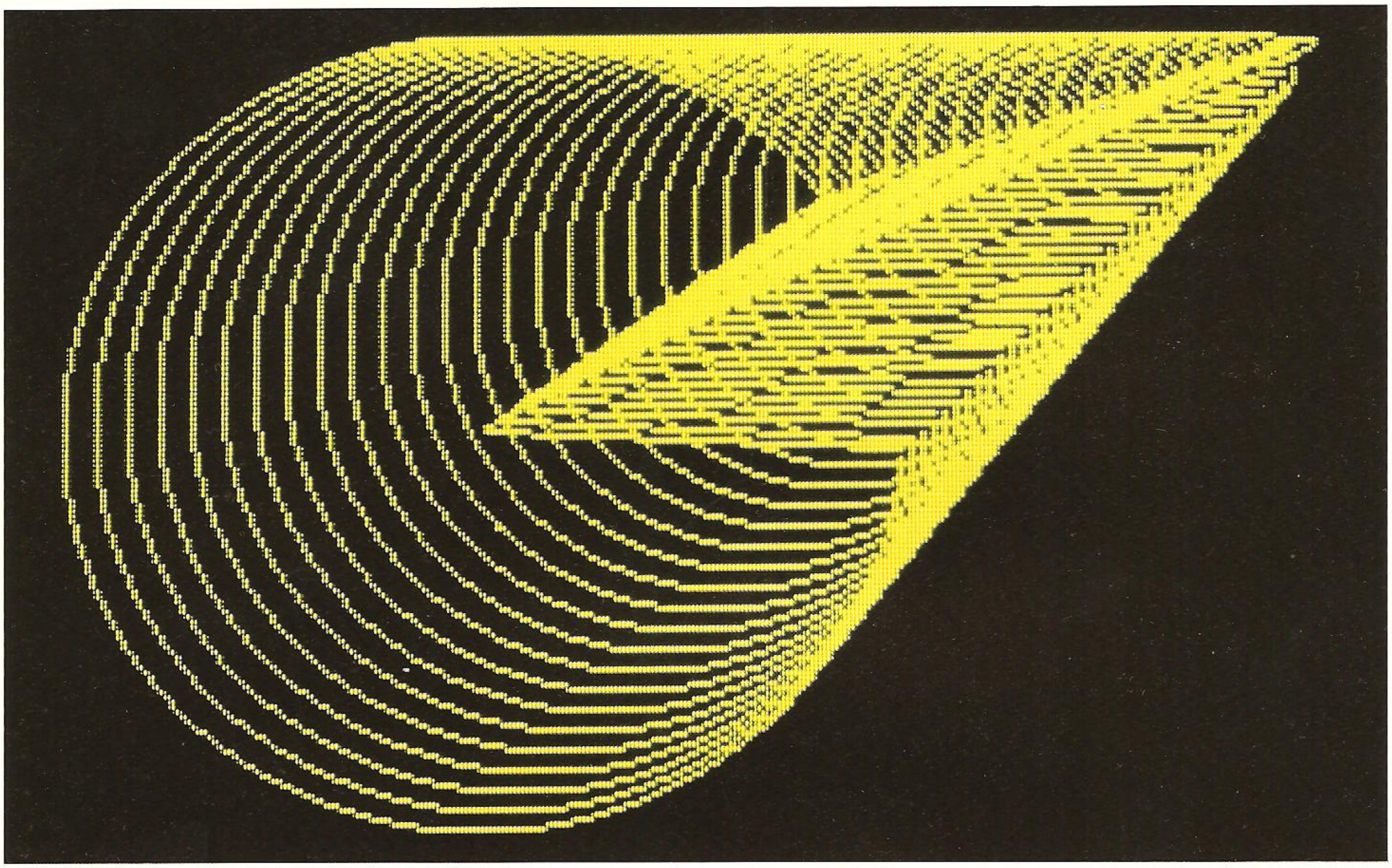
How the program works
Three circle patterns are drawn, using the circle routine within a loop.

Lines 120-140 draw the left-hand circles.
Lines 150-170 repeat the above loop, reversing the x,y co-ordinates.
Lines 180-200 draw the centre circles.

SECTORS AND SEGMENTS

The two routines on this page are useful supplements to the circle routine introduced on pages 30-31. Sectors are constructed by drawing an arc and then joining the end points to the centre from which the arc is drawn. A segment differs from a sector in that the ends of a segment are joined to each other, rather than to a centre point. The advantage of the routine is that they enable you to join the ends of an arc together without having to work out the co-ordinates of those points.

Both the sector and the segment routines call the master circle routine (page 29), the arc routine, FNj (page 31) and the line draw routine, FNn (page 21). This means that the sector and segment routines will not work unless these other routines are present in memory.



SECTOR PROGRAM

```
  10 DEF FN k(x,y,r,s,f)=USR 588
00
 100 BORDER 0: PAPER 0: INK 4: C
LS
 110 LET r=80: LET xc=250: LET y
c=170
 120 FOR i=3 TO 2 STEP -1
 130 FOR y=r TO 1 STEP -i
 140 RANDOMIZE FN k(xc-y*2,yc-IN
T (y),y,20,250)
 150 NEXT y
 160 PAUSE 0: CLS
 170 NEXT i

0 OK, 0:1
```

The sector program on this page creates the illusion of a third dimension by repeatedly drawing smaller and smaller sectors while at the same time moving the centre point upwards and to the right. Try varying i in line 120 to see a different number of sectors displayed.

The segment program repeats a pattern of segments (drawn from a single centre point in lines 140-150) three times across the screen. The number of patterns can be increased by varying the step size of x in line 120.

SECTOR PROGRAM

00:11 seconds

How the program works A sector of a circle is drawn repeatedly with decreasing radius.

Line 120 sets up a loop to vary i, the number of sectors drawn in one display.

Line 130 sets up a loop to vary y, used to calculate the centre point and the radius.

Line 140 draws a single sector.

Line 160 waits for a key to be pressed before drawing the display again.

FNk

SECTOR ROUTINE

Start Address 58800 **Length** 45 bytes
Other routines called Arc and line-draw routines (FNj, FNg).
What it does Draws an arc of specified radius, and joins each end to the centre point.

Using the routine The sector is drawn anti-clockwise from a point to the right of the centre. When the ends of the arc are joined to the centre, the result is a wedge shape if the difference between s and f is less than 127, or a cut pie shape if the difference is greater than 128. Sectors plotted off the screen to left or right may reappear rather unpredictably elsewhere on the screen, so it is best to keep within the parameter limits given below.

ROUTINE PARAMETERS	
DEF FNk(x,y,r,s,f)	
x,y	specify the centre point from which the arc is to be drawn (x<256,y<176)
r	specifies the radius of the arc (r<256)
s,f	specify the length of the arc (s<f, s<256, f<256)

ROUTINE LISTING

```
7900 LET b=58800: LET l=40: LET
z=0: RESTORE 7910
7901 FOR i=0 TO l-1: READ a
7902 POKE (b+i),a: LET z=z+a
7903 NEXT i
7904 LET z=INT (((z/l)-INT (z/l)
)*l)
7905 READ a: IF a<>z THEN PRINT
"??": STOP

7910 DATA 205,20,230,237,91
7911 DATA 106,232,42,110,232
7912 DATA 229,0,34,26,237
7913 DATA 205,51,237,237,91
7914 DATA 108,232,225,34,26
7915 DATA 237,205,51,237,237
7916 DATA 91,110,232,205,72
7917 DATA 240,201,0,0,0
7918 DATA 35,0,0,0,0
```

FNl

SEGMENT ROUTINE

Start address 58700 **Length** 30 bytes
Other routines called Arc and line-draw routines (FNj, FNg).
What it does Draws an arc of specified radius from a centre point, and joins the ends together.

Using the routine This routine works in the same way and with the same restrictions as the sector routine, except that in this case the ends of the arc are joined together, rather than to the centre.
Notice that, like the previous routine, you may get problems trying to connect the ends of the arc together, if either of the end points (and especially if both of them) are off the screen. As before, segments plotted off the edge of the screen to left or right will have unpredictable results: they may reappear on the other side, or cause the Spectrum to crash.

ROUTINE PARAMETERS	
DEF FNl (x,y,r,s,f)	
x,y	specify the centre point from which the arc is to be drawn (x<256,y<176)
r	specifies the radius of the arc (r<256)
s,f	specify the length of the arc (s<f, s<256, f<256)

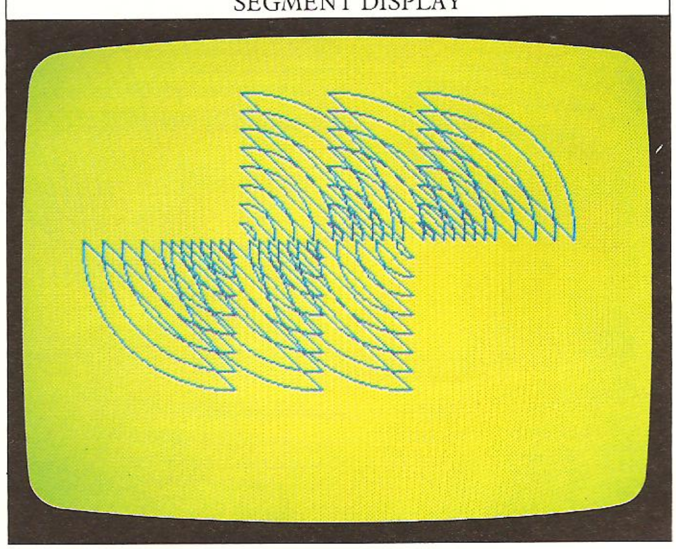
ROUTINE LISTING

```
7950 LET b=58700: LET l=25: LET
z=0: RESTORE 7960
7951 FOR i=0 TO l-1: READ a
7952 POKE (b+i),a: LET z=z+a
7953 NEXT i
7954 LET z=INT (((z/l)-INT (z/l)
)*l)
7955 READ a: IF a<>z THEN PRINT
"??": STOP

7960 DATA 205,20,230,237,91
7961 DATA 106,232,0,42,108
7962 DATA 232,0,0,34,26
7963 DATA 237,205,51,237,0
7966 DATA 201,0,0,0,0
7967 DATA 18,0,0,0,0
```

SEGMENT PROGRAM

```
  10 DEF FN l(x,y,r,s,f)=USR 587
00
 100 BORDER 4: PAPER 4: INK 1: C
LS
 110 LET r=80: LET y=90
 120 FOR x=82 TO 172 STEP 45
 130 FOR i=10 TO 80 STEP 10
 140 RANDOMIZE FN l(x,y,i,128,19
1)
 150 RANDOMIZE FN l(x,y,i,0,63)
 160 NEXT i
 170 NEXT x




0 OK, 0:1
```

SEGMENT DISPLAY



FILLING SHAPES 1

The fill routine given here, FNm, enables you to fill any enclosed shape no matter how irregular. The routine works by looking at the pixels adjacent to the specifed start point. If a pixel INK attribute is set, the routine does not change it, and does not look at pixels adjacent to this one; otherwise, the routine sets the INK attribute to the current INK colour and moves to the next adjacent pixels.

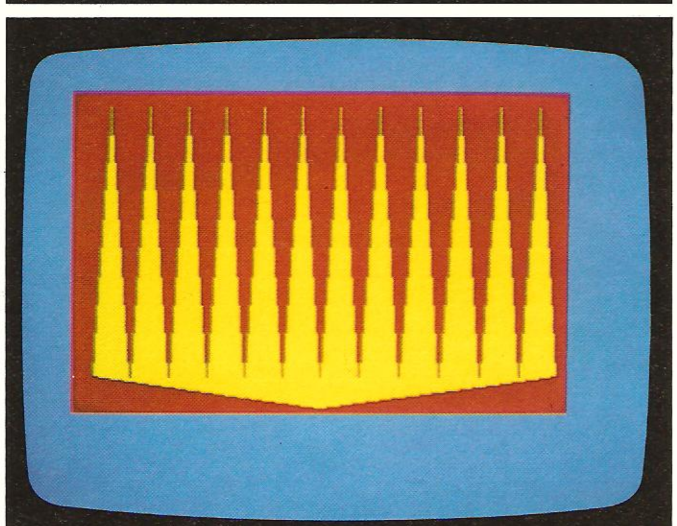
This method is known as the flood or grass-fire method, since, as you can see from its characteristic diamond shape, the INK spreads outwards until it reaches a "trench", which stops it from spreading further. Any shape which is not completely enclosed, even if only by a single pixel, will "leak" when filled.

Colouring irregular shapes

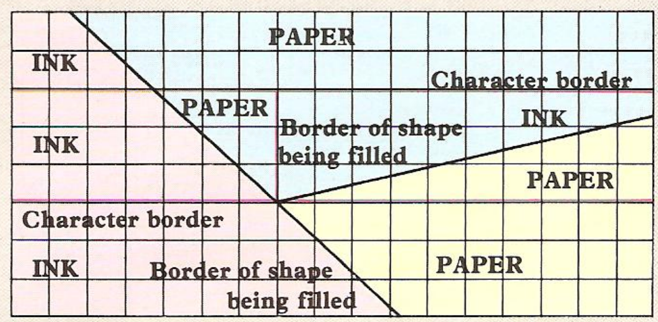
Since the Spectrum can only have one INK and one PAPER colour in each character block, you may have problems when there are more than two colours on the screen, and you call the routine to fill irregular shapes. If, for example, the shape has diagonal edges, you will see a jagged effect corresponding to character borders, instead of a straight line when the shape is filled. The diagram below shows how a combination of INK and PAPER colours can be used to overcome this problem.

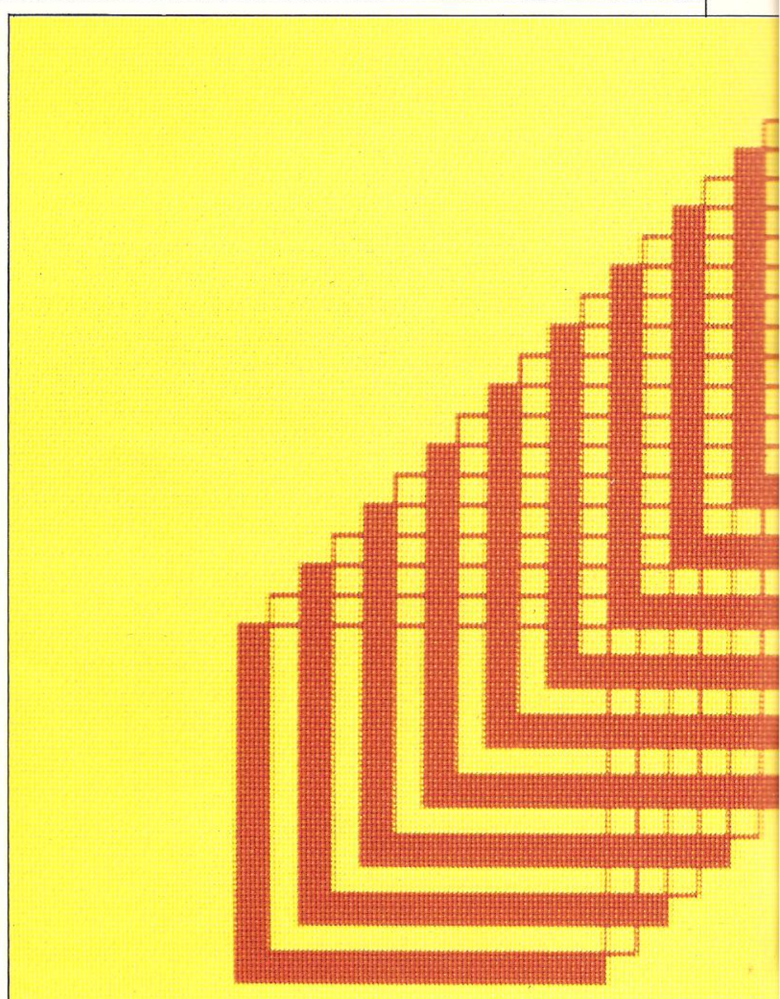
FILL PROGRAM

```
  10 DEF FN f(x,y,p,q)=USR 60700
  20 DEF FN m(x,y)=USR 57700
 100 BORDER 1: PAPER 6: INK 2: C
LS
 110 FOR i=1 TO 12
 120 LET x1=i*20
 130 LET y1=174
 140 LET x2=10+i*20: LET y2=20
 150 RANDOMIZE FN f(x1,y1,x2,y2)
 160 NEXT i
 170 FOR i=1 TO 12
 180 LET x1=i*20
 190 LET y1=174
 200 LET x2=i*20-10: LET y2=20
 210 RANDOMIZE FN f(x1,y1,x2,y2)
 220 NEXT i
 230 RANDOMIZE FN f(10,20,130,2)
 240 RANDOMIZE FN f(250,20,130,2
)
 250 PAUSE 100
 260 RANDOMIZE FN m(10,5)

0 OK, 0:1
```

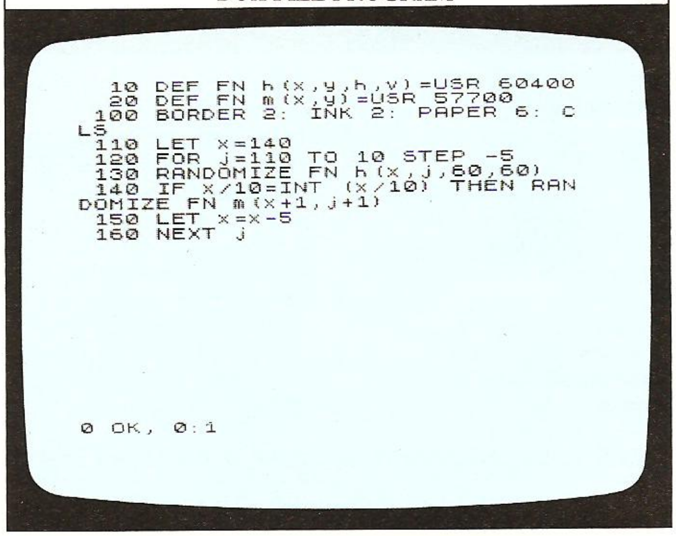


FILLING SHAPES AT CHARACTER BORDERS





BOX FILL PROGRAM

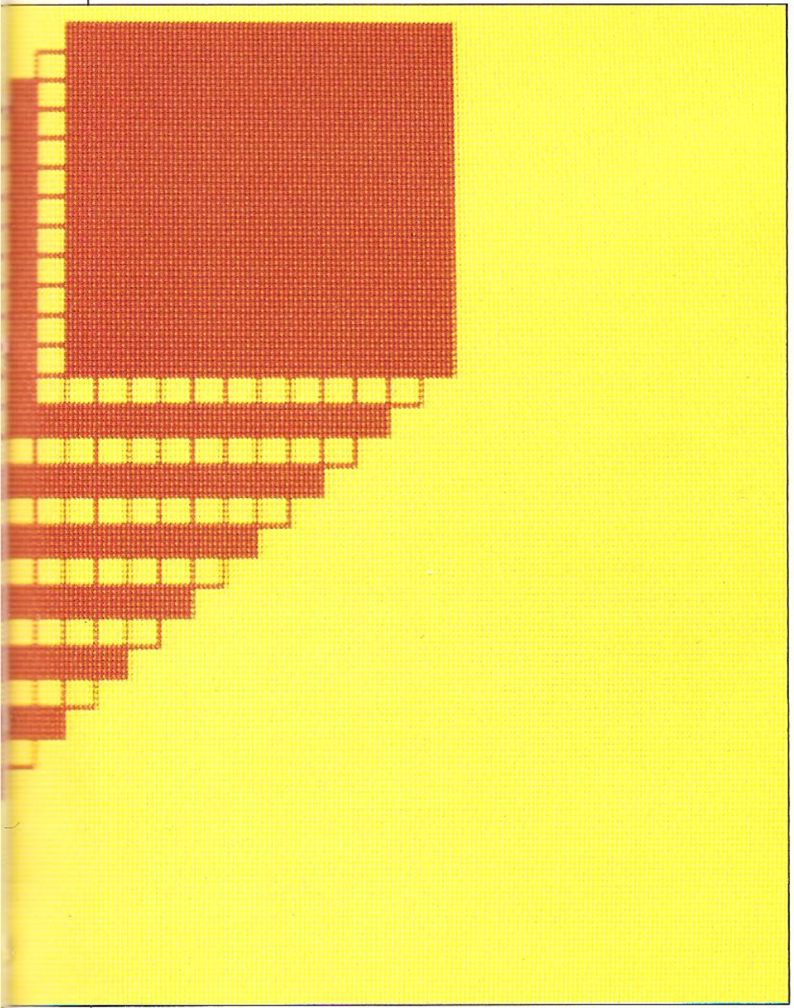


```
  10 DEF FN h(x,y,h,v)=USR 60400
  20 DEF FN m(x,y)=USR 57700
 100 BORDER 2: INK 2: PAPER 6: C
LS
 110 LET x=140
 120 FOR j=110 TO 10 STEP -5
 130 RANDOMIZE FN h(x,j,60,60)
 140 IF x/10=INT (x/10) THEN RAN
DOMIZE FN m(x+1,j+1)
 150 LET x=x-5
 160 NEXT j

0 OK, 0:1
```

BOX FILL PROGRAM

00:05 **seconds**

How the program works
Boxes are drawn in a loop, and filled alternately.
Line 120 sets up a loop.
Line 140 fills a box if variable x is exactly divisible by 10.
Line 150 reduces the value of x by 5.



FNm

FILL ROUTINE

Start address 57700 **Length** 195 bytes
Other routines called Line-draw routine (FNg).
What it does Fills in an area bounded by a solid line of INK, in the current INK colour.

Using the routine This routine fills in an area up to the edge of any shape enclosed by an INK line, or to the screen border. Remember that if there is even a single pixel of PAPER colour at the border, then the INK with which you are filling will leak out, and you may fill the entire screen. Notice also that a "wraparound" effect occurs when filling to left and right of the screen, which means that when the routine reaches the left-hand edge of the screen, it starts filling from the right-hand edge inwards, and the same will happen when the routine reaches the right-hand screen edge.
If some of the attributes for character squares within the area to be filled differ from each other (as will happen, for example, if you change some of the attributes using the window ink routine) then the area will be filled with these colours, rather than in a single colour.

ROUTINE PARAMETERS

DEF FNm(x,y)	
x,y	pixel co-ordinates of the point at which to start filling (x<256, y<176)

ROUTINE LISTING

```
8000 LET b=57700: LET l=190: LET
 z=0: RESTORE 8010
8001 FOR i=0 TO l-1: READ a
8002 POKE (b+i),a: LET z=z+a
8003 NEXT i
8004 LET z=INT (((z/l)-INT (z/l)
)*l)
8005 READ a: IF a<>z THEN PRINT
"??": STOP

8010 DATA 42,11,92,1,4
8011 DATA 0,9,86,14,8
8012 DATA 9,94,237,83,44
8013 DATA 226,237,83,42,226
8014 DATA 33,44,226,229,35
8015 DATA 35,34,40,226,225
8016 DATA 34,38,226,42,38
8017 DATA 226,94,35,86,21
8018 DATA 205,207,225,42,38
8019 DATA 226,94,28,35,86

8020 DATA 205,207,225,42,38
8021 DATA 226,94,35,86,20
8022 DATA 205,207,225,42,38
8023 DATA 226,94,29,35,86
8024 DATA 205,207,225,42,38
8025 DATA 226,35,35,229,1
8026 DATA 76,229,167,237,66
8027 DATA 32,5,225,33,44
8028 DATA 226,229,225,34,38
8029 DATA 226,237,75,40,226

8030 DATA 167,237,66,200,195
8031 DATA 133,225,237,83,42
8032 DATA 226,62,175,147,216
8033 DATA 95,167,31,55,31
8034 DATA 167,31,171,230,248
8035 DATA 171,103,122,7,7
8036 DATA 7,171,230,199,171
8037 DATA 7,7,111,122,230
8038 DATA 7,71,4,62,254
8039 DATA 15,16,253,6,255

8040 DATA 168,71,126,160,192
8041 DATA 126,176,119,42,40
8042 DATA 226,237,91,42,226
8043 DATA 115,35,114,35,229
8044 DATA 1,76,229,167,237
8045 DATA 66,32,5,225,33
8046 DATA 44,226,229,225,34
8047 DATA 40,226,201,193,195
8048 DATA 57,0,0,0,0
```

FILLING SHAPES 2

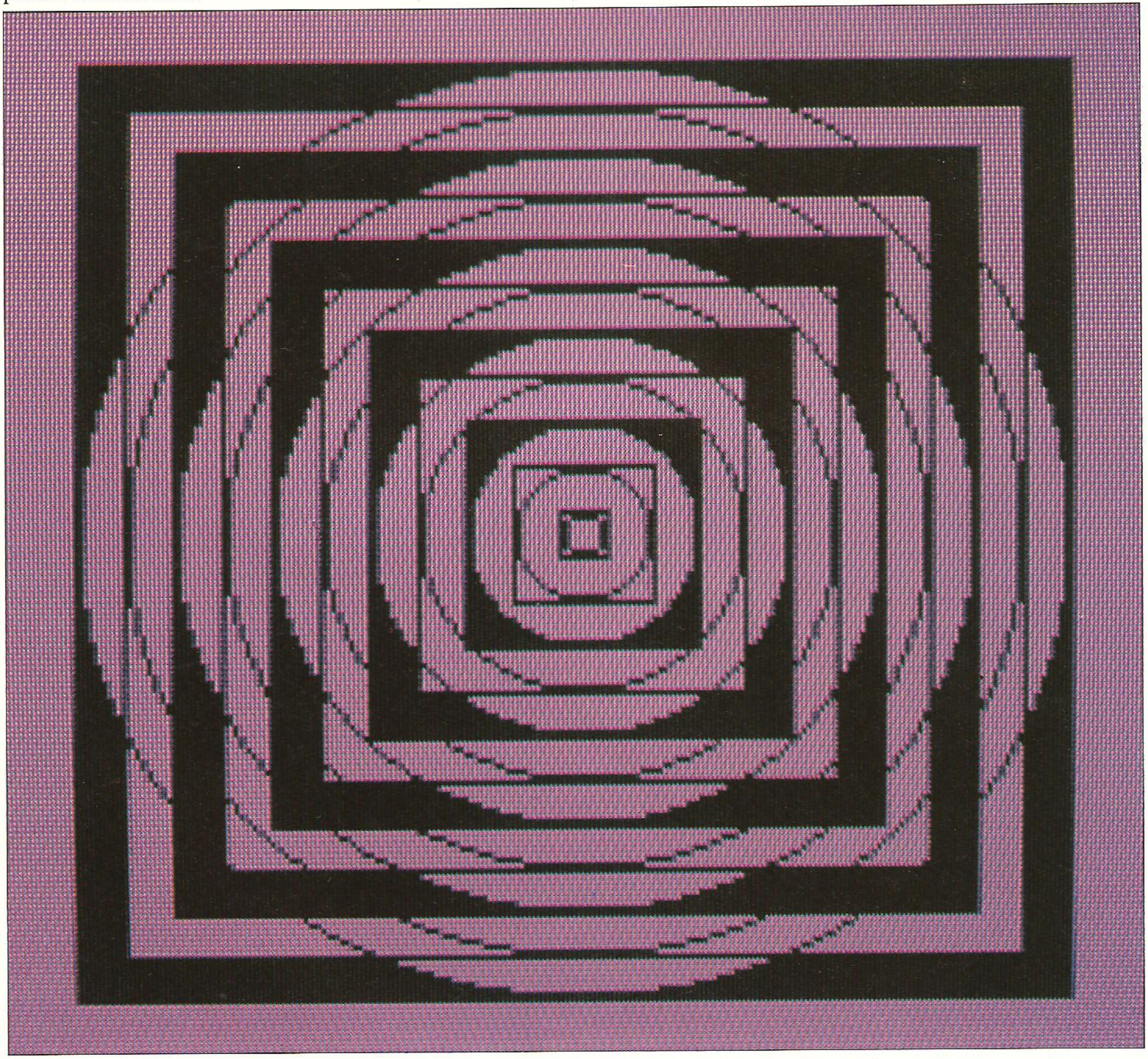
The fill routine really comes into its own when it is given highly irregular shapes to fill. Not only does it cope with these shapes with ease, it also fills them very quickly. The two programs on this page give an idea of the routine's capabilities.

The only complicated detail in each program is the calculation of the point from which the fill routine is to start. Each program has to calculate this point on each pass of the loop. To ensure that whole numbers are passed to the routine, the formula for the co-ordinates of the point is placed in brackets and an INT statement placed in front of it.

SQUARES AND CIRCLES PROGRAM

00:09 **seconds**

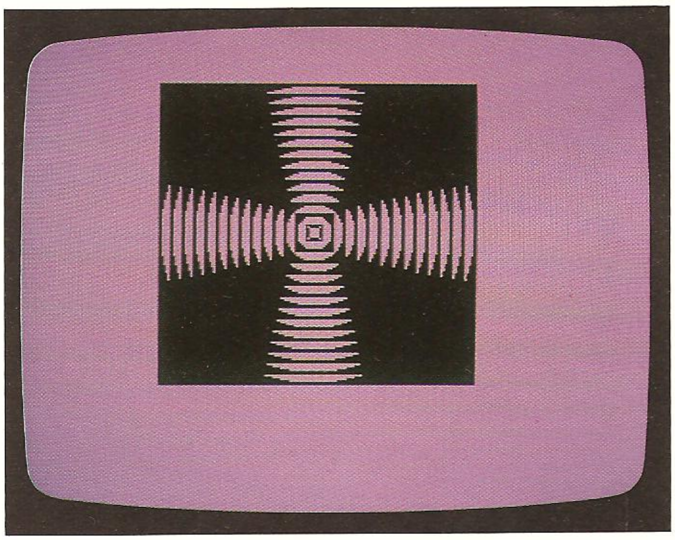
How the program works A series of boxes and circles of increasing size is drawn, and the fill routine called inside areas at which the boxes and circles intersect.
Lines 10-30 define the routines.
Line 120 sets a centre point for the display (x1,y1).
Line 130 starts a loop to draw the boxes and circles.
Line 140 draws a box based on an increment from the centre point.
Line 150 draws a circle.
Line 155 sets a test which calls the fill routine on alternate passes of the loop only.
Lines 160-190 fill four corners of the display.



The squares and circles program fills in the intersections between a series of boxes and circles. The program is interesting for the different final displays which can be obtained by changing the values of a few parameters, caused by different shapes created each time the boxes and circles are drawn. The boxes and circles are drawn in a loop at lines 140-150, and the intersections filled at lines 160-190.

SQUARES AND CIRCLES PROGRAM

```
  10 DEF FN h(x,y,h,v)=USR 60400
  20 DEF FN j(x,y,r,s,f)=USR 589
00
  30 DEF FN m(x,y)=USR 57700
 100 BORDER 3: PAPER 3: CLS
 120 LET x1=123: LET y1=91
 130 FOR i=8 TO 168 STEP 16
 140 RANDOMIZE FN h(x1-INT (i/2)
,y1-INT (i/2),i,i)
 150 RANDOMIZE FN j(x1,y1,INT (i
/2),0,255)
 155 IF (i+8)/32=INT ((i+8)/32)
THEN GO TO 200
 160 RANDOMIZE FN m(x1+1-INT (i/
2),y1+1-INT (i/2))
 170 RANDOMIZE FN m(x1-1+INT (i/
2),y1+1-INT (i/2))
 180 RANDOMIZE FN m(x1-1+INT (i/
2),y1-1+INT (i/2))
 190 RANDOMIZE FN m(x1+1-INT (i/
2),y1-1+INT (i/2))
 200 NEXT i

0 OK, 0:1
```



The spiral program

The spiral program is an effective use of the fill routine to colour in alternate portions of the circle. The two displays were achieved by varying n, which determines the number of spirals to be drawn.

SPIRAL PROGRAM

00:20 **seconds**

How the program works
Only the arc and fill routines are used in this program. A circle is drawn and then two BASIC semicircles are drawn to join the centre point to the circumference. After two of these curves have been drawn, the space between is filled and the sequence repeated. The number of spirals is set by variable n.

Lines 10-20 define the routines.

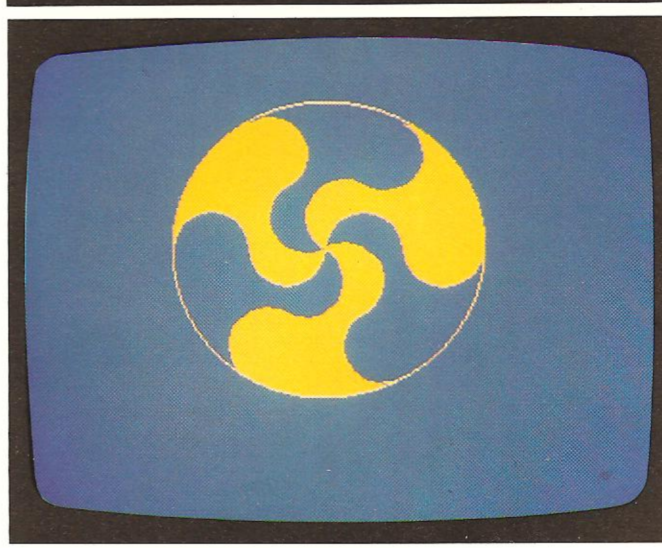
Line 120 draws a complete circle (centre 128,88).

Line 180 draws two curves in BASIC, using PI to specify semicircles.

Line 190 fills the area between two curves on alternate passes of the loop.

SPIRAL PROGRAM

```
  10 DEF FN j(x,y,r,s,f)=USR 589
00
  20 DEF FN m(x,y)=USR 57700
 100 BORDER 1: PAPER 1: INK 6: C
LS
 110 LET n=10
 120 RANDOMIZE FN j(128,88,81,0,
255)
 130 LET p=0: LET pd=2*PI/n
 140 FOR i=1 TO n
 150 LET x=INT (40*COS p)
 160 LET y=INT (40*SIN p)
 170 PLOT 128,88
 180 DRAW x,y,-PI: DRAW x,y,PI
 190 IF i/2=INT (i/2) THEN RANDO
MIZE FN m(128+INT (60*COS (p-pd)
),88+INT (60*SIN (p-pd)))
 200 LET p=p+pd
 210 NEXT i


0 OK, 0:1
```



OVERPRINTING AND ERASING

The OVER command in BASIC is one of four "logical operators" on the Spectrum; its more formal title is Exclusive/Or, or XOR for short. XOR forms the basis of the machine-code routine, FNn, on this page. You will recognize at once the other logical operators, since they occur in Spectrum BASIC with the same titles: AND, OR and NOT. Logical operators give a result depending on the way particular bits are set. The table below shows how the four operators make decisions.

TABLE OF LOGICAL OPERATORS

AND			OR			NOT		XOR		
A	B	**A AND B**	A	B	**A OR B**	A	**NOT A**	A	B	**A XOR B**
0	0	**0**	0	0	**0**	0	**1**	0	0	**0**
0	1	**0**	0	1	**1**	1	**0**	0	1	**1**
1	0	**0**	1	0	**1**			1	0	**1**
1	1	**1**	1	1	**1**			1	1	**0**

Thus, the XOR-line routine (FNn) looks at the screen before setting a pixel. If the pixel is currently set, the routine clears it; if the pixel is not set, however, the routine sets it.

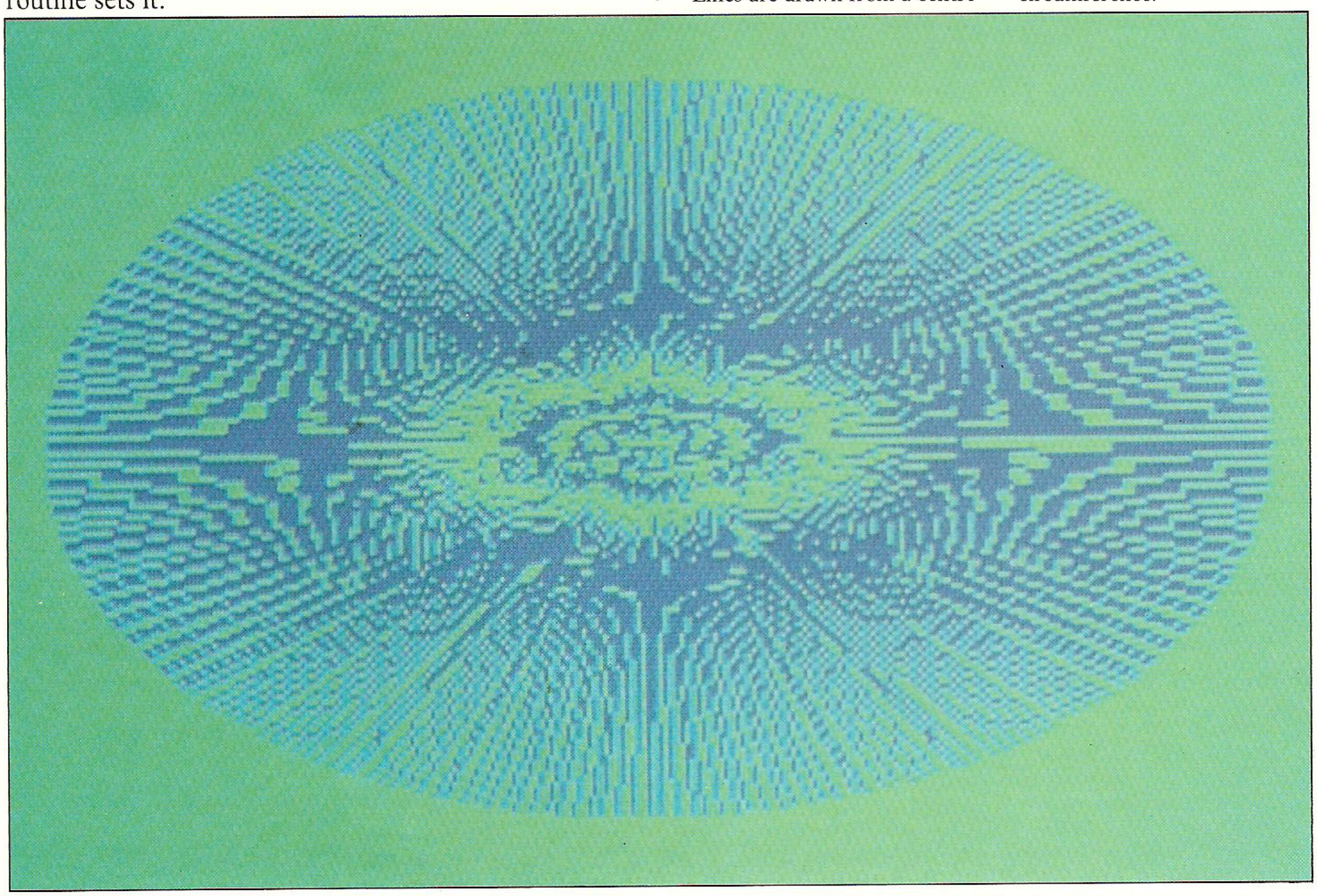
XOR ELLIPSE PROGRAM

```
  10 DEF FN n(x,y,p,q)=USR 57600
 100 BORDER 5: PAPER 5: INK 1: C
LS
 110 LET s=1: LET a=0: LET ad=s*
PI/128
 120 LET x1=127: LET y1=88
 130 FOR i=0 TO 255 STEP s
 140 LET x=x1+INT (110*SIN a)
 150 LET y=y1+INT (70*COS a)
 160 RANDOMIZE FN n(x,y,x1,y1)
 170 LET a=a+ad
 180 NEXT i
 190 PAUSE 0
 200 GO TO 110




0 OK, 0:1
```

XOR ELLIPSE PROGRAM

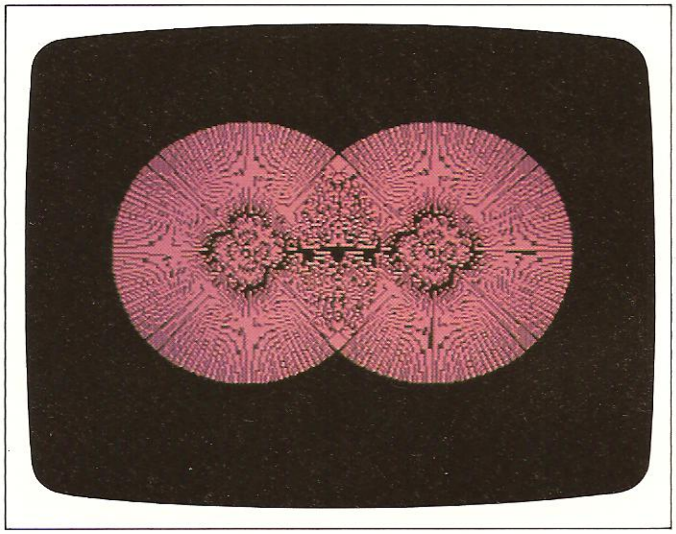
01:10 **minutes**

How the program works
Lines are drawn from a centre to points on an ellipse.
Line 130 specifies how many lines are to be drawn.
Lines 140-150 calculate the co-ordinates of a point on the circumference.



INTERFERENCE CIRCLES PROGRAM

```
  10 DEF FN n(x,y,p,q)=USR 57600
 100 BORDER 0: PAPER 0: INK 3: C
LS
 110 LET s=1: LET a=0: LET ad=s*
PI/128
 120 LET x1=80: LET y1=88
 130 FOR c=1 TO 2
 140 FOR i=0 TO 255 STEP s
 150 LET x=x1+INT (70*SIN a)
 160 LET y=y1+INT (70*COS a)
 170 RANDOMIZE FN n(x,y,x1,y1)
 180 LET a=a+ad
 190 NEXT i
 200 LET x1=x1+96
 210 NEXT c
 220 PAUSE 100
 230 GO TO 110



0 OK, 0:1
```



The interference circles program shows how, by using XOR lines, two overlapping circles can produce an interesting pattern instead of an area of solid colour.

OVERPRINTING PROGRAM

```
  10 DEF FN n(x,y,p,q)=USR 57600
 100 BORDER 1: PAPER 6: INK 2
 110 CLS
 120 FOR i=5 TO 15
 130 PRINT AT i,8;"123456      654
321"
 140 NEXT i
 150 FOR j=18 TO 238 STEP 3
 160 RANDOMIZE FN n(128,90,j,10)
 170 RANDOMIZE FN n(128,90,j,168
)
 180 NEXT j
 190 FOR j=10 TO 168 STEP 10
 200 RANDOMIZE FN n(129,90,18,j)
 210 RANDOMIZE FN n(128,90,238,j
)
 220 NEXT j
 230 PAUSE 0: GO TO 150



0 OK, 0:1
```

FNn

XOR-LINE ROUTINE

Start address 57600 **Length** 20 bytes
Other routines called Line-draw routine (FNg).
What it does Draws an Exclusive/OR line on the screen between two specified points.

Using the routine This routine works in the same way as the line-draw routine, except that Exclusive/OR allows you to erase what has been drawn. Using the routine you can draw lines over an image and then remove them again, without affecting the original image. As for the line-draw routine, the routine incorporates some error-trapping.

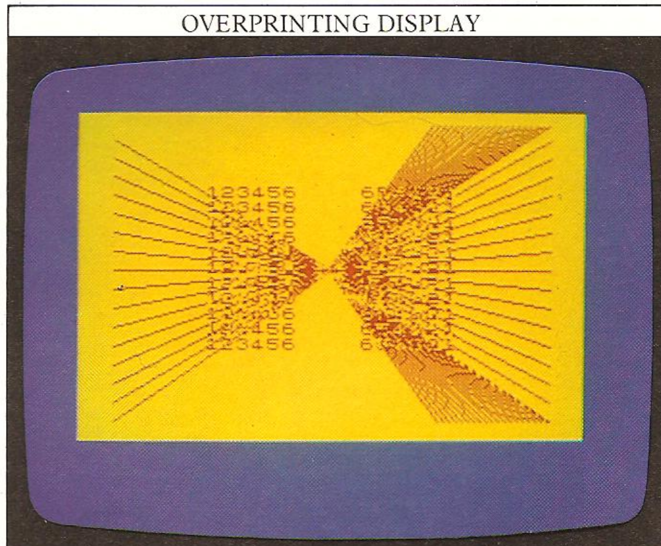
ROUTINE PARAMETERS

DEF FN n(x,y,p,q)	
x,y	specify the start pixel co-ordinates of the XOR-line (x<256,y<176)
p,q	specify the end pixel co-ordinates of the XOR-line (p<256, q<176)

ROUTINE LISTING

```
8050 LET b=57600: LET l=15: LET
z=0: RESTORE 8060
8051 FOR i=0 TO l-1: READ a
8052 POKE (b+i),a: LET z=z+a
8053 NEXT i
8054 LET z=INT (((z/l)-INT (z/l)
)*l)
8055 READ a: IF a<>z THEN PRINT
"??": STOP

8060 DATA 62,168,50,223,237
8061 DATA 205,28,237,62,176
8062 DATA 50,223,237,201,0
8063 DATA 13,0,0,0,0
```

OVERPRINTING DISPLAY



Finally, the overprinting program gives an example of the XOR-line routine being used to draw over some text and cover it (lines 160-170), and then "undraw" the lines by calling the routine again in lines 200 and 210, leaving the text intact.

COMBINING ROUTINES

The programs on this page give some further examples of combining the routines used earlier in this book. You will see from the programs used here that, in a program of any length, it is a good idea to separate the machine-code routines clearly at the beginning of the program, as has been done here.

Although the programs look complicated, they both consist mainly of machine-code calls. The repeated circles program is a symmetrical pattern; the small circles on the circumference of the large ones are drawn in lines 230-380. Variables x,y, which are points on the circumference of a large circle of radius rz, are used to determine the centre of the small circles. The actual centre points of the small circles are obtained by adding or subtracting an offset (x0,y0) from x and y in lines 270-340. Variables xm,ym are used to calculate the co-ordinates for the fill routine.

The kite program is even simpler; the only complicated part is the drawing of the tail (drawn by a subroutine in lines 500-600). The number of bows in the tail can be modified by changing the variable s in line 110.

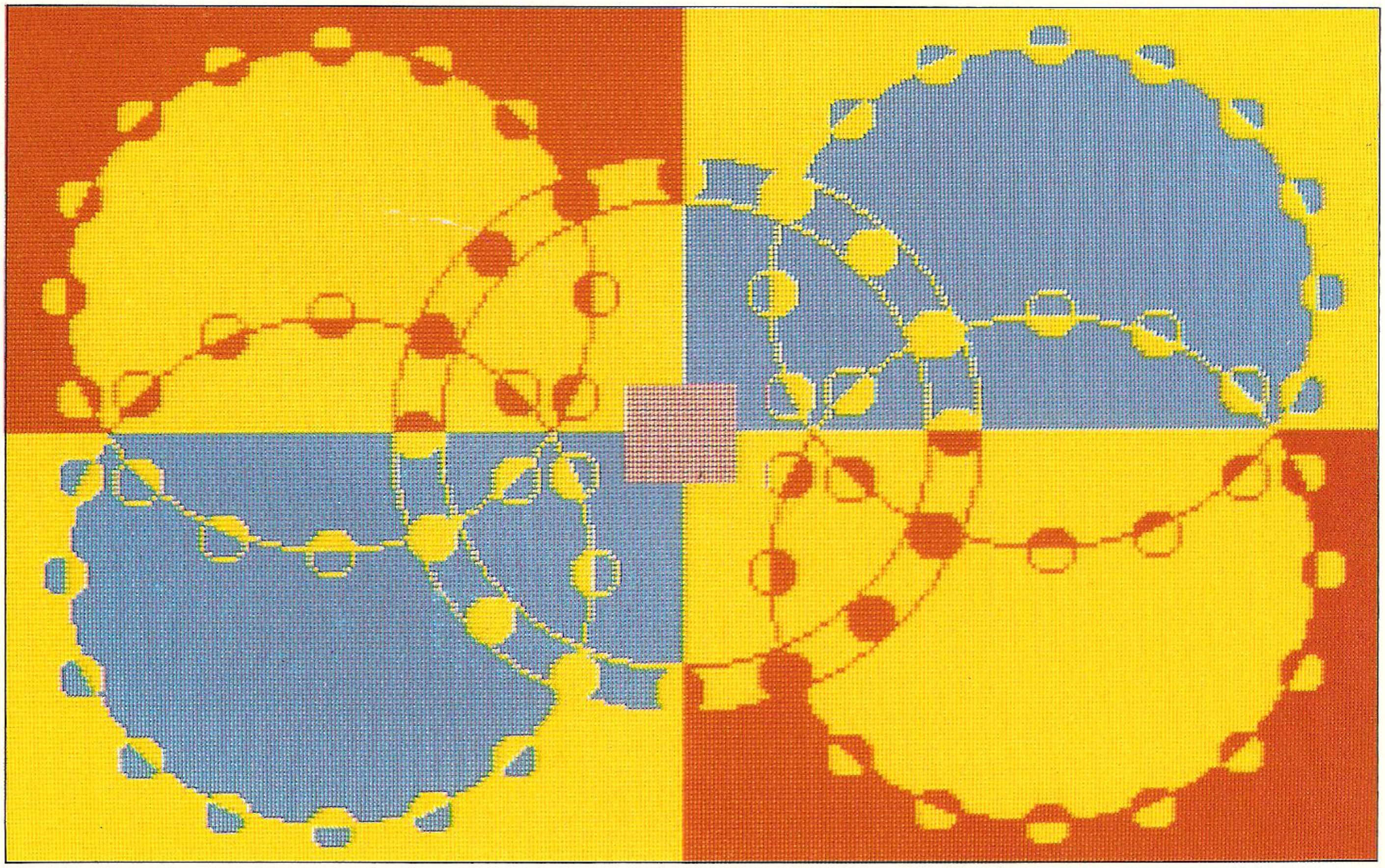
REPEATED CIRCLES PROGRAM

00:18 **seconds**

How the program works
This program displays circles with smaller circles on their circumference. Each of the small circles is then half-filled.
Line 100 defines ad, the step size.
Line 120 defines x0 and y0, the offset from the centre for the four large circles, and rx and rz, the radius of the small and large circles.
Lines 140-200 draw the large circles and the centre box.
Lines 250-380 draw the small circles.
Lines 400-440 set the colours of the four quarters of the screen.

REPEATED CIRCLES PROGRAM

```
  10 DEF FN b(x,y,h,v,c,b,f)=USR
 62800
  20 DEF FN c(x,y,h,v,c,b,f)=USR
 62600
  30 DEF FN h(x,y,h,v)=USR 60400
  40 DEF FN j(x,y,r,s,f)=USR 589
00
  50 DEF FN m(x,y)=USR 57700
 100 LET ad=16*PI/128: LET a=0
 110 LET x1=127: LET y1=87
 120 LET xo=66: LET yo=27: LET r
z=50: LET rx=5
 130 BORDER 0: PAPER 1: INK 6: C
LS
 140 RANDOMIZE FN j(x1-xo,y1-yo,
rz,0,255)
 150 RANDOMIZE FN j(x1+xo,y1-yo,
rz,0,255)
 160 RANDOMIZE FN j(x1+xo,y1+yo,
rz,0,255)
 170 RANDOMIZE FN j(x1-xo,y1+yo,
rz,0,255)

scroll?
```



REPEATED CIRCLES PROGRAM CONTD.

```
 180 RANDOMIZE FN j(x1,y1,rz+rx,
0,255)
 190 RANDOMIZE FN j(x1,y1,rz-rx+
1,0,255)
 200 RANDOMIZE FN h(117,78,20,18
)
 210 RANDOMIZE FN m(128,88)
 220 FOR i=0 TO 255 STEP 16
 230 LET x=x1+INT (rz*SIN a)
 240 LET y=y1+INT (rz*COS a)
 250 LET xm=x1+INT ((rz-3)*SIN a
)
 260 LET ym=y1+INT ((rz-3)*COS a
)
 270 RANDOMIZE FN j(x-xo,y-yo,rx
,0,255)
 280 RANDOMIZE FN m(xm-xo,ym-yo)
 290 RANDOMIZE FN j(x+xo+1,y-yo,
rx,0,255)
 300 RANDOMIZE FN m(xm+xo+1,ym-y
o)
 310 RANDOMIZE FN j(x+xo+1,y+yo,

scroll?
```

```
rx,0,255)
 320 RANDOMIZE FN m(xm+xo+1,ym+y
o)
 330 RANDOMIZE FN j(x-xo,y+yo,rx
,0,255)
 340 RANDOMIZE FN m(xm-xo,ym+yo)
 350 RANDOMIZE FN j(x,y,rx,0,255
)
 360 RANDOMIZE FN m(xm,ym)
 370 LET a=a+ad
 380 NEXT i
 390 RANDOMIZE FN m(1,1)
 400 RANDOMIZE FN b(0,0,16,11,2,
0,0)
 410 RANDOMIZE FN b(16,11,16,11,
2,0,0)
 420 RANDOMIZE FN c(0,0,16,11,6,
0,0)
 430 RANDOMIZE FN c(16,11,16,11,
6,0,0)
 440 RANDOMIZE FN b(14,9,4,4,3,0
,0)

0 OK, 0:1
```

KITE PROGRAM

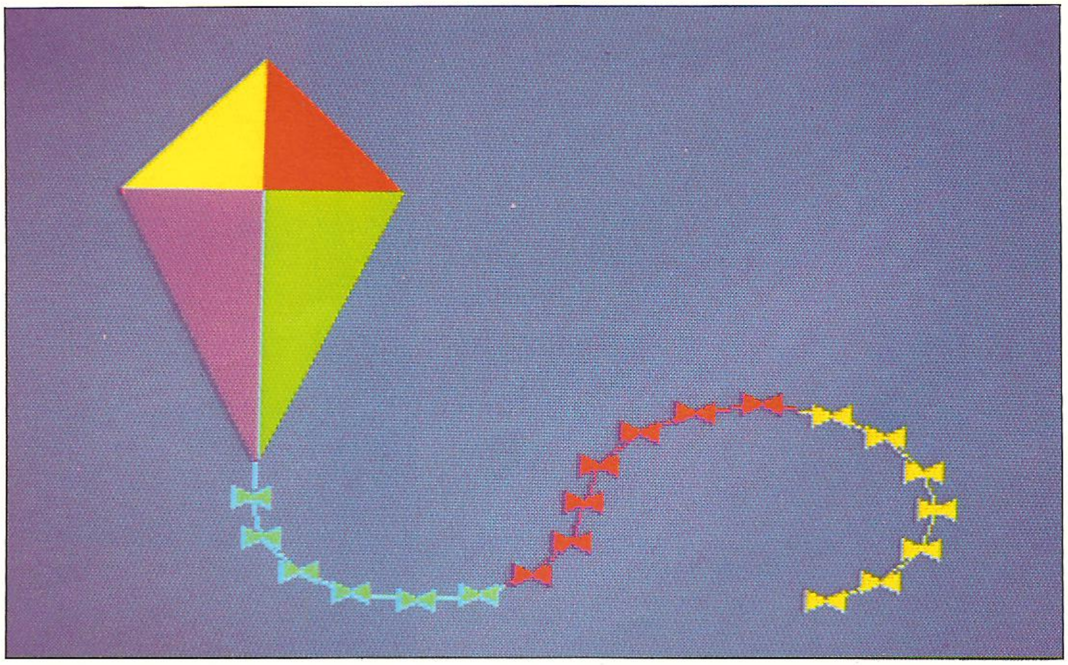
```
  10 DEF FN b(x,y,h,v,c,b,f)=USR
 62800
  20 DEF FN g(x,y,p,q)=USR 60700
  30 DEF FN i(x,y,p,q,r,s)=USR 6
0300
  40 DEF FN m(x,y)=USR 57700
 100 BORDER 1: PAPER 1: INK 2: C
LS
 110 LET s=16: LET ad=s*PI/128:
LET a=ad*64/s: LET lt=128: LET x
1=47: LET y1=127
 120 RANDOMIZE FN i(x1-40,y1,x1+
40,y1,x1,y1+40)
 130 RANDOMIZE FN i(x1-40,y1,x1+
40,y1,x1,y1-80)
 140 RANDOMIZE FN m(x1,y1)
 150 LET x2=143: LET y2=40: LET
x1=x1+47: LET y1=y1-90
 160 GO SUB 500
 170 RANDOMIZE FN g(47,50,47,40)
 180 LET x1=x1+96: LET lt=172: L
ET x2=143: LET y2=40: GO SUB 500

scroll?
```

```
 190 RANDOMIZE FN b(1,1,5,5,6,0,
0): RANDOMIZE FN b(6,6,5,10,4,0,
0): RANDOMIZE FN b(1,6,5,10,3,0,
0)
 200 RANDOMIZE FN b(25,8,6,14,6,
0,0)
 210 RANDOMIZE FN b(5,16,10,6,5,
0,0)
 220 PAUSE 0: STOP
 500 FOR i=0 TO lt STEP s
 510 LET x=x1+INT (48*SIN a)
 520 LET y=y1+INT (30*COS a)
 530 RANDOMIZE FN i(x,y,x-5,y+3,
x-5,y-3)
 540 RANDOMIZE FN m(x-2,y)
 550 RANDOMIZE FN i(x,y,x+5,y+3,
x+5,y-3)
 560 RANDOMIZE FN m(x+2,y)
 570 RANDOMIZE FN g(x,y,x2,y2)
 580 LET x2=x: LET y2=y
 590 LET a=a+ad
 600 NEXT i: RETURN

0 OK, 0:1
```

KITE PROGRAM

00:07 **seconds**

How the program works
The program draws a kite using coloured triangles, and then adds a tail with bows.
Lines 120-130 draw the kite using triangles.
Lines 150 and 180 set values for the subroutine variables.
Line 190 sets colours for the tail.
Line 500 is the start of the tail subroutine.
Lines 510 and 520 calculate the point x,y, at which an ellipse is drawn.
Lines 530-560 draw and fill in each bow.
Line 570 draws a line between each bow.
Lines 580 and 590 set values for the next bow to be drawn.



GRAPHICS EDITOR 1

Perhaps the most effective way of using machine-code routines like those in this book is in a single program which enables you to use the routines together. Although by this stage in the book you have enough routines available to create the kind of sophisticated displays seen in much commercial software, you do not have what the professionals use: a complete graphics editor. This is the purpose of the following program.

The graphics editor program

Each stage of the editor program incorporates routines from this book. The final program includes a facility for SAVEing and LOADing individual screens. The displays accompanying the program on this page and on the following few pages will give you some idea of the sort of pictures you can produce using the completed program.

How the program is built up

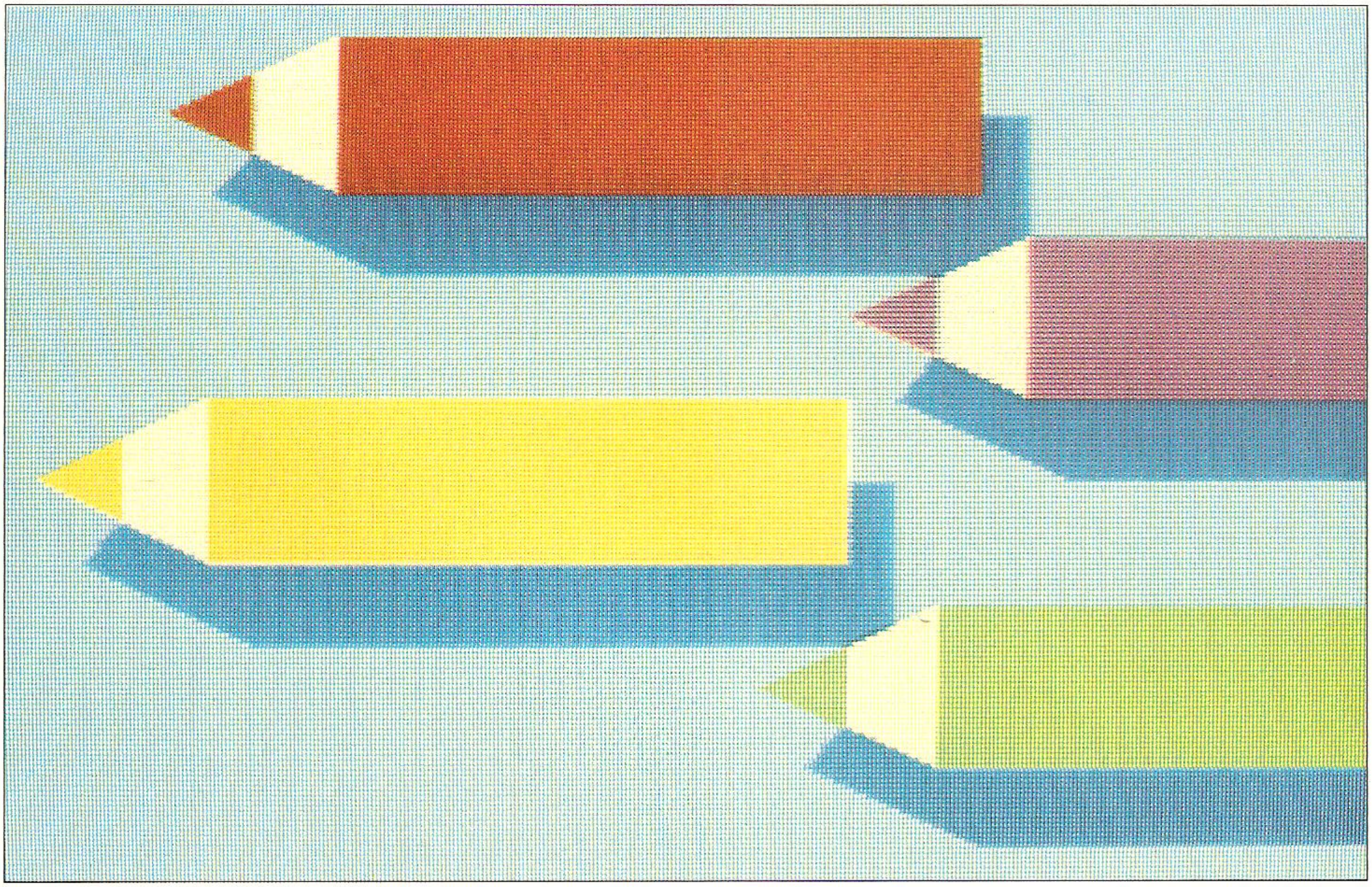
The graphics editor is shown in five stages, with each stage complete in itself. By keying in the lines on this page, you will have enough of the program to be able to move two cursors on the screen. These are used for plotting points and drawing lines in future stages.

GRAPHICS EDITOR STAGE 1

```
  10 DEF FN n(x,y,p,q)=USR 57600
  50 DEF FN h(x,y,h,v)=USR 60400
 100 PAPER 0: BORDER 0: INK 7: C
LS
 110 RANDOMIZE FN h(0,0,255,175)
 120 GO TO 1000
 200 LET cl=20: LET cr=20
 210 IF cx<20 THEN LET cl=cx: GO
 TO 230
 220 IF cx>235 THEN LET cr=255-c
x
 230 RANDOMIZE FN n(cx-cl,cy,cx+
cr,cy)
 240 LET cl=20: LET cr=20
 250 IF cy<20 THEN LET cl=cy: GO
 TO 270
 260 IF cy>155 THEN LET cr=175-c
y
 270 RANDOMIZE FN n(cx,cy-cl,cx,
cy+cr)
 280 RETURN
 300 LET me=7: LET mw=7: LET mu=

scroll?
```

How stage one works

Only two machine-code routines are used in stage one. The box-draw routine, FNh, draws a line round the edge of the drawing area, to prevent the fill routine (added later) from "wrapping around" the screen edge.



GRAPHICS EDITOR STAGE 1 CONTD.

```
7: LET md=7
 310 IF mx<7 THEN LET mw=0: LET
mu=mw: GO TO 330
 320 IF mx>248 THEN LET me=0: LE
T md=me
 330 IF my<mw OR my<md THEN LET
mw=0: LET md=mw: GO TO 350
 340 IF 175-my<mu OR 175-my<me T
HEN LET mu=0: LET me=mu
 350 RANDOMIZE FN n(mx-mw,my-mw,
mx+me,my+me)
 360 RANDOMIZE FN n(mx+md,my-md,
mx-mu,my+mu)
 370 RETURN
 900 LET mx=cx: LET my=cy
 910 GO SUB cur: GO SUB mar
 920 GO TO 1100
1000 LET kol=0: LET ink=0: LET p
ap=7: LET d=0: LET cur=200: LET
mar=300: LET g=0: LET b=0: LET f
l=0: LET f=255
1010 LET cx=160: LET cy=100: LET

scroll?
```

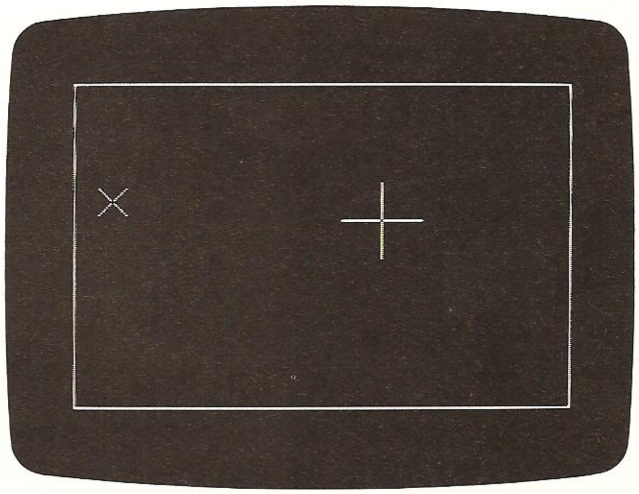
```
 mx=20: LET my=110
1020 GO SUB cur: GO SUB mar
1100 LET a$=INKEY$
1110 IF a$="" THEN LET d=0: GO T
O 1100
1120 LET ke=CODE a$: LET d=d+1
1130 IF ke<>9 THEN GO TO 1170
1140 GO SUB cur: LET cx=cx+d
1150 IF cx>255 THEN LET cx=255:
LET d=0
1160 GO SUB cur: GO TO 1100
1170 IF ke<>8 THEN GO TO 1210
1180 GO SUB cur: LET cx=cx-d
1190 IF cx<0 THEN LET cx=0: LET
d=0
1200 GO SUB cur: GO TO 1100
1210 IF ke<>10 THEN GO TO 1250
1220 GO SUB cur: LET cy=cy-d
1230 IF cy<0 THEN LET cy=0: LET
d=0
1240 GO SUB cur: GO TO 1100
1250 IF ke<>11 THEN GO TO 1290

scroll?
```

```
1260 GO SUB cur: LET cy=cy+d
1270 IF cy>175 THEN LET cy=175:
LET d=0
1280 GO SUB cur: GO TO 1100
1290 IF ke<>77 THEN GO TO 1320
1300 GO SUB mar: LET mx=cx: LET
my=cy
1310 GO SUB mar: GO TO 1100
1320 GO TO 1100





0 OK, 0:1
```

The other routine included here is the XOR line routine, FNn, used to draw the two cursors. Exclusive/OR plotting is used because the cursors have to be able to move around the screen and remain visible, without disturbing whatever has already been drawn. Try moving one of the cursors in this program to a corner of the screen to see the XOR effect. The screen border remains unchanged when the cursor is moved away.

Line 1000 is the beginning of the main routine. It gives initial values to all the variables used in the program. These, for example, store values for INK, PAPER, FLASH and BRIGHT. After setting initial co-ordinates for the two cursors (points cx,cy and mx,my), the program moves to the subroutines. These are stored early in the program to increase the running speed.

CURSORS DISPLAY



The cursor subroutine is at lines 200-280, and the cursor is positioned at point cx,cy. The second cursor is placed on the screen using the subroutine at lines 300-370 (points mx,my). These cursor subroutines (called cur, mar) are used to delete the cursors before any routine is called, and again to put the cursors back on the screen afterwards.

GRAPHICS EDITOR PARAMETERS

A	attribute edit	**L**	line
I	ink	**Q**	window paper
P	paper	**S**	save screen
O	bright/flash	**T**	triangle (press T again for second corner of triangle)
ENTER	to quit attribute edit	**W**	partial screen clear
B	box	**X**	text
C	circle	ENTER	to quit text
S	start		
F	finish		
D	dot		
E	window ink		
F	fill		
G	grid		
J	load screen		

All these instructions require you to press CAPS SHIFT followed by the letter shown, in upper case.

GRAPHICS EDITOR 2

The second stage of the graphics editor adds routines for points, lines and boxes, as well as adding the ink, paper and partial screen clear routines.

Colour is set by the subroutines in lines 400-850. These allow you to select colour, BRIGHT and FLASH values. Points are drawn using the point-plot routine, in lines 1320 to 1350. Some of the details in these lines reappear throughout the program. Line 1320, for example, checks to see if key D has been pressed (ASCII code number 68). If it has, the two cursor subroutines are called, and the values of cx and cy are used as the co-ordinates of the point to be plotted. Lines 1360 to 1490 work in a similar way for the line draw, fill and box routines. Line 1500 is a "dummy" line, where other routines will be inserted.

The grid subroutine

Lines 1730 to 1790 set up a grid on the screen, by

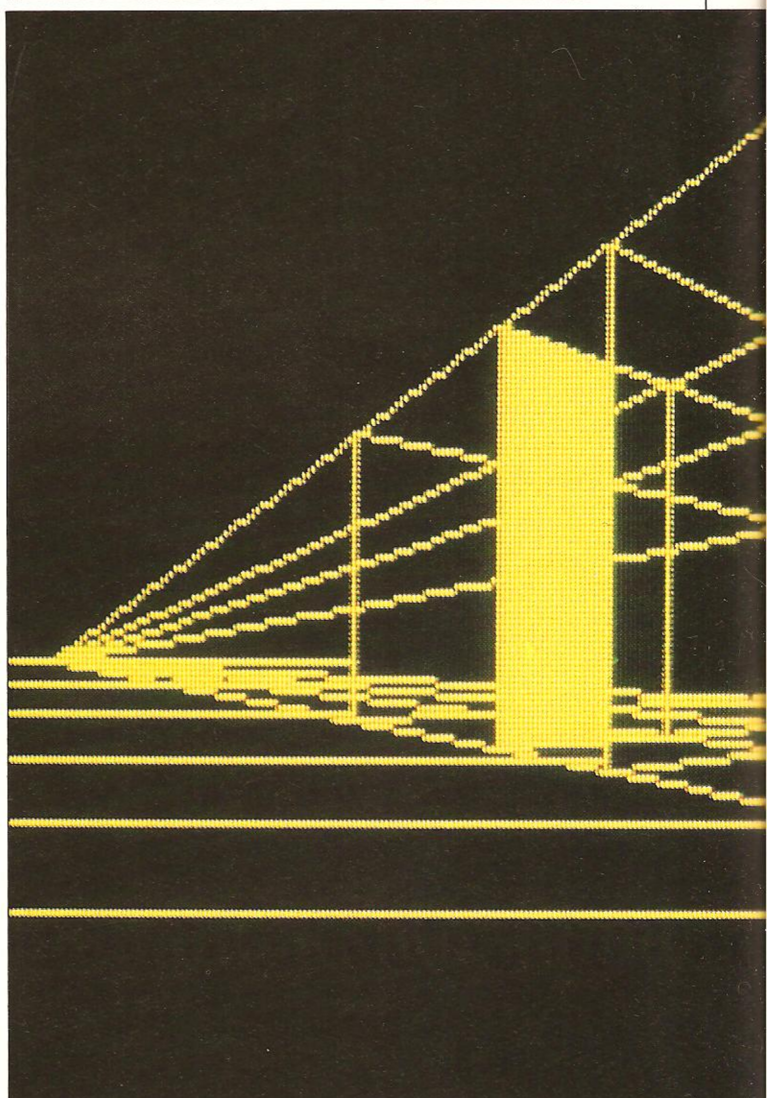
GRAPHICS EDITOR STAGE 2

```
  20 DEF FN f(x,y)=USR 61500
  30 DEF FN g(x,y,p,q)=USR 60700
  40 DEF FN m(x,y)=USR 57700
  80 DEF FN b(x,y,h,v,c,b,f)=USR
 62800
  90 DEF FN c(x,y,h,v,c,b,f)=USR
 62600
  95 DEF FN a(x,y,h,v)=USR 63000
 120 GO SUB 6000: GO TO 1000
 400 PRINT #0;"0-7?"
 410 PAUSE 0: LET a$=INKEY$: IF
a$<"0" OR a$>"7" THEN GO TO 410
 420 INPUT "": LET c=VAL a$
 430 PRINT #0;"Bright?"
 440 PAUSE 0: LET a$=INKEY$: IF
a$<"0" OR a$>"7" THEN GO TO 440
 450 INPUT "": LET b=VAL a$
 460 PRINT #0;"Flash?"
 470 PAUSE 0: LET a$=INKEY$: IF
a$<"0" OR a$>"7" THEN GO TO 470
 480 INPUT "": LET fl=VAL a$
 490 RETURN

scroll?
```

```
 600 LET xc=INT (cx/8): LET yc=2
1-INT (cy/8)
 610 LET xm=INT (mx/8): LET ym=2
1-INT (my/8)
 620 LET x=xc: IF xc>xm THEN LET
 x=xm
 630 LET y=yc: IF yc>ym THEN LET
 y=ym
 640 LET h=ABS (xc-xm)+1: LET v=
ABS (yc-ym)+1
 650 RETURN
1320 IF ke<>68 THEN GO TO 1360
1330 GO SUB cur: GO SUB mar
1340 RANDOMIZE FN f(cx,cy)
1350 GO TO 900
1360 IF ke<>76 THEN GO TO 1400
1370 GO SUB cur: GO SUB mar
1380 RANDOMIZE FN g(mx,my,cx,cy)
1390 GO TO 900
1400 IF ke<>70 THEN GO TO 1440
1410 GO SUB cur: GO SUB mar
1420 RANDOMIZE FN m(cx,cy): LET

scroll?
```

GRAPHICS EDITOR STAGE 2 CONTD.

```
mx=cx: LET my=cy
1430 GO TO 910
1440 IF ke<>66 THEN GO TO 1500
1450 GO SUB cur: GO SUB mar
1460 LET x=cx: IF cx>mx THEN LET
 x=mx
1470 LET y=cy: IF cy>my THEN LET
 y=my
1480 RANDOMIZE FN h(x,y,ABS (cx-
mx),ABS (cy-my))
1490 GO TO 900
1500 GO TO 1730
1730 IF ke<>71 THEN GO TO 1800
1740 IF g=1 THEN GO TO 1780
1750 LET g=1
1760 RANDOMIZE USR 55500
1770 GO TO 1100
1780 RANDOMIZE USR 55531
1790 LET g=0: GO TO 1100
2140 IF ke<>81 THEN GO TO 2190
2150 GO SUB 600
2160 GO SUB 400: GO SUB cur: GO

scroll?
```



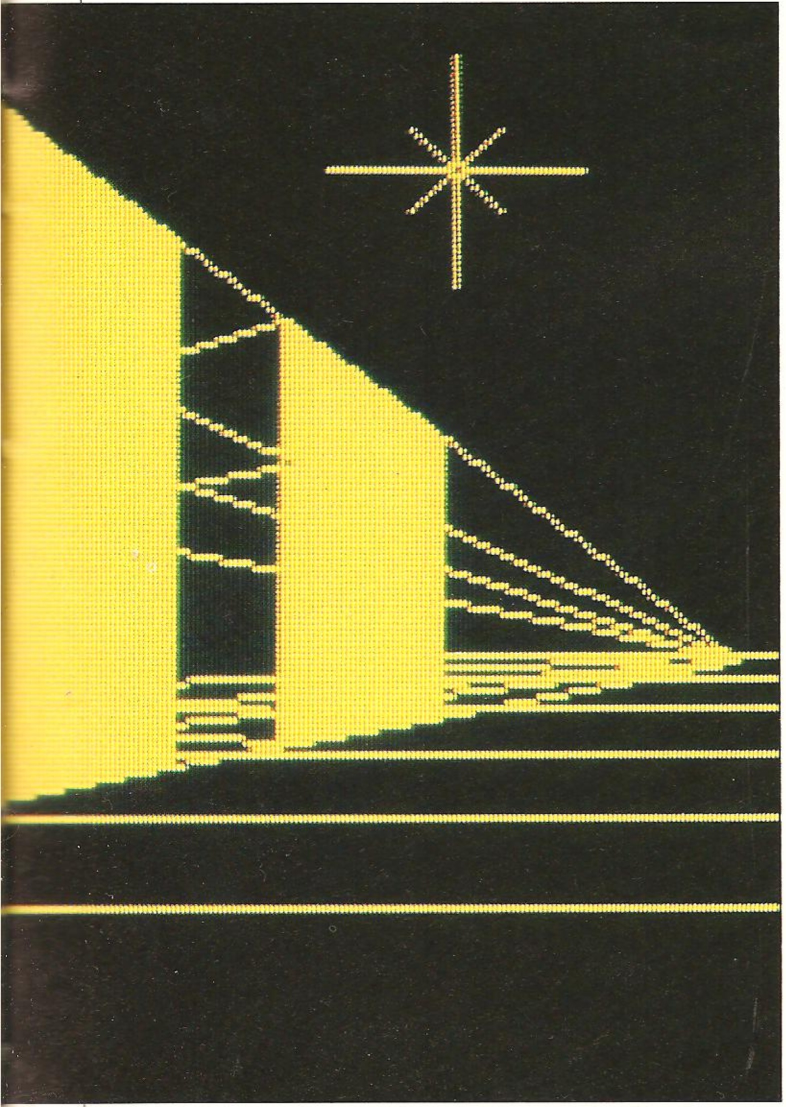
printing character squares in normal and BRIGHT alternately. When key G is pressed, a grid drawn by a machine-code routine appears. The routine is POKEd into memory by the subroutine at lines 6000 (called in line 120). The grid is used to show character borders on the screen while you are drawing a display.

GRAPHICS EDITOR STAGE 2 CONTD.

```
SUB mar
2170 RANDOMIZE FN c(x,y,h,v,c,b,
f()
2180 GO TO 910
2190 IF ke<>87 THEN GO TO 2240
2200 GO SUB 600
2210 GO SUB cur: GO SUB mar
2220 RANDOMIZE FN a(x,y,h,v)
2230 GO TO 910
2240 IF ke<>69 THEN GO TO 1100
2250 GO SUB 600
2260 GO SUB 400: GO SUB cur: GO
SUB mar
2270 RANDOMIZE FN b(x,y,h,v,c,b,
f()
2280 GO TO 910
6000 RESTORE 6030: FOR n=55500 T
O 55542
6010 READ a: POKE n,a
6020 NEXT n
6030 DATA 33,0,88,1,192,2,17,55,
217,237,176

scroll?
```

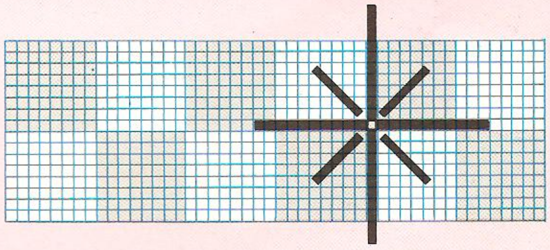
GRAPHICS EDITOR STAGE 2 CONTD.

```
6040 DATA 33,247,216,17,0,88,1,6
4,0,237,176
6050 DATA 33,0,88,1,128,2,237,17
6,201
6060 DATA 33,55,217,17,0,88,1,19
2,2,237,176,201
6070 FOR n=55543 TO 55574 STEP 2
6080 POKE n,120: POKE n+1,56
6090 NEXT n
6100 FOR n=55575 TO 55606 STEP 2
6110 POKE n,56: POKE n+1,120
6120 NEXT n
6130 RETURN




0 OK, 0:1
```

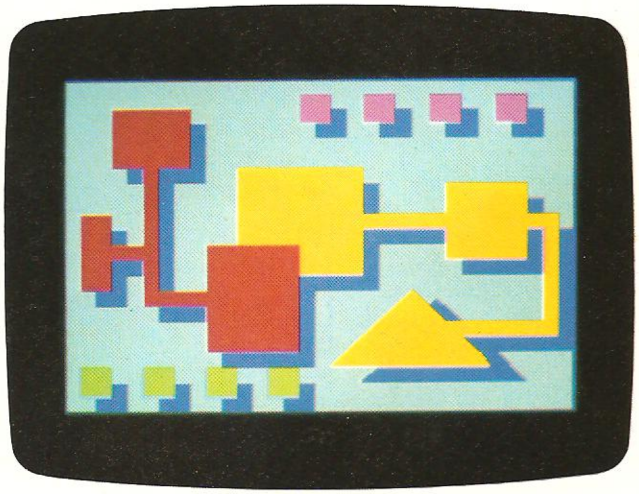


USING THE GRAPHICS EDITOR GRID

The graphics editor grid (CAPS SHIFT and G) is used to display character borders by setting the BRIGHT attributes of alternate characters. The grid does not delete anything currently on the screen. The cursor shows the current pixel position, superimposed on the grid. In the diagram, the cursor is on the leftmost pixel of a character square.



GRAPHICS EDITOR FILL DISPLAY



GRAPHICS EDITOR 3

The third stage of the graphics editor adds routines for drawing circles and triangles. These routines give you many new possibilities for your displays, as you can see from those shown here.

Drawing circles

Line 1500 is the start of the circle routine. Lines 1510 to 1530 enable the user to enter start and finish parameters between 0 to 360 degrees, rather than the machine code's 0-255 parameter values. Lines 1570 and 1580 contain some BASIC error-trapping to prevent a circle being drawn too far off the screen and causing the Spectrum to crash. These lines could be incorporated into any BASIC programs which call the circle routines.

Adding triangles

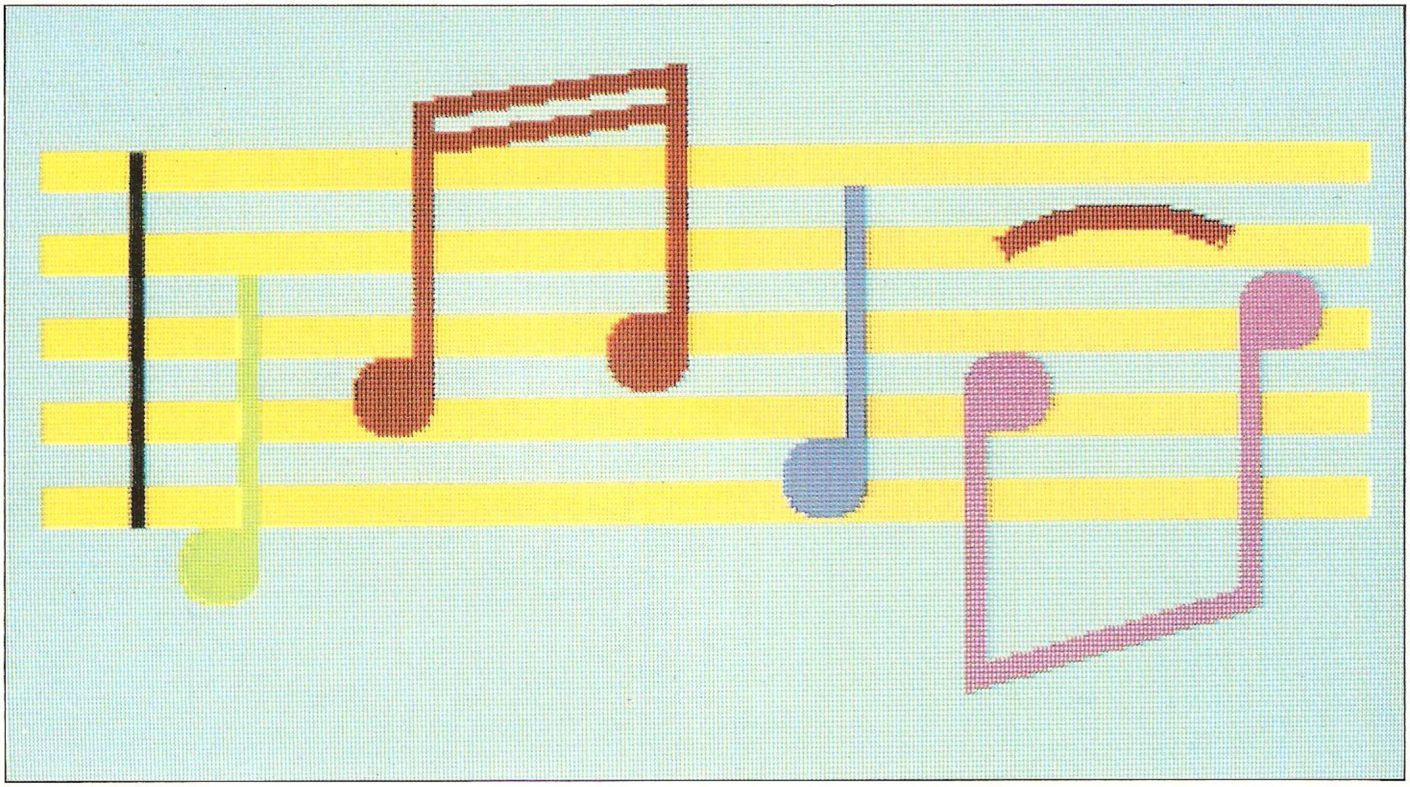
Lines 1610 to 1720 are used to store corner co-ordinates for a triangle before calling the triangle routine, FNi. The parameters of the three corner points are held as variables cx,cy, mx,my and tx,ty.

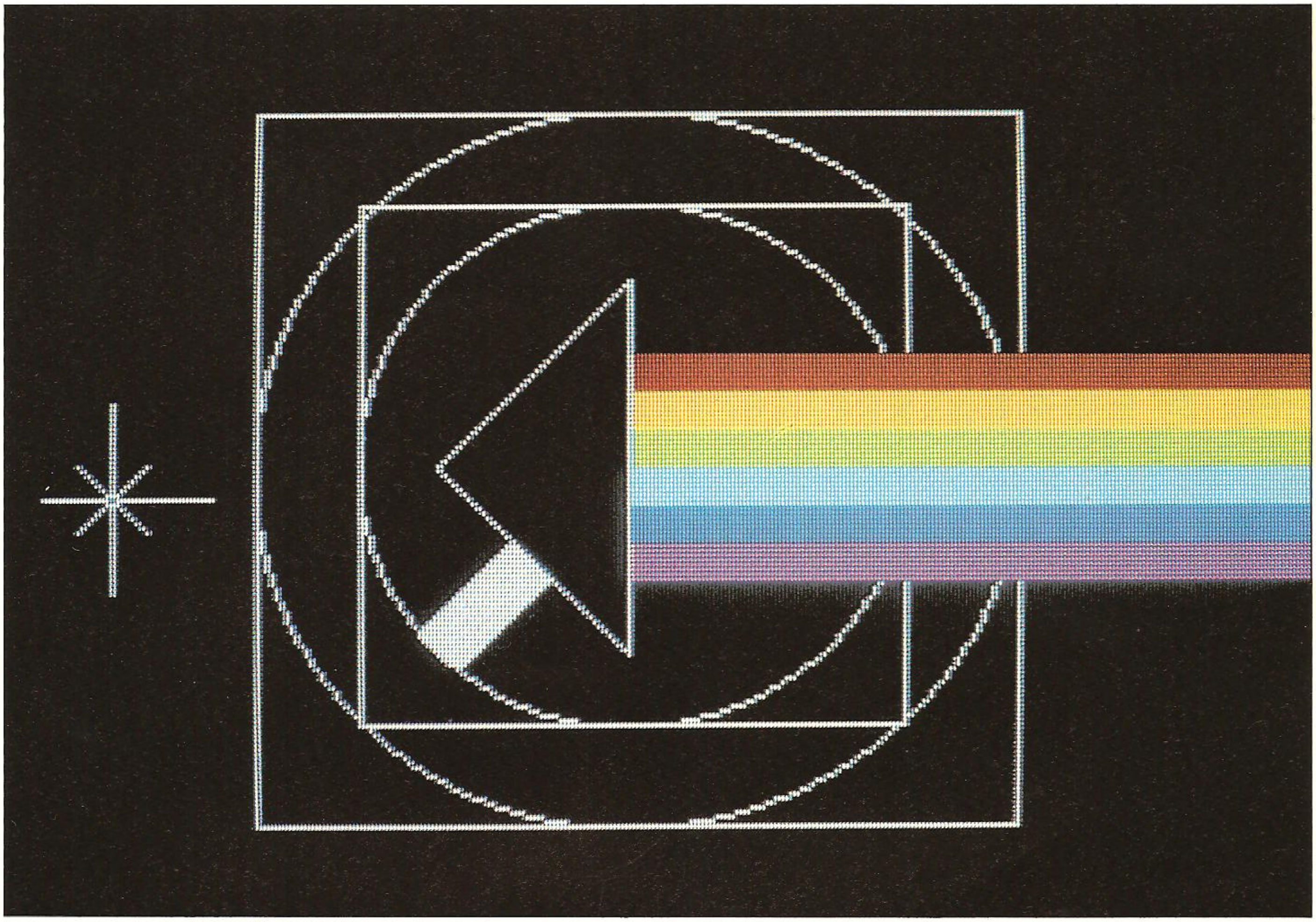
GRAPHICS EDITOR STAGE 3

```
  60 DEF FN j(x,y,r,s,f)=USR 589
00
  70 DEF FN i(x,y,p,q,r,s)=USR 6
0300
1500 IF ke<>67 THEN GO TO 1610
1510 INPUT "s = ";s;"  f = ";f
1520 IF s<0 OR f<0 OR s>360 OR f
>360 OR s<>INT s OR f<>INT f THE
N GO TO 1510
1530 LET s=INT (255*(s/360)): LE
T f=INT (255*(f/360))
1540 GO SUB cur
1550 LET x=ABS (cx-mx): LET y=AB
S (cy-my)
1560 LET r=INT (SQR (x↑2+y↑2)+0.
5)
1570 IF r>255 THEN BEEP 2,3: INP
UT "": GO SUB cur: GO TO 1100
1580 IF r+mx>275 OR mx-r<-20 OR
r+my>195 OR my-r<-20 THEN LET r=
275: GO TO 1570
1590 RANDOMIZE FN j(mx,my,r,s,f)

scroll?
```

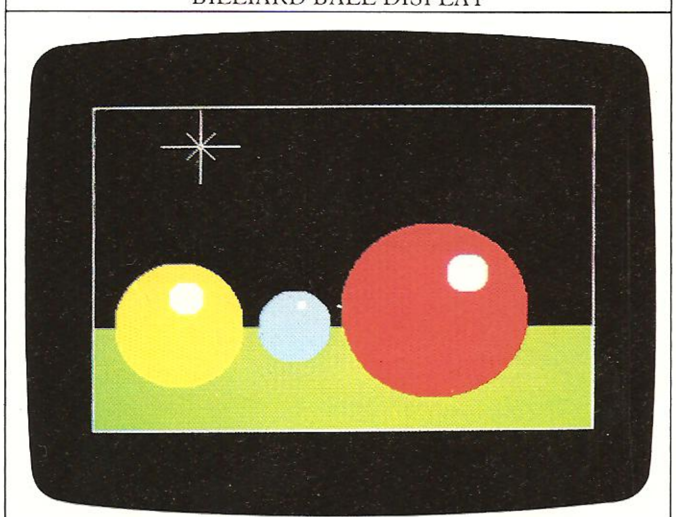
GRAPHICS EDITOR STAGE 3 CONTD.

```
1600 GO SUB cur: GO TO 1100
1610 IF ke<>84 THEN GO TO 1730
1620 LET p=mx: LET q=my: LET mx=
cx: LET my=cy: GO SUB mar
1630 LET a$=INKEY$: IF a$="" THE
N GO TO 1630
1640 LET a=CODE a$
1650 GO SUB cur
1660 LET cx=cx+1*(a=9 AND cx<255
)-1*(a=8 AND cx>0)
1670 LET cy=cy+1*(a=11 AND cy<17
5)-1*(a=10 AND cy>0)
1680 GO SUB cur
1690 IF a<>84 THEN GO TO 1630
1700 GO SUB cur: LET tx=mx: LET
ty=my: GO SUB mar: LET mx=p: LET
 my=q: GO SUB mar
1710 RANDOMIZE FN i(cx,cy,mx,my,
tx,ty)
1720 GO TO 900

0 OK, 0:1
```

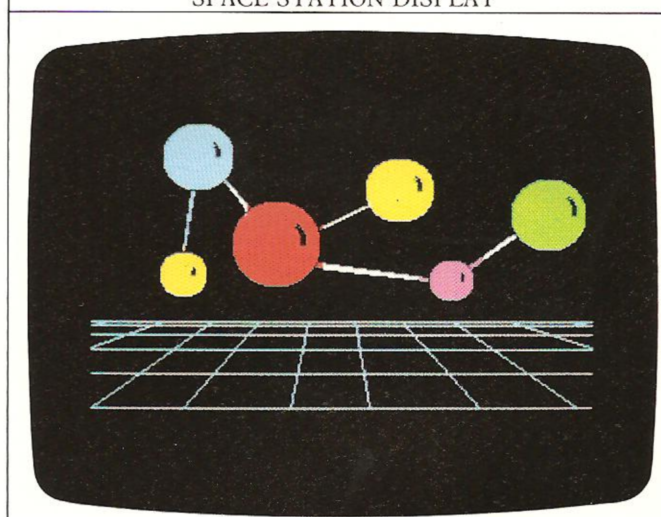




BILLIARD BALL DISPLAY



SPACE STATION DISPLAY



Lines 1660 and 1670 take advantage of Spectrum BASIC's facility for writing conditional statements in an abbreviated form. Line 1660 could be rewritten as:

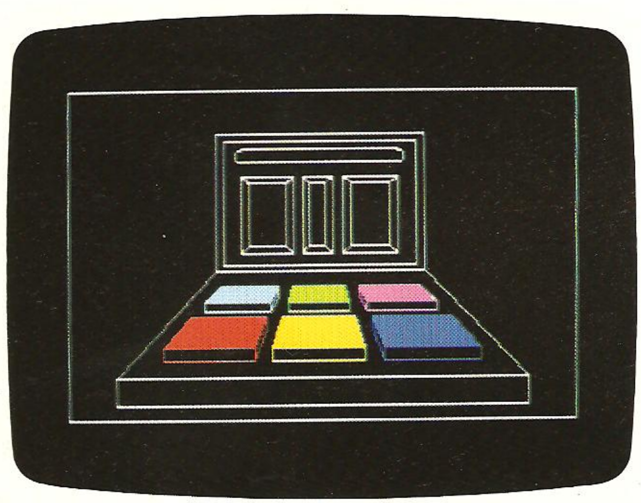
```
1660 IF a=9 AND cx<255 THEN LET cx=cx+1
1665 IF a=8 AND cx>0 THEN  LET cx=cx-1
```

These lines are used to move the cursor in BASIC to the third corner of the triangle, by calculating new values for cx and cy as a key is pressed. You will notice from the movement of this cursor how much slower BASIC movement is than the usual cursor speed, which is carried out by machine code. This speed advantage alone would be sufficient justification for using machine-code routines rather than BASIC.

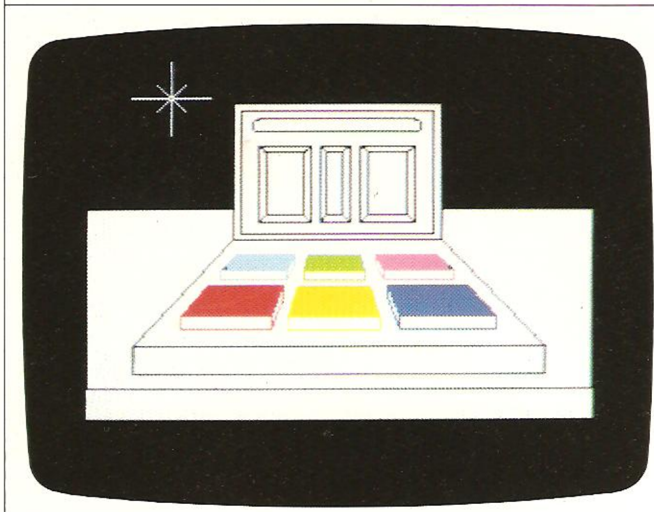
GRAPHICS EDITOR 4

The designs on this page show one method of producing a typical display. One general point is worth noting before beginning any large-scale graphics editor display. Each photograph of the cocktail display represents a point where the screen was SAVEd before going further to add more details. The reason for this is simple: even when you have a little experience with the editor, it is easy to ruin a display by adding an unintended line, or by filling a shape that is not totally

BLACK PAINTBOX DISPLAY



PAINTBOX DISPLAY



enclosed. Rather than risk losing an entire display, it is sensible to take a few seconds and SAVE what has been drawn before continuing.

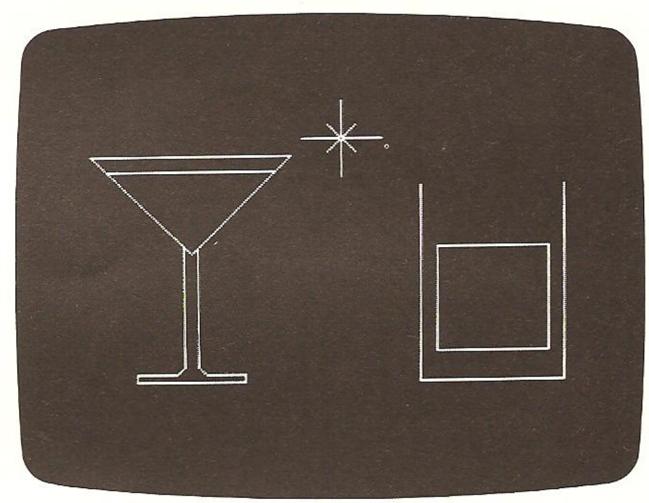
The paintbox displays

The paintbox displays above show the difference in effect which can be obtained by drawing in white ink on a black background, rather than using black ink on a white background. This effect can be obtained by changing the initial graphics editor screen to black INK and white PAPER colours. A simple change like this can be very effective.

Building up a display

The cocktail display shown here is an example of how a graphics editor display can be developed in stages. Stage one of the display uses only lines, squares and triangles.

COCKTAIL DISPLAY 1



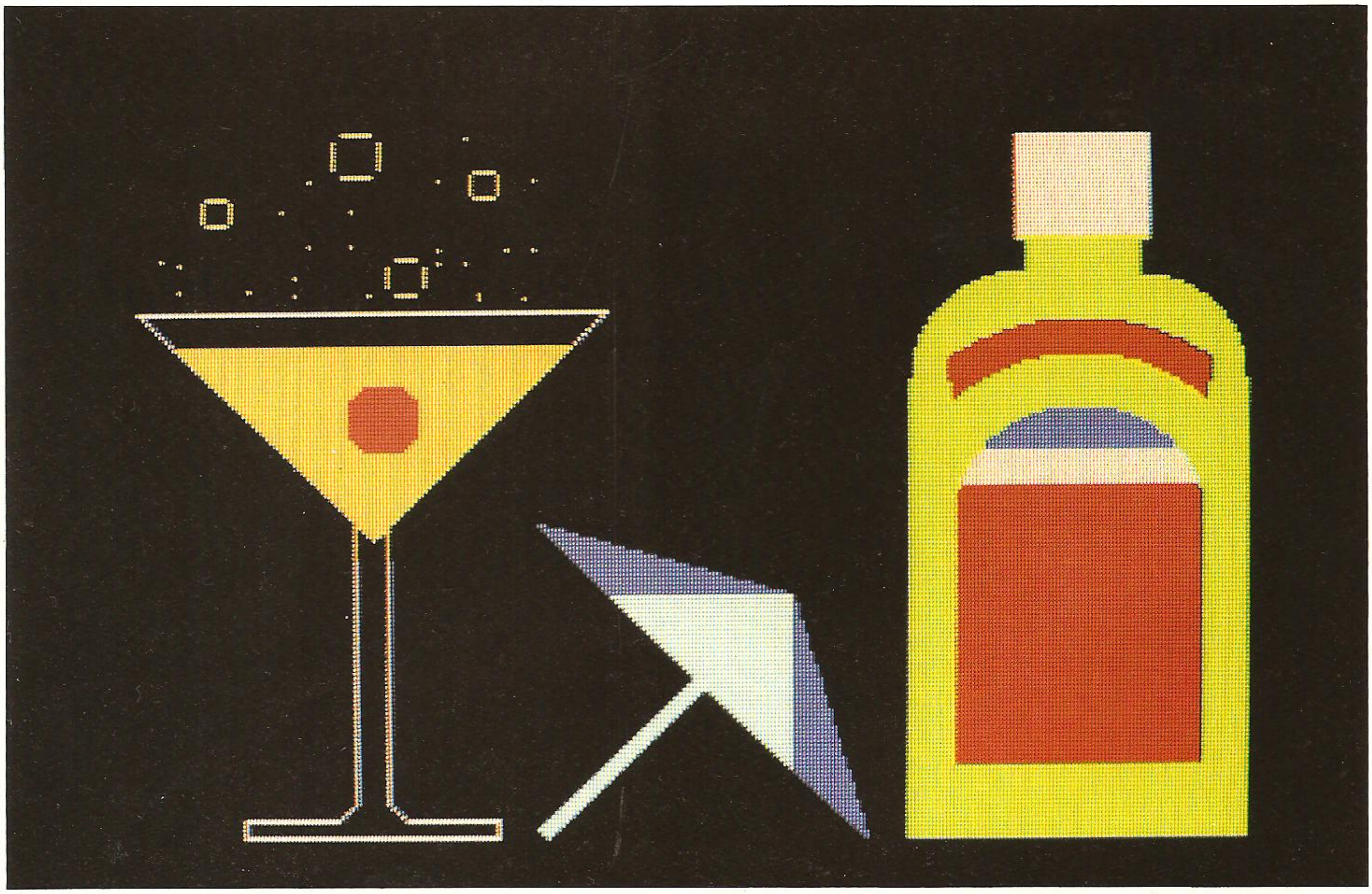
COCKTAIL DISPLAY 2

Even at this early stage, however, the design has been planned so that there will be no problem with character borders when colour is added. The position of character borders can of course be checked by using the grid (CAPS SHIFT and G). The grid does not delete anything which has been drawn, so it is a simple matter to flick between the grid and the normal screen as necessary at this stage to ensure that lines and points are drawn in the correct position on either side of a character border.

The second stage uses circles and arcs to draw, for example, the cherry in the glass. Because of the relatively low resolution of the Spectrum screen display, a small circle such as that which forms the cherry may not be completely enclosed. Four single points were plotted on this circle to prevent the INK from "leaking" when the shape is filled. The umbrella in stage three was also drawn to take advantage of character borders when filled with colour. The colour change on the umbrella lies along a horizontal and vertical character border, although it appears from the display to be diagonal. The picture was completed by filling areas and then adding colours. When drawing a complex display, it is always best to keep the filling and colouring operations until last. Remember also that colours should not be added while you are using the grid.

COCKTAIL DISPLAY 3

COCKTAIL DISPLAY 4



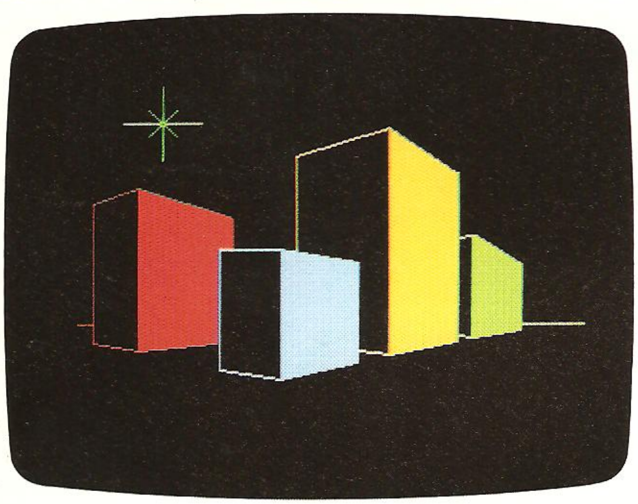
GRAPHICS EDITOR 5

Text is added by lines 1800-1890 of the program. Line 1810 takes the current pixel position (variables x,y) and converts these to character co-ordinates. This is because text is printed in character positions rather than using pixel co-ordinates. Line 1820 deletes the cursors, and line 1830 prints a flashing text prompt at the character position. Text is then entered in lines 1840-1870, which include BASIC controls for deleting mistakes in keying, and ending the text string when ENTER is pressed.

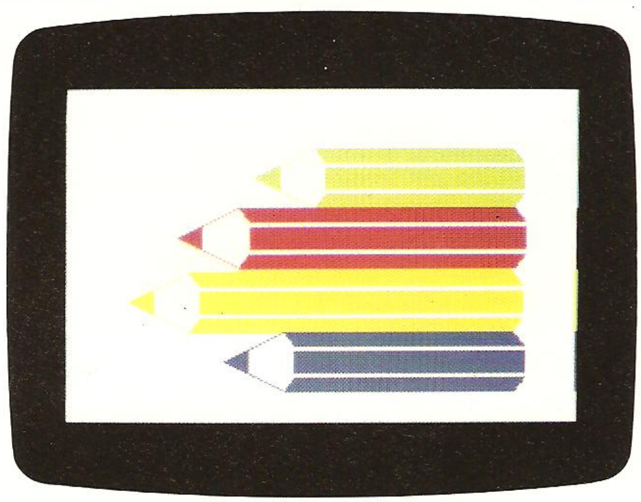
Attribute editing

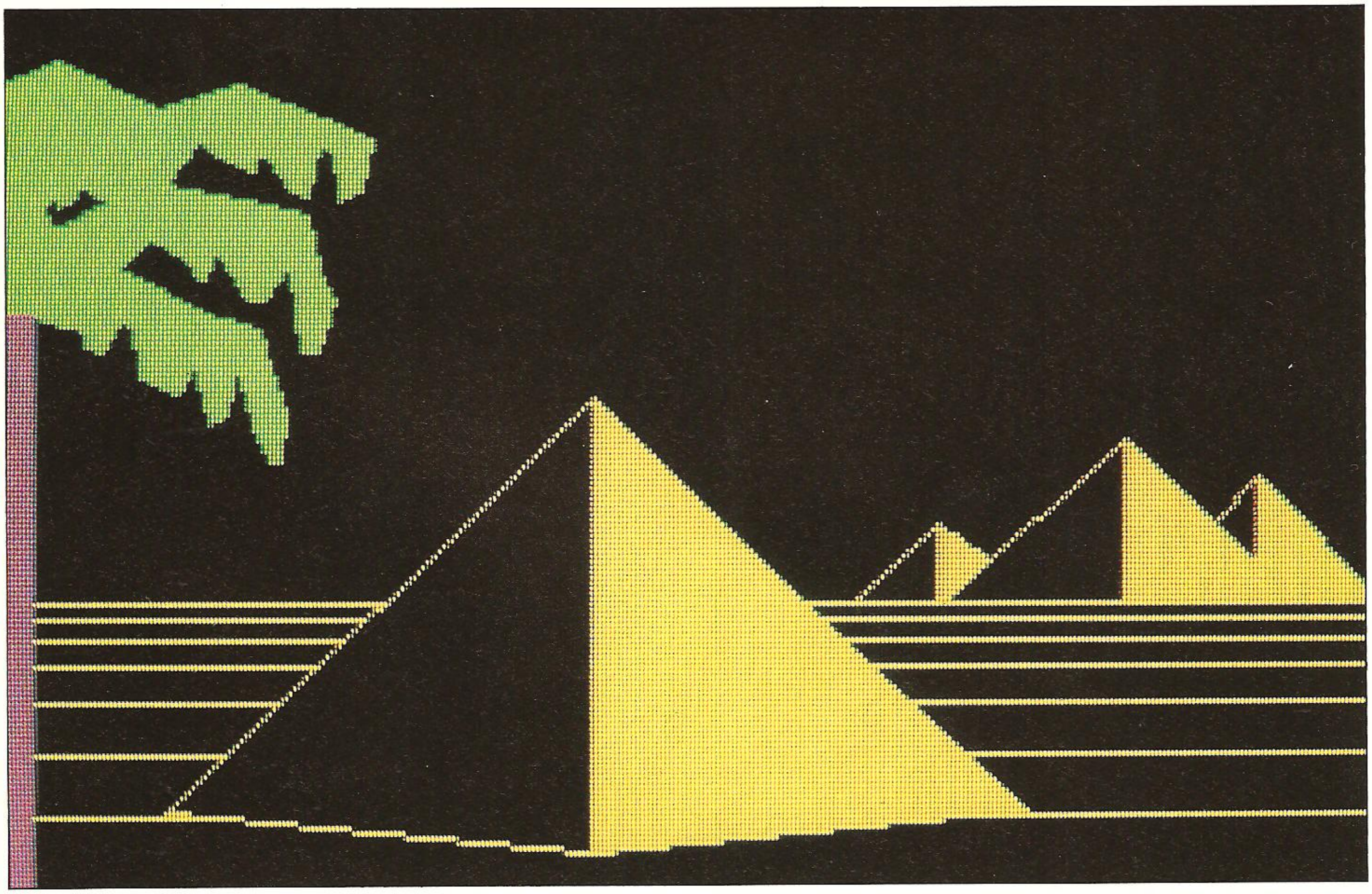
Lines 1900-2030 give you the option of setting INK,

BOXES DISPLAY



PENCILS DISPLAY





PAPER, BRIGHT and FLASH attribùtes of any character square on the screen. As with text, the current pixel position is converted to character co-ordinates (held as variables lin, col) for this routine. Lines 1970 and 1980 simply move the cursor (a flashing character square, printed in line 1960) onto the next line or column when the end of either is reached.

Saving and loading screens

Finally, lines 2040-2130 of the program enable you to SAVE and LOAD your displays, using the Spectrum SCREEN$ command. An advantage of this method is that you should be able to load onto the graphics editor the title display of many commercial games, since these programs often begin with a SCREEN$ display. This will enable you to make your own versions of these screens.

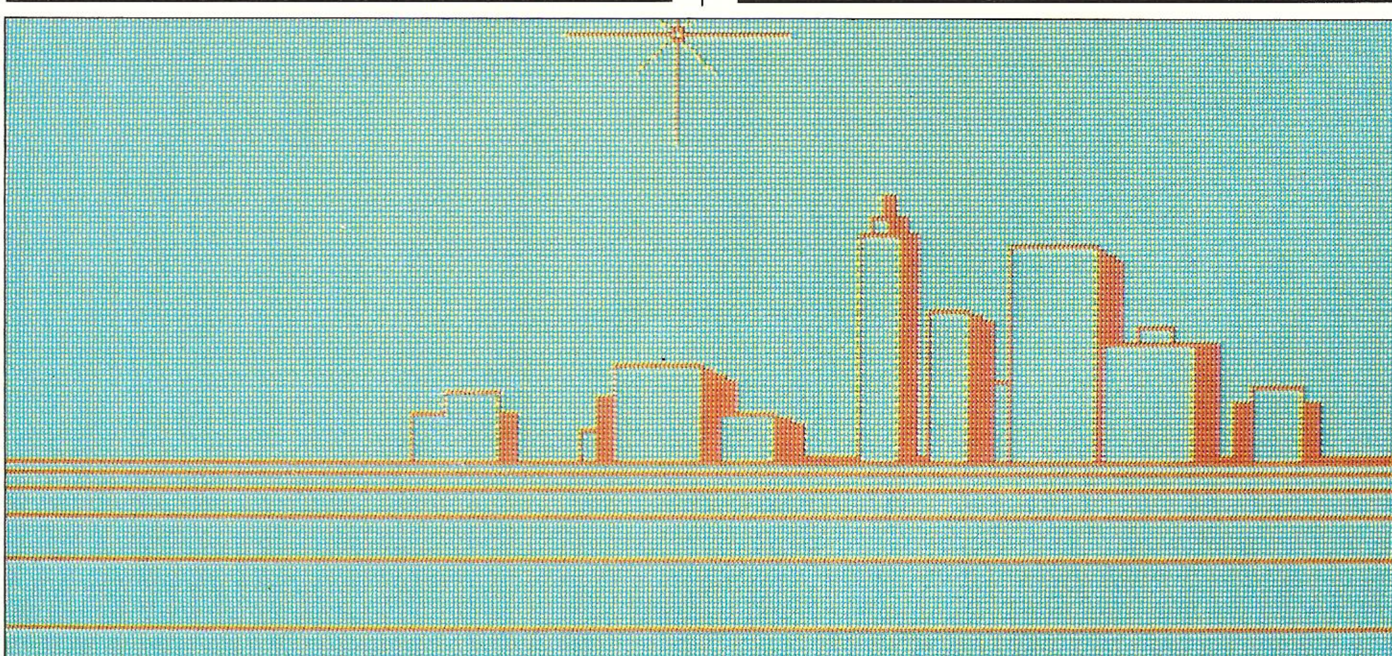
GRAPHICS EDITOR STAGE 5

```
 500 PRINT #0;"0-7 ?"
 510 LET a$=INKEY$: IF a$="" THE
N GO TO 510
 520 LET asc=CODE a$
 530 IF asc<47 OR asc>55 THEN GO
 TO 510
 540 INPUT ""
 550 RETURN
1800 IF ke<>88 THEN GO TO 1900
1810 LET x=INT (cx/8): LET y=21-
INT (cy/8)
1820 GO SUB cur: GO SUB mar
1830 PRINT AT y,x; OVER 1; FLASH
 1;">";CHR$ 8;
1840 PAUSE 0: LET a$=INKEY$: IF
a$="" OR a$<CHR$ 12 THEN GO TO 1
840
1850 IF a$=CHR$ 13 THEN GO TO 18
80
1860 IF a$=CHR$ 12 THEN PRINT CH
R$ 8+"  "+CHR$ 8+CHR$ 8;: LET a$
=""

scroll?
```

GRAPHICS EDITOR STAGE 5 CONTD.

```
1870 PRINT a$; OVER 1; FLASH 1;"
>";CHR$ 8;: GO TO 1840
1880 PRINT OVER 1;">": GO TO 910
1890 GO TO 910
1900 IF ke<>65 OR g=1 THEN GO TO
 2040
1910 GO SUB cur: GO SUB mar
1920 LET col=INT (cx/8): LET lin
=21-INT (cy/8)
1930 PRINT AT lin,col; INK ink;
PAPER pap; OVER 1; FLASH 1;" "
1940 LET a$=INKEY$: IF a$="" THE
N GO TO 1940
1950 LET asc=CODE a$
1960 PRINT AT lin,col; INK ink;
PAPER pap; BRIGHT b; FLASH fl; O
VER 1;" "
1970 LET col=col+1*(col<31 AND a
sc=9)-1*(col>0 AND asc=8)
1980 LET lin=lin+1*(lin<21 AND a
sc=10)-1*(lin>0 AND asc=11)
1990 IF asc=13 THEN GO TO 910

scroll?
```

```
2000 IF asc=73 THEN GO SUB 500:
LET ink=asc-48
2010 IF asc=80 THEN GO SUB 500:
LET pap=asc-48
2020 IF asc=79 THEN GO SUB 430
2030 GO TO 1930
2040 IF ke<>83 THEN GO TO 2090
2050 INPUT "SAVE ";p$
2060 GO SUB cur: GO SUB mar
2070 INPUT "": SAVE p$SCREEN$
2080 GO TO 910
2090 IF ke<>74 THEN GO TO 2140
2100 INPUT "LOAD ";p$
2110 GO SUB cur: GO SUB mar
2120 LOAD p$SCREEN$
2130 GO TO 910

0 OK, 0:1
```



MULTIPLE LINES

When using the line-draw routine (FNg), you must specify both the start and the end points for each line drawn. Where only a few lines are involved, this is not difficult, but if you are drawing a complicated shape with many lines joined together, you will find yourself continually specifying each point twice: once as the end of a line, and then again as the start of the next line. This can be avoided by using the multiple line-draw routine (FNo). This routine takes a series of co-ordinates which have been stored in memory, and joins each point in turn to the one before.

Having drawn your complex series of lines so quickly, you now need a way of wiping them off without damaging the rest of the display. For this reason an Exclusive/OR version of the routine is also included (FNp). This routine is the same as the multiple line-draw routine, but plots XOR lines. The XOR routine will enable you to repeatedly draw and undraw a whole series of lines on the screen in a few seconds.

Putting the points in memory

Before using the routines, you must specify the co-ordinates of the points to be linked, which are stored in a buffer. In operation, the routine takes a point from memory and joins it to the next point, and continues until it reaches a y co-ordinate of 255. Points can be POKEd into memory by using a loading routine such as the one below, which accepts pairs of co-ordinates:

```
10 LET n = 57200
20 INPUT "x = " ; x : INPUT "y = "; y
30 POKE n,y : POKE n + 1,x
40 LET n = n + 2
50 GOTO 20
```

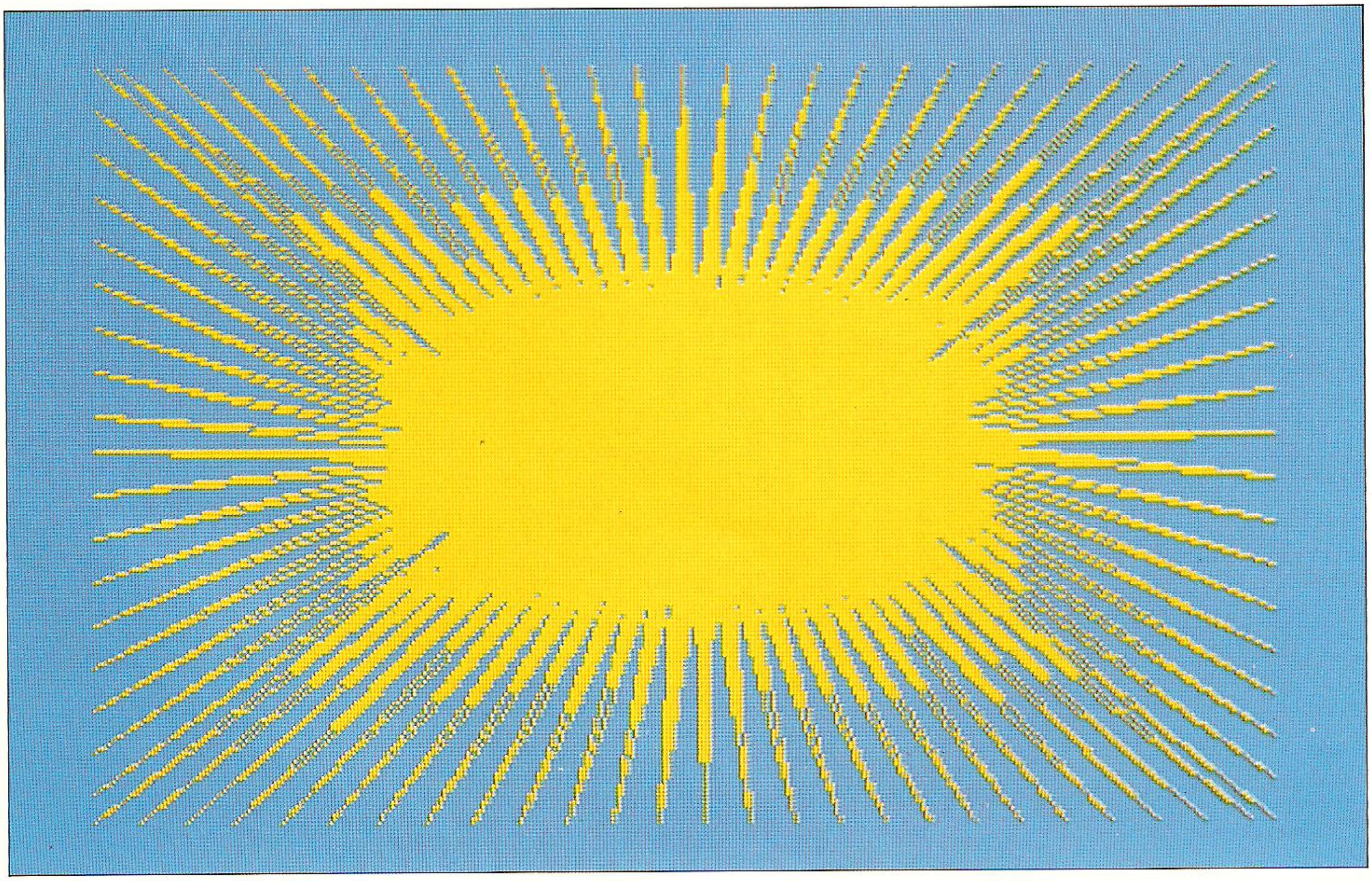
400 bytes from 57200 are reserved in memory for this purpose, so you can draw 199 lines with the routines.

These routines are especially useful for plotting the same shape on the screen repeatedly, since points once stored in memory can be called by the routine almost instantaneously.

MULTILINE PROGRAM

00:22 **seconds**

How the program works An explosion effect is obtained by drawing a continuous line joining points on the edge of the screen with points on an ellipse. The shape is then filled. **Lines 140-330** POKE into memory the values of points around the edges of the screen. **Lines 350 to 400** POKE into memory points on an ellipse. **Lines 410 and 420** POKE values of 255 to complete the table of points.



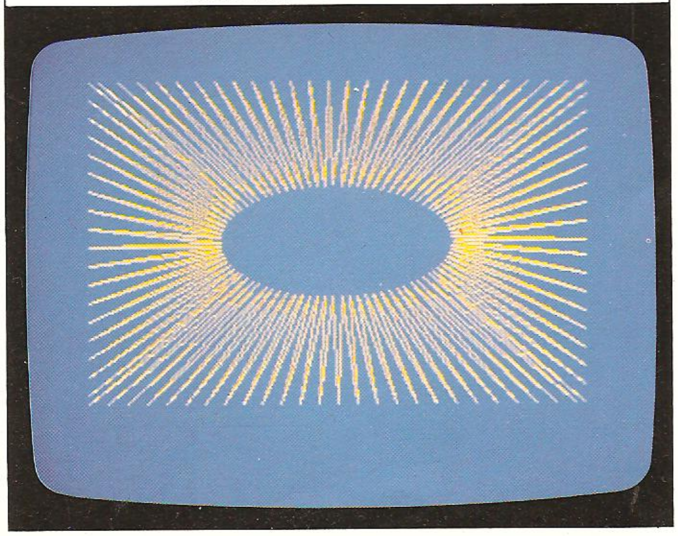
MULTILINE PROGRAM

```
  10 DEF FN o()=USR 57100
  20 DEF FN m(x,y)=USR 57700
 100 BORDER 1: PAPER 1: INK 6: C
LS
 110 LET x1=128: LET y1=88
 120 FOR s=10 TO 100 STEP 10
 130 LET z=57200: LET l=0: LET a
=0
 140 FOR i=88 TO 175 STEP s
 150 POKE z+2+l,i
 160 POKE z+3+l,255: LET l=l+4
 170 NEXT i
 180 FOR i=0 TO 255 STEP s
 190 POKE z+2+l,175
 200 POKE z+3+l,255-i: LET l=l+4
 210 NEXT i
 220 FOR i=0 TO 175 STEP s
 230 POKE z+2+l,175-i
 240 POKE z+3+l,0: LET l=l+4
 250 NEXT i
 260 FOR i=0 TO 255 STEP s
 270 POKE z+2+l,0: POKE z+3+l,i

scroll?
```

```
 280 LET l=l+4
 290 NEXT i
 300 FOR i=0 TO 87 STEP s
 310 POKE z+2+l,i
 320 POKE z+3+l,255: LET l=l+4
 330 NEXT i
 340 LET ad=8*PI/l
 350 FOR i=0 TO l+4 STEP 4
 360 LET x=x1+INT (60*COS a)
 370 LET y=y1+INT (30*SIN a)
 380 POKE z+i,y: POKE z+i+1,x
 390 LET a=a+ad
 400 NEXT i
 410 POKE z+l+2,255
 420 POKE z+l+3,255
 430 RANDOMIZE FN o()
 440 PAUSE 100
 450 RANDOMIZE FN m(x1,y1)
 460 PAUSE 100: CLS
 470 NEXT s

0 OK, 0:1
```

The multiline program draws a long sequence of lines, which are then filled by the fill routine. Both routines must be in memory for the program to RUN.

DISPLAY BEFORE FILLING



FNo

MULTIPLE LINE-DRAW ROUTINE

Start address 57100 **Length** 40 bytes
Other routines called Line-draw routine (FNg).
What it does Draws a series of lines on the screen, from a specified list of co-ordinates.

Using the routine Co-ordinates of lines to be plotted are stored in a table at memory location 57200. Up to 200 lines can be stored in this area of memory. Points in the table must be specified by the y co-ordinate (0<=y<=175) followed by the x co-ordinate (o<=x<=255), rather than the other way round.
To stop the routine POKE specify a y co-ordinate of 255. The routine will continues plotting points until it reaches this y co-ordinate; if you omit the 255, the routine will continue to plot points using whatever numbers are in memory after the co-ordinate table.

ROUTINE LISTING

```
8100 LET b=57100: LET l=35: LET
z=0: RESTORE 8110
8101 FOR i=0 TO l-1: READ a
8102 POKE (b+i),a: LET z=z+a
8103 NEXT i
8104 LET z=INT (((z/l)-INT (z/l)
)*l)
8105 READ a: IF a<>z THEN PRINT
"??": STOP

8110 DATA 33,112,223,94,35
8111 DATA 86,237,83,26,237
8112 DATA 35,126,254,255,32
8113 DATA 1,201,95,35,86
8114 DATA 43,229,42,26,237
8115 DATA 205,51,237,225,24
8116 DATA 228,0,0,0,0
8117 DATA 17,0,0,0,0
```

FNp

MULTIPLE XOR-LINE ROUTINE

Start address 57000 **Length** 20 bytes
Other routines called Multiple line-draw routine (FNo).
What it does Draws a series of Exclusive/OR lines on the screen, using points specified in a table.

Using the routine This routine works in the same way as the multiple line-draw routine, but can be called twice with the same table of co-ordinates to erase the lines drawn. Remember as before to POKE points in the order y,x. Co-ordinates are stored in memory from location 57200, and the final point must be followed by a y co-ordinate of 255.

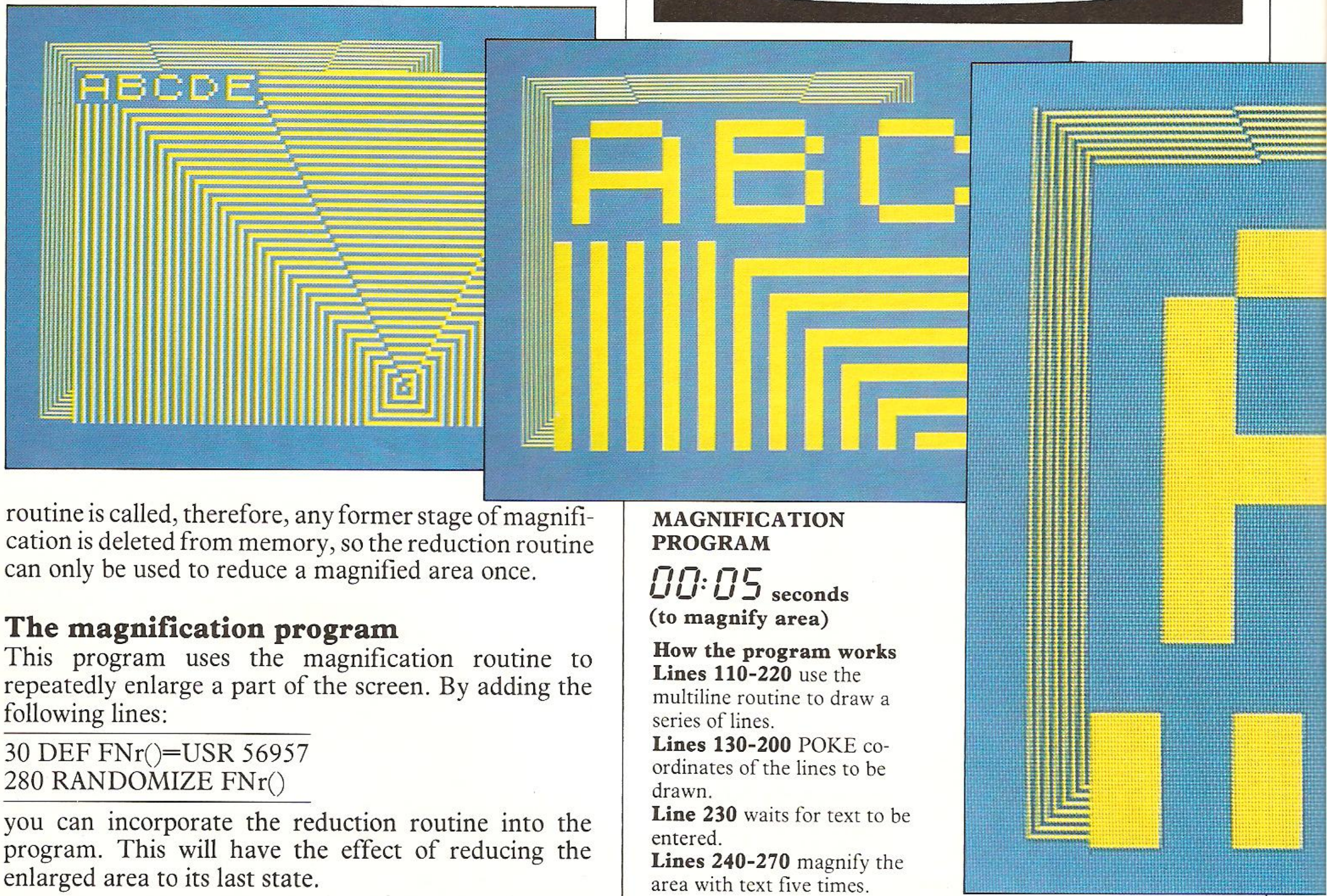
ROUTINE LISTING

```
8150 LET b=57000: LET l=15: LET
z=0: RESTORE 8160
8151 FOR i=0 TO l-1: READ a
8152 POKE (b+i),a: LET z=z+a
8153 NEXT i
8154 LET z=INT (((z/l)-INT (z/l)
)*l)
8155 READ a: IF a<>z THEN PRINT
"??": STOP

8160 DATA 62,168,50,223,237
8161 DATA 205,12,223,62,176
8162 DATA 50,223,237,201,0
8163 DATA 13,0,0,0,0
```

MAGNIFICATION AND REDUCTION 1

One of the most dramatic uses for machine code is to magnify a portion of the Spectrum screen. The principle behind magnification is straightforward. To double the size of a single byte, for example 00110010 (a value of 50 in decimal), simply rotate it left one bit, thus making 01100100 (which is equivalent to 100 in decimal). Using this principle of doubling, you can magnify whole sections of a screen. The magnification routine, FNq, given here, is based on this idea. The routine simply requires you to specify the screen area to be enlarged.

The magnification routine is accompanied by a reduction routine, FNr, which is used to reduce an already-magnified area. The reduction routine actually forms part of the magnification routine, with the start address of the second routine being a call which "hooks" into the main routine. The reduction routine restores the screen as it was before the magnification, and works by the magnification routine saving the entire screen each time it is called before magnifying any area; the reduction routine simply displays this area from memory on the screen. Each time the magnification routine is called, therefore, any former stage of magnification is deleted from memory, so the reduction routine can only be used to reduce a magnified area once.



The magnification program

This program uses the magnification routine to repeatedly enlarge a part of the screen. By adding the following lines:

```
30 DEF FNr()=USR 56957
280 RANDOMIZE FNr()
```

you can incorporate the reduction routine into the program. This will have the effect of reducing the enlarged area to its last state.

Remember that the magnification program calls the multiline routine, FNo, to draw the background pattern; this routine must also be present in memory for the program to RUN correctly.

MAGNIFICATION PROGRAM

```
  10 DEF FN o()=USR 57100
  20 DEF FN q(x,y,h,v)=USR 56700
 100 BORDER 1: PAPER 1: INK 6: C
LS
 110 LET x1=2: LET x2=174
 120 LET y1=174: LET y2=2
 130 FOR i=57200 TO 57598 STEP 8
 140 POKE i,y1: POKE i+1,x1
 150 POKE i+2,y2: POKE i+3,x1
 160 POKE i+4,y2: POKE i+5,x2
 170 POKE i+6,y1: POKE i+7,x2
 180 LET y1=y1-2: LET x1=x1+2
 190 LET y2=y2+2: LET x2=x2-2
 200 NEXT i
 210 POKE 57548,255
 220 RANDOMIZE FN o()
 230 INPUT a$: PRINT AT 2,2;a$
 240 FOR i=1 TO 5
 250 RANDOMIZE FN q(2,2,15,10)
 260 PAUSE 100
 270 NEXT i
 280 GO TO 230

0 OK, 0:1
```

MAGNIFICATION PROGRAM

00:05 seconds (to magnify area)

How the program works

Lines 110-220 use the multiline routine to draw a series of lines.

Lines 130-200 POKE co-ordinates of the lines to be drawn.

Line 230 waits for text to be entered.

Lines 240-270 magnify the area with text five times.

FNq

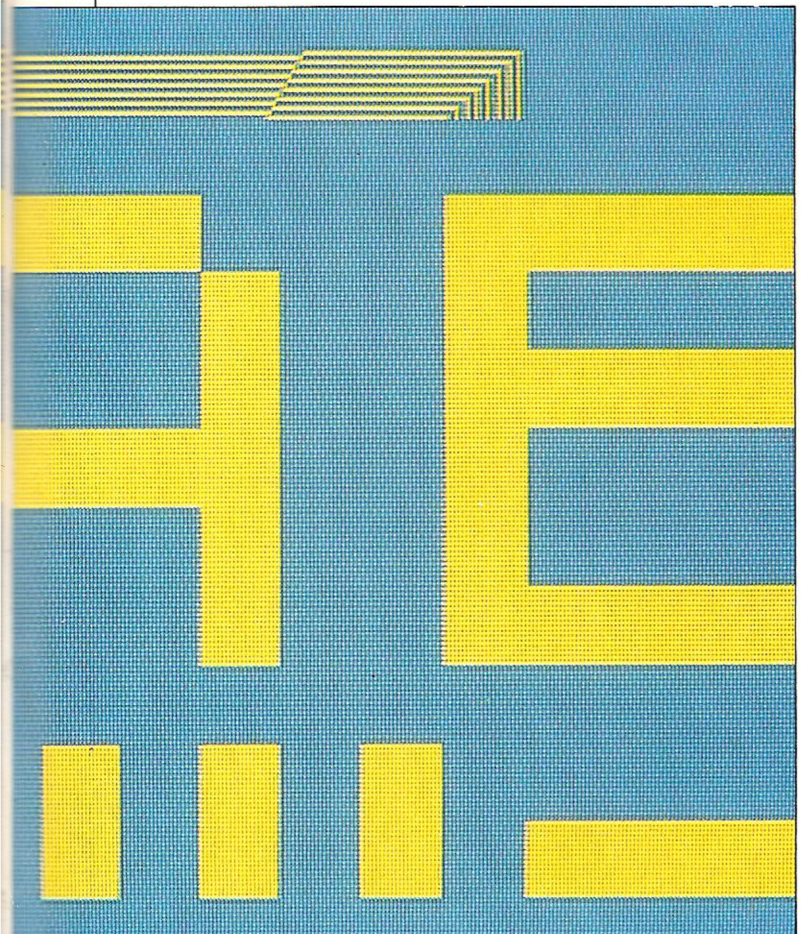
MAGNIFICATION ROUTINE

Start address 56700 **Length** 290 bytes
What it does Magnifies a specified screen area to double its previous size.

Using the routine The routine uses character co-ordinates, as in the window ink and paper routines (FNb and FNc), rather than pixel co-ordinates. Remember that these start from the top left-hand corner of the screen. The routine can be used to magnify the same area repeatedly, increasing the enlargement each time.
Since an area is doubled in size by the routine it is easy when magnifying to make part of the area disappear off the screen. To prevent crashes occurring, use the tests in the parameter table to make sure that the area when enlarged will not be off the screen. These tests can be incorporated into your programs.

ROUTINE PARAMETERS

DEF FN q(x,y,h,v)	
x,y	specify top left-hand corner of area to be magnified (x<32, y<22)
h,v	specify horizontal and vertical sizes of area (x+(2*h)<32,y+(2*v)<22)



LISTING FOR BOTH ROUTINES

```
8200 LET b=56700: LET l=285: LET
 z=0: RESTORE 8210
8201 FOR i=0 TO l-1: READ a
8202 POKE (b+i),a: LET z=z+a
8203 NEXT i
8204 LET z=INT (((z/l)-INT (z/l)
)*l)
8205 READ a: IF a<>z THEN PRINT
"??": STOP

8210 DATA 42,11,92,1,4
8211 DATA 0,9,86,14,8
8212 DATA 9,94,237,83,137
8213 DATA 222,9,126,50,140
8214 DATA 222,9,126,50,139
8215 DATA 222,58,138,222,71
8216 DATA 58,140,222,128,230
8217 DATA 224,40,6,62,31
8218 DATA 144,50,140,222,58
8219 DATA 137,222,71,58,139

8220 DATA 222,128,214,22,56
8221 DATA 6,62,21,144,50
8222 DATA 139,222,237,91,137
8223 DATA 222,123,230,24,246
8224 DATA 64,103,123,230,7
8225 DATA 183,31,31,31,31
8226 DATA 130,111,34,141,222
8227 DATA 17,0,64,167,237
8228 DATA 82,17,0,118,25
8229 DATA 34,143,222,205,113

8230 DATA 222,42,141,222,237
8231 DATA 91,143,222,58,139
8232 DATA 222,71,197,1,2
8233 DATA 4,197,205,20,222
8234 DATA 193,16,249,42,141
8235 DATA 222,205,79,222,34
8236 DATA 141,222,6,4,13
8237 DATA 32,235,237,91,143
8238 DATA 222,205,89,222,237
8239 DATA 83,143,222,193,16

8240 DATA 217,201,58,140,222
8241 DATA 71,34,145,222,237
8242 DATA 83,147,222,197,205
8243 DATA 54,222,193,16,249
8244 DATA 42,145,222,237,91
8245 DATA 147,222,229,205,99
8246 DATA 222,225,36,36,20
8247 DATA 201,26,1,2,4
8248 DATA 197,245,175,119,241
8249 DATA 23,245,203,22,241

8250 DATA 203,22,16,247,35
8251 DATA 193,13,32,237,19
8252 DATA 201,62,32,133,111
8253 DATA 208,62,8,132,103
8254 DATA 201,62,32,131,95
8255 DATA 208,62,8,130,87
8256 DATA 201,58,140,222,203
8257 DATA 39,71,126,36,119
8258 DATA 37,35,16,249,201
8259 DATA 33,0,64,17,0

8260 DATA 118,1,0,26,237
8261 DATA 176,201,33,0,118
8262 DATA 17,0,64,1,0
8263 DATA 26,237,176,201,2
8264 DATA 2,5,10,130,72
8265 DATA 226,118,98,78,194
8266 DATA 125,0,0,0,0
8267 DATA 38,0,0,0,0
```

FNr

REDUCTION ROUTINE

Start address 56957 **Length** 290 bytes
What it does Reduces a previously enlarged routine to its original size.

Using the routine Each time the magnification routine is called, it saves in memory the screen as it was before magnification. The reduction routine simply displays this saved screen. Thus the reduction routine cannot repeatedly reduce an area on the screen; it will only show whatever was on the screen before the last magnification.

MAGNIFICATION AND REDUCTION 2

The program on this page gives an indication of the capabilities of the magnification routine. From an initial display, the magnification routine is called three times to enlarge different areas of the screen. As a further sophistication, these enlarged areas are then coloured using the window paper routine, and a line is drawn around the edge of each area.

The various shapes are drawn by different subroutines at lines 300, 400, 500 and 600, and each shape is drawn higher up a column by increasing the y co-ordinate before calling the subroutine. After a single column has been completed with five shapes, the display is repeated by the loop beginning at line 110 which sets a new value for the x co-ordinate.

After the subroutines have been completed, line 700 uses the magnification routine for the first time. Each time an area is magnified, the box routine is then called to draw a black line around the edge of the enlarged area (in this case, line 710). After a pause, the new area is coloured, and another area is enlarged.

SWEETS PROGRAM

```
  10 DEF FN c(x,y,h,v,c,b,f)=USR
 62600
  20 DEF FN g(x,y,p,q)=USR 60700
  30 DEF FN h(x,y,h,v)=USR 60400
  40 DEF FN j(x,y,r,s,f)=USR 589
00
  50 DEF FN m(x,y)=USR 57700
  60 DEF FN q(x,y,h,v)=USR 56700
 100 BORDER 1: PAPER 6: INK 0: C
LS
 110 FOR x=9 TO 209 STEP 40
 120 LET x1=x: LET y1=10
 130 GO SUB 500
 140 LET y1=y1+35
 150 GO SUB 600
 160 LET y1=y1+30
 170 GO SUB 400
 180 LET y1=y1+30
 190 GO SUB 300
 200 LET y1=y1+40
 210 GO SUB 400
 220 NEXT x

scroll?
```

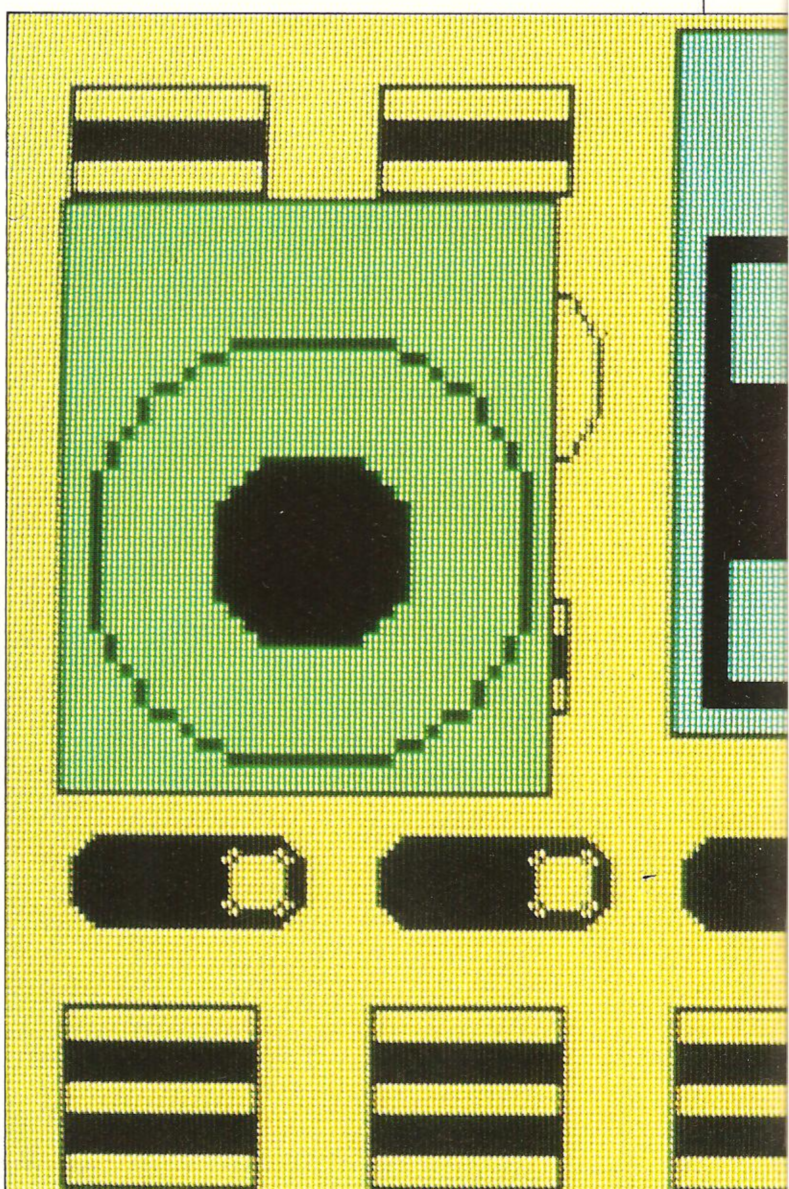
```
 230 PAUSE 50
 240 GO TO 700
 250 STOP
 300 RANDOMIZE FN j(x1+15,y1+15,
7,0,255)
 310 RANDOMIZE FN m(x1+15,y1+15)
 320 RANDOMIZE FN j(x1+15,y1+15,
15,0,255)
 330 RETURN
 400 RANDOMIZE FN h(x1,y1,25,15)
 410 RANDOMIZE FN h(x1,y1+5,25,5
)
 420 RANDOMIZE FN m(x1+1,y1+6)
 430 RETURN
 500 RANDOMIZE FN h(x1,y1,25,25)
 510 RANDOMIZE FN h(x1,y1+5,25,5
)
 520 RANDOMIZE FN m(x1+1,y1+6)
 530 RANDOMIZE FN h(x1,y1+15,25,
5)
 540 RANDOMIZE FN m(x1+1,y1+16)
 550 RETURN

scroll?
```

Relocating areas in memory

The magnification routine works by storing in memory whatever is on the screen in the specified area. It uses 8K of memory, stored from location 30208. If your BASIC program is particularly long, you may find that this area is required by your program. By POKEing the following three numbers:

POKE 56793,176
POKE 56950,176
POKE 56959,176

you can place the code about 18K higher in memory.

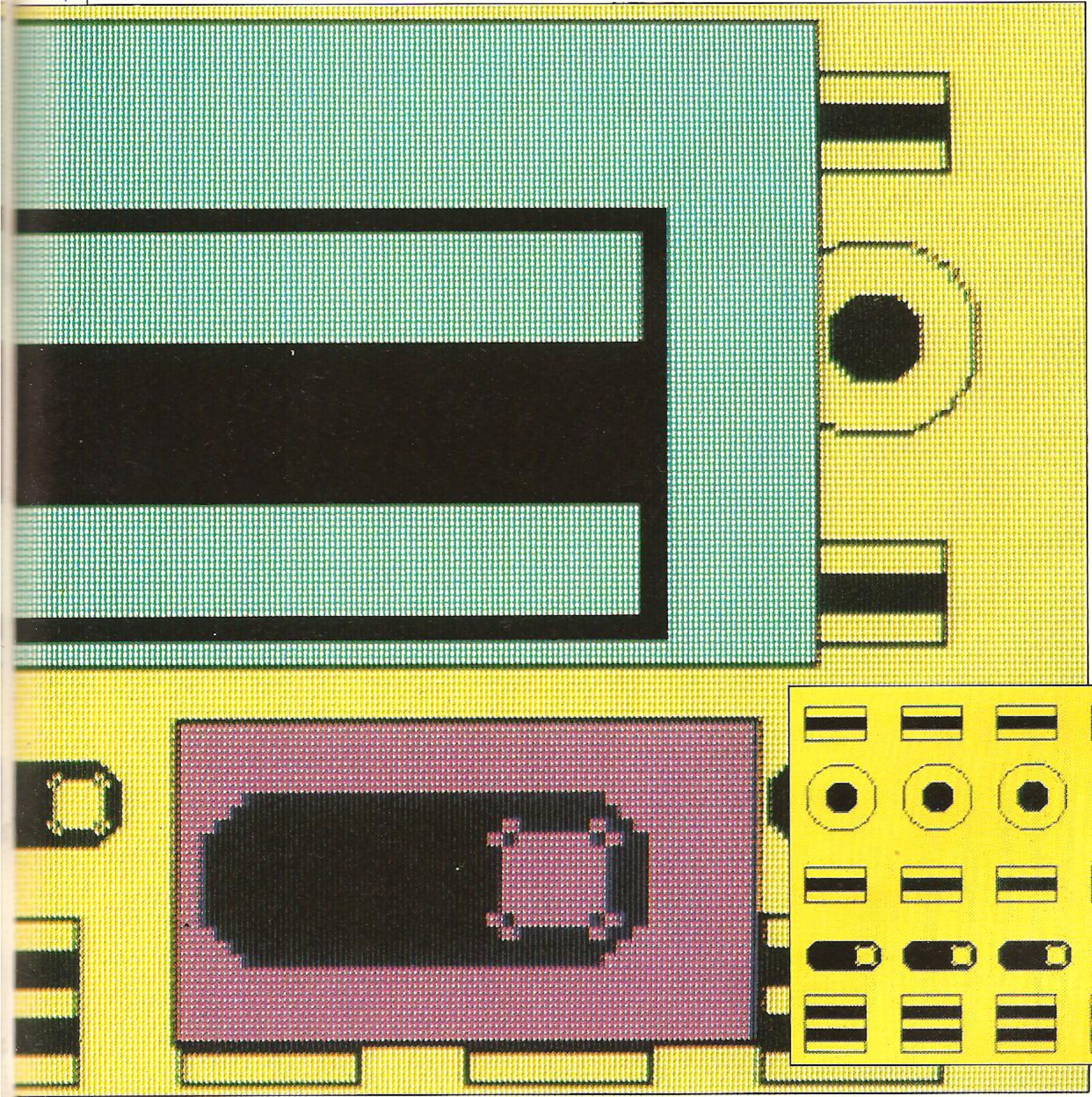


SWEETS PROGRAM CONTD.

```
 600 RANDOMIZE FN j(x1+25,y1+7,6
,0,255)
 610 RANDOMIZE FN j(x1+25,y1+7,7
,0,255)
 620 RANDOMIZE FN j(x1+25,y1+7,5
,0,255)
 630 RANDOMIZE FN g(x1+5,y1+1,x1
+21,y1+1)
 640 RANDOMIZE FN g(x1+5,y1+13,x
1+21,y1+13)
 650 RANDOMIZE FN j(x1+7,y1+7,7,
63,192)
 660 RANDOMIZE FN m(x1+7,y1+7)
 670 RETURN
 700 RANDOMIZE FN q(1,4,4,5)
 710 RANDOMIZE FN h(8,176-112,64
,80)
 720 PAUSE 100
 730 RANDOMIZE FN c(1,4,8,10,4,0
,0)
 740 PAUSE 100
 750 RANDOMIZE FN q(11,1,4,3)

scroll?
```

SWEETS PROGRAM CONTD.

```
 760 RANDOMIZE FN q(11,1,8,6)
 770 RANDOMIZE FN h(88,176-104,1
28,96)
 780 PAUSE 100
 790 RANDOMIZE FN c(11,1,16,12,5
,0,0)
 800 PAUSE 100
 810 RANDOMIZE FN q(16,14,5,3)
 820 RANDOMIZE FN h(128,176-160,
80,48)
 830 PAUSE 100
 840 RANDOMIZE FN c(16,14,10,6,3
,0,0)
 850 STOP

0 OK, 0:1
```



SWEETS PROGRAM

00:13 **seconds**

How the program works

A series of objects is placed on the screen, and the magnification routine used to enlarge various parts of the display.

Lines 130-220 call subroutines to draw the pattern of sweets.

Lines 300-670 form the subroutines which draw the sweets.

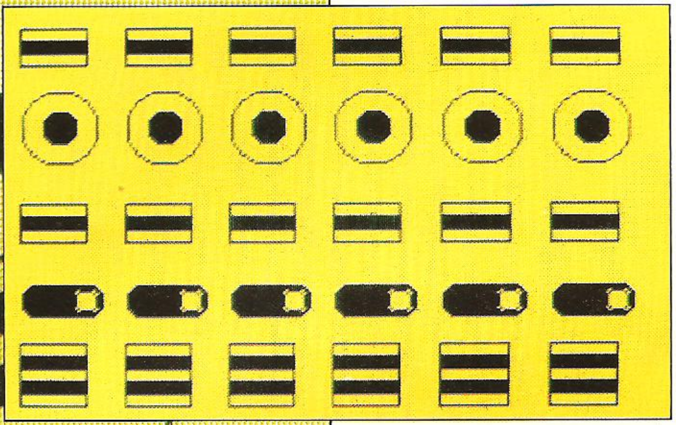
Lines 700-710 magnify the top left-hand area (coloured green in line 730).

Lines 750-770 magnify the top right-hand area (to four times its normal size).

Line 790 colours this area cyan.

Lines 810-820 magnify the third area.

Line 840 colours this area magenta.



SAVING AND LOADING DISPLAYS

The BASIC commands SAVE SCREEN$ and LOAD SCREEN$, used to save and load displays, have the disadvantage that they require nearly 8K of memory to save any display, no matter how simple. The screen compaction routine, FNs, allows you to store screens in a fraction of this space: the simpler the display, the less memory is required by the routine to store it. Even highly complex screen displays are stored in considerably less than 8K. As a guide, the three displays on this page require a total of just under 12K.

Previously saved displays can be displayed again using the decompaction routine, FNt. For both routines, the memory location of a display is specified by two parameters, containing the high and low bytes respectively of the start address.

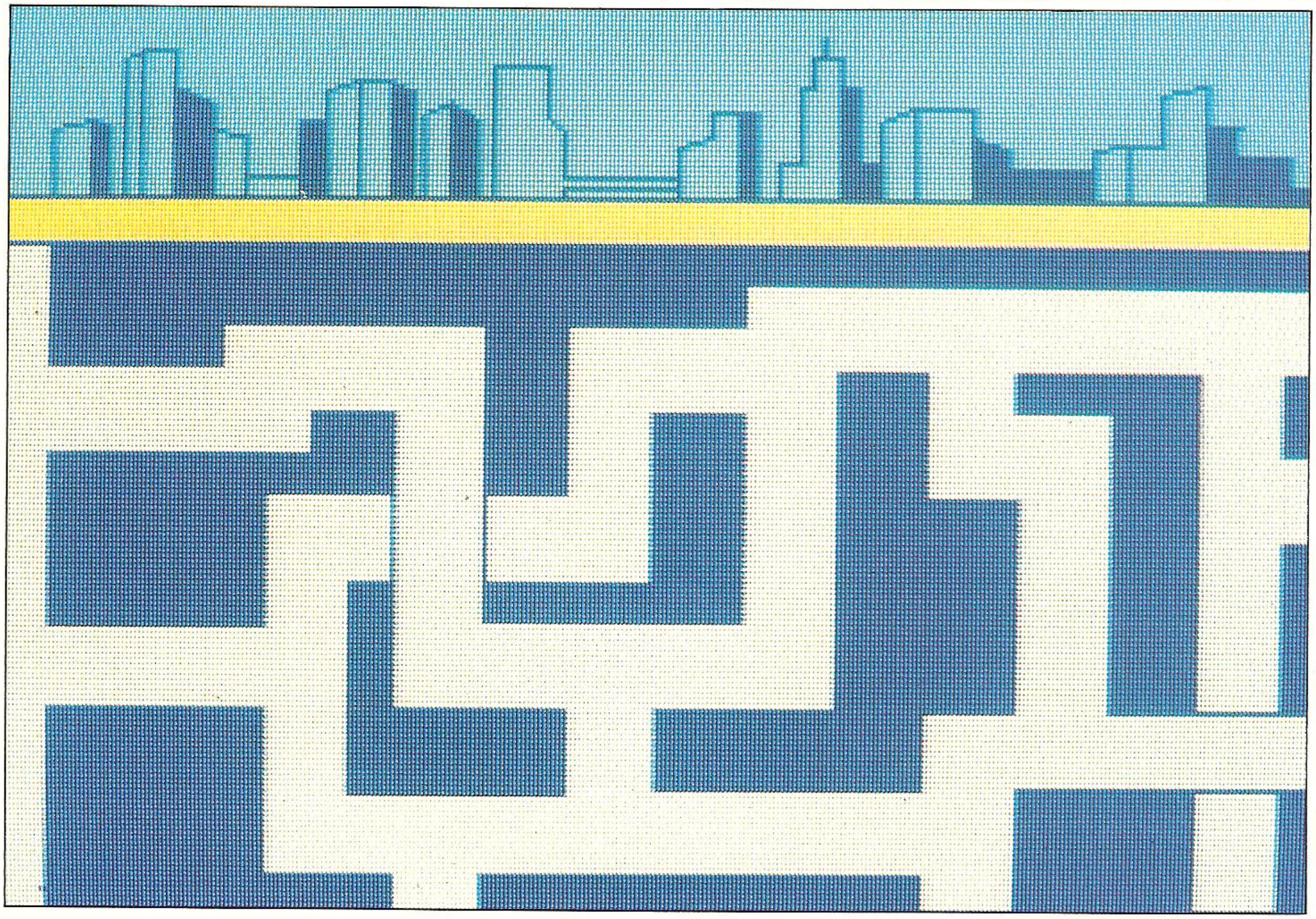
COMPACTION PROGRAM

00:03 **seconds**

How the program works
Three screens are loaded using SCREEN$, compacted, and then displayed again in quick succession.
Line 110 sets values for high and low bytes of the address (30000) where the first display is to be stored.
Line 120 PRINTs this address.
Line 140 compacts the display.
Line 150 PEEKs start values for the next display.
Lines 160-200 repeat the operation for the other screens.
Lines 300-350 display the screens in turn.

COMPACTION/DECOMPACTION PROGRAM

```
  10 DEF FN s(h,l)=USR 56600
  20 DEF FN t(h,l)=USR 56500
 100 LET a=56598: LET b=56599
 110 LET h1=117: LET l1=48
 120 LET ad=l1+(h1*256)
 130 PRINT ad: LOAD ""SCREEN$
 140 RANDOMIZE FN s(h1,l1)
 150 LET l2=PEEK a: LET h2=PEEK
b: LET ad=l2+(h2*256)
 160 PRINT ad: LOAD ""SCREEN$
 170 RANDOMIZE FN s(h2,l2)
 180 LET l3=PEEK a: LET h3=PEEK
b: LET ad=l3+(h3*256)
 190 PRINT ad: LOAD ""SCREEN$
 200 RANDOMIZE FN s(h3,l3)
 210 PAUSE 50
 300 RANDOMIZE FN t(h1,l1)
 310 PAUSE 50
 320 RANDOMIZE FN t(h2,l2)
 330 PAUSE 50
 340 RANDOMIZE FN t(h3,l3)
 350 PAUSE 50: GO TO 300

0 OK, 0:1
```



FNs

SCREEN COMPACTION ROUTINE

Start address 56600 **Length** 65 bytes
What it does Saves the current screen in a compacted form in memory.

Using the routine Parameters h and l are calculated by the formula

10 LET h=INT (store/256) : LET l=store−256*h

where "store" is the address in memory at which the screen is to be stored.
After storing a screen, you can find the start address for the next screen to be stored by PEEKing locations 23297 and 23296 (for h and l, the high and low bytes respectively of the address). This should be done immediately after compacting a display.

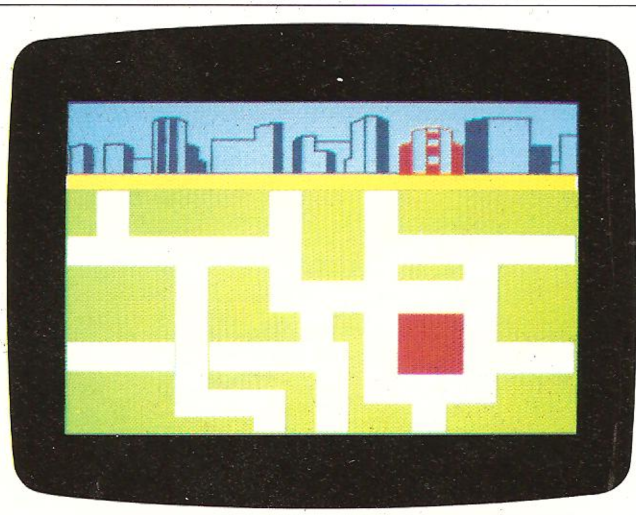
ROUTINE PARAMETERS

DEF FN s(h,l)	
h,l	specify the high and low bytes of the data for the screen respectively (h,l<255)

ROUTINE LISTING

```
8300 LET b=56600: LET l=60: LET
z=0: RESTORE 8310
8301 FOR i=0 TO l-1: READ a
8302 POKE (b+i),a: LET z=z+a
8303 NEXT i
8304 LET z=INT (((z/l)-INT (z/l)
)*l)
8305 READ a: IF a<>z THEN PRINT
"??": STOP

8310 DATA 42,11,92,1,4
8311 DATA 0,9,86,14,8
8312 DATA 9,94,237,83,82
8313 DATA 221,33,0,64,6
8314 DATA 1,126,44,32,8
8315 DATA 36,245,124,254,91
8316 DATA 40,16,241,78,185
8317 DATA 32,4,4,32,238
8318 DATA 5,18,19,120,18
8319 DATA 19,24,227,241,18
8320 DATA 19,120,18,237,83
8321 DATA 22,221,201,0,0
8322 DATA 56,0,0,0,0
```

COMPACTION PROGRAM: SAMPLE DISPLAY 2



FNt

SCREEN DECOMPACTION ROUTINE

Start address 56500 **Length** 45 bytes
What it does Decompacts a screen previously stored at a specified address.

Using the routine This routine puts back onto the screen a routine previously stored by the compaction routine. The decompaction routine loads screens much more quickly than the LOAD SCREEN$ command.
To obtain the start addresses (h and l) of each screen compacted by the compaction routine (FNs), PEEK locations 23297 and 23296 (for h and l) after compacting a screen.

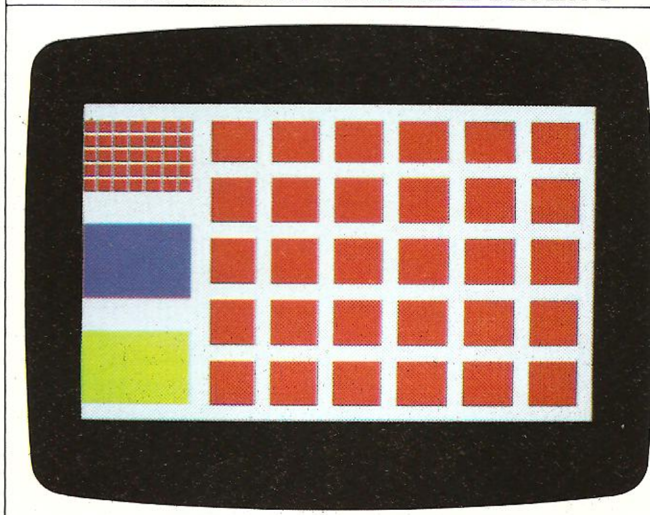
ROUTINE PARAMETERS

DEF FN t(h,l)	
h,l	specify the high and low bytes of the data for the screen respectively (h,l<255)

ROUTINE LISTING

```
8350 LET b=56500: LET l=35: LET
z=0: RESTORE 8360
8351 FOR i=0 TO l-1: READ a
8352 POKE (b+i),a: LET z=z+a
8353 NEXT i
8354 LET z=INT (((z/l)-INT (z/l)
)*l)
8355 READ a: IF a<>z THEN PRINT
"??": STOP

8360 DATA 42,11,92,1,4
8361 DATA 0,9,86,14,8
8362 DATA 9,94,33,0,64
8363 DATA 26,245,19,26,19
8364 DATA 71,241,119,35,16
8365 DATA 252,124,254,91,32
8366 DATA 240,201,0,0,0
8367 DATA 28,0,0,0,0
```

COMPACTION PROGRAM: SAMPLE DISPLAY 3



The displays on this page were produced using the graphics editor, stored by the compaction program and then displayed in succession. The start addresses in memory (variables h,l) of any screens you draw will obviously differ from those given here.

ROUTINES CHECKLIST

The table shown below gives a summary of all the machine-code routines used in this book. This table does not explain every detail of using each routine; it is intended only as an aid when using the routines in your programs. If you have not used a routine before, you are recommended to read the introduction to the routine on the appropriate page of the book before using it in your program.

page	title	parameters		parameters	co-ordinates
11	partial screen clear	FNa(x,y,h,v)	x,y h,v	start co-ordinates horiz. + vert. increment	character character
11	window ink	FNb(x,y,h,v,c,b,f)	x,y h,v c b,f	start co-ordinates horiz. + vert. increment colour BRIGHT or FLASH	character character — —
13	window paper	FNc(x,y,h,v,c,b,f)	x,y h,v c b,f	start co-ordinates horiz. + vert. increment colour BRIGHT or FLASH	character character — —
15	enlarged horizontal text	FNd(x,y)	x,y	start co-ordinates	character
15	enlarged vertical text	FNe(x,y)	x,y	start co-ordinates	character
17	point-plot	FNf(x,y)	x,y	start co-ordinates	pixel
21	line-draw	FNg(x,y,p,q)	x,y p,q	start co-ordinates end point co-ordinates	pixel pixel
25	box-draw	FNh(x,y,h,v)	x,y h,v	start co-ordinates horiz. + vert. increment	pixel pixel
27	triangle	FNi(x,y,p,q,r,s)	x,y p,q r,s	start co-ordinates co-ordinates of second point co-ordinates of third point	pixel pixel pixel
29	squares table				
29	master curve				
31	arc	FNj(x,y,r,s,f)	x,y r s,f	start co-ordinates length of radius start and finish point of arc	pixel pixel —
33	sector	FNk(x,y,r,s,f)	x,y r s,f	start co-ordinates length of radius start and finish point of arc	pixel pixel —
33	segment	FNl(x,y,r,s,f)	x,y r s,f	start co-ordinates length of radius start and finish point of arc	pixel pixel —
35	fill	FNm(x,y)	x,y	start co-ordinates	character
39	XOR-line	FNn(x,y,p,q)	x,y p,q	start co-ordinates co-ordinates of second point	pixel pixel
53	multiple line-draw	FNo()			
53	multiple XOR-line draw	FNp()			
55	magnification	FNq(x,y,h,v)	x,y h,v	start co-ordinates horiz. + vert. increment	character character
55	reduction	FNr()			
59	compaction	FNs(h,l)	h,l	high and low bytes	—
59	decompaction	FNt(h,l)	h,l	high and low bytes	—

Before using a routine you must first define it in your program using DEF FN followed by the correct number of parameters. Parameters passed to machine-code routines must always be whole numbers; if a parameter value is calculated by your program, then put an INT statement in front of it to ensure a whole-number value is passed to the routine.

ranges	bytes	address	check
0-32 and 0-24 0-32 and 0-24	100	63000	82
0-32 and 0-24 0-32 and 0-24 0-7 0=off, 1=on	135	62800	53
0-32 and 0-24 0-32 and 0-24 0-7 0=off, 1=on	150	62600	19
0-32 and 0-24	220	62200	0
0-32 and 0-24	215	61900	125
0-255 and 0-176	65	61500	24
0-255 and 0-176 0-255 and 0-176	215	60700	192
0-255 and 0-176 0-255 and 0-176	110	60400	86
0-255 and 0-176 0-256 and 1-176 0-256 and 1-176	80	60300	68
	60	59600	3
	525	59000	234
0-255 and 0-176 0-255 0-255	45	58900	17
0-255 and 0-176 0-255 0-255	45	58800	35
0-255 and 0-176 0-255 0-255	30	58700	18
0-32 and 0-24	195	57700	57
0-255 and 0-176 0-256 and 1-176	20	57600	13
	40	57100	17
	20	57000	13
0-32 and 0-24 0-32 and 0-24	290	56700	38
		56957	
0-255	65	56600	56
0-255	40	56500	28

MEMORY MAP

This chart shows how the Spectrum memory is organised when all the routines are present in memory. RAMTOP should be set to 55500 using a CLEAR command.

Code	Title	Address
FNa	partial screen clear	63000
FNb	window ink	62800
FNc	window paper	62600
	100-byte buffer	62500
FNd	enlarged horizontal text	62200
FNe	enlarged vertical text	61900
	300-byte buffer	61600
FNf	point plot	61500
FNg	line draw	60700
FNh	box draw	60400
FNi	triangle draw	60300
	600-byte buffer	59700
	squares table	59600
	master curve	59000
FNj	arc	58900
FNk	sector	58800
FNl	section	58700
FNm	fill	57700
FNn	exclusive/OR-line draw	57600
	400-byte buffer	57200
FNo	multiple line draw	57100
FNp	multiple XOR-line draw	57000
FNq	magnification	56700
FNr	reduction	56957
FNs	compaction	56600
FNt	decompaction	56500

ERROR TRAPPING

Error trapping in BASIC is carried out when an error message is displayed to show a mistake has occurred. This message is produced by a routine in the Spectrum ROM which prints on the screen the nature of the error.

When using machine code, however, it is often difficult to place restrictions on the way the routines are used. In most cases a determined user will be able to make the routine crash simply by passing it information which it does not understand. This could be checked within the machine-code routine itself to ensure that whatever is inputted is within the possible ranges you can type in, but to do this for all the routines in this book would require each routine to be perhaps doubled in length to incorporate the error checking required.

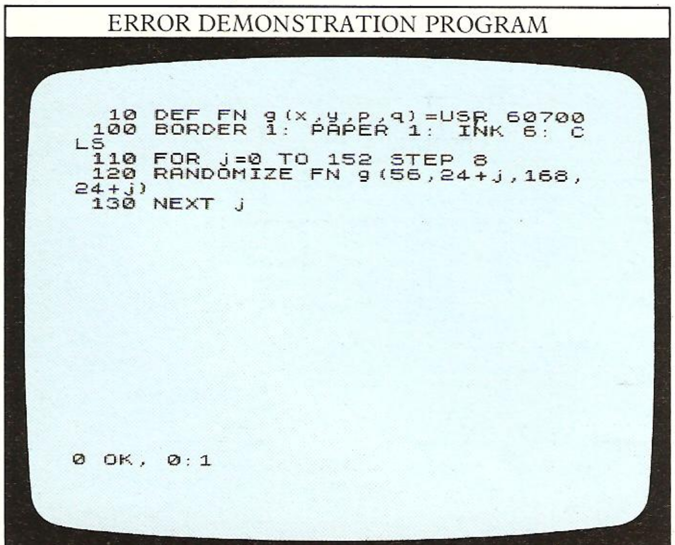
Preventing likely errors

In some cases it is quite easy, as well as helpful for the user, to add at least some error checking. A check routine has been added to the point-plot routine, for example, which means that although you may get rather unexpected results when you plot off-screen points, the routine is unlikely to crash. Try plotting a point which is off the screen and you will see the effects — a point will appear, but since the point specified is off the screen the routine will try to make sense of the data and plot a point at a position on the screen.

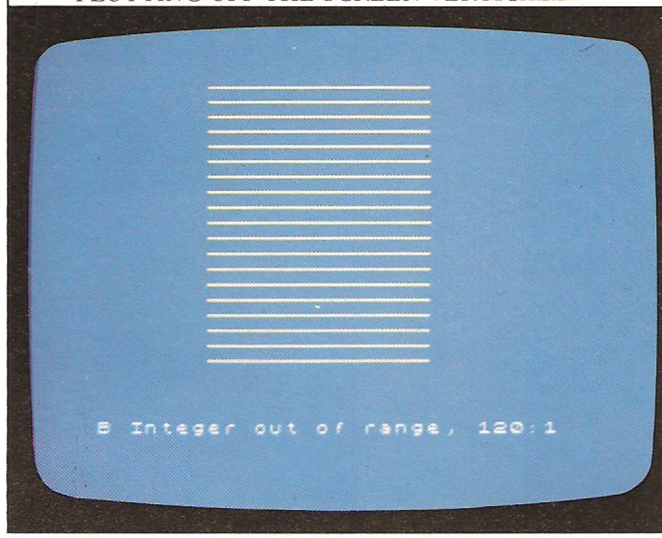
Error-trapping with the line-draw routine

Another part of the machine code which contains some error checking is the line-draw routine, as you can see from the error demonstration program shown here. Specifying lines off the top or bottom of the screen will result in "Integer out of range" being displayed, as happens when the program below is RUN with the co-

ERROR DEMONSTRATION PROGRAM

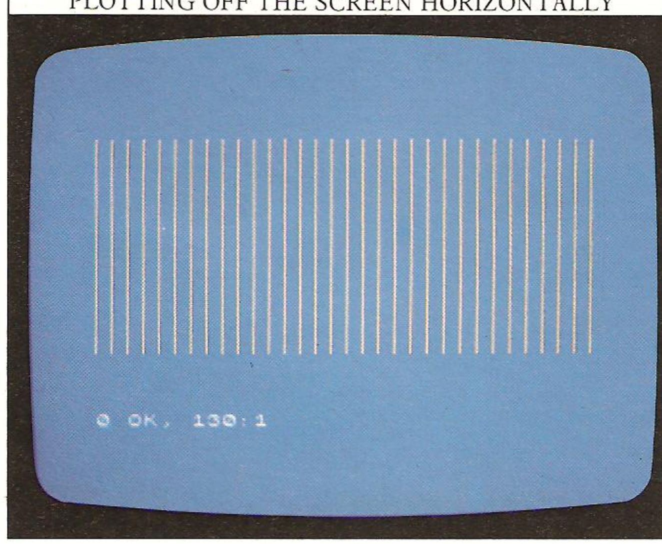


PLOTTING OFF THE SCREEN VERTICALLY



ordinates shown. Errors in horizontal co-ordinates are more difficult to trap, since these co-ordinates lie between 0 and 255, the range of numbers that can be contained in one byte. If you use a number larger than 255 it is likely to be treated as if it were 255 less than its actual value, with the result that the line wraps round to the other side of the screen. This effect can be seen in the screen below, the result of specifying parameter values which are off the screen horizontally.

PLOTTING OFF THE SCREEN HORIZONTALLY

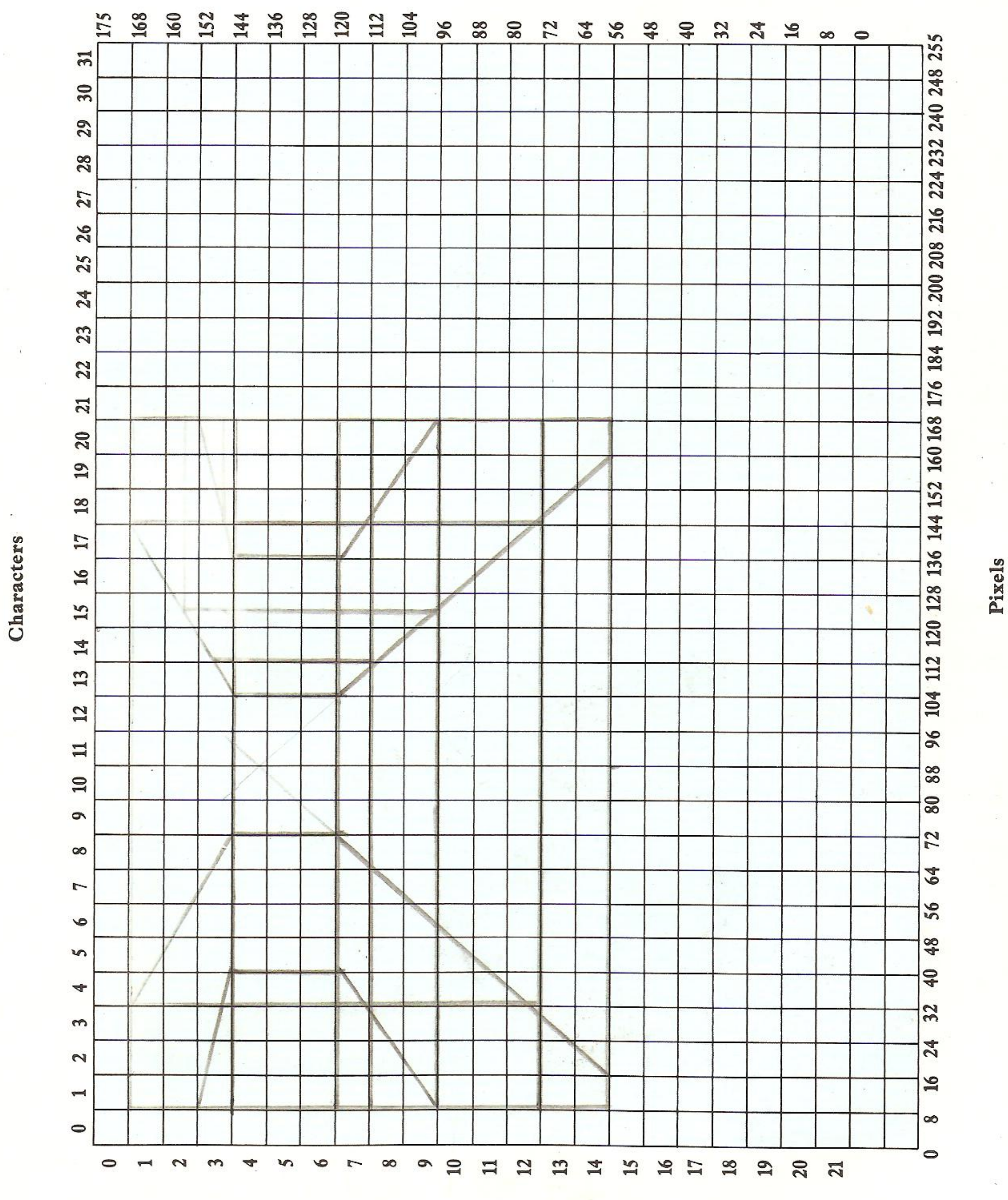


A similar effect can be seen with the curve routines, which cause odd effects when they go off the sides of the screen, but which will work within limits off the top and bottom of the screen.

The error trapping in these routines covers only the most likely errors you may make when using the machine code. As far as possible, you should keep to the limits and parameters specified for each routine.

GRAPHICS GRIDS

This grid shows screen divisions for both pixel and character co-ordinates. Points on the screen are defined by co-ordinates x (horizontal) and y (vertical). Character co-ordinates are measured from the top left-hand corner of the screen across and down. Pixel co-ordinates are measured from the bottom left-hand corner across and upwards. Pixel co-ordinates do not cover the bottom two lines of the screen.



INDEX

Main entries are given in **bold** type

Acknowledgments

A number of people helped and encouraged me with this book. Thanks to Alan and Michael at Dorling Kindersley, to Jacqui Lyons for her representation and to Andy Werbinski for reluctant assistance. I am particularly grateful, as always, to my parents, and to Martine.

Piers Letcher
Spring 1985

DK

Screen Shot
PROGRAMMING SERIES

The bestselling teach-yourself programming course now takes you beyond BASIC to the world of advanced machine-code graphics.

Using a combination of simple BASIC programming and a collection of tailor-made, ready-to-run machine-code routines, this book shows you how to produce precision, high-resolution graphics in a fraction of the time they would take in BASIC alone. A keyboard-driven graphics editor and a wide variety of demonstration programs will help you open up the full potential of the ZX Spectrum – without the need for any knowledge of machine-code programming.

Together, Books Three and Four in this series form a complete, self-contained graphics system for Spectrum-owners.

All the programs in this book run on both 48K ZX Spectrum and ZX Spectrum+ machines.

“Far better than anything else reviewed on these pages . . . Outstandingly good”
BIG K

“As good as anything else that is available, and far better than most”
COMPUTING TODAY

“Excellent . . . As a series they could form the best 'basic introduction' to programming I've seen”
POPULAR COMPUTING WEEKLY

DORLING KINDERSLEY

ISBN 0-86318-103-1

£5.95

9 780863 181030