Screen Shot
PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

COMMODORE 64

GRAPHICS

PHIL CORNES

BOOK THREE

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

DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

COMMODORE 64 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 Screen-Shot style, the series continues to set new standards in the world of computer books.

BOOKS ABOUT THE COMMODORE 64

This is Book Three in a series of guides to programming the Commodore 64. It contains a complete BASIC-and-machine-code graphics language for the Commodore, 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 **ZX Spectrum+**

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**

PHIL CORNES

After taking a B.A. in Mathematics and Computing, Phil Cornes has been involved in system development of computer-based education at British Telecom's National Training College. He has been a part-time technical author since 1978, and has become a regular contributor to personal computer magazines such as *Personal Computer World, Computing Today* and *Electronics Today International.* He has written a book and a large number of articles on programming and using the Commodore 64.

BOOK THREE


POWER
INST DEL
CLR HOME
RESTORE
RETURN
SHIFT
CRSR
CRSR

DK

Screen Shot

PROGRAMMING SERIES

STEP-BY-STEP PROGRAMMING

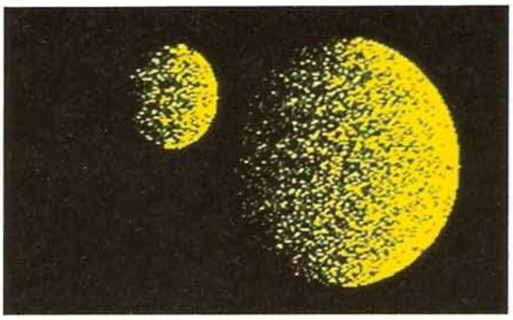
COMMODORE 64 GRAPHICS

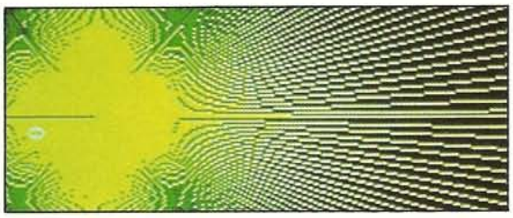
PHIL CORNES

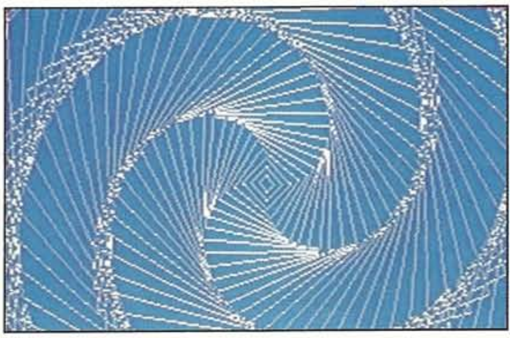
GUILD PUBLISHING · LONDON

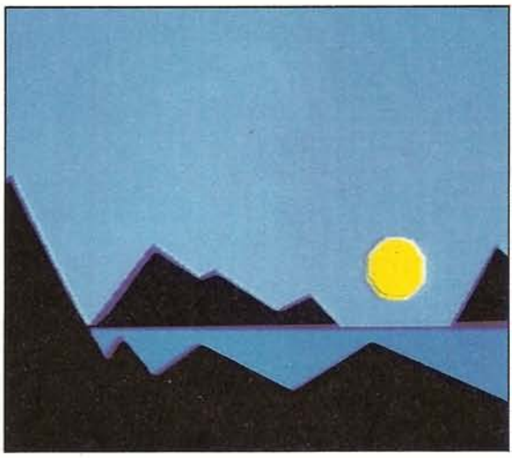
BOOK THREE

CONTENTS









The DK Screen-Shot Programming Series was conceived, edited and designed by Dorling Kindersley Limited.

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

This edition published 1985 by Book Club Associates by arrangement with Dorling Kindersley Limited.

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

HOW TO USE THIS BOOK

The Commodore 64 is one of the most powerful microcomputers currently available, and it has many facilities that set it apart from other machines. However, it does have one drawback. The Commodore's BASIC interpreter — the part of the computer which "understands" the BASIC programming language — has a rather limited vocabulary. Because it doesn't understand keywords like PLOT, DRAW or COLOR, complex graphics often require long programs full of difficult-to-use POKE statements. Furthermore, when you try out programs like these, the chances are that the Commodore will take quite a long time to run them because so many instructions are needed.

This book provides you with the tools you need to get much more out of the Commodore 64's graphics by taking you beyond BASIC. With the routines and programs on the following pages, you can make graphics both easier to key in and much faster to run.

Speeding up Commodore BASIC

Each time you run a BASIC graphics program, the Commodore will almost certainly have to perform a number of different operations, repeating each one over and over again. Take as an example plotting a single point on the screen. Each time the computer plots a point, it must interpret quite a long sequence of BASIC instructions. A considerable amount of "thinking" time is needed before the point appears. Imagine how much work the Commodore has to do if you want it to draw a series of lines made up of individual point-plots.

Program sequences that are to be repeated are usually written as routines. Indeed, this book gives you a whole package of routines to produce graphics. However, they are not ordinary BASIC routines activated by the command GOSUB. Instead, they are written so that they produce instructions in machine code, and these instructions are triggered by the command SYS. Using these routines, graphics programs run tens or hundreds of times faster than they would with pure BASIC.

The machine-code graphics routines

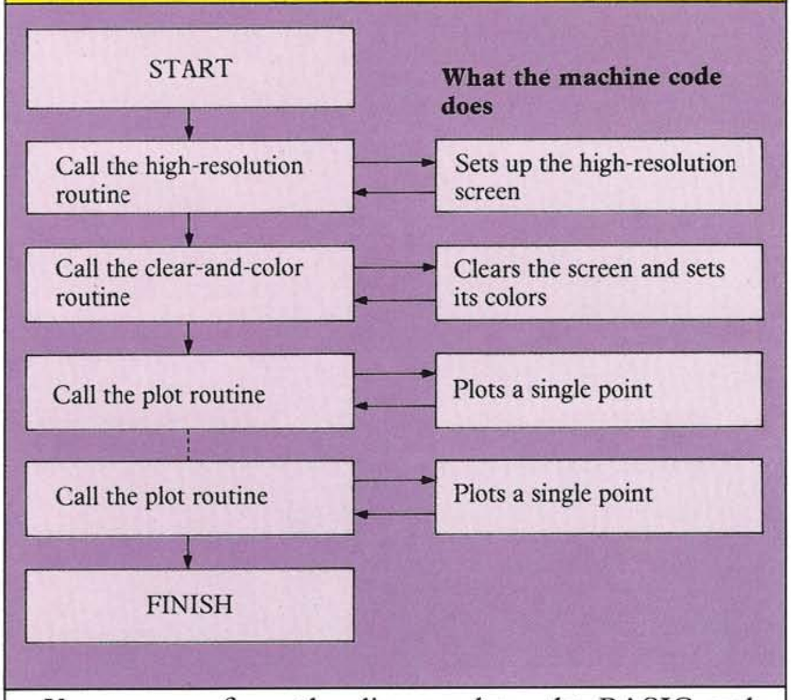
Pages 10-43 introduce 19 machine-code graphics routines that together build up into a complete graphics system. The routines are arranged in blocks, with each block coded by a letter from A to L. Each block contains from one to five separate graphics routines coded by a number (A1, A2 and so on). You can find a complete list of the routines on pages 62-63.

To try out any program in this book, all you have to do is key in the main BASIC program together with the routine blocks that it uses. You can find full details of how to do this on pages 8-9. By saving each of the routine blocks on tape or disk as you go through the book, you'll have a new and flexible graphics capability at your fingertips.

Linking BASIC and machine code

When you try out a program in this book, the computer will run it by using a number of the machine-code routines. Here is a program sequence which would plot a series of points:

HOW A PROGRAM USES THE GRAPHICS ROUTINES



You can see from the diagram how the BASIC and machine code work together. A typical example of a main BASIC graphics listing which plots points in this way looks like this:

MAIN BASIC LISTING

```
LIST
10000 SYS A1 : SYS B1,112 : POKE 53280,0
10010 R=60 : XC=180 : YC=100
10020 GOSUB 10060
10030 R=25 : XC=90 : YC=70
10040 GOSUB 10060
10050 GOTO 10050
10060 FOR Y=-R TO R
10070 X1=INT(SQR(R*R-Y*Y))
10080 FOR X=-X1 TO X1
10090 N=INT(RND(1)*X1*2)+1
10100 IF N<X1+X THEN SYS C1,X+XC,Y+YC
10110 NEXT X : NEXT Y : RETURN
READY.
```

You will notice that the listing contains a number of lines which feature the SYS command. This makes the computer carry out instructions that start not at a specified line number, but instead at a specified address in memory (you can see the address numbers in the routines checklist on pages 62-63).

In the previous program, when the computer reaches line 10100, for example, it goes to memory address C1. C1 is simply a variable holding the number of the memory address for routine 1 in block C. When the computer goes to this address it follows the instructions that begin there. This is where the graphics routine comes in. It puts instructions that code a graphics operation directly into the memory. Once there, the computer can read them without using BASIC.

You can see the routine in the panel below. It plots points. Because it has low line numbers, its instructions are POKEd into memory before the main BASIC listing calls on them. This means that from the moment the routine has been run, the potential for plotting points with machine code is in memory ready to be used.

When the computer encounters the command SYS C1 in the main BASIC program, it will jump to the correct machine code and plot a point. This will happen very quickly because BASIC is not used.

What is machine code?

Although you can program the Commodore with BASIC keywords, ultimately it works only with numbers. In every one of the 65,536 addresses in the Commodore's RAM, there is room for one number with a value between 0 and 255. It is these numbers that control all the programmable operations that the computer can perform.

Before a program can be carried out, the Commodore must convert all BASIC instructions, strings and variables into numbers and put them into specific memory addresses. Once this has been done, the computer can then start working on them to produce results.

The number language that the computer uses is known as machine code. Machine-code instructions are carried out very rapidly. A large amount of the time your Commodore needs to run a BASIC program is taken up with converting from BASIC into machine code, rather than computing results.

This book speeds up programs by cutting down the amount of BASIC-to-machine-code conversion that the Commodore needs to do. The graphics routines use the POKE command to put DATA numbers directly into memory, and these produce machine-code results.

WHAT A MACHINE-CODE PANEL CONTAINS

Block title Each machine-code block is labeled by a single letter from A to L (block M does not contain machine-code). The program line numbers in the blocks form a single sequence which follows their alphabetical order.

Syntax and parameters This section tells you how to key in each routine and also what parameters, if any, you need to specify.

Parameters This section tells you what the parameters specify, and what limits they must fall between.

BLOCK C

PLOT routine

What the routine does

The routine plots a single pixel at a specified point on the high-resolution screen (the screen coordinates are shown in the grid on page 61).

SYNTAX AND PARAMETERS

SYS C1,X,Y

X,Y	Horizontal and vertical coordinates of the point to be plotted (ranges 0-319 and 0-199).

ROUTINE LISTING

```
1200 IF PEEK(49712)=32 THEN 1230
1210 SYS A3,1240 : FOR C=49712 TO 49787
1220 READ B : POKE C,B : NEXT C
1230 C1=49712
1240 DATA 32,40,192,32,224,192,169
1250 DATA 0,144,2,165,1,141,24
1260 DATA 192,142,9,192,140,8,192
1270 DATA 32,40,192,32,236,192,169
1280 DATA 0,144,2,169,1,13,24
1290 DATA 192,141,24,192,142,13,192

1300 DATA 140,12,192,240,1,96,32
1310 DATA 0,193,160,0,173,29,192
1320 DATA 201,0,240,8,177,253,77
1330 DATA 0,192,145,253,96,177,253
1340 DATA 13,0,192,145,253,96
```

Routine names This tells you which routine or routines the block contains.

Routine function This tells you what each of the routines in the block does, and also any special information you need in order to use them.

Syntax This shows the general form in which the routine(s) must be keyed in. Here SYS C1 is the part of the syntax which calls the plot routine, while X,Y are the two parameters it requires.

Routine listing This is the listing you must key in and run in order to be able to call the machine-code routine(s) it codes.

HOW TO KEY IN THE PROGRAMS

Because the programs in this book all depend on machine code, it is crucial that you know what to key in before you start. To run any program in this book, you need to do three things:

1 Make sure that your Commodore is set up for high resolution. You can see how to do this in the panel on the right of this page. You only have to do this once every time you switch on, but failing to do it will prevent the programs from running.

2 Find out which machine-code routine blocks the program needs, and load or key them in. You can either load the separate routines one by one, or, if you have worked through the book already, you can load the complete set. Both ways work equally well (it doesn't matter if you have unused routines in memory). Make sure that you add all the routines in order, with the lowest line numbers being keyed in or loaded first.

3 Add the main BASIC listing and run the complete program. If you have trouble with a program, consult the panels on the opposite page.

Writing your own programs

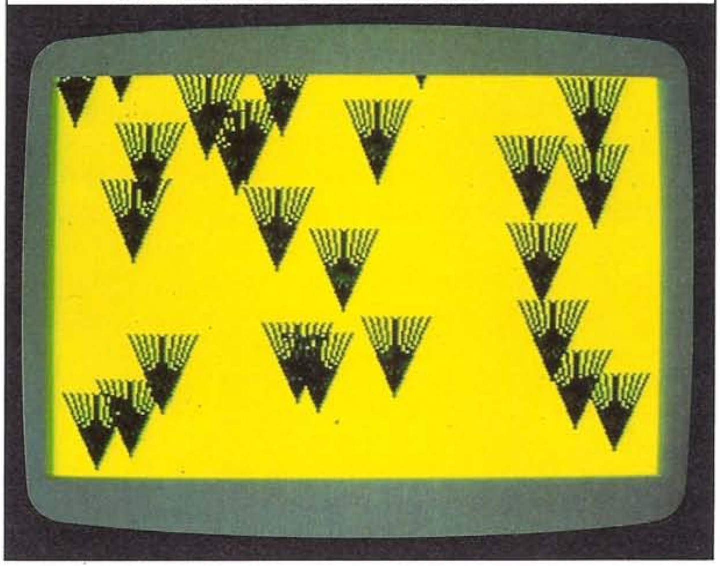
Once you have a complete set of the graphics routines, writing your own high-resolution graphics programs is easy. All you have to do is load the routines and then add a main BASIC program, starting at line 10000.

Here for example is a program which will draw fan-shapes on the screen. You can see from this how simple the graphics system is to use, and how much more straightforward the listing is than a pure BASIC equivalent.

HIGH-RESOLUTION GRAPHICS PROGRAM

```
LIST

10000 SYS A1 : SYS B1,7
10010 POKE 53280,12
10020 FOR P=1 TO 25
10030 X=RND(0)*319
10040 Y=RND(0)*199
10050 SYS C1,X,Y
10060 FOR N=1 TO 20 STEP 2
10070 SYS D1,(X-20)+2*N,Y-40
10080 SYS C1,X,Y
10090 NEXT N
10100 NEXT P
10110 GOTO 10110
READY.
```

HIGH-RESOLUTION GRAPHICS DISPLAY



IMPORTANT

After you turn on the computer and before you try out the programs in this book, you must key in the following series of direct commands:

HIGH-RESOLUTION COMMANDS SCREEN

```
POKE 642,64
READY.
POKE 44,64
READY.
POKE 16384,0
READY.
NEW
READY.
```

The Commodore does not have a section of memory specifically reserved for high-resolution graphics. These commands make the computer rearrange its memory so that it can store high-resolution graphics information in the area normally used by the BASIC system. The programs in this book will not work if you forget to key in these commands.

This sequence must be typed in as direct commands and not as part of a program. If you try to enter the commands as statements in a program, there is a chance that the program may destroy itself before it has finished running.

Program line numbers

When you key in a program from this book, the complete listing will fall into two parts. The machine-code routines, which make up the first part of the program, will always start at line number 100. The main program, which makes up the second part and calls the routines, will always start at line number 10000. In addition to this, ordinary BASIC subroutines (subroutines which do not involve machine code and which are called by GOSUB) will normally appear at line number 20000 onward.

How to store and re-load the routines

The value of machine code is that it is very fast. However, some of the blocks of machine-code routines in this book are quite lengthy, and there would be little point in using this system of speeding up programs if it meant a great deal of extra time on the keyboard. But if you have a cassette or disk storage system, there is no need for you to key in any machine-code block more than once.

As you encounter each block for the first time, after testing that it works correctly you should store it. Then, when you want to try out a program, you can re-load the routine blocks needed, add the program, and you are ready to run.

Block A of the machine-code routines contains a merge routine which allows you to add together routine blocks and programs (the Commodore normally erases any program in memory if you load another). If you key in and store every routine block separately as you go through the book, the merge routine will let you later add all the routine blocks together. The resulting complete set of machine-code routines will enable you to run any high-resolution program in this book without having to load routine blocks separately. You can find instructions for using the merge routine on page 11.

How to alter a program

The system of program numbering is designed to help you to distinguish between the machine-code routines and the main program. You can alter the main program just as you would any other BASIC program. You can edit, extend or reorganize in any way you like, as long as the main program continues to use only the routines that precede it. To modify a main program, simply delete or alter the lines you want to change, and the revised version will be ready to run.

PROGRAMMING TROUBLESHOOTER

The program won't RUN or LIST
Check that you have set your Commodore up for high resolution. If not, switch off and begin again.

The program won't RUN but it will LIST
Check that the lines are in the right order. If you merge routines and programs out of line order, the complete program will probably crash. This is particularly important to remember with turtle graphics.

The program only RUNs partially
You may have keyed in the routines incorrectly. On page 61, you will find an error-detection system which will help you to track down any typing errors in your machine-code DATA. Errors in BASIC can be detected by switching to low resolution (see page 10), after which programming faults will be revealed by the usual Commodore error reports on screen.

The program just produces a READY message
Check that your machine code is in memory. Try typing SYS A1 in as a direct command. If the screen doesn't switch to high resolution, your routines are either incorrectly keyed, or have not been run.

The program works, but won't re-RUN
If you key in RUN on a line which already has some graphics on it, the computer won't understand the instruction. Press the RUN/STOP and RESTORE keys together, and key in RUN again.

GRAPHICS QUESTIONS AND ANSWERS

When do I have to set up high resolution?
You must do this every time you turn the computer on, if you want to use the high-resolution screen. Get into the habit of doing this automatically after you switch on.

Can I start anywhere in the book?
Yes, you can start anywhere you like. However, you will find it much easier to work through pages 10-43 first, building up a library of the routine blocks on tape or disk as you try them out. If you do this, you will avoid having to key them in more than once.

How can I add together routines and programs from a tape or disk?
There's no MERGE command on the Commodore, but this book gives you a machine-code equivalent. This is contained in routine block A. If you load this block and then run it, you can merge anything else with SYS 49297. Full details are on page 11.

Can I load just one routine from a block?
No. You must always load or key in complete routine blocks.

Can I adapt the machine-code routines?
No. If you alter the DATA numbers in the routines it is highly unlikely that the routines will work.

How do I stop a program?
Press the RUN/STOP and RESTORE keys together.

THE GRAPHICS TOOLKIT

The machine-code routines featured on these two pages don't produce graphics themselves, but they are either essential for the routines that do, or else they give you program-handling facilities which Commodore BASIC does not have.

Before you can produce graphics, you need to turn on the high-resolution screen. To do this, set your Commodore up for high resolution, if you haven't already done so (see page 8), and then type in the whole of the listing in the block A panel on the opposite page. Before you go any further, you should store the listing on tape or disk. If you save a block of routines before running it, you can re-load it if it fails to work the first time so that it can be debugged.

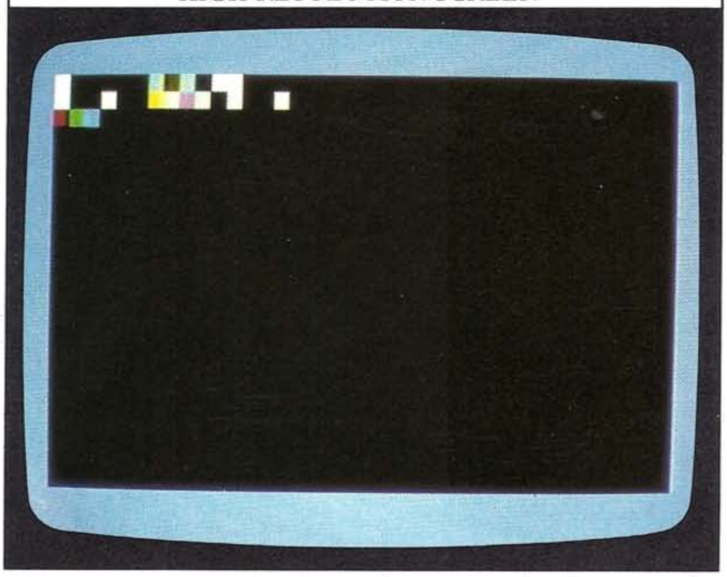
High- and low-resolution routines

After you have saved block A, add this very short main program to it:

```
10000 SYS A1
10010 GOTO 10010
```

You now have a program which when run will turn on the high-resolution graphics screen. The program consists of two parts. Line 10000 calls a machine-code routine for high resolution, while line 10010 forms an endless loop, preventing the computer from trying to produce a READY message on the high-resolution screen. If you now key in RUN and then press RETURN, you should see a display like this:

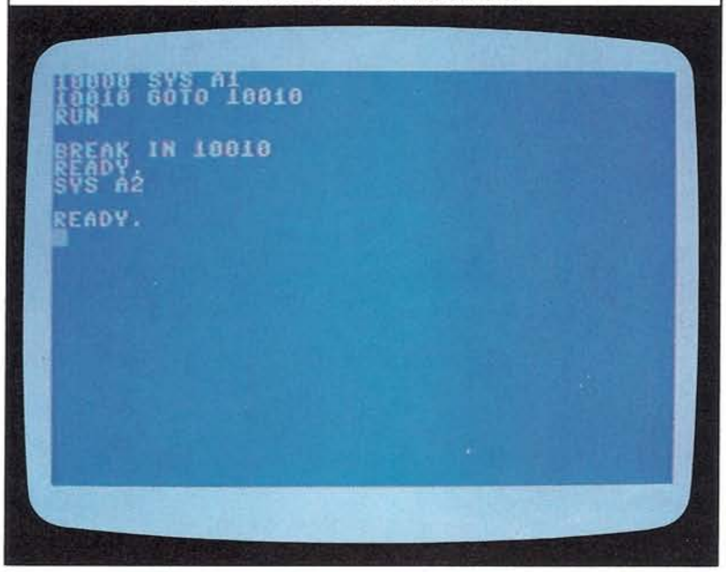
HIGH-RESOLUTION SCREEN



The program works in a few seconds. It uses the command SYS A1 to activate the first machine-code routine in block A, turning on high resolution.

If you now press the RUN/STOP key and type in SYS A2 followed by pressing RETURN, you will return to the low-resolution (text mode) screen:

LOW-RESOLUTION SCREEN



How the routines check themselves

When you first run the program, you may notice a short time delay before the high-resolution screen is turned on. This delay is the time taken for the five routines in block A to load their machine code into memory. This only has to happen once each time you switch on, because all the routines check to see if they have already been placed in memory.

Line 100 in block A checks to see if the machine code has already been loaded. If it hasn't, control is passed to the loop in lines 110 and 120 which READs all the DATA and POKEs it into memory starting at location 49152. But if the DATA is already in memory then this loop is skipped.

Lines 130 to 150 set up the variables that tell the computer where each set of machine-code instructions, from A1 to A5, starts in memory. There are five variables — one for each routine.

The restore routine

The next routine in the block doesn't produce anything on its own, but is very helpful in graphics programming. Because all the routines in this book contain DATA statements, you can run into trouble if your main programs also contain DATA. This is because the computer READs DATA as a single series, starting at the beginning of a program, and working through to the end. If the DATA is to be READ more than once, the RESTORE statement is normally used. However, the Commodore's RESTORE statement cannot be used to specify READing of just part of the DATA, so normally you cannot make the computer use DATA selectively.

The restore routine provides just this facility. It's quite simple to use in programs. The command:

```
SYS A3,15000
```

for example makes the computer READ the DATA beginning at line 15000. Watch out for this routine where a main program uses DATA.

The rescue routine

The fourth routine in the graphics toolkit will help you if you accidentally erase a program. It reverses the effect of the BASIC command NEW (in some BASICs, this is carried out by the command OLD). The rescue routine lets you recall a BASIC program. This is possible because NEW does not in fact clear a program completely, but merely alters some memory pointers so that the program is ignored by the computer. To cancel NEW, simply type:

```
SYS 49271
```

This routine has to be called by a number instead of by SYS A4 because all variables are "forgotten" if you use NEW.

The merge routine

Throughout this book you will want to join separate programs together, usually to combine a main program with one or a group of routines. The merge routine allows you to add one program onto the end of another.

Suppose you have a program in memory, and then want to add another program to it. Typing:

```
SYS 49297, "FILENAME"
```

or, if you are using a disk drive,

```
SYS 49297, "FILENAME",8
```

is all that is needed. The program you have identified with the filename will now be loaded onto the end of the program in memory. Notice that again the routine has to be called by its address number instead of by SYS A5.

How to add the routines together

You can use the merge routine to build up a complete set of all the machine-code graphics routines in this book. All you have to do is load or key in block A, run it, and then any other routine block can be added to it with SYS 49297. You should add the routine blocks in alphabetical order as you encounter them in the book. Their line numbers are designed to form a single, non-overlapping sequence.

Remember whenever you use the merge routine to add programs in strict line order.

What next?

Now you have saved your graphics toolkit on tape or disk, you can add some more routines which will start graphics on the screen. Turn over the page to find out about quick ways to use color.

BLOCK A

HIGH-RESOLUTION, LOW-RESOLUTION, RESTORE, RESCUE and MERGE routines

What the routines do

High-resolution turns on the high-resolution screen. All the programs in this book need this routine in order to work.
Low-resolution returns the screen to low resolution (text mode).
Restore resets the DATA pointer to a specified line of DATA in a program. This allows DATA to be READ selectively.
Rescue cancels the effect of the NEW command. It is equivalent to the OLD command of other BASICs.
Merge adds a second program onto the end of the one currently in memory. The merge routine does not arrange lines in numerical sequence. The line numbers of the second program must therefore be higher than and not overlapping those of the first.

SYNTAX AND PARAMETERS	
High-resolution: SYS A1 Low-resolution : SYS A2 Restore: SYS A3,N Rescue: SYS 49271 Merge: SYS 49297,A$(,8)	
N	(**Restore** only) Line number at which the program is to start READing DATA.
A$	(**Merge** only) Filename. This can be the name of any program currently on tape or disk.
,8	(**Merge** only) Device number. Add ,8 if you are using a disk drive; omit if you are using a tape.

ROUTINE LISTING

```
100 IF PEEK(49192)=32 THEN 130
110 RESTORE : FOR C=49152 TO 49404
120 READ B : POKE C,B : NEXT C
130 A1=49237 : A2=49254
140 A3=49209 : A4=49271
150 A5=49297
160 DATA 0,0,0,0,0,0,0
170 DATA 0,0,0,0,0,0,0
180 DATA 0,0,0,0,0,0,0
190 DATA 0,0,0,0,0,0,0

200 DATA 0,0,0,0,0,0,0
210 DATA 0,0,0,0,0,32,121
220 DATA 0,32,253,174,32,138,173
230 DATA 32,191,177,166,100,164,101
240 DATA 96,32,40,192,132,63,132
250 DATA 20,134,64,134,21,32,19
260 DATA 166,165,95,56,233,1,133
270 DATA 65,165,96,233,0,133,66
280 DATA 96,169,8,13,24,208,141
290 DATA 24,208,169,32,13,17,208

300 DATA 141,17,208,96,169,247,45
310 DATA 24,208,141,24,208,169,223
320 DATA 45,17,208,141,17,208,96
330 DATA 169,8,160,1,145,43,32
340 DATA 51,165,165,34,133,45,133
350 DATA 47,133,49,165,35,133,46
360 DATA 133,48,133,50,96,32,121
370 DATA 0,32,253,174,169,0,133
380 DATA 10,32,212,225,165,43,72
390 DATA 165,44,72,56,165,45,233

400 DATA 2,133,43,165,46,233,0
410 DATA 133,44,169,0,133,185,166
420 DATA 43,164,44,32,213,255,176
430 DATA 14,134,45,132,46,32,51
440 DATA 165,104,133,44,104,133,43
450 DATA 96,170,201,4,208,5,164
460 DATA 186,136,240,206,104,133,44
470 DATA 104,133,43,24,108,0,3
480 DATA 224,2,176,7,224,1,176
490 DATA 1,96,192,64,96,56,138
500 DATA 208,3,152,201,200,96,105
510 DATA 0,133,46,133,48,133,50
520 DATA 96
```

HIGH-RESOLUTION COLOR

Now that you can switch the Commodore to high resolution, the next step is to decide which colors you want to use. There are two ways in which you can do this — you can either specify colors for the whole of the screen, or for just part of it. As you will see later on, coloring the screen is best done in a particular order. First you set up overall colors, then you draw your design, then, if you want to, you can change areas of color.

Coloring is done with two machine-code routines that are programmed by block B on the opposite page. They are the clear-and-color routine and the block-color routine.

Commodore color codes

If you have read Book Two in this series, you will be familiar with the color-coding that the Commodore uses in high resolution. There are 16 colors available. Their codes are summarized in the table on page 63.

In high resolution, there are some restrictions in the way that these colors can be used. In the bit-map mode only two colors can be used in each 8x8 pixel block, but resolution is very good. In the multicolor mode four colors can be used in each 8x8 block, but the resolution drops by half. All the programs in this book use the bit-map mode to give the best resolution.

On the screen one color is known as the background color, and the other the foreground color. Graphics images appear in the foreground color surrounded by the background color. Any combination of background and foreground colors can be selected by adding together two of the color control numbers shown in the table.

To see how the clear-and-color and block-color routines work, you will need to load or type in the block A listing on page 11, if it is not in memory already, and then type block B onto the end. When you have done this, save the combined blocks so that you have a total of seven routines safely stored on tape or disk. You will find that every program in this book uses routines from both these blocks.

The clear-and-color routine

To specify a foreground and background color for the whole screen, you use the clear-and-color routine. This routine is called by the command SYS B1, together with a color code, like this:

```
SYS B1,80
```

(don't forget the comma). If you have blocks A and B in memory (that is, if you have loaded and run them) this command would clear the screen and set up a foreground color of green and a background color of black. You wouldn't actually see the foreground color until you had plotted or drawn with it.

If you have only carried out this procedure before with BASIC, you'll see a huge reduction in the amount of time it takes.

The block-color routine

You can color a selected area of the screen with the block-color routine. This is called into action with a command like:

```
SYS B2,X,Y,C
```

where C is the required color combination, and X and Y are a pair of high-resolution coordinates (you will find a high-resolution coordinates grid on page 60). This routine controls the colors in one 8x8 pixel block. It takes the two coordinates, rounds them down to get the corner of an 8x8 block, and then colors the block. So, for example, the line:

```
SYS B2,100,100,118
```

would set the color of just the 8x8 pixel block containing the high-resolution coordinates 100,100 to yellow on blue (color codes 112+6). If you had previously set the overall background color to something other than blue, you would now see a single blue block standing out on

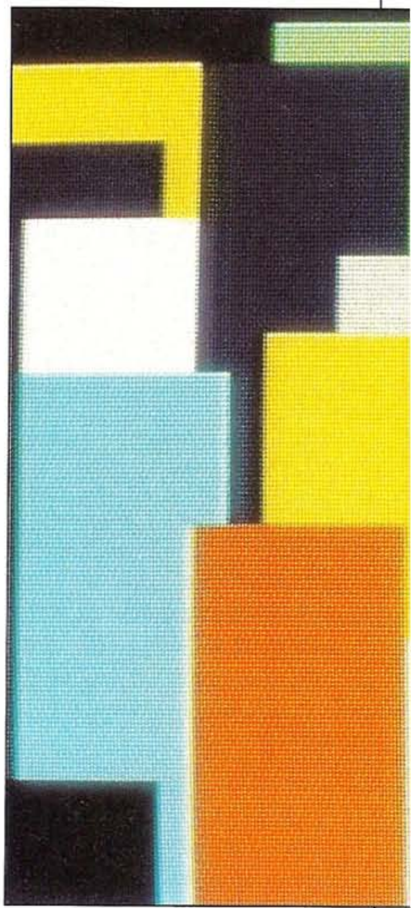
RANDOM BLOCK-COLOR PROGRAM

01:05

How the program works
Three values are selected at random — a color combination and a pair of coordinates. These fix the color and position of each bar. The height is random, while the width is fixed at 40 pixels for each bar. (The time above is that taken for 50 bars to be displayed.)
Line 10000 sets up the high-resolution screen and colors it black with the clear-and-color routine.
Lines 10010-10070 make up a loop which produces random bars in random colors.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
B Clear-and-color	13
Block-color	



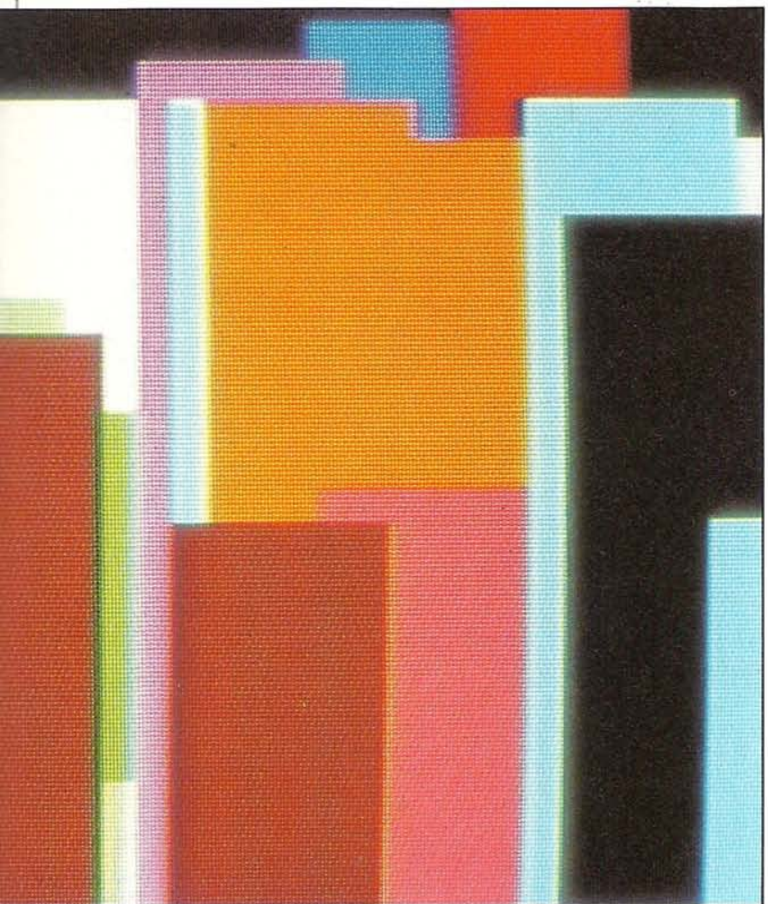
the screen. Anything plotted or drawn in this block would be in the new foreground color — yellow. This might sound complicated, but it's easy to master.

Random block-coloring

You can test out the block-color routine with the following program. It sets up random colors in bars over the screen. Using the routine by having SYS B2 inside a loop allows you to color any size of rectangle. To try it, load blocks A and B if they are not currently in memory, add this listing and then run the complete program.

RANDOM BLOCK-COLOR PROGRAM

```
LIST

10000 SYS A1 : SYS B1,0
10010 C=RND(0)*255
10020 LX=INT(RND(0)*39)*8 : UX=LX+40
10030 LY=INT(RND(0)*24)*8 : UY=196
10040 FOR X=LX TO UX STEP 8
10050 FOR Y=LY TO UY STEP 8
10060 SYS B2,X,Y,C
10070 NEXT Y : NEXT X : GOTO 10010
READY.
```



BLOCK B

CLEAR-AND-COLOR and BLOCK-COLOR routines

What the routines do

Clear-and-color clears the high-resolution screen, and sets up the initial overall foreground and background colors for the whole screen. It is normally used at the beginning of a high-resolution graphics program, immediately following the high-resolution routine.

Block-color sets up the foreground and background colors for a single 8x8 pixel block. It is often used within a loop to color rectangles containing a number of blocks. The block-color routine can be used to reset the colors of an area of the screen as often as required within a program, so that a range of colors can be built up in a display.

Both these routines use the standard Commodore color combination codes. For details, see the chart on page 63.

SYNTAX AND PARAMETERS

Clear-and-color: SYS B1,C Block-color: SYS B2,X,Y,C

C	Color combination code (range 0-255).
X,Y	(**Block-color only)** Horizontal and vertical coordinates of any point within the 8x8 pixel block that is to be colored (ranges 0-319 and 0-199).

ROUTINE LISTING

```
600 IF PEEK(49408)=173 THEN 630
610 SYS A3,650 : FOR C=49408 TO 49682
620 READ B : POKE C,B : NEXT C
630 B1=49559
640 B2=49634
650 DATA 173,8,192,72,41,7,141
660 DATA 1,192,104,41,248,133,253
670 DATA 173,12,192,72,41,7,141
680 DATA 2,192,104,41,248,72,74
690 DATA 74,74,141,3,192,74,74

700 DATA 24,109,3,192,141,3,192
710 DATA 104,10,10,10,13,2,192
720 DATA 24,101,253,133,253,173,3
730 DATA 192,109,9,192,24,105,32
740 DATA 133,254,169,128,174,1,192
750 DATA 232,202,240,4,74,56,176
760 DATA 249,141,0,192,96,173,26
770 DATA 192,41,248,141,26,192,24
780 DATA 110,27,192,110,26,192,110
790 DATA 27,192,110,26,192,110,27

800 DATA 192,110,26,192,169,0,133
810 DATA 252,173,28,192,74,74,74
820 DATA 141,4,192,10,10,24,109
830 DATA 4,192,10,10,38,252,10
840 DATA 38,252,24,109,26,192,133
850 DATA 251,169,4,101,252,109,27
860 DATA 192,133,252,96,32,40,192
870 DATA 140,5,192,169,32,133,252
880 DATA 169,0,133,251,165,252,201
890 DATA 63,208,6,165,251,201,64

900 DATA 240,13,162,0,138,129,251
910 DATA 230,251,208,235,230,252,208
920 DATA 231,169,4,133,252,169,0
930 DATA 133,251,165,252,201,7,208
940 DATA 6,165,251,201,232,240,15
950 DATA 162,0,173,5,192,129,251
960 DATA 230,251,208,233,230,252,208
970 DATA 229,96,32,40,192,32,224
980 DATA 192,144,7,32,6,169,32
990 DATA 251,168,96,142,27,192,140

1000 DATA 26,192,32,40,192,32,236
1010 DATA 192,176,235,140,28,192,32
1020 DATA 40,192,140,5,192,32,82
1030 DATA 193,160,0,173,5,192,145
1040 DATA 251,96
```

PICTURES WITH POINTS

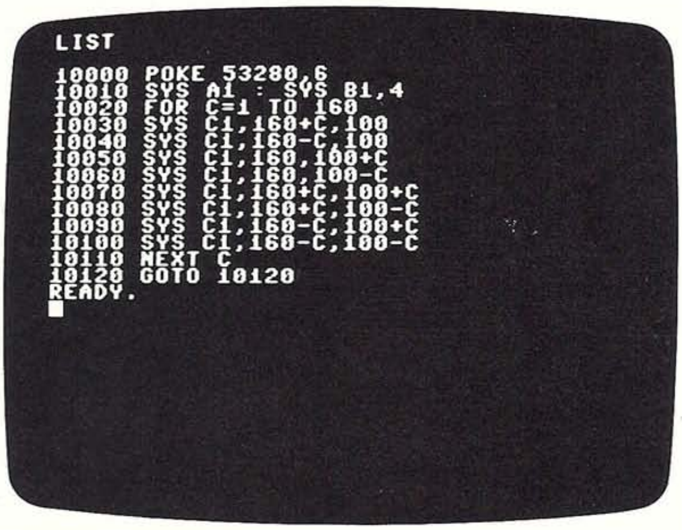
In any high-resolution picture there are a number of elements from which the display is constructed. The fundamental element is a simple point, a single lit pixel. Once you can plot points, all the other graphics objects like lines and circles can be produced by plotting in a specific way.

The Commodore doesn't have a PLOT command in its BASIC. However, the single routine in block C on the opposite page gives you this facility. Once this routine's machine code is in memory, you can use it with the following kind of command:

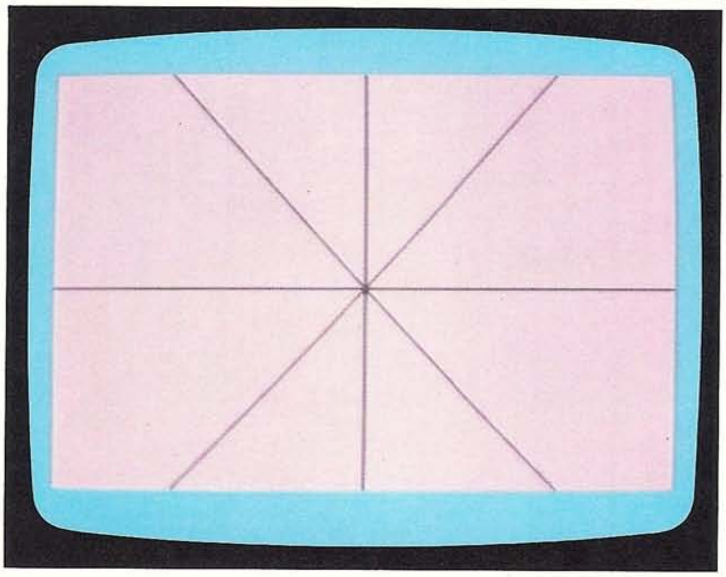
```
SYS C1,X,Y
```

This will plot a single point at position X,Y on the high-resolution grid. You can see the plot routine at work in the pair of programs on these two pages. Each of them uses the routine in a different way. The first plots predictably, while the second is semi-random.

POINT STAR PROGRAM



```
LIST
10000 POKE 53280,6
10010 SYS A1 : SYS B1,4
10020 FOR C=1 TO 160
10030 SYS C1,160+C,100
10040 SYS C1,160-C,100
10050 SYS C1,160,100+C
10060 SYS C1,160,100-C
10070 SYS C1,160+C,100+C
10080 SYS C1,160+C,100-C
10090 SYS C1,160-C,100+C
10100 SYS C1,160-C,100-C
10110 NEXT C
10120 GOTO 10120
READY.
```

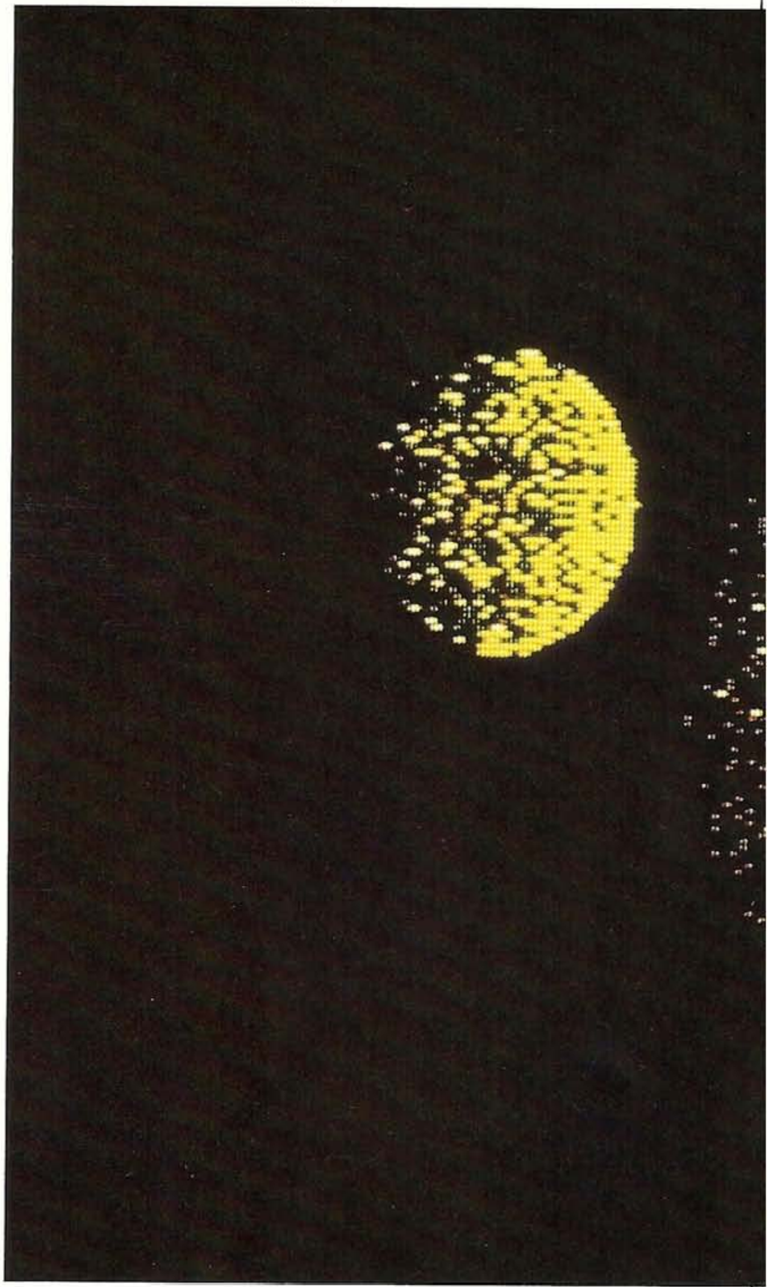


Drawing lines with the plot routine

One simple way of using the point-plotter is to make it produce lines by plotting rows of points close together. The Point Star program makes the plot routine produce a star. It uses routines in blocks A, B and C, so you will need all these in memory as well as the listing before you run it.

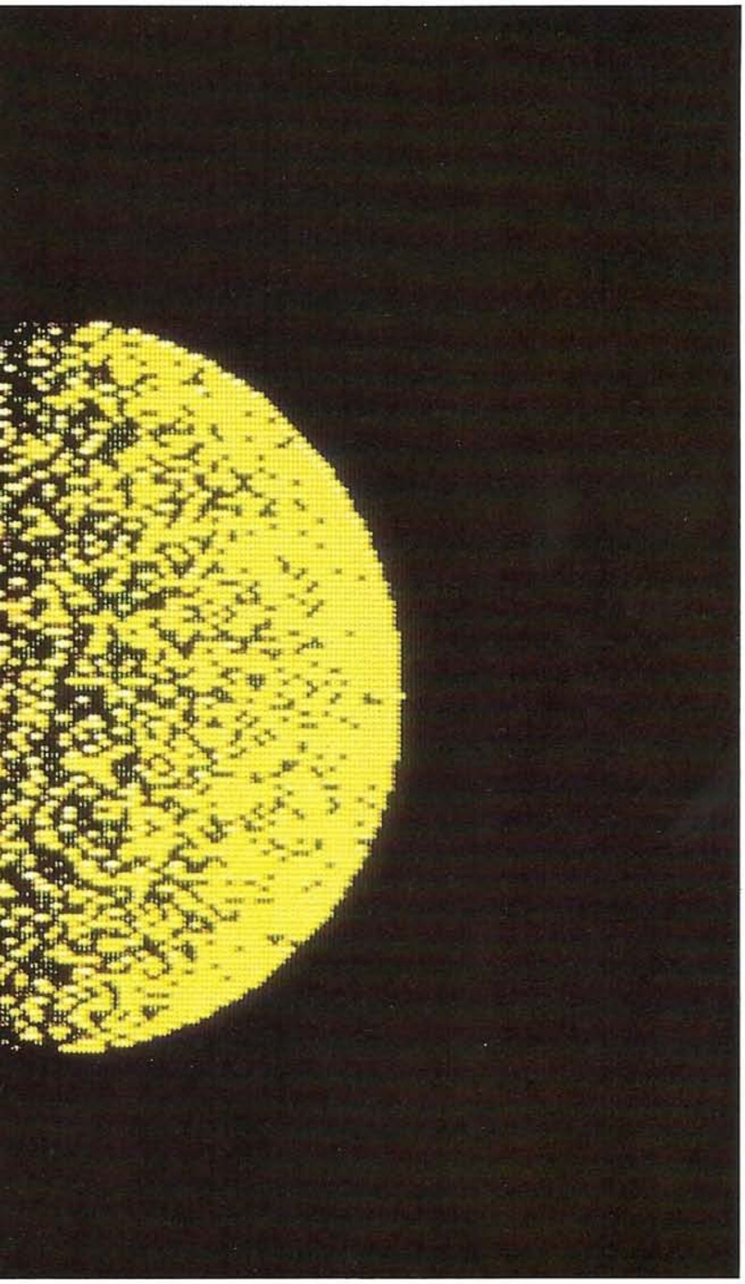
Shading with the plot routine

If you draw lines with the plot routine, pixels are plotted in a regular way. But another technique that you can try out with the routine involves plotting points more densely in one part of an object than in another. The Planets program uses this method to produce an almost three-dimensional display.



PLANETS PROGRAM

```
LIST
10000 SYS A1 : SYS B1,112 : POKE 53280,0
10010 R=60 : XC=180 : YC=100
10020 GOSUB 10060
10030 R=25 : XC=90 : YC=70
10040 GOSUB 10060
10050 GOTO 10050
10060 FOR Y=-R TO R
10070 X1=INT(SQR(R*R-Y*Y))
10080 FOR X=-X1 TO X1
10090 N=INT(RND(1)*X1*2)+1
10100 IF N<X1+X THEN SYS C1,X+XC,Y+YC
10110 NEXT X : NEXT Y : RETURN
READY.
```



BLOCK C

PLOT routine

What the routine does
The routine plots a single pixel at a specified point on the high-resolution screen (the screen coordinates are shown in the grid on page 61).

SYNTAX AND PARAMETERS	
SYS C1,X,Y	
X,Y	Horizontal and vertical coordinates of the point to be plotted (ranges 0-319 and 0-199).

ROUTINE LISTING

```
1200 IF PEEK(49712)=32 THEN 1230
1210 SYS A3,1240 : FOR C=49712 TO 49787
1220 READ B : POKE C,B : NEXT C
1230 C1=49712
1240 DATA 32,40,192,32,224,192,169
1250 DATA 0,144,2,165,1,141,24
1260 DATA 192,142,9,192,140,8,192
1270 DATA 32,40,192,32,236,192,169
1280 DATA 0,144,2,169,1,13,24
1290 DATA 192,141,24,192,142,13,192

1300 DATA 140,12,192,240,1,96,32
1310 DATA 0,193,160,0,173,29,192
1320 DATA 201,0,240,8,177,253,77
1330 DATA 0,192,145,253,96,177,253
1340 DATA 13,0,192,145,253,96
```

Random numbers and shading

The Planets program uses random numbers to decide which pixels within a boundary should be plotted. As each pixel in each horizontal line of the planet shape is considered in turn, the random function determines whether or not the pixel should be plotted. The probability of any point being plotted is made to depend on how far along the line, from left to right, the point lies. The left-most point is never plotted while the right-most point almost certainly is. In this way the brightness increases across the width in a realistic fashion. The program is written so that the total length of each line of pixels varies, producing a circular outline. However, the shading technique will work with any regular outline.

PLANETS PROGRAM

06:30

How the program works
The program uses a BASIC subroutine which plots rows of points, varying the width of each row to produce a circular outline. The plot routine is called so that it comes into operation most frequently toward the right of each row.
Line 10000 sets up the high-resolution screen and selects the foreground and background colors.
Lines 10010 and 10030 select two pairs of coordinates which specify the center of each planet.
Lines 10020 and 10040 call the subroutine which carries out the plotting.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
B Clear-and-color	13
C Plot	15

LINE GRAPHICS 1

As you saw in the Point Star program on page 14, you can use FOR . . . NEXT loops to plot straight lines, as long as the X and Y coordinates are related to each other in a simple way. But this is rather limiting, because often you will want to draw lines with slopes that are difficult to work out in this way.

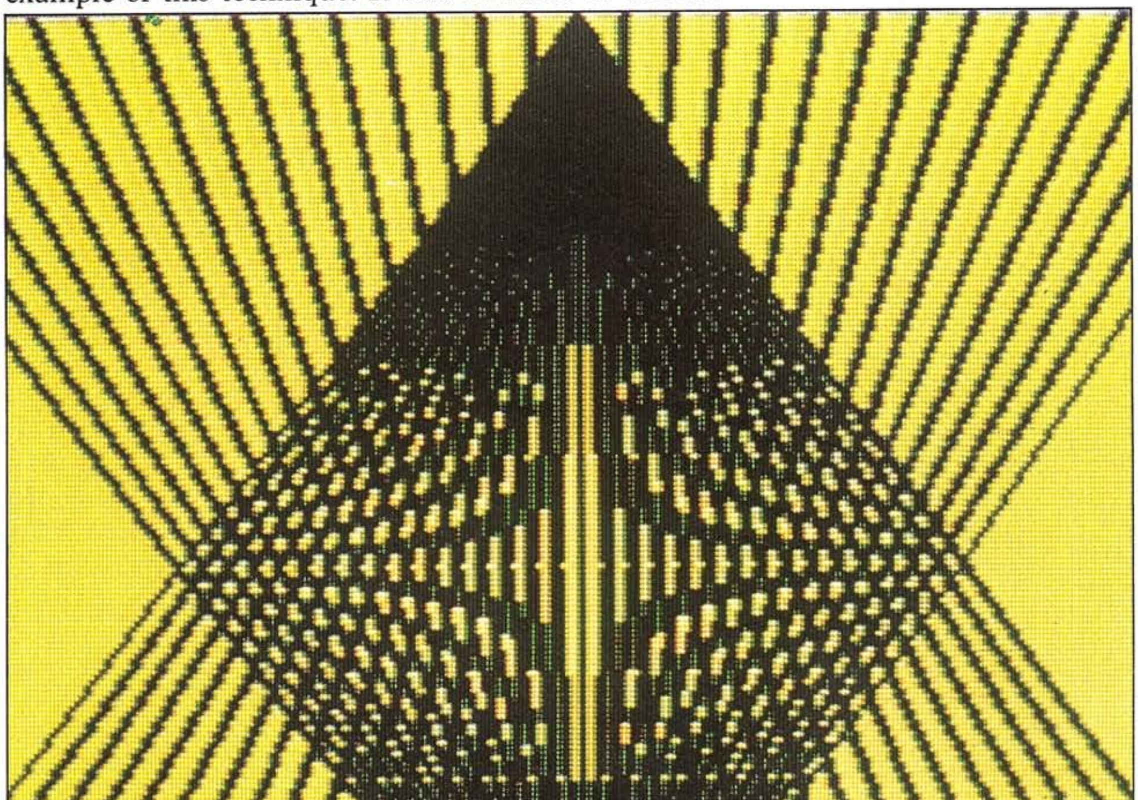
To draw a straight line between any pair of points, you need to use the draw routine. This is the single routine contained in block D on the opposite page. It's much longer than the plot routine in block C because it has a lot more work to do. With this block, the command SYS D1,X,Y will make the computer draw a line from the last point it visited to the point at X,Y.

Line designs

With computers that feature a DRAW command, it's easy to produce designs that use the command with STEP to make interesting displays. With the draw routine in memory, you can do this with the Commodore. The following program is a simple example of this technique. It uses routines in all four blocks from A to D, so these blocks must all be in memory before you can run the program. It draws a lattice-like web of lines.

LINE WEB PROGRAM

```
LIST

10000 POKE 53280,4
10010 SYS A1 : SYS B1,7 : SYS C1,0,0
10020 FOR X=0 TO 319 STEP 10
10030 SYS D1,X,0
10040 SYS D1,150,200
10050 NEXT X
10060 FOR X=0 TO 319 STEP 10
10070 SYS D1,150,0
10080 SYS D1,X,199
10090 NEXT X
10100 GOTO 10100
READY.
```



LINE WEB PROGRAM

00:12

How the program works
The program uses the draw routine to produce interconnecting lines. The plot routine (line 10010) is used to reset the draw routine's last coordinate to 0,0. Try removing the SYS C1 command and see what happens if you run the program more than once.
Lines 10000-10010 set up and color the high-resolution screen.
Lines 10020-10050 form the first loop that draws lines from the top of the screen down to a single point at the bottom.
Lines 10060-10090 form a loop which repeats the process upside-down.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
B Clear-and-color	13
C Plot	15
D Draw	17

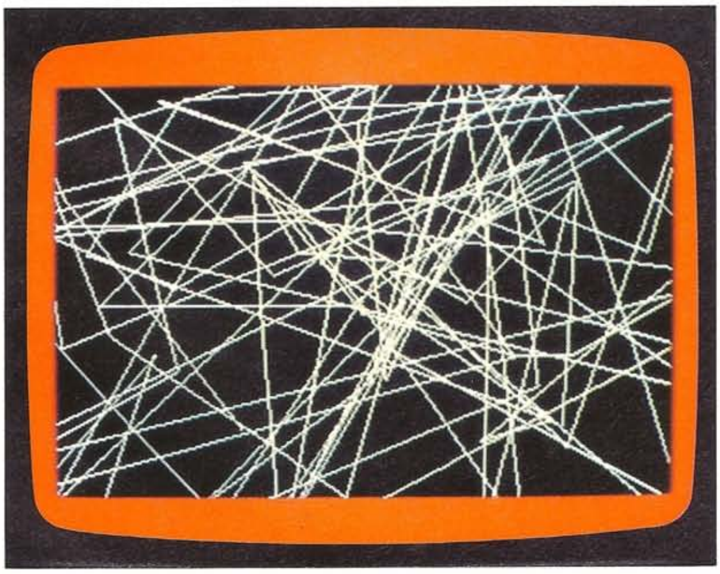
Testing the draw routine's memory

Making the computer draw a line is substantially more complex than instructing it to plot a point, as the length of block D shows. Much of the machine-code in this block is concerned with making the computer remember which was the last point it visited. You can see how it uses this information if you try out the next program. To run it, you will need to load or key in blocks A, B and D, if you don't still have them in memory from the previous program. If you have all the blocks in memory, and block C as well, you don't have to do anything with the machine-code routines. Just change the main program from line 10000 upward.

After line 10010 has set up the high-resolution screen, lines 10020-10070 select two random coordinates, and then draw a line to this point from the last point visited. The draw routine has to remember and update the current last point. If a point is off-screen, the routine will still remember where it is although it cannot actually be seen. This means that the program can continue even if its results are invisible.

RANDOM LINE PROGRAM

```
LIST

10000 POKE 53280,2
10010 SYS A1 : SYS B1,27
10020 FOR C=1 TO 100
10030 X=INT(RND(1)*400-40)
10040 Y=INT(RND(1)*280-40)
10050 SYS D1,X,Y
10060 FOR T=1 TO 300 : NEXT T
10070 NEXT C
10080 GOTO 10080
READY.
```



BLOCK D

DRAW routine

What the routine does

The routine draws a line from the last point visited to the point specified. The routine accepts a pair of coordinates which set the final point in the line (the screen coordinates are shown in the grid on page 63). The draw routine is not restricted to working only with points that lie within the screen boundary. If either or both of the points involved are off the screen, the line will still appear correctly, but will be "clipped" by the screen edges. The draw routine is essential for the operation of the circle and arc routines (see page 21).

SYNTAX AND PARAMETERS	
SYS D1,X,Y	
X,Y	Horizontal and vertical coordinates of the line end-point (ranges 0-319 and 0-199; higher values will be accepted but will produce off-screen images).

ROUTINE LISTING

```
1500 IF PEEK(49792)=173 THEN 1530
1510 SYS A3,1540 : FOR C=49792 TO 50180
1520 READ B : POKE C,B : NEXT C
1530 D1=49792
1540 DATA 173,24,192,141,25,192,32
1550 DATA 40,192,32,224,192,169,0
1560 DATA 144,2,169,1,141,24,192
1570 DATA 142,7,192,140,6,192,32
1580 DATA 40,192,32,236,192,169,0
1590 DATA 144,2,169,1,13,24,192

1600 DATA 141,24,192,13,25,192,141
1610 DATA 25,192,142,11,192,140,10
1620 DATA 192,56,173,6,192,237,8
1630 DATA 192,170,173,7,192,237,9
1640 DATA 192,168,16,17,56,138,73
1650 DATA 255,105,0,170,152,73,255
1660 DATA 105,0,168,169,255,208,17
1670 DATA 208,4,224,0,240,9,169
1680 DATA 0,141,15,192,169,1,208
1690 DATA 5,169,0,141,15,192,141

1700 DATA 14,192,142,18,192,140,19
1710 DATA 192,56,173,10,192,237,12
1720 DATA 192,170,173,11,192,237,13
1730 DATA 192,168,16,17,56,138,73
1740 DATA 255,105,0,170,152,73,255
1750 DATA 105,0,168,169,255,208,17
1760 DATA 208,4,224,0,240,9,169
1770 DATA 0,141,17,192,169,1,208
1780 DATA 5,169,0,141,17,192,141
1790 DATA 16,192,142,20.192,140,21

1800 DATA 192,142,22,192,140,23,192
1810 DATA 173,29,192,208,3,32,205
1820 DATA 195,173,23,192,48,45,208
1830 DATA 5,173,22,192,240,38,56
1840 DATA 173,22,192,237,18,192,141
1850 DATA 22,192,173,23,192,237,19
1860 DATA 192,141,23,192,24,173,12
1870 DATA 192,109,16,192,141,12,192
1880 DATA 173,13,192,109,17,192,141
1890 DATA 13,192,173,23,192,48,7

1900 DATA 208,43,173,22,192,208,38
1910 DATA 24,173,22,192,109,20,192
1920 DATA 141,22,192,173,23,192,109
1930 DATA 21,192,141,23,192,24,173
1940 DATA 8,192,109,14,192,141,8
1950 DATA 192,173,9,192,109,15,192
1960 DATA 141,9,192,32,205,195,173
1970 DATA 8,192,205,6,192,208,145
1980 DATA 173,9,192,205,7,192,208
1990 DATA 137,173,12,192,205,10,192

2000 DATA 208,129,173,13,192,205,11
2010 DATA 192,208,230,96,173,25,192
2020 DATA 208,28,32,0,193,160,0
2030 DATA 173,29,192,201,0,240,8
2040 DATA 177,253,77,0,192,145,253
2050 DATA 96,177,253,13,0,192,145
2060 DATA 253,96,174,9,192,172,8
2070 DATA 192,32,224,192,176,11,174
2080 DATA 13,192,172,12,192,32,236
2090 DATA 192,144.206,96
```

LINE GRAPHICS 2

When you use the draw routine, you can either specify separately each line that you want the computer to draw or you can use a program sequence to specify a number of lines. Drawing parallel lines a set distance apart is quite easy — you just use a FOR . . . NEXT loop with STEP. However, with a slightly different kind of program, you can produce series of lines which combine to produce special visual effects.

Radiating patterns

If you make the Commodore draw radiating lines close together, you will find that it produces some interesting patterns. This is because the screen resolution, although good, is limited. Sloping lines are actually drawn as a series of steps, and these sometimes combine to give unexpected secondary shapes. You can see this kind of pattern if you try out the Radiating Pattern program below. It uses routine blocks A-D. The type of pattern produced depends on how close together the lines are. Try altering the STEP value and see what happens.

Drawing diamonds

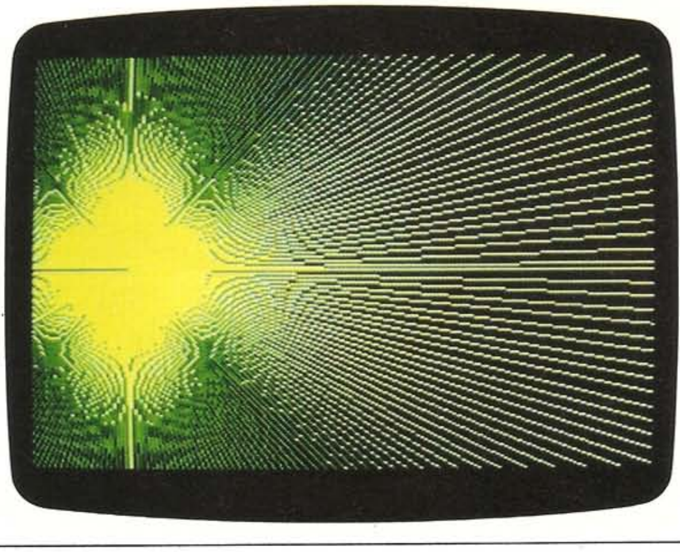
You can make the draw routine build up shapes if you use a loop. The Diamond program below does this — it draws a succession of diamonds, starting with the widest that will fit on the screen and then narrowing down. Note that if you change the STEP size to an even number, the shaded effect on the two diagonally opposite faces will disappear. The program uses blocks A-D.

Line landscapes

By using the draw routine and then adding different colors to parts of the screen with the block-color routine, you can build up quite complex pictures. The Line Landscape program shows you one way you can do this.

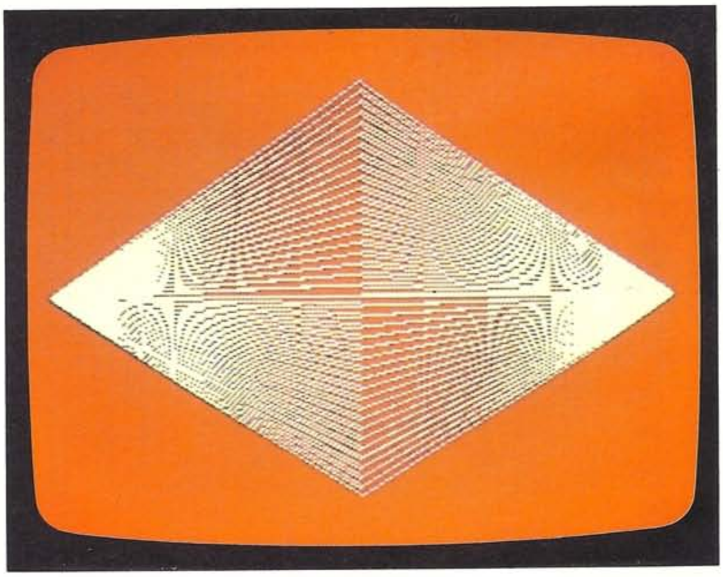
RADIATING PATTERN PROGRAM

```
LIST
10000 SYS A1 : SYS B1,208 : POKE 53280,0
10010 FOR A=0 TO 2*π STEP 0.025
10020 SYS C1,50,100
10030 SYS D1,50+300*COS(A),100+300*SIN(A
)
10040 NEXT A
10050 GOTO 10050
READY.
```



DIAMOND PROGRAM

```
LIST
10000 SYS A1 : SYS B1,18 : POKE 53280,2
10010 SYS C1,0,100
10020 FOR Y=0 TO 199 STEP 5
10030 SYS D1,160,Y
10040 SYS D1,319,100
10050 SYS D1,160,199-Y
10060 SYS D1,0,100
10070 NEXT Y
10080 GOTO 10080
READY.
```



LINE LANDSCAPE PROGRAM

```
10000 SYS A1 : SYS B1,112
10010 FOR X=0 TO 320 STEP 3
10020 SYS C1,X,144
10030 SYS D1,X*4,0
10040 SYS C1,X+1,144
10050 SYS D1,X*4+1,0
10060 NEXT X : Y=144
10070 FOR C=2 TO 10
10080 SYS C1,0,Y
10090 SYS D1,319,Y
10100 Y=Y+C : NEXT C
10110 LX=0 : UX=312 : LY=144
10120 UY=192 : C=13 : GOSUB 10190
10130 POKE 53280,12
10140 LX=184 : UX=208 : LY=88
10150 UY=144 : C=68: GOSUB 10190
10160 LX=248 : UX=296 : LY=56
10170 UY=168 : C=68: GOSUB 10190
10180 GOTO 10180
10190 FOR X=LX TO UX STEP 8
10200 FOR Y=LY TO UY STEP 8
10210 SYS B2,X,Y,C
10220 NEXT Y : NEXT X : RETURN
READY.
```

This program draws radiating lines that seem to come from a point hidden by a "horizon". Then, by drawing horizontal lines that get closer together away from the bottom of the screen, an illusion of distance is created. Finally, two "buildings" are produced by the block-color routine, using the same color for foreground and background so the underlying display is blanked out.

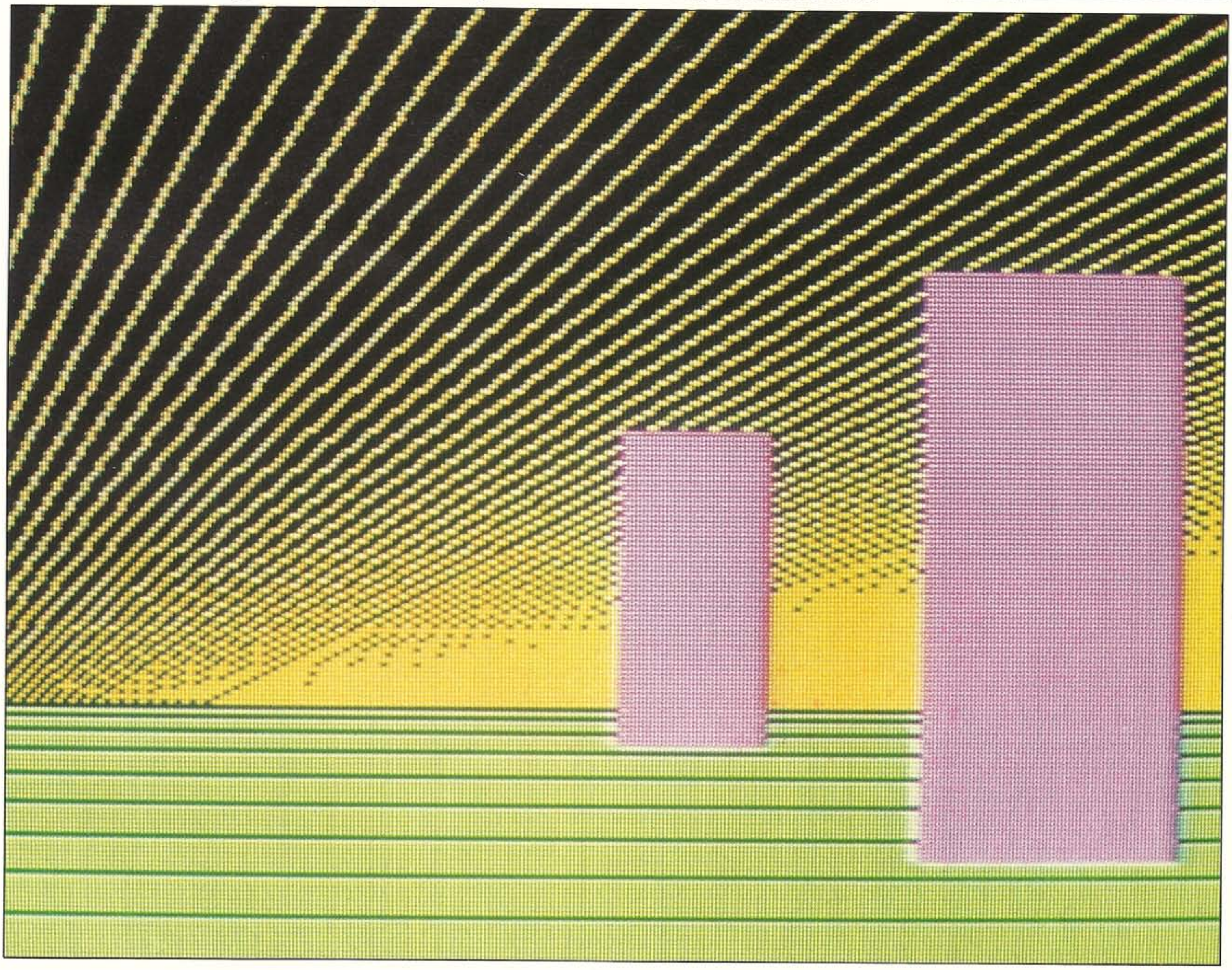
LINE LANDSCAPE PROGRAM

00:35

How the program works
The draw routine is used to display lines in two different ways. The sunset pattern is produced by gradually decreasing the slope of the lines above the horizon, while in the foreground the space between the lines is successively increased.
Lines 10010-10060 form a loop which draws lines of decreasing gradient between the top of the screen and a vertical value of 144.
Lines 10070-10100 draw the perspective lines in the bottom part of the screen.
Lines 10110-10220 use the block-color routine three times to color the shapes.

ROUTINES USED BY THIS PROGRAM

Page	Block Routine(s)
11	A High-resolution
13	B Clear-and-color Block-color
15	C Plot
17	D Draw



CIRCLES AND ARCS 1

The two routines in block E on the opposite page let you draw circles and partial circles, or arcs. Both of them work by drawing small straight lines, so it's essential that you always have block D, containing the draw routine, in memory when you use them.

To try the program that follows, load routine blocks A-D (remember that if you run block A first, you can use the merge routine to do this) and then add block E. Now key in the BASIC listing that follows. The program activates the circle routine with the command SYS E1, and the arc routine with the command SYS E2, using them a total of nine times. The circle routine uses three parameters while the arc routine uses five. They are all explained on the next page.

The routines you have loaded in will also enable the program to use DATA to control plotting. This facility is provided by the restore routine. Here it makes the program READ DATA from line 15000.

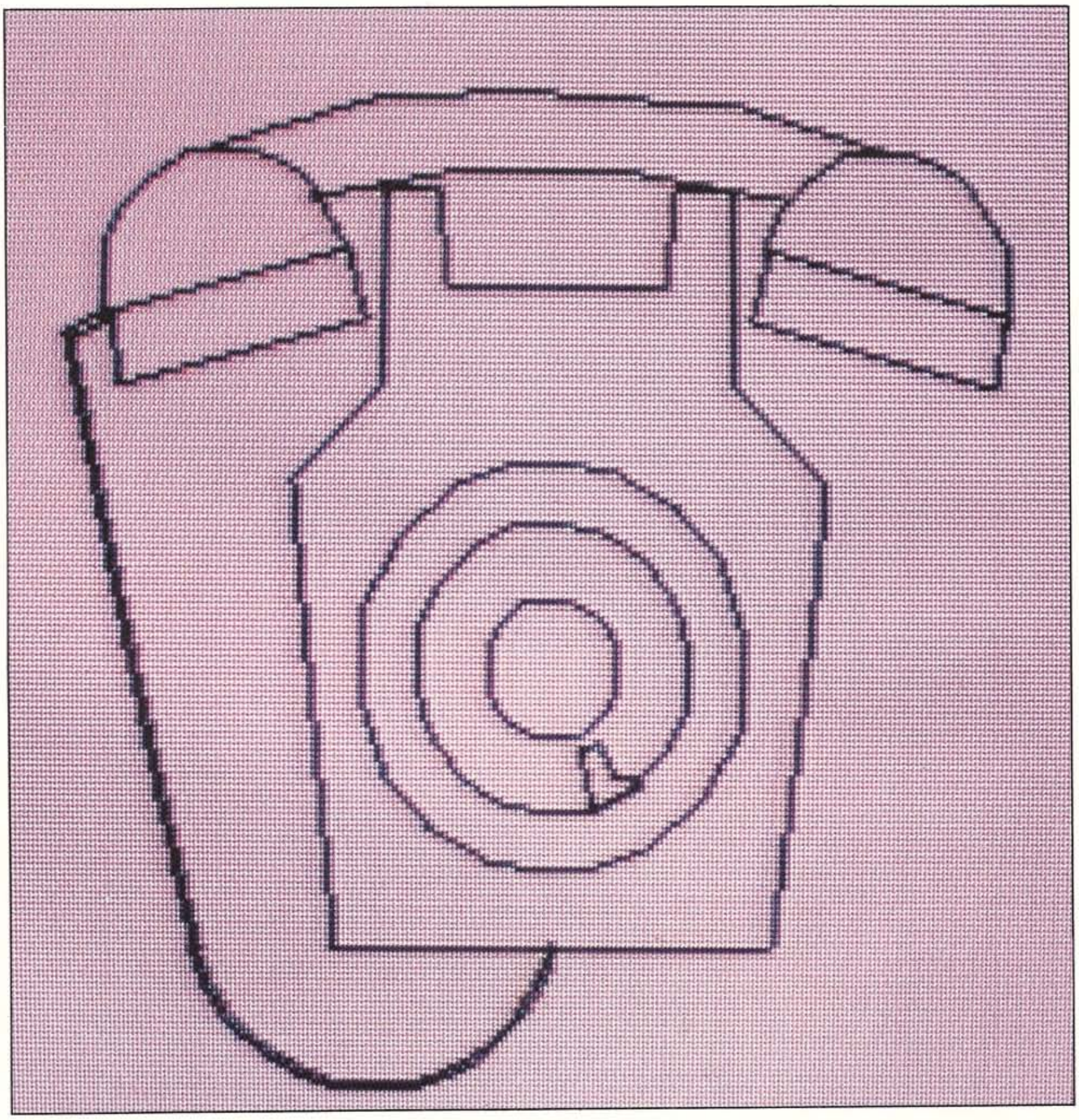
TELEPHONE PROGRAM

```
LIST 10000-10190

10000 SYS A1 : SYS B1,4
10010 SYS A3,15000
10020 F=0 : POKE 53280,6
10030 READ X,Y
10040 IF X=-1 THEN F=Y : GOTO 10030
10050 IF X<0 THEN 10090
10060 IF F=1 THEN SYS D1,X,Y
10070 IF F=0 THEN SYS C1,X,Y
10080 GOTO 10030
10090 SYS E2,160,220,196,251,289
10100 SYS E2,160,220,183,256,283
10110 SYS E2,100,56,22,165,345
10120 SYS E2,220,56,22,195,15
10130 SYS E1,160,120,35
10140 SYS E1,160,120,25
10150 SYS E1,160,120,12
10170 SYS E2,128,160,34,14,166
10180 SYS E2,128,160,33,14,166
10190 GOTO 10190

READY.
```

TELEPHONE PROGRAM

00:17

How the program works
All the instructions for the straight lines in the display are held in DATA statements. The program READs these in sequence, and then uses the plot or draw routines. It then adds the circles and arcs.
Line 10010 makes the program READ the DATA from line 15000 onward.
Lines 10030-10080 activate the plot and draw routines.
Lines 10090-10180 produce a total of nine circles and arcs.
Line 10190 stops the READY message spoiling the display.
Lines 15000-15140 contain DATA that select routines and fix coordinates.



ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
Restore	
B Clear-and-color	13
C Plot	15
D Draw	17
E Circle	21
Arc	

The second screen of the listing consists entirely of the DATA needed for plotting and drawing the straight parts of the display.

TELEPHONE PROGRAM (CONTD.)

```
LIST 15000-

15000 DATA 120,168,-1,1,112,88
15010 DATA 128,72,128,40,138,40
15020 DATA 140,56,180,56,182,40
15030 DATA 192,40,192,72,208,88
15040 DATA 200,168,120,168,-1,0
15050 DATA 168,144,-1,1,176,140
15060 DATA 175,138,171,138,168,132
15070 DATA 165,133,168,144,-1,0
15080 DATA 122,50,-1,1,125,61
15090 DATA 81,72,79,61,122,50,-1,0
15100 DATA 198,50,-1,1,195,61
15110 DATA 239,72,241,61,198,50,-1,0
15120 DATA 95,168,-1,1,71,64,80,60
15130 DATA 80,61,73,64,97,168
15140 DATA -2,0
READY.
```

How to use the circle and arc routines

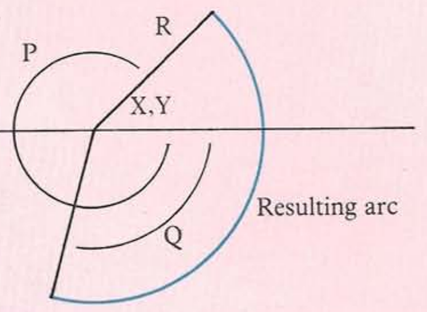
The circle routine is very straightforward to use. All you have to do is decide where you want the center of the circle to be, and how long you want its radius. If you want to draw a circle at the center of the screen (160,100) with a radius of 50 pixels, you would key in:

```
SYS E1,160,100,50
```

Using the arc routine requires a bit more planning. The parameters that you need to specify are the same as those for the circle, but in addition you need to supply two numbers — a starting angle (P) and a finishing angle (Q). This enables the routine to draw just part of a complete circle. Both the angles are measured in degrees starting at the positive horizontal axis, and turning clockwise around to the angles' radii. Suppose you wanted to draw an arc like this:

ARC ROUTINE PARAMETERS

X,Y=160,100
R=50
P=315
Q=105



The complete command for the arc would be:

```
SYS E2,160,100,50,315,105
```

Remember that all positions and lengths are in pixels, and all angles in degrees.

BLOCK E

CIRCLE and ARC routines

What the routines do
Circle draws a circle of a specified radius and center.
Arc draws part of a circle. Both use the draw routine.

SYNTAX AND PARAMETERS

Circle: SYS E1,X,Y,R Arc: SYS E2,X,Y,R,P,Q

X,Y	Horizontal and vertical coordinates of the center of the circle or arc (ranges 0-319 and 0-199; values higher than these will still be accepted but may produce off-screen images).
R	Length of radius in pixels (no range limit).
P	(**Arc** only) Angle at which the arc is to start, measured in degrees clockwise from positive horizontal axis (no range limit).
Q	(**Arc** only) Angle at which the arc is to finish, measured in degrees clockwise from positive horizontal axis (no range limit).

ROUTINE LISTING

```
2500 IF PEEK(50202)=169 THEN 2530
2510 SYS A3,2540 : FOR C=50192 TO 50549
2520 READ B : POKE C,B : NEXT C
2530 E1=50202 : E2=50225
2540 DATA 0,0,0,0,0,0,0
2550 DATA 0,0,0,169,1,141,4
2560 DATA 192,162,0,142,22,196,142
2570 DATA 23,196,162,104,142,24,196
2580 DATA 141,25,196,208,5,169,0
2590 DATA 141,4,192,32,40,192,140

2600 DATA 16,196,142,17,196,32,40
2610 DATA 192,140,18,196,142,19,196
2620 DATA 32,40,192,140,20,196,142
2630 DATA 21,196,173,4,192,208,18
2640 DATA 32,40,192,140,22,196,142
2650 DATA 23,196,32,40,192,140,24
2660 DATA 196,142,25,196,169,1,141
2670 DATA 24,192,141,25,192,32,229
2680 DATA 196,173,6,192,141,8,192
2690 DATA 173,7,192,141,9,192,173

2700 DATA 10,192,141,12,192,173,11
2710 DATA 192,141,13,192,32,211,196
2720 DATA 144,19,173,24,196,105,104
2730 DATA 141,24,196,173,25,196,105
2740 DATA 1,141,25,196,76,139,196
2750 DATA 173,22,196,105,9,141,22
2760 DATA 196,173,23,196,105,0,141
2770 DATA 23,196,32,211,196,176,9
2780 DATA 32,229,196,32,185,194,76
2790 DATA 163,196,173,24,196,141,22

2800 DATA 196,173,25,196,141,23,196
2810 DATA 32,229,196,76,185,194,56
2820 DATA 173,24,196,237,22,196,173
2830 DATA 25,196,237,23,196,56,48
2840 DATA 1,24,96,169,0,141,4
2850 DATA 192,32,70,197,32,100,226
2860 DATA 32,12,188,172,20,196,173
2870 DATA 21,196,32,145,179,165,97
2880 DATA 32,43,186,32,191,177,24
2890 DATA 165,101,109,16,196,141,6

2900 DATA 192,165,100,109,17,196,141
2910 DATA 7,192,169,1,141,4,192
2920 DATA 32,70,197,32,100,226,32
2930 DATA 12,188,172,20,196,173,21
2940 DATA 196,32,145,179,165,97,32
2950 DATA 43,186,32,191,177,24,165
2960 DATA 101,109,18,196,141,10,192
2970 DATA 165,100,109,19,196,141,11
2980 DATA 192,96,169,224,160,226,32
2990 DATA 140,186,160,90,169,0,32

3000 DATA 145,179,165,97,32,18,187
3010 DATA 32,12,188,172,22,196,24
3020 DATA 173,4,192,240,6,152,105
3030 DATA 14,168,169,1,109,23,196
3040 DATA 32,145,179,165,97,76,43
3050 DATA 186
```

CIRCLES AND ARCS 2

One technique which may be new to you, but which can be used to great effect, is recursion. Recursion means repetition, but it's repetition of a special kind. On these two pages, you can develop a program that produces recursive patterns with circles.

Recursion with circles

The listings opposite show you one of the big advantages of recursion — programs that are quite short can produce complex displays. To see the first display below, load blocks A, B, D and E, if you don't have them in memory already, key in the first listing opposite, and then run it. The program repeatedly draws smaller and smaller circles until it has produced a sequence of seven, and then it starts the process again from another position.

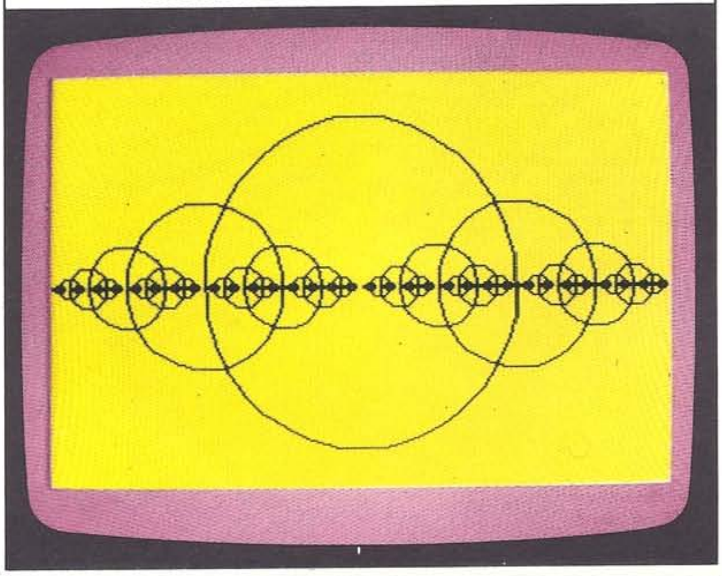
Once you have run this program, you can start to alter it. The most obvious change you can make is to the limit set by line 20000. After you have done that, try altering the values in line 20010. Here is one way to do it:

```
20010 SYS E1,X(L),100+13*L,R(L)
```

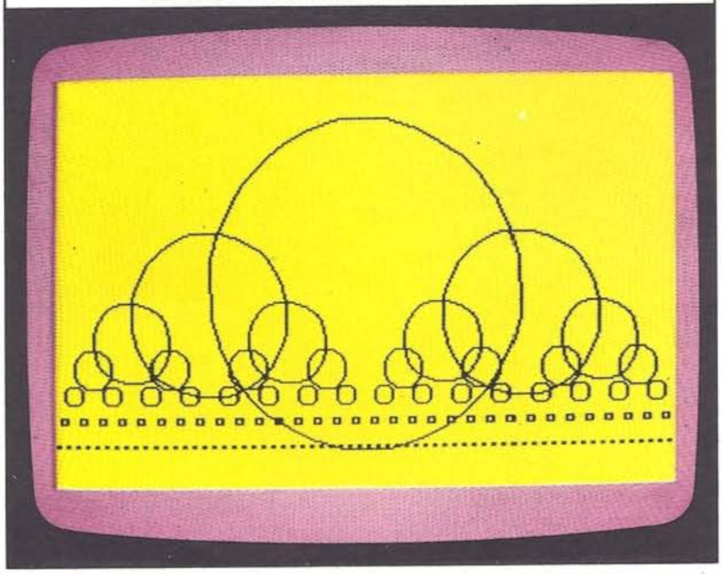
This makes the height of each circle above the bottom of the screen vary, as you can see from the second of the small displays below.

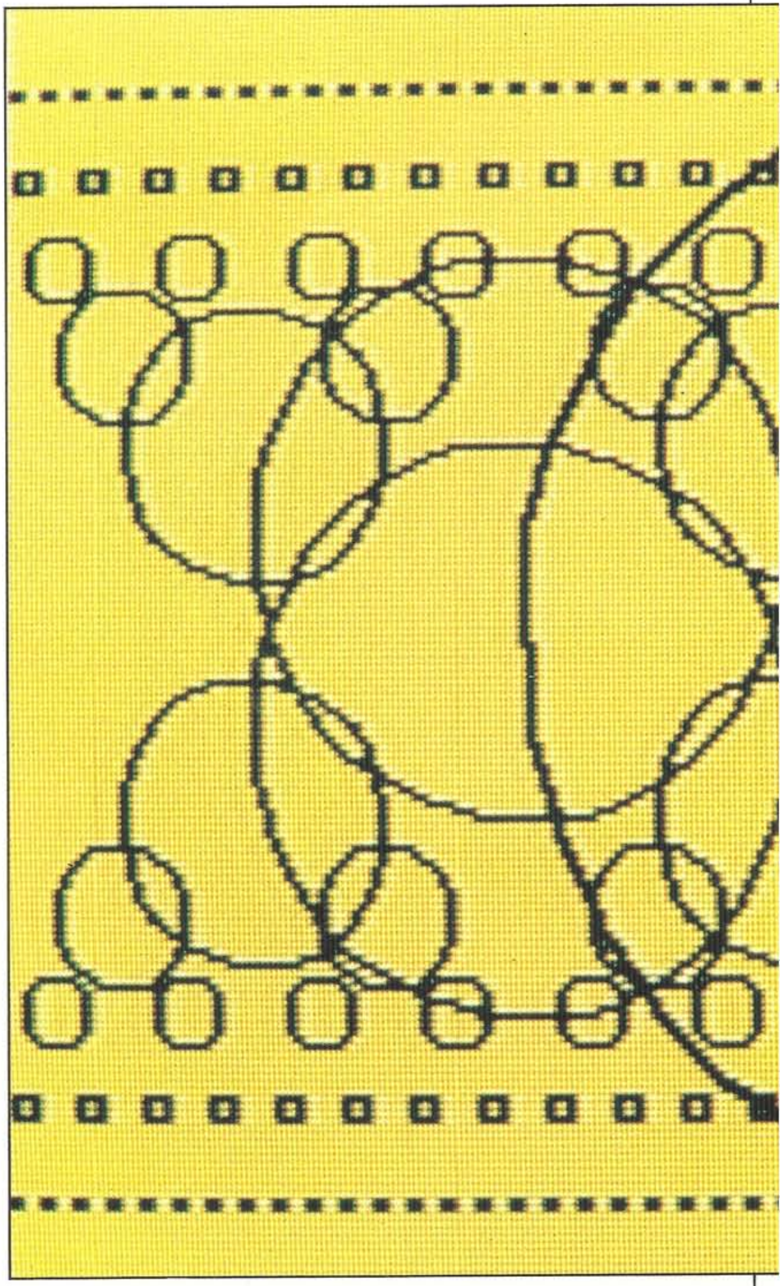
After you have made the program draw circles at different heights, you can extend it so that the pattern is reflected in the horizontal axis as well. All that is needed is a second BASIC subroutine, starting at line 30000, and a line to call it. The altered program is the second listing opposite. It produces the big display shown below.

RECURSIVE CIRCLES DISPLAY



ADAPTED RECURSIVE CIRCLES



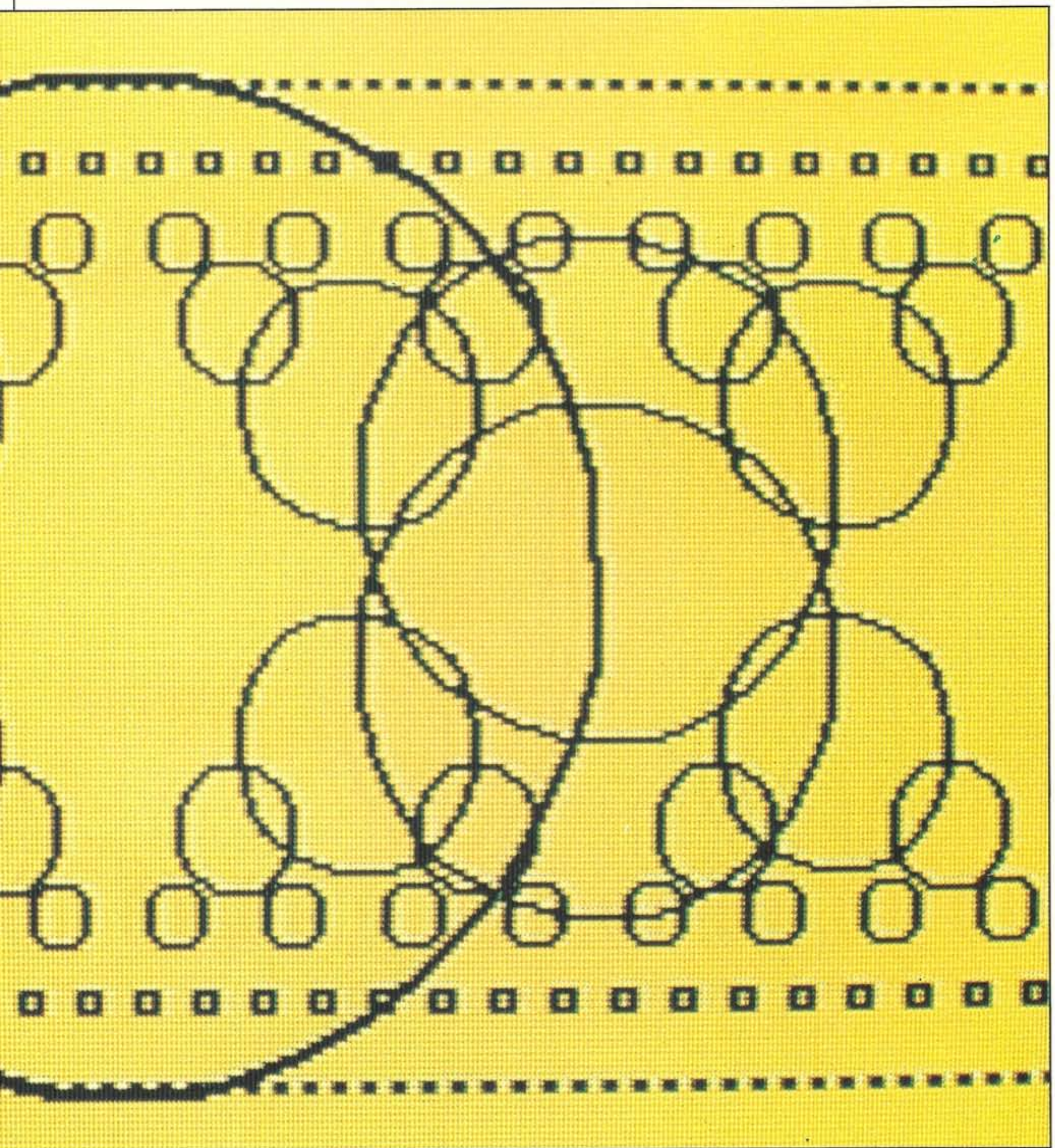


RECURSIVE CIRCLES PROGRAM

```
LIST

10000 SYS A1 : SYS B1,7
10010 L=0 : X(L)=160 : R(L)=80
10020 POKE 53280,4
10030 GOSUB 20000
10040 GOTO 10040
20000 IF L=7 THEN L=L-1 : RETURN
20010 SYS E1,X(L),100,R(L)
20020 R(L+1)=R(L)/2
20030 X(L+1)=X(L)+R(L)
20040 L=L+1 : GOSUB 20000
20050 X(L+1)=X(L)-R(L)
20060 L=L+1 : GOSUB 20000
20070 L=L-1 : RETURN
READY.
```

DOUBLE RECURSION PROGRAM

```
10000 SYS A1 : SYS B1,7
10010 L=0 : X(L)=160 : R(L)=80
10020 POKE 53280,4
10030 GOSUB 20000
10040 L=0 : X(L)=160 : R(L)=80
10050 GOSUB 30000
10060 GOTO 10060
20000 IF L=7 THEN L=L-1 : RETURN
20010 SYS E1,X(L),100+13*L,R(L)
20020 R(L+1)=R(L)/2
20030 X(L+1)=X(L)+R(L)
20040 L=L+1 : GOSUB 20000
20050 X(L+1)=X(L)-R(L)
20060 L=L+1 : GOSUB 20000
20070 L=L-1 : RETURN
30000 IF L=7 THEN L=L-1 : RETURN
30010 SYS E1,X(L),199-(100+13*L),R(L)
30020 R(L+1)=R(L)/2
30030 X(L+1)=X(L)+R(L)
30040 L=L+1 : GOSUB 30000
30050 X(L+1)=X(L)-R(L)
30060 L=L+1 : GOSUB 30000
30070 L=L-1 : RETURN
READY.
```



DOUBLE RECURSION PROGRAM

06:30

How the program works
The program repeatedly calls the circle routine. Every time it does so, the horizontal position and radius of the circle is set by values stored as array variables.
Line 10000 sets up the high-resolution screen and the colors.
Line 10060 loops back on itself after both sets of recursions have been completed.
Lines 20000-20070 form a BASIC subroutine which repeatedly draws circles with different radii at different coordinates, until the limiting condition in line 20000 is met.
Lines 30000-30070 form a second subroutine which produces a mirror-image of the display created by the first.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
B Clear-and-color	13
D Draw	17
E Circle	21

OVERPRINTING AND ERASING

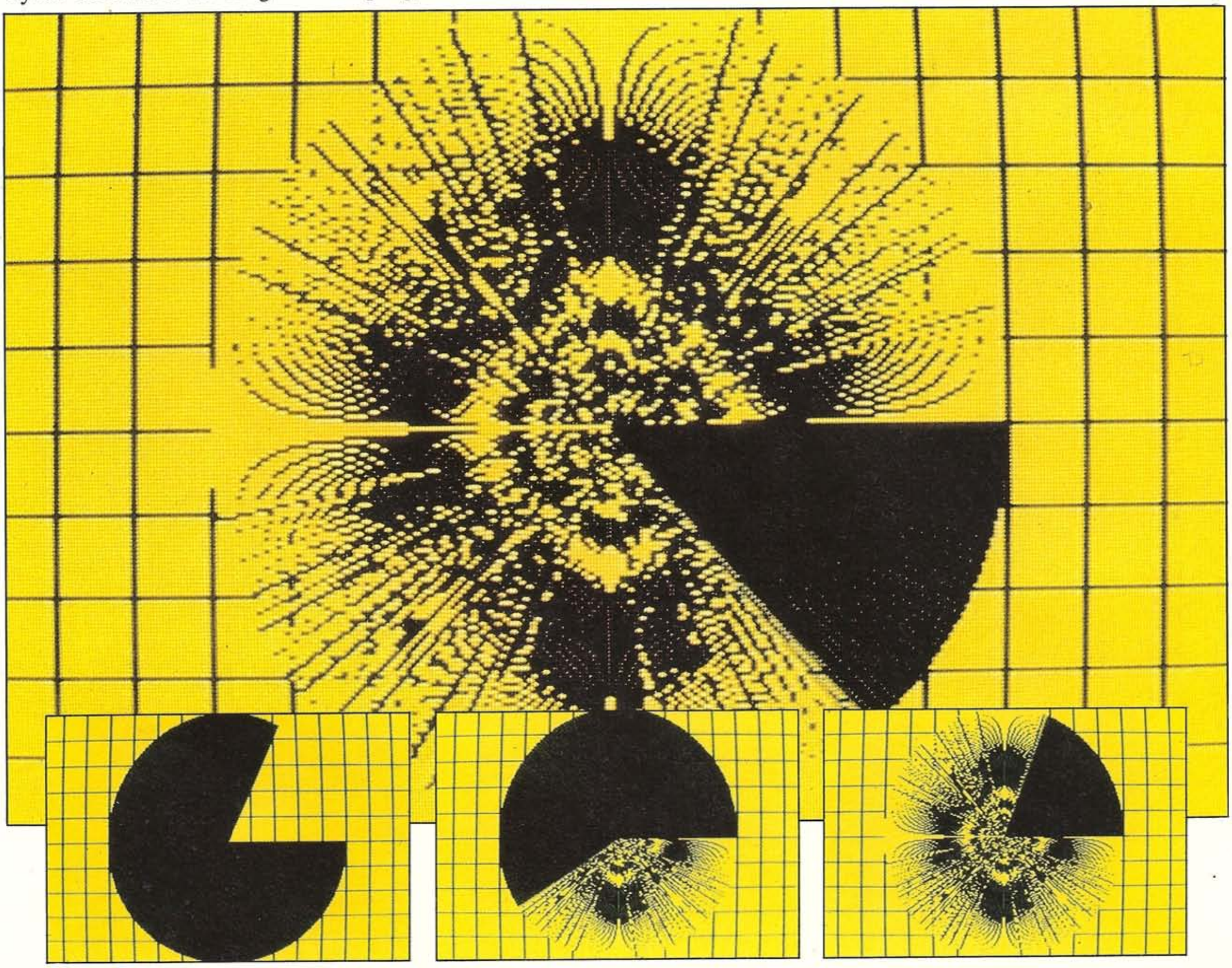
Normally if you print one shape over another on the Commodore, the second simply replaces the first. However, with the routine in block F on the opposite page you can achieve some different effects. This routine is activated by SYS F1,1 and is turned off by SYS F1,0. It's called the erase routine, but as you will see, this is a simplification because its effects can be quite subtle.

Patterns with the erase routine

With heavily colored shapes, you can sometimes produce some interesting patterns by using the erase routine to overprint them. The next program produces a practically solid circle on a grid by drawing hundreds of radiating lines. When the circle is first drawn, it is solid, but when it is overprinted the erase routine creates a pattern as the lines are canceled out. This cycle continues as long as the program runs.

OVERPRINTED CIRCLE PROGRAM

```
LIST

10000 SYS A1 : SYS B1,7 : SYS F1,0 : POK
E 53280,5
10010 FOR X=0 TO 319 STEP 20
10020 SYS C1,X,0
10030 SYS D1,X,199
10040 NEXT X
10050 FOR Y=0 TO 199 STEP 20
10060 SYS C1,0,Y
10070 SYS D1,319,Y
10080 NEXT Y
10090 FOR A=0 TO 2*π STEP 0.01
10100 SYS C1,160,100
10110 SYS D1,160+100*COS(A),100+100*SIN(
A)
10120 NEXT A
10130 SYS F1,1 : GOTO 10090
READY.
```



ROTATING SQUARES PROGRAM

```
LIST

10000 SYS A1 : SYS B1,22
10010 POKE 53280,8 : SYS F1,1
10020 FOR C=250 TO 0 STEP -5
10030 A=5-C/50
10040 X=C*COS(A) : Y=C*SIN(A)
10050 SYS C1,X+160,Y+100
10060 SYS C1,X+160,Y+100
10070 SYS D1,160-Y,100+X
10080 SYS D1,160-X,100-Y
10090 SYS D1,160+Y,100-X
10100 SYS D1,160+X,100+Y
10110 NEXT C : GOTO 10020
READY.
```

ROTATING SQUARES DISPLAY

How to use the erase routine

The Rotating Squares program above shows you how the erase routine can be used to wipe away a design. In this program, a nest of squares is built up on the screen.

OVERPRINTED CIRCLE PROGRAM

01:20

How the program works
The grid and solid circle are drawn while the erase routine is turned off. It is then turned on, so that when the circle is repeated it produces a canceled-pixel pattern. The time given above is for the program to draw through 360 degrees.
Line 10000 switches off the erase routine.
Lines 10090-10120 draw the solid circle.
Line 10130 starts the process again, but this time with the erase routine switched on.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
B Clear-and-color	13
C Plot	15
D Draw	17
F Erase	25

BLOCK F

ERASE routine

What the routine does
The routine is used to affect the operation of the previous graphics routines by activating an "exclusive-OR" mode on the screen. This means that after the erase routine is turned on, a pixel plotted over one already lit will cancel it out. A pixel plotted on an unlit (background) pixel will appear normally. The routine can therefore be used to erase all or part of a display by redrawing it. Note that this routine must be turned off when drawing closed shapes which are later to be filled.

SYNTAX AND PARAMETERS

SYS F1,N

N	Off or on (0=off, 1=on).

ROUTINE LISTING

```
3200 IF PEEK(50560)=32 THEN 3230
3210 SYS A3,3240 : FOR C=50560 TO 50578
3220 READ B : POKE C,B : NEXT C
3230 F1=50560
3240 DATA 32,40,192,152,208,7,138
3250 DATA 208,4,141,29,192,96,169
3260 DATA 1,141,29,192,96
```

When the program repeats itself, instead of just overprinting the design, the computer starts to erase it. This is caused by line 10010. The SYS F1,1 command turns on an "exclusive-OR" drawing facility. What this means is that whenever the computer overprints a pixel lit in the foreground color, it cancels it out, turning it off. Because the lines are quite far apart, they just disappear, unlike those in the Overprinted Circle program, which are close enough together to affect each other.

Points to watch with the erase routine

If you want to remove a display from the screen, always erase by drawing again in exactly the same order. You can do this quite simply by looping the program back with the erase routine switched on. However, remember that if your display contains many lines close together or overlapping, you may not be able to erase them all without producing an effect of the kind shown by the Overprinted Circle program.

When you are drawing closed shapes with the erase routine turned on, you will find that final points plotted on complete outlines are canceled out, leaving a single pixel gap. This is a problem if you later want to fill a shape. Therefore always keep the erase routine turned off when you are not using it. It's a good idea to switch it off from within a program when a display is completed. However, if you suspect that you have left the routine on, you can switch it off by a direct command.

FILLING SHAPES 1

Having found out how to draw outlines, the next step is filling them in. Block G on the opposite page contains a flood-fill routine, that is, it rapidly fills closed shapes with solid color. It will fill almost any regular or irregular shape. All you have to do is specify any point inside the shape and the routine will fill all around the point until it reaches the boundaries.

How the flood-fill routine works

What the flood-fill routine does is to search for background pixels and light them in the foreground color. It continues to search in each particular direction until it meets a boundary of lit pixels or until it reaches the edge of the screen. So when you are designing your own pictures using the flood-fill routine, you must take care that there are no gaps in the boundary surrounding the area to be filled, or the color will "leak" out into areas you may not have planned to fill.

You will find that this routine will fill almost any shape that you want it to. It "remembers" to go back to regions of a shape so that all of it is filled in. However, due to the limited memory space available for the routine's calculations, you may come across shapes that the routine cannot deal with. If this happens, you will get an ILLEGAL QTY ERR message. To overcome this problem, all you have to do is split up the area to be filled into a number of smaller, simpler areas and flood-fill each one separately.

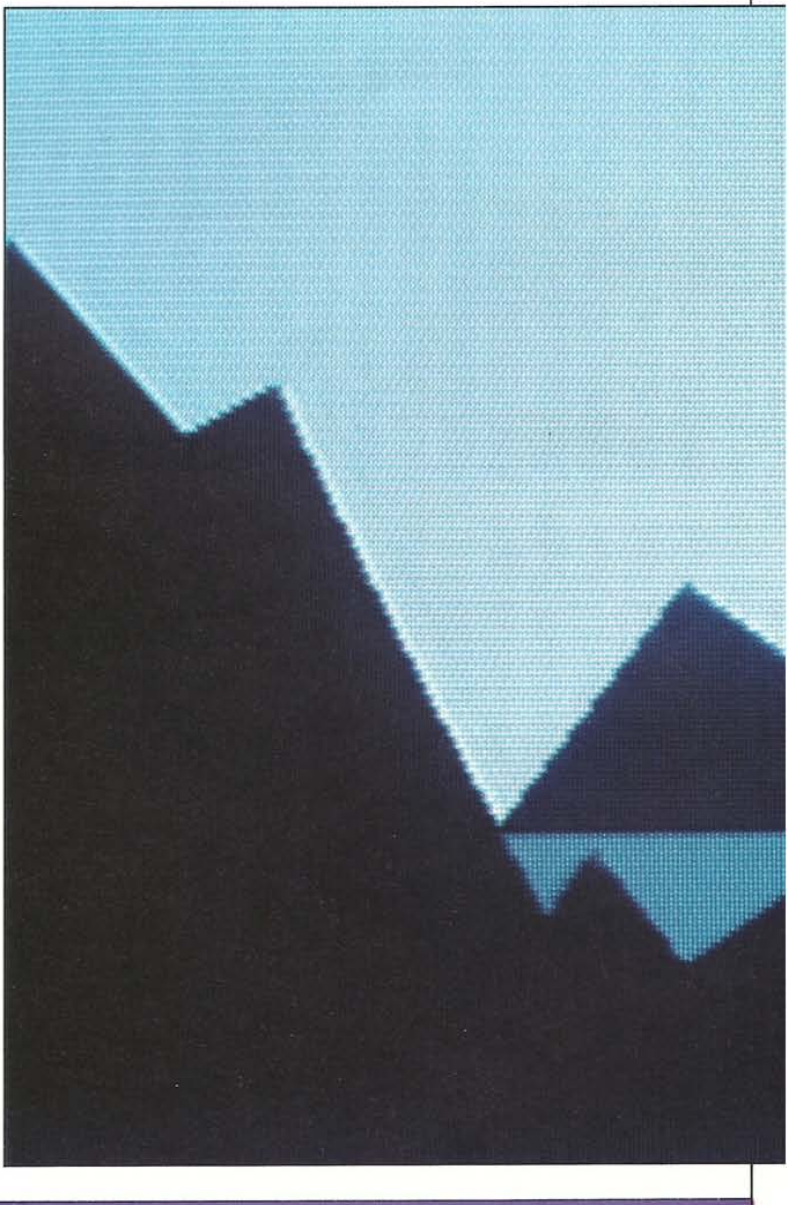
Filling in a seascape

The program on these two pages shows a quite complex display being filled and then colored. The block-color routine is used to set up separate areas each with a different color combination. Remember to make sure that the erase routine is not still switched on if you have

SEASCAPE PROGRAM

```
10000 SYS A1 : SYS B1,14 : SYS A3,10060
10010 SYS C1,0,56
10020 FOR N=1 TO 11
10030 READ X,Y
10040 SYS D1,X,Y
10050 NEXT N
10060 DATA 0,56,16,40,48,72,64,64,112,15
2,120,140,136,160
10070 DATA 160,144,208,168,248,144,319,1
76
10080 SYS C1,104,136
10090 FOR N=1 TO 6
10100 READ X,Y
10110 SYS D1,X,Y
10120 NEXT N
10130 DATA 136,96,160,112,168,108,200,12
8,216,120,232,136
10140 SYS C1,289,135
10150 SYS D1,312,96
10160 SYS D1,319,104
10170 SYS G1,48,160
10180 SYS C1,105,136

READY.
```

```
LIST 10190-

10190 SYS D1,319,136
10200 SYS G1,136,112
10210 SYS G1,312,112
10220 SYS E1,260,108,15
10230 SYS G1,260,108
10240 FOR X=104 TO 312 STEP 8
10250 FOR Y=32 TO 192 STEP 8
10260 SYS B2,X,Y,190
10270 NEXT Y : NEXT X
10280 FOR X=104 TO 312 STEP 8
10290 FOR Y=136 TO 196 STEP 8
10300 SYS B2,X,Y,6
10310 NEXT Y : NEXT X
10320 FOR X=232 TO 280 STEP 8
10330 FOR Y=80 TO 128
10340 SYS B2,X,Y,126
10350 NEXT Y : NEXT X
10360 GOTO 10360
READY.
```



just tried out the programs on pages 24-25. If it is, you will find that the flood-fill routine "escapes" and fills the whole screen.

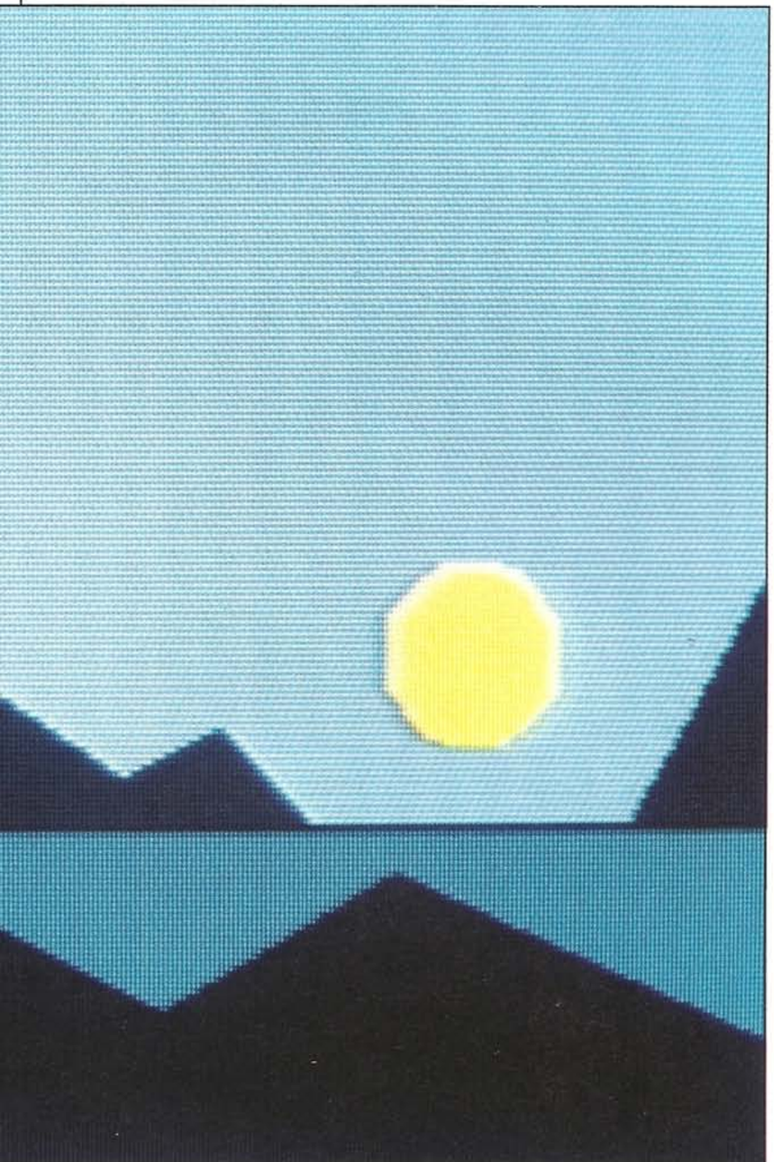
SEASCAPE PROGRAM

01:00

How the program works
First an outline is drawn in black over blue. This is then filled with solid black. Finally the block color routine is used to color the display selectively. In some places, the colors have to be changed more than once.
Line 10000 calls the high-resolution routine, sets up the colors and resets the DATA pointer to line 10060.
Lines 10010-10230
use the plot, draw and flood-fill routines to produce and fill the outline.
Lines 10240-10350 use the block-color routine to color the result.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution Restore	11
B Clear-and-color Block-color	13
C Plot	15
D Draw	17
E Circle	21
G Flood-fill	27



BLOCK G

FLOOD-FILL routine

What the routine does
Flood-fill fills a closed regular or irregular shape with the current foreground color, given a single starting point within the shape. The routine operates by rapidly drawing lines horizontally until a boundary is detected, and then by repeating the process above and below the original line until the shape is filled. Because the routine operates on the single-pixel level, a pixel missing in any boundary will eventually allow the routine to escape and start filling outside the shape. For this reason it will only fill a complete shape. Very complex shapes may generate an ILLEGAL QTY ERR report. This can be avoided by splitting an area up into smaller parts.

SYNTAX AND PARAMETERS

SYS G1,X,Y

X,Y	Horizontal and vertical coordinates of point where filling is to start (ranges 0-319 and 0-199).

ROUTINE LISTING

```
3500 IF PEEK(50694)=32 THEN 3530
3510 SYS A3,3540 : FOR C=50694 TO 51086
3520 READ B : POKE C,B : NEXT C
3530 G1=50694
3540 DATA 32,40,192,32,224,192,144
3550 DATA 5,162,14,76,55,164,142
3560 DATA 5,198,140,4,198,32,40
3570 DATA 192,32,236,192,176,237,140
3580 DATA 3,198,173,4,198,174,5
3590 DATA 198,32,118,199,240,1,96

3600 DATA 169,0,141,0,198,141,1
3610 DATA 198,141,2,198,56,173,4
3620 DATA 198,233,1,141,4,198,168
3630 DATA 173,5,198,233,0,141,5
3640 DATA 198,170,32,224,192,176,9
3650 DATA 152,172,3,198,32,118,199
3660 DATA 240,223,24,173,4,198,105
3670 DATA 1,141,4,198,141,8,192
3680 DATA 169,0,141,13,192,109,5
3690 DATA 198,141,5,198,141,9,192

3700 DATA 173,3,198,141,12,192,32
3710 DATA 0,193,160,0,177,253,13
3720 DATA 0,192,145,253,172,3,198
3730 DATA 136,174,5,198,173,4,198
3740 DATA 32,118,199,208,18,173,1
3750 DATA 198,208,13,238,1,198,174
3760 DATA 3,198,202,142,4,192,32
3770 DATA 70,199,172,3,198,136,174
3780 DATA 5,198,173,4,198,32,118
3790 DATA 199,240,10,173,1,198,240

3800 DATA 5,169,0,141,1,198,172
3810 DATA 3,198,200,174,5,198,173
3820 DATA 4,198,32,118,199,208,18
3830 DATA 173,2,198,208,13,238,2
3840 DATA 198,174,3,198,232,142,4
3850 DATA 192,32,70,199,172,3,198
3860 DATA 200,174,5,198,173,4,198
3870 DATA 32,118,199,240,10,173,2
3880 DATA 198,240,5,169,0,141,2
3890 DATA 198,24,173,4,198,105,1

3900 DATA 168,173,5,198,105,0,170
3910 DATA 32,224,192,176,12,152,172
3920 DATA 3,198,32,118,199,208,3
3930 DATA 76,92,198,173,0,198,208
3940 DATA 1,96,169,0,141,1,198
3950 DATA 141,2,198,172,0,198,136
3960 DATA 185,160,197,141,4,198,136
3970 DATA 185,160,197,141,5,198,136
3980 DATA 185,160,197,141,3,198,140
3990 DATA 0,198,76,59,198,172,4

4000 DATA 192,162,0,32,236,192,176
4010 DATA 37,173,0,198,201,96,144
4020 DATA 3,76,14,198,172,0,198
4030 DATA 173,4,192,153,160,197,200
4040 DATA 173,5,198,153,160,197,200
4050 DATA 173,4,198,153,160,197,200
4060 DATA 140,0,198,96,140,12,192
4070 DATA 160,0,140,13,192,141,8
4080 DATA 192,142,9,192,32,0,193
4090 DATA 173,0,192,160,0,49,253
4100 DATA 96
```

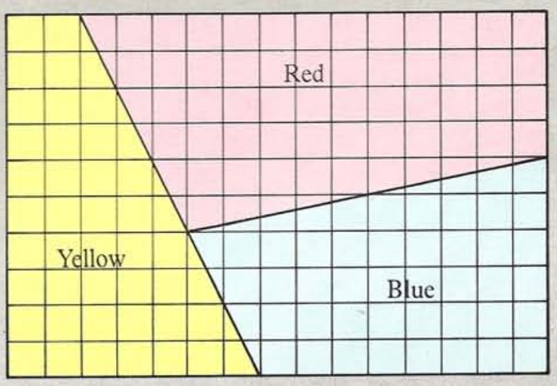
FILLING SHAPES 2

If you want to produce a colored and filled picture, it's important to bear in mind the Commodore's color restrictions when you are designing the display. You can use all of the Commodore's 16 colors on the screen at once, but if you are using the bit-map mode (as all the programs in this book do) you cannot get more than two colors into a single 8x8 pixel block.

Consider the following problem — suppose that the first of the following diagrams is part of a picture. How can you color the three areas shown if you can use only one foreground color and one background color in each 8x8 pixel block?

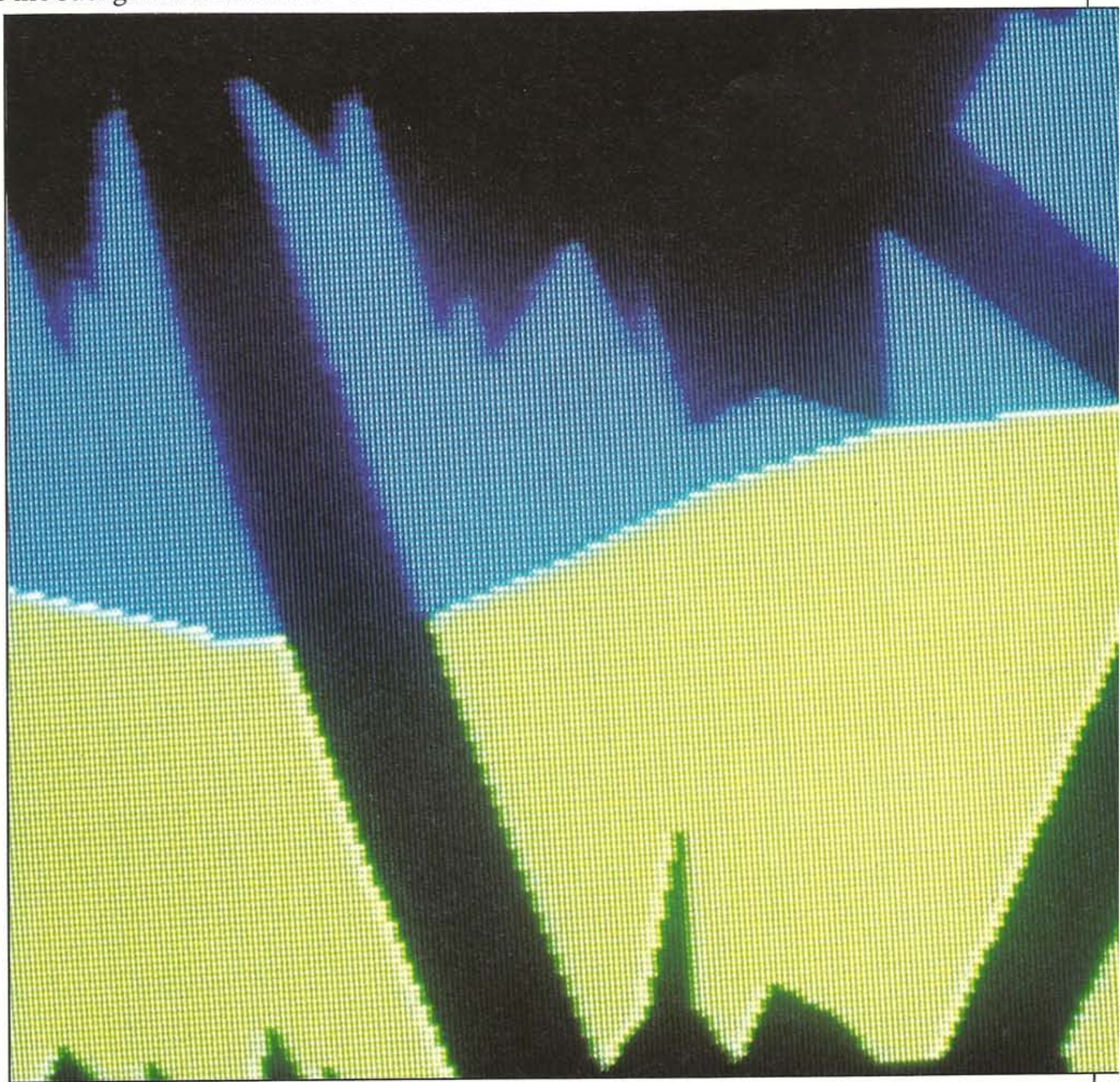
It sounds easy but imagine what would be involved. Suppose you decide to fill just the area colored yellow. Yellow becomes the foreground color, so, because it is adjacent to the blue area at the bottom of the design, blue needs to be the background color. Now look at the red area at the top. On the boundary with the yellow foreground area, red must be the background color. But on the boundary with the blue background area, red must be the foreground color. How can red be foreground and background at once? It sounds impossible, but there is a way of producing this result. It depends how you divide up the screen.

A COLORING PROBLEM



JUNGLE PROGRAM

01:15



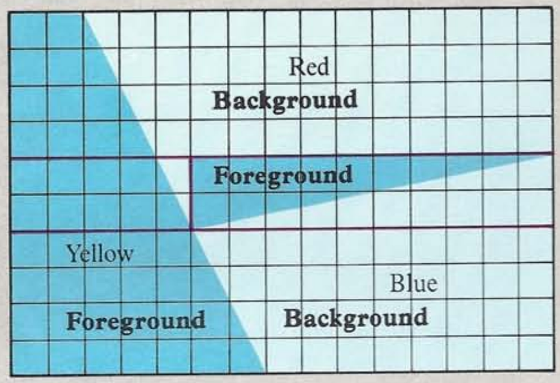
How the program works
The program first draws and fills the forest in black and white. The block-color routine is then used to set up the blue and green colors. Any one color may be either foreground or background in different parts of the display.
Line 10010 resets the DATA pointer to ensure that reading starts at the right point (line 15000).
Lines 10030-10080 form a loop which interprets the DATA as instructions to plot and draw.
Lines 10090-10151 fill and color the result.
Lines 20000-20030 form the block-coloring subroutine.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution Restore	11
B Clear-and-color Block-color	13
C Plot	15
D Draw	17
G Flood-fill	27

To solve this problem, you need to think of the screen as being divided up into rectangles with different color combinations set up by the block-color routine. Each rectangle contains two colors. One color may be background in one rectangle and foreground in another. In this way you can use block-coloring to produce the kind of coloring needed. The diagram below shows how it could be done.

THE COLORING PROBLEM SOLUTION



Filling and coloring a complex picture

The program below produces a complex filled and colored display. It's a view from a jungle clearing, looking out through the trees to a distant hilltop.

JUNGLE PROGRAM

```
10000 SYS A1 : SYS B1,16 : POKE 53280,0
10010 SYS A3,15000
10020 F=0
10030 READ X,Y
10040 IF X=-1 THEN F=Y : GOTO 10030
10050 IF X<0 THEN 10090
10060 IF F=1 THEN SYS D1,X,Y
10070 IF F=0 THEN SYS C1,X,Y
10080 GOTO 10030
10090 SYS G1,160,80
10100 SYS G1,160,180
10110 POKE 53280,0
10120 FOR K=1 TO 10
10130 READ LX,UX,LY,UY,C
10140 GOSUB 20000 : NEXT K
10150 GOTO 10150
15000 DATA 12,168,-1,1,14,95
15010 DATA 47,95,15,87,15,24
15020 DATA 28,56,28,38,32,48
15030 DATA 36,16,48,63,84,168
15040 DATA 80,172,68,164,64,168
15050 DATA 61,156,58,160,53,150

READY.
```

```
15060 DATA 48,168,35,154,32,162
15070 DATA 22,151,12,168,-1,0
15080 DATA 100,160,-1,1,79,96
15090 DATA 53,20,56,20,69,33
15100 DATA 72,20,84,56,88,48
15110 DATA 91,60,104,43,112,60
15120 DATA 112,44,120,71,132,66
15130 DATA 148,71,148,44,184,71
15140 DATA 192,80,184,96,150,160
15150 DATA 140,160,136,154,126,148
15160 DATA 120,160,114,150,114,126
15170 DATA 108,150,100,160,-1,0
15180 DATA 168,160,-1,1,204,96
15190 DATA 227,128,244,166,228,172
15200 DATA 216,160,208,164,196,148
15210 DATA 190,160,170,166,168,160
15220 DATA -1,0,204,71,-1,1,160,34
15230 DATA 172,17,178,20,196,11
15240 DATA 204,24,212,24,214,30
15250 DATA 224,24,228,30,244,14
15260 DATA 250,22,204,71,-1,0
15270 DATA 216,84,-1,1,231,63,266,28

READY.
```

```
15280 DATA 268,44,278,32,281,56
15290 DATA 288,48,292,56,298,40
15300 DATA 308,34,305,64,272,64
15310 DATA 232,71,303,71,286,168
15320 DATA 280,168,272,160,264,168
15330 DATA 216,84,-1,0,183,95,-1,1
15340 DATA 80,95,100,88,120,80
15350 DATA 144,74,183,72,183,95,-2,0
15360 DATA 0,312,0,56,6
15370 DATA 0,312,96,192,13
15380 DATA 0,8,64,88,0
15390 DATA 16,40,64,88,214
15400 DATA 48,72,64,88,6
15410 DATA 80,176,72,88,214
15420 DATA 80,224,64,64,6
15430 DATA 184,296,72,88,13
15440 DATA 232,296,64,64,214
15450 DATA 304,312,64,88,0
20000 FOR X=LX TO UX STEP 8
20010 FOR Y=LY TO UY STEP 8
20020 SYS B2,X,Y,C
20030 NEXT Y : NEXT X : RETURN
READY.
```

HIGH-RESOLUTION TEXT

One problem with using the Commodore 64 in high resolution is that there are no facilities for printing text. Obviously, if you have large volumes of text to display on the screen, then it is easier to do this in low resolution (text mode). Sometimes, however, there is a need to put a few characters on a display. The listing in block H on the opposite page contains two new routines which enable you to do this. They are the ROM-copy routine and the text routine. If you want to put text on the screen, you need to use both of these routines.

Copying the Commodore's character set

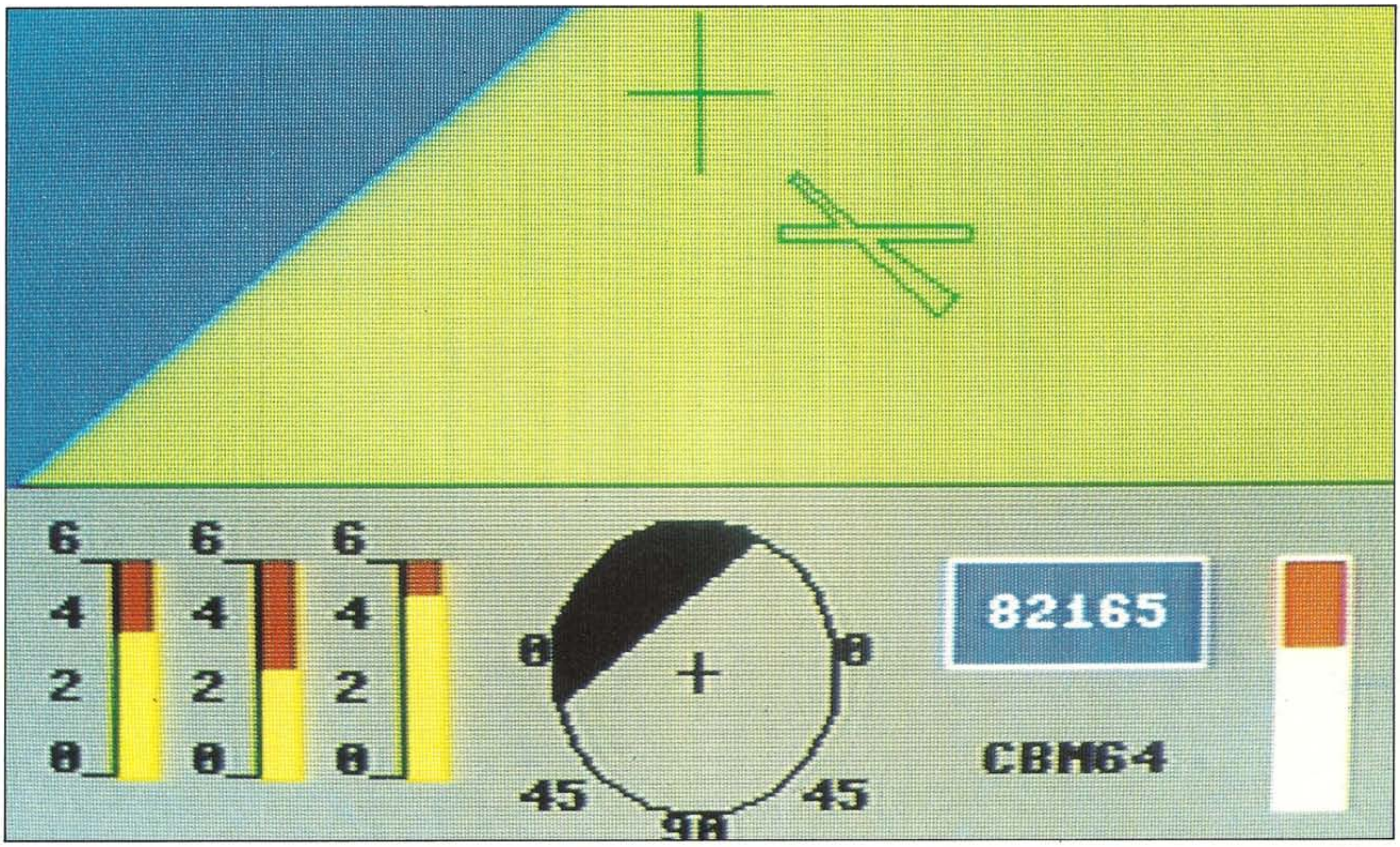
If you have read Book Two in this series, you will know that it is possible to make a copy of the Commodore's standard character set, which is held in ROM, and put this copy into RAM, where it can be modified. This copying is performed by the ROM-copy routine. When the block H machine code is in memory, you can use SYS H1 to copy the ROM character set. You won't see anything after activating the ROM-copy routine. However, afterwards you can use the text routine to take any of the copied characters and print them on the screen. To display text, all you have to do is use SYS H2, followed by the text. The routine prevents the characters appearing as colored blocks.

Using high-resolution text in games

There are many times during a game where text on the high-resolution screen can be used. The following program produces a favorite — a flight simulator. Although the display is only static, it is very detailed. To run it, load or key in the routines listed next to the display, and then add the program. The routines and coordinates are all keyed in as DATA.

FLIGHT SIMULATOR PROGRAM

```
10000 SYS A1 : SYS B1,16 : SYS A3,15000
10010 SYS E1,160,160,32
10020 F=0 : POKE 53280,0 : SYS H1
10030 READ X,Y
10040 IF X=-1 THEN F=Y : GOTO 10030
10050 IF X<0 THEN 10090
10060 IF F=1 THEN SYS D1,X,Y
10070 IF F=0 THEN SYS C1,X,Y
10080 GOTO 10030
10090 FOR C=1 TO 19 : READ X,Y,T$
10100 SYS H2,X,Y,T$ : NEXT C
10110 SYS G1,152,146
10120 SYS G1,40,40 : SYS G1,296,160
10130 FOR K=1 TO 10
10140 READ LX,LY,UX,UY,C
10150 GOSUB 20000 : NEXT K
10160 SYS B2,96,136,153
10170 GOTO 10170
15000 DATA 0,120,-1,1,319,120,-1,0
15010 DATA 8,120,-1,1,160,0,-1,0
15020 DATA 160,24,-1,1,160,56,-1,0
15030 DATA 144,40,-1,1,176,40,-1,0

READY.
```



FLIGHT SIMULATOR PROGRAM (CONTD.)

```
15040 DATA 182,56,-1,1,196,67,220,67
15050 DATA 220,70,200,70,217,81,212,85
15060 DATA 194,70,178,70,178,67,191,67
15070 DATA 180,58,182,56,-1,0
15080 DATA 24,136,-1,1,31,136,31,183
15090 DATA 24,183,-1,0,56,136,-1,1
15100 DATA 63,136,63,183,56,183,-1,0
15110 DATA 88,136,-1,1,95,136,95,183
15120 DATA 88,183,-1,0,216,136,-1,1
15130 DATA 271,136,271,159,216,159
15140 DATA 216,136,-1,0,288,136,-1,1
15150 DATA 303,136,303,191,288,191
15160 DATA 288,136,-1,0,288,156,-1,1
15170 DATA 303,156,-1,0,129,167,-1,1
15180 DATA 173,132,-1,0,160,156,-1,1
15190 DATA 160,164,160,160,156,160
15200 DATA 164,160,-1,0,-2,0
16000 DATA 16,176,"0",16,160,"2"
16010 DATA 16,144,"4",16,128,"6"
16020 DATA 48,176,"0",48,160,"2"
16030 DATA 48,144,"4",48,128,"6"
16040 DATA 80,176,"0",80,160,"2"

READY.
```

```
LIST 16050-

16050 DATA 80,144,"4",80,128,"6"
16060 DATA 120,152,"0",120,184,"45"
16070 DATA 152,192,"90",184,184,"45"
16080 DATA 192,152,"0",224,144,"82165"
16090 DATA 224,176,"CBM64"
17000 DATA 0,0,312,112,101,144,24,216
17010 DATA 80,5,0,120,312,192,12
17020 DATA 216,136,264,152,22,288,136
17030 DATA 296,184,18,32,136,32,144,153
17040 DATA 32,152,32,176,119,64,136,64
17050 DATA 152,153,64,160,64,176,119
17060 DATA 96,144,96,176,119
20000 FOR X=LX TO UX STEP 8
20010 FOR Y=LY TO UY STEP 8
20020 SYS B2,X,Y,C
20030 NEXT Y : NEXT X : RETURN
READY.
```

FLIGHT SIMULATOR PROGRAM

00:35

How the program works
The display is first drawn, and then the program uses the text routine to add numbers and labels to parts of the display. It is finally flood-filled and then colored.
Line 10000 sets up the high-resolution screen and the overall colors.
Line 10020 copies the ROM character set so it can be used by the text routine.
Lines 10030-10120 READ the DATA block, either producing points or lines or (line 10100) putting text from the DATA lines onto the screen.
Lines 10130-10150 call the block-coloring subroutine.
Lines 20000-20030 form the subroutine which contains the block-color routine.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution Restore	11
B Clear-and-color Block-color	13
C Plot	15
D Draw	17
E Circle	21
G Flood-fill	27
H ROM-copy Text	31

BLOCK H

ROM-COPY and TEXT routines

What the routines do
ROM-copy copies the standard character set from ROM into RAM so that the characters can be used in high-resolution displays with the text routine.
Text displays any text on the high-resolution screen. The text is displayed starting at an 8x8 pixel block which can be fixed by specifying any point which lies within it.

SYNTAX AND PARAMETERS

ROM-copy: SYS H1   Text: SYS H2,X,Y,A$

X,Y	(**Text** only) Horizontal and vertical coordinates of any point within the 8x8 pixel block where the first character is to appear.
A$	(**Text** only) Any text.

ROUTINE LISTING

```
4200 IF PEEK(51104)=173 THEN 4230
4210 SYS A3,4240 : FOR C=51104 TO 51306
4220 READ B : POKE C,B : NEXT C
4230 H1=51104 : H2=51167
4240 DATA 173,14,220,41,254,141,14
4250 DATA 220,165,1,41,251,133,1
4260 DATA 160,0,132,253,132,251,169
4270 DATA 17,133,254,169,209,133,252
4280 DATA 32,215,199,230,254,169,208
4290 DATA 133,252,32,215,199,165,1

4300 DATA 9,4,133,1,173,14,220
4310 DATA 9,1,141,14,220,96,177
4320 DATA 251,145,253,200,208,249,96
4330 DATA 32,40,192,32,224,192,144
4340 DATA 3,76,234,193,152,41,248
4350 DATA 141,8,192,142,9,192,32
4360 DATA 40,192,32,236,192,176,236
4370 DATA 152,41,248,141,12,192,142
4380 DATA 13,192,32,121,0,32,253
4390 DATA 174,32,158,173,32,163,182

4400 DATA 141,2,198,169,0,141,1
4410 DATA 198,133,252,32,0,193,165
4420 DATA 254,201,64,144,7,165,253
4430 DATA 201,64,144,1,96,172,2
4440 DATA 198,208,1,96,136,140,2
4450 DATA 198,172,1,198,177,34,200
4460 DATA 140,1,198,10,38,252,10
4470 DATA 38,252,10,38,252,133,251
4480 DATA 165,252,105,16,133,252,160
4490 DATA 7,177,251,145,253,136,16
4500 DATA 249,24,165,253,105,8,133
4510 DATA 253,165,254,105,0,133,254
4520 DATA 169,0,133,252,76,29,200
```

Using the text routine

The text routine will put any text you like at the point with coordinates X,Y on the high-resolution screen. It rounds down the coordinates you specify to the nearest 8 so that they bring the character onto a low-resolution boundary. This is done to make changing character foreground and background colors more easy.

With this routine, the text doesn't have to be a letter or word — it can be a string expression that is evaluated into text. So you could use something as simple as "FRED", or you could use something as complex as CHR$(27)+"="+A$. However, the text routine cannot deal with numeric expressions. These must be converted to string expressions first.

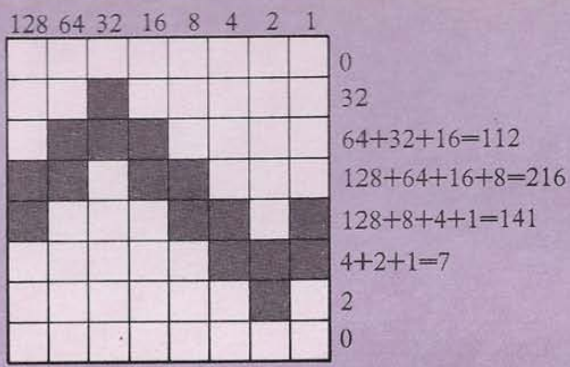
DESIGNING CHARACTERS

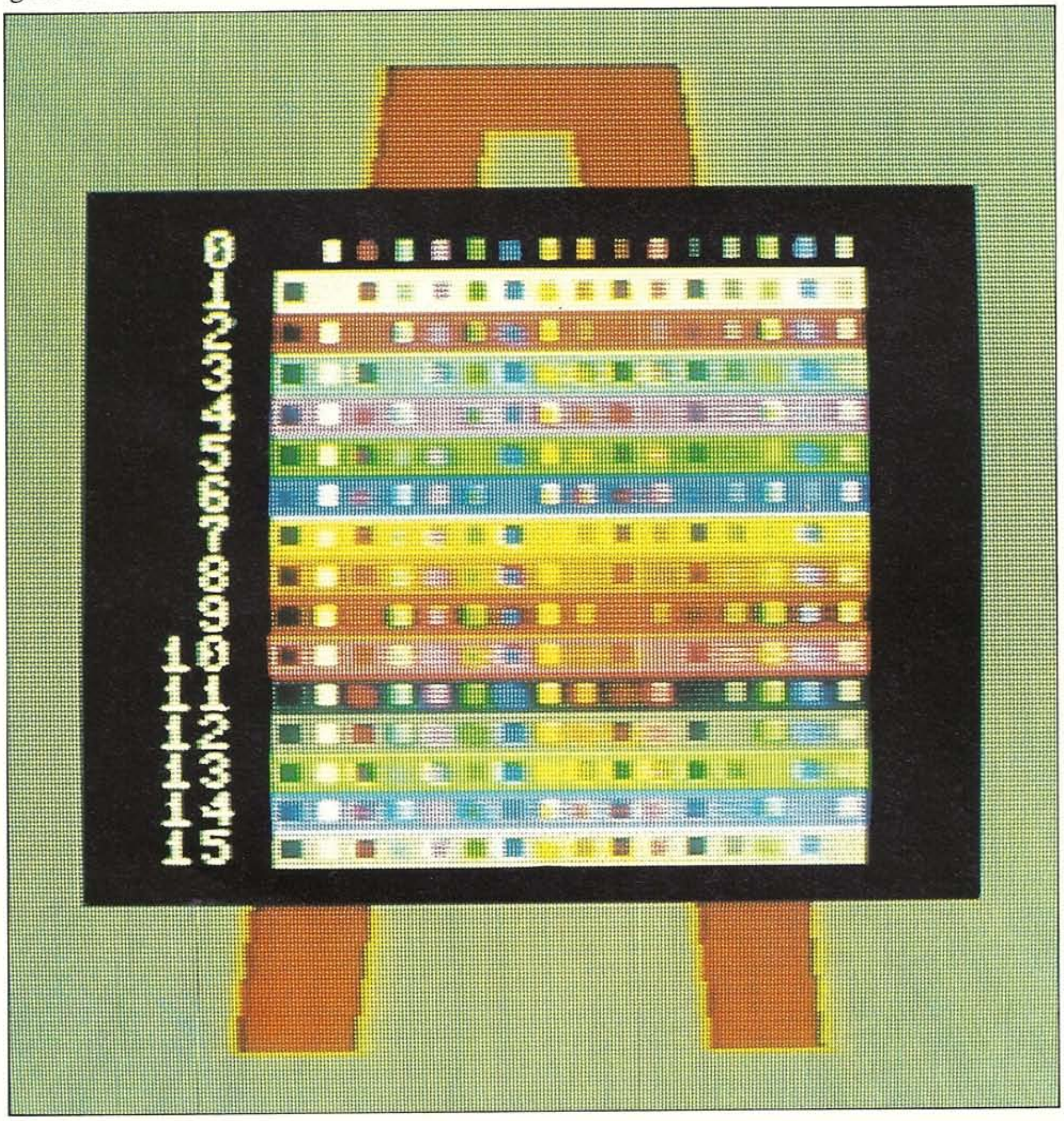
On the previous two pages you saw how the ROM-copy routine can be used to copy characters from ROM into RAM. One advantage of doing this is that it is a simple matter to go on from there to create your own characters. Once you have done that, you can use these shapes to fill up spaces, giving you the facility to "pattern-fill". The listing in block I on the opposite page contains a single routine. This is the define-character routine which enables you to define any of the 8x8 characters stored in RAM.

How to make a high-resolution character

If you already know how an 8x8 character is coded, you will find this routine quite simple to use as it just accepts 8 bytes, one for each row of the character. If you haven't already coded a character on the Commodore, all you need to know is that the 8x8 pixel grid can be broken down into 8 rows. Each of these is coded by a row total. The row total is arrived at by adding together the "bit values" of all the filled-in pixels in the row. Once you have 8 row totals, these can be used with the define-character routine.

CODING A CHARACTER





COLOR CHART PROGRAM

00:20

How the program works
The program first produces a blackboard and an easel by drawing, flood-filling and block-coloring. After the color numbers have been printed by the text routine, the same routine is used to print a specially-defined character. The block-color routine makes the character appear in all the Commodore's colors.
Line 10200 defines the square color-test character.
Lines 10140-10250 print the numbers using the STR$ command, and then print the color-test characters.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
Restore	
B Clear-and-color	13
Block-color	
C Plot	15
D Draw	17
G Flood-fill	27
H ROM-copy	31
Text	
I Define-character	33

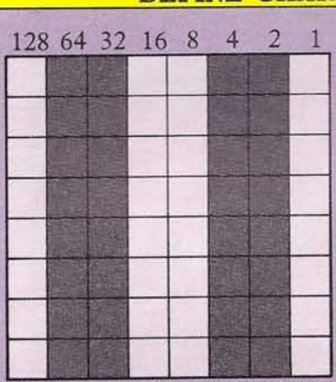
If you give your design character code C, and the row totals are X1-X8, then you can put it into the computer's memory with

SYS I1,C,X1,X2,X3,X4,X5,X6,X7,X8

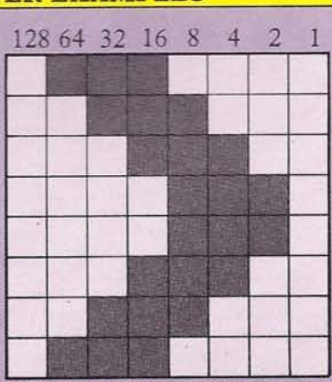
Once the new character is in RAM, you can display it on the high-resolution screen with the text routine. Furthermore, as you will see on the next four pages, you can use the character with the pattern-fill routine to fill designs.

Below are two examples of user-defined characters, together with the complete define-character routines that produce them.

DEFINE-CHARACTER EXAMPLES



SYS I1,60,102,102,102,102,102,102,102,102



SYS I1,61,112,56,28,14,14,28,56,112

Color chart program

Now that you have seen how to create your own high-resolution characters, you should have no trouble understanding how the next program works. This listing creates a character and then displays it in each of the 256 color combinations that the Commodore is capable of producing. The numbers down the left-hand side of the chart set up the background colors. The small square character then appears in all of the foreground colors.

COLOR CHART PROGRAM

```
LIST 10000-10190

10000 SYS A1 : SYS B1,156
10010 SYS H1 : POKE 53280,12
10020 SYS A3,15000 : F=0
10030 READ X,Y
10040 IF X=-1 THEN F=Y : GOTO 10030
10050 IF X<0 THEN 10090
10060 IF F=1 THEN SYS D1,X,Y
10070 IF F=0 THEN SYS C1,X,Y
10080 GOTO 10030
10090 SYS G1,120,180
10100 SYS G1,208,180
10110 SYS G1,160,1
10120 READ LX,LY,UX,UY,C
10130 GOSUB 20000
10140 FOR C=0 TO 9
10150 SYS H2,80,32+8*C,STR$(C)
10160 NEXT C
10170 FOR C=10 TO 15
10180 SYS H2,72,32+8*C,STR$(C)
10190 NEXT C

READY.
```

BLOCK I

DEFINE-CHARACTER routine

What the routine does

The routine defines a character which is then held in RAM. This user-defined character can then be called by its CHR$ number. The routine accepts nine parameters — a character code, and eight row totals which specify the character. Each row total consists of the sum of the bit values for all the pixels that are to be lit on that row of the character. A row that has no pixels lit has a row total of 0, whereas one that has all its pixels lit has a row total of 255. The eight rows of a character give eight row totals.

SYNTAX AND PARAMETERS	
SYS I1,C,X1,X2,X3,X4,X5,X6,X7,X8	
C	Character code (range 0-255).
X1-X8	Row bit totals (range 0-255 each).

ROUTINE LISTING

```
4600 IF PEEK(51328)=169 THEN 4630
4610 SYS A3,4640 : FOR C=51328 TO 51377
4620 READ B : POKE C,B : NEXT C
4630 I1=51328
4640 DATA 169,0,133,252,32,40,192
4650 DATA 152,10,38,252,10,38,252
4660 DATA 10,38,252,133,251,24,165
4670 DATA 252,105,16,133,252,169,0
4680 DATA 141,4,192,32,40,192,152
4690 DATA 172,4,192,145,251,200,140
4700 DATA 4,192,152,201,8,208,238
4710 DATA 96
```

COLOR CHART PROGRAM (CONTD.)

```
LIST 10200-

10200 SYS I1,0,0,0,60,60,60,60,0,0
10210 FOR X=0 TO 15 : FOR Y=0 TO 15
10220 X1=104+X*8 : Y1=32+Y*8
10230 SYS B2,X1,Y1,X*16+Y
10240 SYS H2,X1,Y1,CHR$(0)
10250 NEXT Y : NEXT X
10260 GOTO 10260
15000 DATA 128,0,-1,1,124,23,148,23
15010 DATA 150,12,170,12,172,23,196,23
15020 DATA 192,0,-1,0,96,199,-1,1
15030 DATA 100,168,124,168,120,200
15040 DATA 200,200,196,168,220,168
15050 DATA 224,199,-1,0,-2,0
16000 DATA 64,24,248,160,16
20000 FOR X=LX TO UX STEP 8
20010 FOR Y=LY TO UY STEP 8
20020 SYS B2,X,Y,C
20030 NEXT Y : NEXT X : RETURN
READY.
```

Mixing text and defined characters

You can use defined characters just like ordinary text, so that if you want to, you can mix them with text. But remember that to do this, you must first use the ROM-copy routine so that the text characters are in RAM, where you can use the define-character routine to change them.

PATTERN-FILLING 1

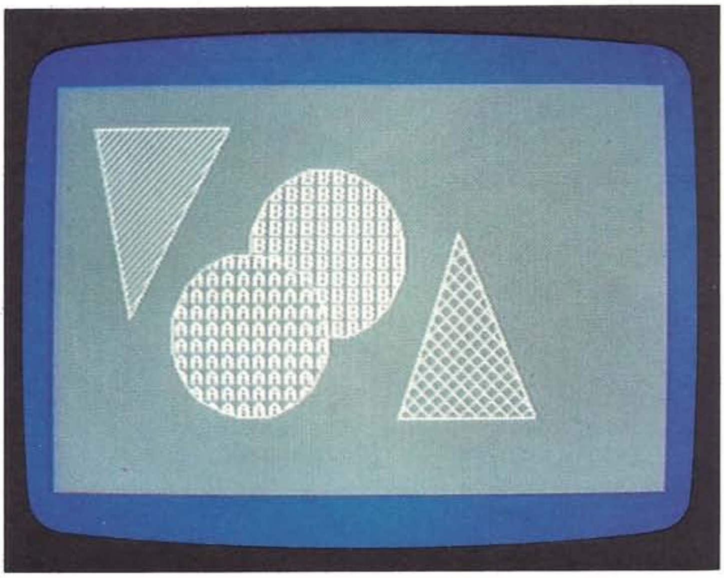
Over the last few pages you have seen how you can use the graphics routines to put text up on the screen and how to define your own characters. Now you have got this far, you can add another facility whose usefulness is out of all proportion to its simplicity — a routine which fills irregular shapes with a pattern which you can specify.

The listing in block J opposite contains the pattern-fill routine. To use the routine you need to specify the coordinates of a point where filling is to begin, as with the flood-fill routine. However, after the coordinates, you then need to specify a character number. The computer will use this character to fill the shape. If you first define a character with the define-character routine, you can then use it to pattern-fill.

To see the pattern-fill routine in action, load or key in routine blocks A-E and H, add block J and then the program below.

PATTERN-FILL PROGRAM

```
LIST

10000 SYS A1 : SYS B1,28
10010 SYS H1 : POKE 53280,6
10020 SYS E1,103,120,40
10030 SYS E2,143,80,40,180,90
10040 SYS C1,20,20 : SYS D1,90,20
10050 SYS D1,40,110 : SYS D1,20,20
10060 SYS C1,180,160 : SYS D1,250,160
10070 SYS D1,210,70 : SYS D1,180,160
10080 SYS I1,0,68,136,17,34,68,136,17,34
 : SYS I1,1,28,34,65,128,193,34,20,8
10090 SYS J1,103,81,65
10100 SYS J1,143,41,66
10110 SYS J1,40,21,0
10120 SYS J1,210,73,1
10130 GOTO 10130
READY.
```



Pattern-filling complex shapes

Unlike the flood-fill routine, pattern-fill will only work with relatively simple shapes. It starts at the point you specify and then works vertically downward until it reaches a boundary or the bottom of the screen. In a complex shape, you may have to use it more than once, as the next program shows. It draws a map which is both flood-filled and pattern-filled.

PATTERN-FILLED MAP PROGRAM

```
10000 SYS A1 : SYS B1,118
10010 SYS A3,10060
10020 SYS C1,0,0 : SYS D1,319,0 : SYS D1
 319,199:SYS D1,0,199:SYS D1,0,0
10030 SYS C1,48,0
10040 FOR N=1 TO 12 : READ X,Y
10050 SYS D1,X,Y : NEXT N
10060 DATA 54,27,64,56,59,90,64,112,80,1
28,99,164,108,172,110,170,92,136,128,179

10070 DATA 128,186,163,199
10080 SYS C1,64,56 : SYS D1,228,56
10090 SYS C1,80,128 : SYS D1,149,150
10100 SYS C1,252,0
10110 FOR N=1 TO 22 : READ X,Y
10120 SYS D1,X,Y : NEXT N
10130 DATA 232,20,240,36,248,24,256,28,2
44,50,238,40,225,61,224,96,214,118
10140 DATA 224,140,220,144,208,128,172,1
36,149,150,160,176,174,180,188,184
10150 DATA 192,170,204,168,200,192,224,1
88,224,199

READY.
```

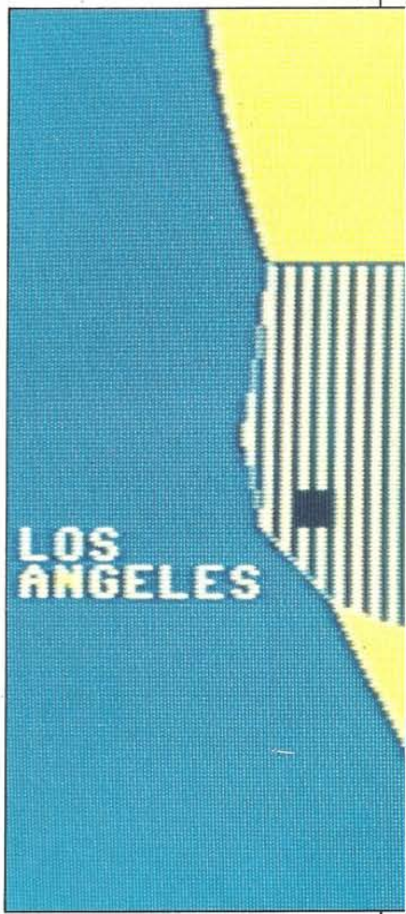
PATTERN-FILLED MAP PROGRAM

00:25

How the program works
The map is filled using both types of fill routine.
Line 10220 produces the flood-filling.
Line 10242 calls the pattern-fill routine.

ROUTINES USED BY THIS PROGRAM

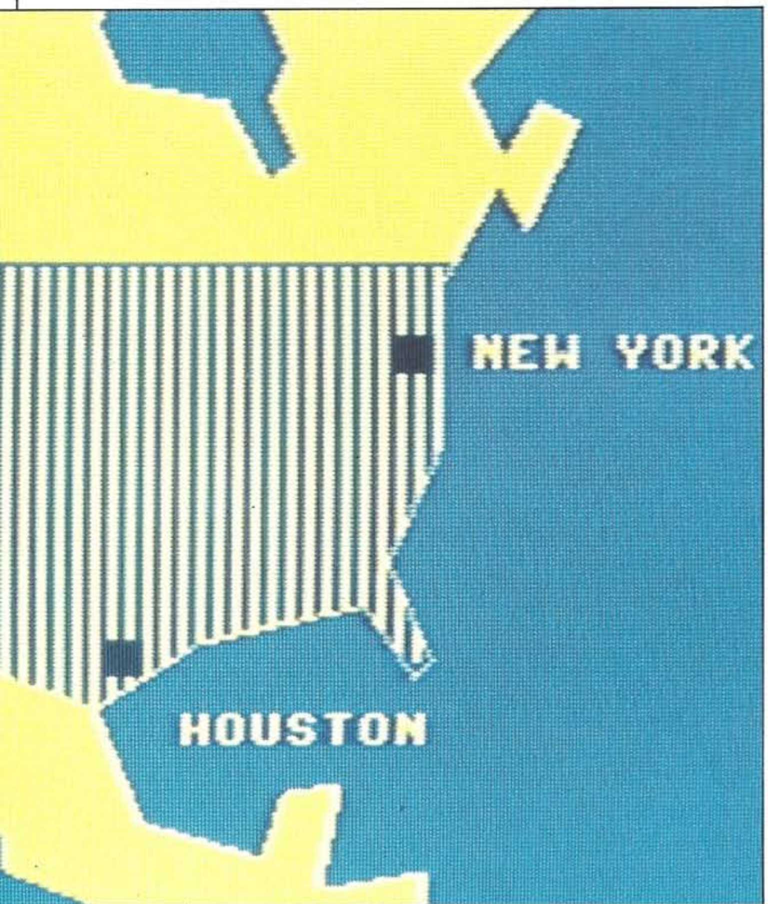
Block Routine(s)	Page
A High-resolution Restore	11
B Clear-and-color Block-color	13
C Plot	15
D Draw	17
G Flood-fill	27
H ROM-copy Text	31
I Define-character	33
J Pattern-fill	35



PATTERN-FILLED MAP PROGRAM (CONTD.)

```
10160 SYS C1,150,0
10170 FOR N=1 TO 8 : READ X,Y
10180 SYS D1,X,Y : NEXT N
10190 DATA 148,8,154,20,178,24,188,40,19
4,36,188,24,192,16,184,0
10200 SYS H1
10210 SYS I1,50,204,204,204,204,204,204,
204,204
10220 SYS G1,100,1 : SYS G1,140,180
10230 SYS J1,66,58,50 : SYS J1,150,114,5
0
10240 SYS J1,214,128,50 : SYS J1,218,134
,50
10250 C=0
10260 SYS B2,72,108,C : SYS B2,216,72,C
: SYS B2,152,136,C
10270 SYS H1
10280 SYS H2,232,72,"NEW YORK"
10290 SYS H2,168,152,"HOUSTON"
10300 SYS H2,8,112,"LOS"
10310 SYS H2,8,120,"ANGELES"
10320 GOTO 10320
READY.
```

After each time that the pattern-fill routine moves down a line, it fills as far as it can to the left and to the right. However, unlike the flood-fill routine, it won't "remember" to go back to unfilled areas in complex shapes. Instead you need to start the routine again from the top of any unfilled areas. It's rather like painting a wall: just start from the top of each area you want to paint, and go back to any that are missed out. You will find that the routine is written so that the pattern will match exactly where it meets any part that was filled previously.



BLOCK J

PATTERN-FILL routine

What the routine does

The routine fills a closed regular or irregular shape with an 8x8 pixel RAM text character (this may be a standard Commodore character, or one previously created using the define-character routine). Although the routine fills with 8x8 characters, it will fill partial character spaces. The routine should be started at the top of any shape to be filled. Complex shapes may have to be pattern-filled by calling the routine a number of times, breaking the shapes up into simpler areas.

SYNTAX AND PARAMETERS	
SYS J1,X,Y,C	
X,Y	Horizontal and vertical coordinates of the point where pattern- filling is to begin (ranges 0-319 and 0-199).
C	Code number of character used for filling (0-255).

ROUTINE LISTING

```
4800 IF PEEK(51394)=32 THEN 4830
4810 SYS A3,4840 : FOR C=51392 TO 51608
4820 READ B : POKE C,B : NEXT C
4830 J1=51394
4840 DATA 160,0,32,40,192,32,224
4850 DATA 192,144,5,162,14,76,55
4860 DATA 164,142,5,198,140,4,198
4870 DATA 32,40,192,32,236,192,176
4880 DATA 237,140,3,198,32,40,192
4890 DATA 162,0,134,252,152,10,38

4900 DATA 252,10,38,252,10,38,252
4910 DATA 133,251,24,165,252,105,16
4920 DATA 133,252,172,3,198,173,4
4930 DATA 198,174,5,198,141,192,200
4940 DATA 142,193,200,32,128,201,240
4950 DATA 1,96,56,173,4,198,233
4960 DATA 1,141,4,198,168,173,5
4970 DATA 198,233,0,141,5,198,170
4980 DATA 32,224,192,176,26,152,172
4990 DATA 3,198,32,128,201,240,223

5000 DATA 76,65,201,172,2,192,177
5010 DATA 251,45,0,192,160,0,17
5020 DATA 253,145,253,24,173,4,198
5030 DATA 105,1,141,4,198,168,173
5040 DATA 5,198,105,0,141,5,198
5050 DATA 170,32,224,192,176,9,152
5060 DATA 172,3,198,32,128,201,240
5070 DATA 209,173,192,200,141,4,198
5080 DATA 173,193,200,141,5,198,172
5090 DATA 3,198,200,140,3,198,162

5100 DATA 0,32,236,192,176,3,76
5110 DATA 250,200,96,140,12,192,160
5120 DATA 0,140,13,192,141,8,192
5130 DATA 142,9,192,32,0,193,173
5140 DATA 0,192,160,0,49,253,96
```

Copying and defining a pattern

The pattern-fill routine will only fill with characters that already exist in RAM. To fill with a character other than one that can be copied from ROM into RAM by the ROM-copy routine, you must define it first. If you forget to use the ROM-copy or define-character routines, you will find that the pattern-fill routine takes whatever is in RAM at the particular character address that you have specified. If your pattern-filling just produces a pattern of random dots and lines repeated over the screen, you have probably forgotten to use the routines in block H.

PATTERN-FILLING 2

The Commodore has a character set with 256 elements, and you can pattern-fill with all the characters that can be printed on the screen. This means that a single program can have areas filled with different patterns. The programs on these two pages show what you can do with just three different kinds of pattern-filling.

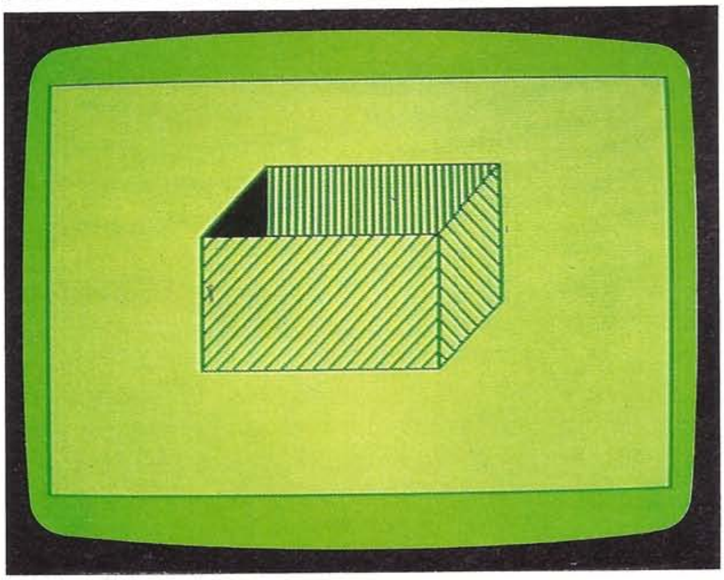
Cross-hatching a design

Many designs produce a 3-D effect by cross-hatching, that is, by having areas filled with parallel lines. You can use this technique to shade a diagram to give an impression of its three-dimensional shape.

The program below makes a 3-D box. Each type of cross-hatching is produced by a separate character used with the pattern-filling routine. The numbers that code the three characters are in lines 10120, 10150 and 10180. To try the program out, you will need routines A-D and G-J in memory.

PATTERN-FILLED BOX PROGRAM

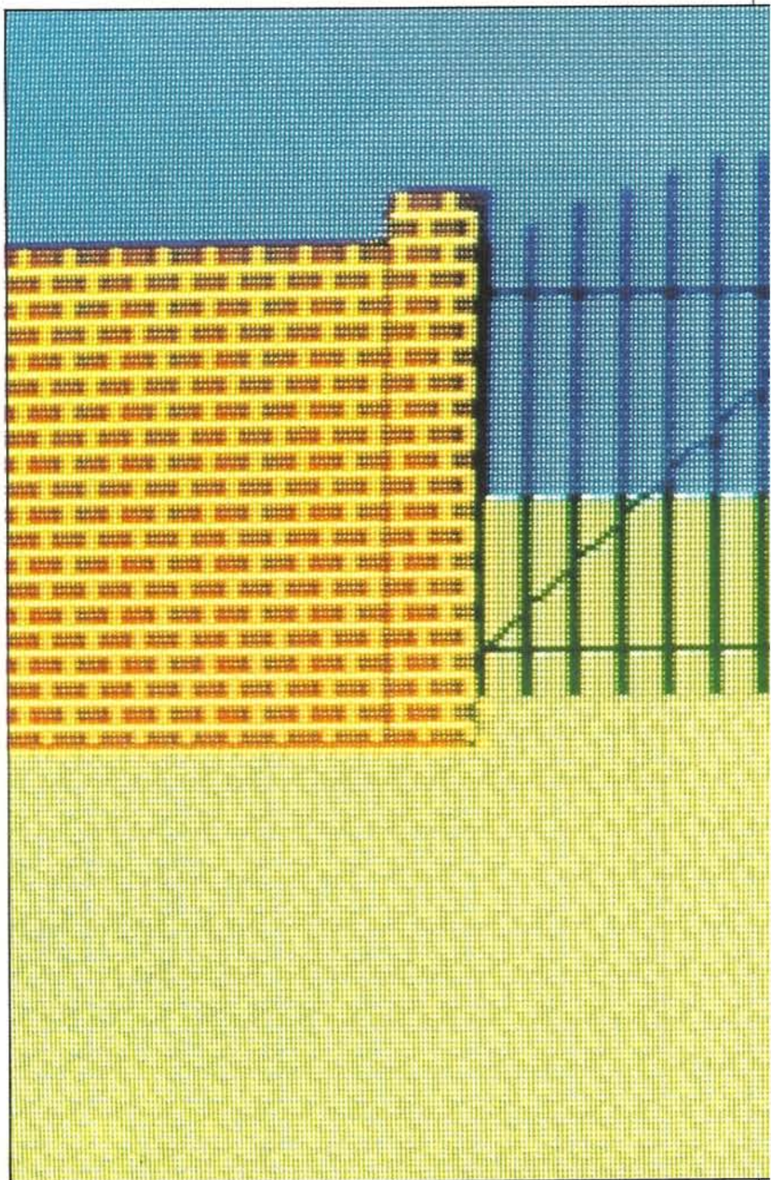
```
10000 SYS A1: SYS B1,13: POKE 53280,5
10010 SYS C1,0,0 : SYS D1,319,0
10020 SYS D1,319,199 : SYS D1,0,199
10030 SYS D1,0,0
10040 SYS C1,80,136 : SYS D1,80,72
10050 SYS D1,200,72 : SYS D1,200,136
10060 SYS D1,80,136 : SYS C1,80,72
10070 SYS D1,112,40 : SYS D1,232,40
10080 SYS D1,200,72 : SYS C1,232,40
10090 SYS D1,232,104 : SYS D1,200,136
10100 SYS C1,112,40 : SYS D1,112,72
10110 SYS H1
10120 SYS I1,10,1,2,4,8,16,32,64,128
10130 SYS J1,114,73,10
10140 SYS J1,114,73,10
10150 SYS I1,12,128,64,32,16,8,4,2,1
10160 SYS J1,231,43,12
10170 SYS J1,201,104,12
10180 SYS I1,13,17,17,17,17,17,17,17,17
10190 SYS  J1,113,41,13
10200 SYS G1,104,64
10210 GOTO 10210
READY.
```



Pattern-filling a wall and gate

The next program uses all the routines so far to draw and pattern-fill a wall and a gate. It uses three characters produced by the define-character routine, and then colors the result with the block-color routine.

When you run this program, you will see a new technique at work. Sometimes you may want to pattern-fill an area but not show the outline or boundary that surrounds the pattern. This program shows you one way of achieving this effect. It draws one particular boundary (the curved top of the gate), pattern-fills the shape that it surrounds, and then takes this boundary away by drawing it again with the erase routine turned on. The gate is pattern-filled without the top boundary appearing in the final display. It's much easier than trying to work out a programming routine to draw lines that together form an arc. You can use it with any shape that has a programmable outline.



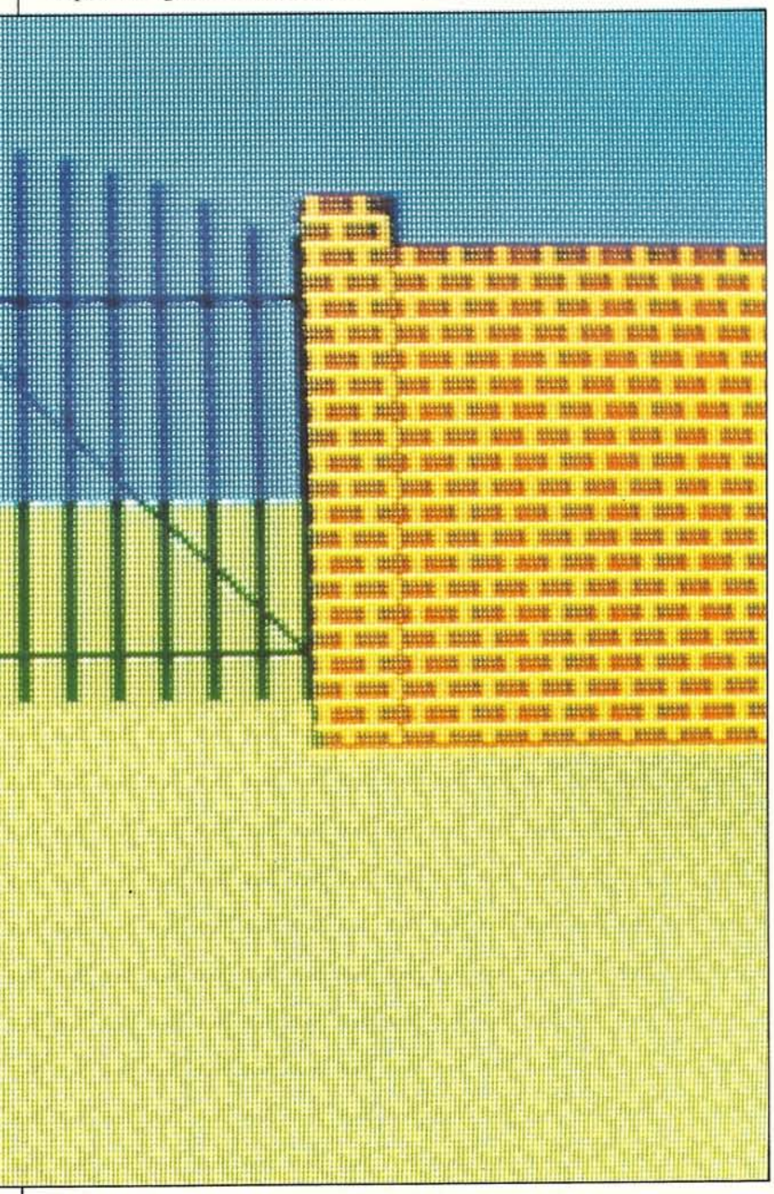
WALL AND GATE PROGRAM

00:40

How the program works
The program uses three specially defined characters to produce a pattern-filled display. The vertical bars of the gate are not drawn but pattern-filled within a boundary which is later removed.
Lines 10010-10040 define three characters (the bars of the gate, the bricks in the wall and the pattern on the path).
Line 10050 READs the DATA in lines 15000-16070. This controls all the drawing, filling and coloring.
Line 10120 turns on the erase routine so that the arc at the top of the gate is erased after the bars have been created by pattern-filling in line 10200. **Line 10230** turns the erase routine off again.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution Restore	11
B Clear-and-color Block-color	13
C Plot	15
D Draw	17
E Arc	21
F Erase	25
G Flood-fill	27
H ROM-copy	31
I Define-character	33
J Pattern-fill	35



The program uses two sets of DATA. To ensure that READing always begins from the correct point, the restore routine is used twice (lines 10050 and 10280) to reset the DATA pointer.

WALL AND GATE PROGRAM

```
10000 SYS A1 : SYS B1,16
10010 SYS H1 : POKE 53280,0
10020 SYS I1,0,129,129,129,129,129,129,1
29,129
10030 SYS I1,1,0,251,251,251,0,191,191,1
91
10040 SYS I1,2,37,37,164,160,136,8,41,33

10050 F=0 : SYS A3,15000
10060 READ X,Y
10070 IF X=-1 THEN F=Y : GOTO 10060
10080 IF X<0 THEN 10120
10090 IF F=1 THEN SYS D1,X,Y
10100 IF F=0 THEN SYS C1,X,Y
10110 GOTO 10060
10120 SYS F1,1
10130 SYS E2,160,160,136,242,298
10140 SYS C1,96,112 : SYS D1,223,112
10150 SYS J1,160,113,2
10160 SYS J1,40,41,1
10170 SYS J1,280,41,1
10180 SYS J1,88,33,1

READY.
```

```
LIST 10190-16000

10190 SYS J1,232,33,1
10200 SYS J1,155,25,0
10210 SYS E2,160,160,136,242,298
10220 SYS C1,96,112 : SYS D1,223,112
10230 SYS F1,0
10240 SYS C1,96,104 : SYS D1,223,104
10250 SYS C1,96,48 : SYS D1,223,48
10260 SYS C1,96,104 : SYS D1,160,48
10270 SYS D1,223,104
10280 SYS A3,16000 : FOR K=1 TO 8
10290 READ LX,UX,LY,UY,C
10300 GOSUB 20000 : NEXT K
10310 GOTO 10310
15000 DATA 0,119,-1,1,95,119,95,32
15010 DATA 80,32,80,119,-1,0,0,40
15020 DATA -1,1,80,40,-1,0,319,40
15030 DATA -1,1,239,40,-1,0,319,119
15040 DATA -1,1,224,119,224,32,239,32
15050 DATA 239,119,-1,0,-2,0
16000 DATA 0,312,0,32,102

READY.
```

```
LIST 16010-

16010 DATA 0,312,112,192,213
16020 DATA 0,72,40,112,151
16030 DATA 80,88,32,112,151
16040 DATA 224,232,32,112,151
16050 DATA 240,312,40,112,151
16060 DATA 96,216,0,72,6
16070 DATA 96,216,80,104,5
20000 FOR X=LX TO UX STEP 8
20010 FOR Y=LY TO UY STEP 8
20020 SYS B2,X,Y,C
20030 NEXT Y : NEXT X : RETURN
READY.
```

BLOCK-COPYING 1

There are many times when it is useful to be able to copy some object that you have drawn on the screen without having to repeat the code to program it. The listing in block K on the opposite page contains a block-copy routine which will copy one 8x8 block, taking it from a specified position on the screen and displaying the copy in another position. The routine doesn't simply redraw whatever is to be copied—instead it makes a direct copy of it from the original in memory, a process which is much faster. Although the routine only copies one block at a time, it's an easy matter to put it inside a FOR . . . NEXT loop so that rectangular blocks of any size can be copied from one position to another.

When you use this routine, the originating block is left unchanged (unless of course you copy back onto the space it occupies). Furthermore, the routine only copies the part of the memory that holds the high-resolution information, and does not copy the color memory. This means that the object simply takes on whatever color combination has been defined in the area to which it is to be copied. However, the block-color routine, which also deals with 8x8 blocks, can be used in combination with the block-copy routine to set the colors of all the copies made.

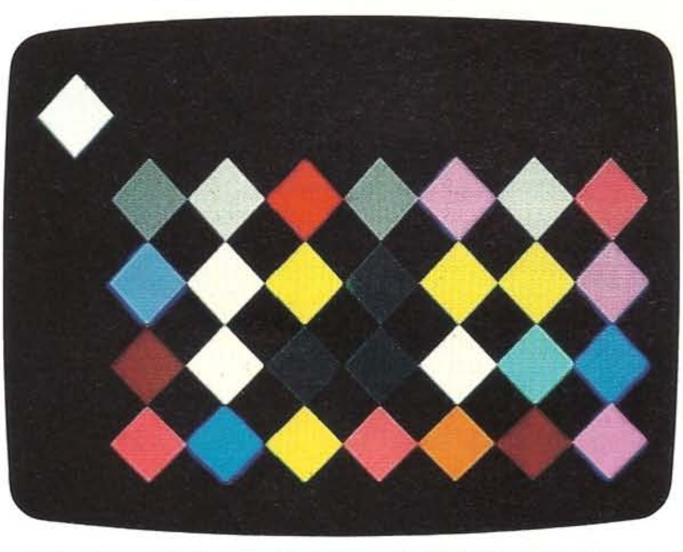
If you compare the Planet Copier program with the Planets program on page 15, you can see that the block-copy routine is simply looped to repeat the design.

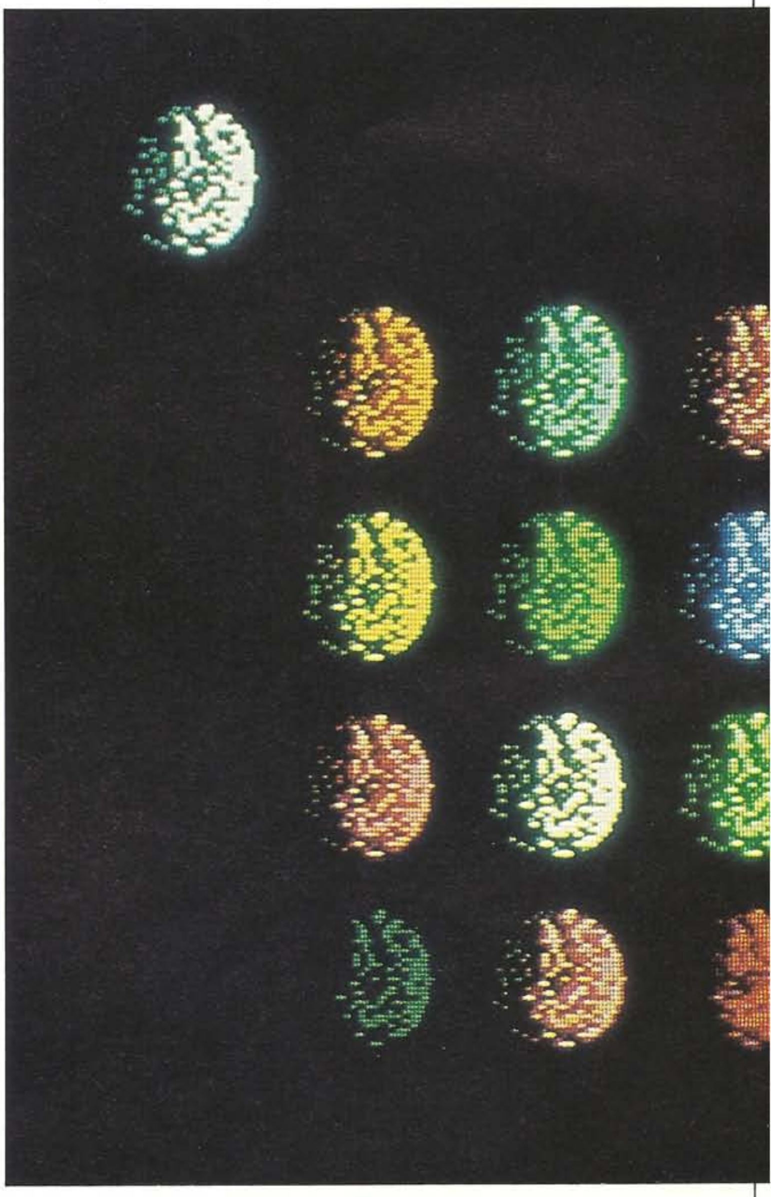
Copying a design

Each of the programs on these two pages produces a shape and then copies it over the screen. It's a simple way of using the block-copy routine to produce a pattern. The first program draws and fills a diamond, and copies it. The second produces a planet by plot-shading, using the method shown on pages 14-15, and then copies this design onto other parts of the screen.

DIAMOND COPIER PROGRAM

```
LIST

10000 SYS A1 : SYS B1,16 : POKE 53280,0
10010 SYS C1,20,0
10020 SYS D1,40,20 : SYS D1,20,40 : SYS
D1,0,20 : SYS D1,20,0 : SYS G1,20,20
10030 LX=0 : LY=0
10040 FOR NX=40 TO 280 STEP 40
10050 FOR NY=40 TO 160 STEP 40
10060 GOSUB 21000 : NEXT NY : NEXT NX
10070 GOTO 10070
20010 X1=INT(SQR(R*R-Y*Y))
21000 K=INT(RND(0)*15+1)*16
21010 FOR MX=0 TO 32 STEP 8
21020 FOR MY=0 TO 32 STEP 8
21030 SYS K1,LX+MX,LY+MY,NX+MX,NY+MY
21040 SYS B2,NX+MX,NY+MY,K
21050 NEXT MY : NEXT MX : RETURN
READY.
```





To try either program, first load the routines (the Diamond Copier needs routine blocks A-D and G) and then add block K. Next add the listings.

PLANET COPIER PROGRAM

```
LIST

10000 SYS A1 : SYS B1,16 : POKE 53280,0
10010 R=15 : XC=16 : YC=16
10020 GOSUB 20000
10030 LX=0 : LY=0
10040 FOR NX=40 TO 280 STEP 40
10050 FOR NY=40 TO 160 STEP 40
10060 GOSUB 21000 : NEXT NY : NEXT NX
10070 GOTO 10070
20000 FOR Y=-R TO R
20010 X1=INT(SQR(R*R-Y*Y))
20020 FOR X=-X1 TO X1
20030 N=INT(RND(1)*X1*2)+1
20040 IF N<X1+X THEN SYS C1,X+XC,Y+YC
20050 NEXT X : NEXT Y : RETURN
21000 K=INT(RND(0)*15+1)*16
21010 FOR MX=0 TO 24 STEP 8
21020 FOR MY=0 TO 24 STEP 8
21030 SYS K1,LX+MX,LY+MY,NX+MX,NY+MY
21040 SYS B2,NX+MX,NY+MY,K
21050 NEXT MY : NEXT MX : RETURN
READY.
```

BLOCK K

BLOCK-COPY routine

What the routine does
The routine makes a copy of whatever is displayed in a specified 8x8 pixel block. The copy can then be displayed in any other 8x8 block on the screen. The routine can be used within a loop like the block-color routine to copy rectangles made up of a number of 8x8 pixel blocks. Note that colors are not copied: a copy has whatever colors are already set in its destination block.

SYNTAX AND PARAMETERS

SYS K1,X,Y,A,B

X,Y	Horizontal and vertical coordinates of any point within the block to be copied (ranges 0-319 and 0-199).
A,B	Horizontal and vertical coordinates of any point within the destination block (ranges 0-319 and 0-199).

ROUTINE LISTING

```
5200 IF PEEK(51616)=32 THEN 5230
5210 SYS A3,5240 : FOR C=51616 TO 51678
5220 READ B : POKE C,B : NEXT C
5230 K1=51616
5240 DATA 32,184,201,165,253,133,251
5250 DATA 165,254,133,252,32,184,201
5260 DATA 160,7,177,251,145,253,136
5270 DATA 16,249,96,32,40,192,32
5280 DATA 224,192,144,5,162,14,76
5290 DATA 55,164,142,9,192,140,8
5300 DATA 192,32,40,192,32,236,192
5310 DATA 176,237,152,41,248,142,13
5320 DATA 192,141,12,192,76,0,193
```

You might notice with these programs that if you break them with the RUN/STOP key, and then type RUN followed by pressing RETURN, sometimes nothing happens. Instead, you just get an error message in high-resolution blocks. The reason for this is that the line you type RUN on probably has some of the display further along it. The computer treats this as part of your instruction, fails to understand it, and comes to a halt. To re-run the program, press the RUN/STOP and RESTORE keys together first.

PLANET COPIER PROGRAM

00:40

How the program works
First a planet is plotted in the top-left corner. The block-copy routine is then used to display copies of the planet over the screen, and these are colored by the block-color routine.
Lines 10100-10200 plot the planet to be copied.
Lines 10030-10700 copy the planet, using the block-copy routine inside a FOR...NEXT loop, and then add the colors each time.
Lines 20000-20050 form the planet-plotting subroutine.
Lines 21000-21050 form the copying and coloring subroutine.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
B Clear-and-color Block-color	13
C Plot	15
K Block-copy	39

BLOCK-COPYING 2

As well as using the block-copy routine to make interesting patterns, you can use it to alter displays. The program on these two pages produces a simulation of a sliding numbers game. It's a simple example of the game using a 3x3 grid, but once you have seen how the program works, you can make it more complex.

In the game, you move the numbers around until they are in order. To make the computer simulate this, you could program it to redraw the display after a move is made, showing the numbers in their new positions. However, if you use the block-copy routine, a lot of programming is avoided. The listing is still quite complex because the computer has to hold a lot of information in arrays. But without block-copying, it would be much more difficult.

The parts of the puzzle

The Puzzle program starts by drawing and filling the puzzle's border and then drawing just one puzzle piece. This is all carried out by lines 10000-10090 in the screen below, although you will need to key in the whole program before this is carried out.

PUZZLE PROGRAM

```
10000 SYS A1 : SYS B1,100
10010 SYS H1 : POKE 53280,6
10020 SYS C1,104,48 : SYS D1,104,151
10030 SYS D1,207,151 : SYS D1,207,48
10040 SYS D1,104,48 : SYS C1,119,63
10050 SYS D1,119,136 : SYS D1,192,136
10060 SYS D1,192,63 : SYS D1,119,63
10070 SYS G1,112,56 : SYS C1,121,65
10080 SYS D1,121,86 : SYS D1,142,86
10090 SYS D1,142,65 : SYS D1,121,65
10100 LX=120 : LY=64 : SYS A3,15000
10110 FOR C=2 TO 8 : READ X,Y,W
10120 P(C,0)=X : P(C,1)=Y : P(C,2)=C
10130 P(C,3)=W : GOSUB 20000 : NEXT C
10140 P(9,0)=168 : P(1,0)=LX : P(1,2)=1
10150 P(9,1)=112 : P(1,1)=LY : P(9,2)=9
10160 P(1,3)=8196 : P(9,3)=1664
10170 FOR C=1 TO 8 : GOSUB 22000
10180 SYS H2,P(C,0)+8,P(C,1)+8,CHR$(P(C,
2)+48) : NEXT C : K=9
10190 GET A$ : IF A$="" THEN 10190
10200 FOR C=1 TO 30 : N=INT(RND(0)*4)

READY.
```

You won't see the block-copy routine in the first part of the program, but it is there, in the form of subroutines. Every time the subroutine at line 20000 is called, the block-copy routine copies a 3x3 character block from coordinates LX,LY to X,Y. Lines 10100-10160 use this subroutine to copy the top-left puzzle piece to the coordinates READ by line 10110. This section of the program also sets up the nine puzzle piece positions.

When you have keyed in the machine-code routines and the complete Puzzle program, pressing RETURN after the puzzle has first appeared will rearrange the the puzzle pieces at random. You can then use the cursor keys to move the pieces back into the right order.

PUZZLE PROGRAM (CONTD.)

```
LIST 10210-15030

10210 M=INT(16↑N+0.2) : R=M*15 : IF R>32
767 THEN R=R-65536
10220 IF (P(K,3) AND R)=0 THEN 10240
10230 GOSUB 21000
10240 NEXT C
10250 GET A$ : IF A$="" THEN 10250
10260 A=ASC(A$) : R=-15*(A=145)-240*(A=2
9)-3840*(A=17)+4096*(A=157)
10270 IF R=0 THEN 10250
10280 N=-1*(A=29)-2*(A=17)-3*(A=157)
10290 IF (P(K,3)AND R)=0 THEN 10250
10300 GOSUB 21000
10310 FOR C=1 TO 9
10320 IF P(C,2)<>C THEN 10250
10330 NEXT C
10340 GOTO 10340
15000 DATA 144,64,12309,168,64,38
15010 DATA 120,88,20743,144,88,25160
15020 DATA 168,88,857,120,112,-31744
15030 DATA 144,112,-27280

READY.
```

```
LIST 20000-

20000 FOR MX=0 TO 16 STEP 8
20010 FOR MY=0 TO 16 STEP 8
20020 SYS K1,LX+MX,LY+MY,X+MX,Y+MY
20030 SYS B2,X+MX,Y+MY,PEEK(1024+40*INT(
(LY+MY)/8)+INT((LX+MX)/8))
20040 NEXT MY : NEXT MX : RETURN
21000 S=INT(P(K,3)/16↑N)-16*INT(P(K,3)/1
6↑(N+1))
21010 LX=P(S,0) : LY=P(S,1)
21020 X=P(K,0) : Y=P(K,1) : GOSUB 20000
21030 X=LX : Y=LY : LX=0 : LY=0
21040 GOSUB 20000 : TMP=P(K,2)
21050 P(K,2)=P(S,2) : P(S,2)=TMP
21060 K=S : RETURN
22000 READ K : FOR X=0 TO 16 STEP 8
22010 FOR Y=0 TO 16 STEP 8
22020 SYS B2,X+P(C,0),Y+P(C,1),K
22030 NEXT Y : NEXT X : RETURN
READY.
```

PUZZLE PROGRAM

00:15

How the program works

The block-copy routine is used in a subroutine to rearrange the pieces of the puzzle. The computer will produce a random arrangement which can then be sorted into sequence with the cursor keys.

Lines 10000-10160 produce the initial puzzle by calling the block-copy routine which is contained in a subroutine.

Lines 10190-10240 scan the keyboard and then rearrange the puzzle at random.

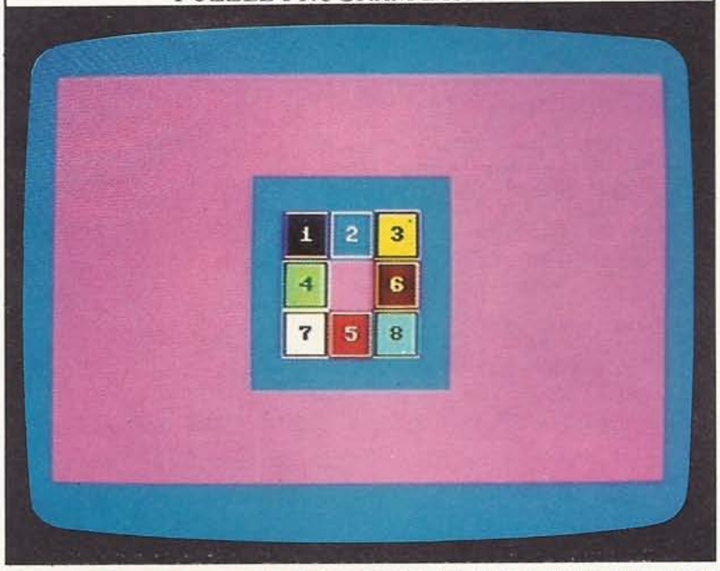
Lines 10250-10300 move the pieces in reponse to the player's instructions.

Lines 10310-10340 test to see if the solution has been reached.

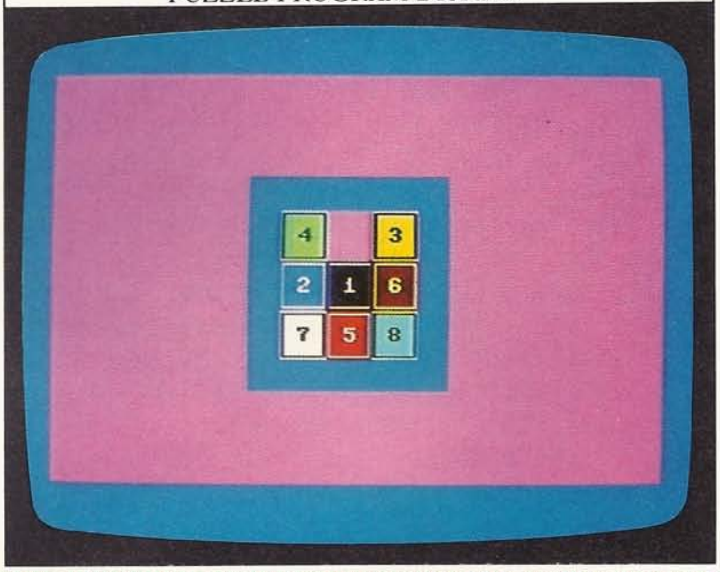
ROUTINES USED BY THIS PROGRAM

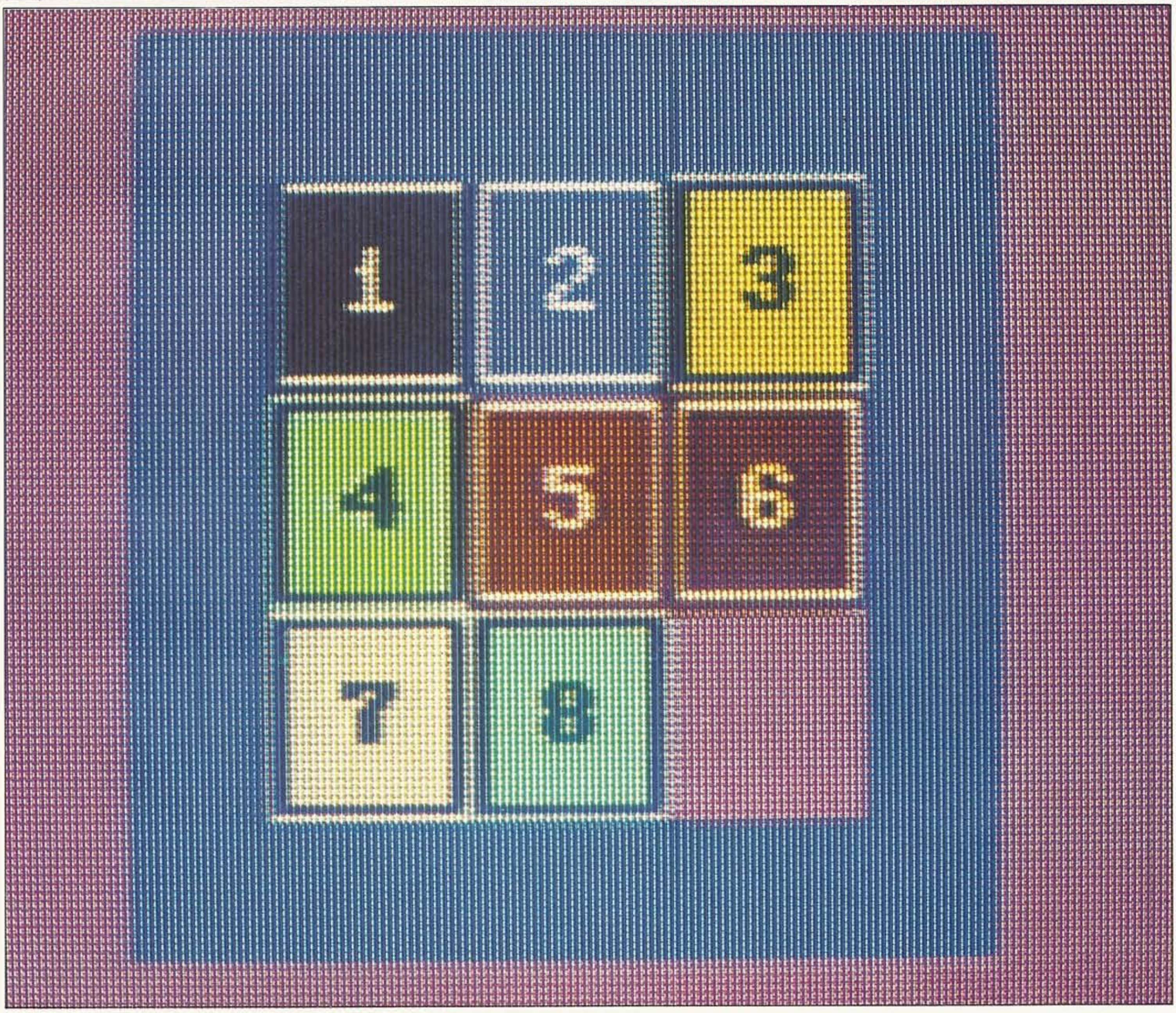
Block Routine(s)	Page
A High-resolution Restore	11
B Clear-and-color Block-color	13
C Plot	15
D Draw	17
Flood-fill	27
H ROM-copy Text	31
K Block-copy	39

PUZZLE PROGRAM DISPLAY



PUZZLE PROGRAM DISPLAY





SCREEN-SCROLLING

The final facility provided by the machine-code graphics routines is screen scrolling. Scrolling is the movement of the whole screen display, usually horizontally or vertically. Left and right scroll are particularly useful in games and other programs where you want a moving background.

You could use the block-copy routine to produce scrolling, but if you try to do this, you will find that it's unacceptably slow. Although the block-copy routine itself works quite rapidly, the Commodore BASIC ROM takes time to interpret and carry out the FOR . . . NEXT loops needed to block-copy the contents of the entire screen. However, the scroll routine in block L opposite completely eliminates the need for any BASIC. It carries out horizontal scrolling at high speed by replacing the BASIC FOR . . . NEXT statements with machine code.

Scrolling and wrap-around

With the screen-scroll routine's machine code in memory, you can scroll a display with a command like this:

```
SYS L1,D,C
```

where D is the direction of the scroll (1=left and 0=right) and C is the code of the color that will be left behind in the newly-created strip down the side of the screen when the scroll is carried out. Since you would not often want to use the color combination 0 (black foreground and black background) this has been pressed into service in another way. When color combination 0 is used, the character positions that are vacated by the scroll are filled by the characters that have just been pushed off the other end of the screen. This makes the screen "wrap around", a facility that is useful in animating backgrounds to give more complex displays.

How to make a display scroll

If you have a program which finishes with a line like:

```
10400 GOTO 10400
```

all you have to do to make it scroll is to change the final line and add one more. To wrap-around the display, you would need the following lines:

```
10400 SYS L1,1,0
10410 GOTO 10400
```

This repeats the scroll routine, moving the display to the left and wrapping it around.

To make the program move to the right, but produce a white screen instead of wrapping around, you would need to add:

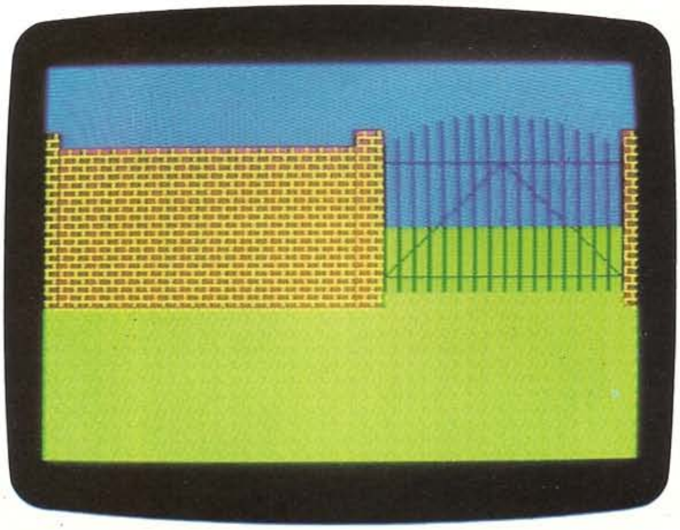
```
10400 SYS L1,0,17
10410 GOTO 10400
```

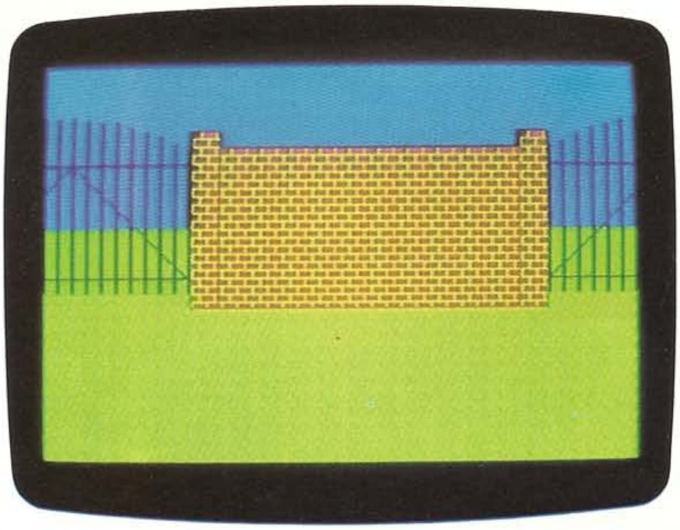
All the effects so far use GOTO to produce a loop which endlessly scrolls the display. You can however scroll a display a set distance to the right or left by using a FOR . . . NEXT loop. You can even link together a series of these loops so that a display moves from side to side by a specified or random amount.

Different effects with the scroll routine

The displays on these two pages have been produced by taking two programs from earlier in this book, and then adding the scroll routine. The Wall and Gate program shows the wrap-around scroll which repeats the original design. The Line Landscape displays show both wrap-around and scrolling to reveal a color—in this case purple, the same color as the original two "buildings".

WRAP-AROUND SCROLLING





BLOCK L

SCREEN-SCROLL routine

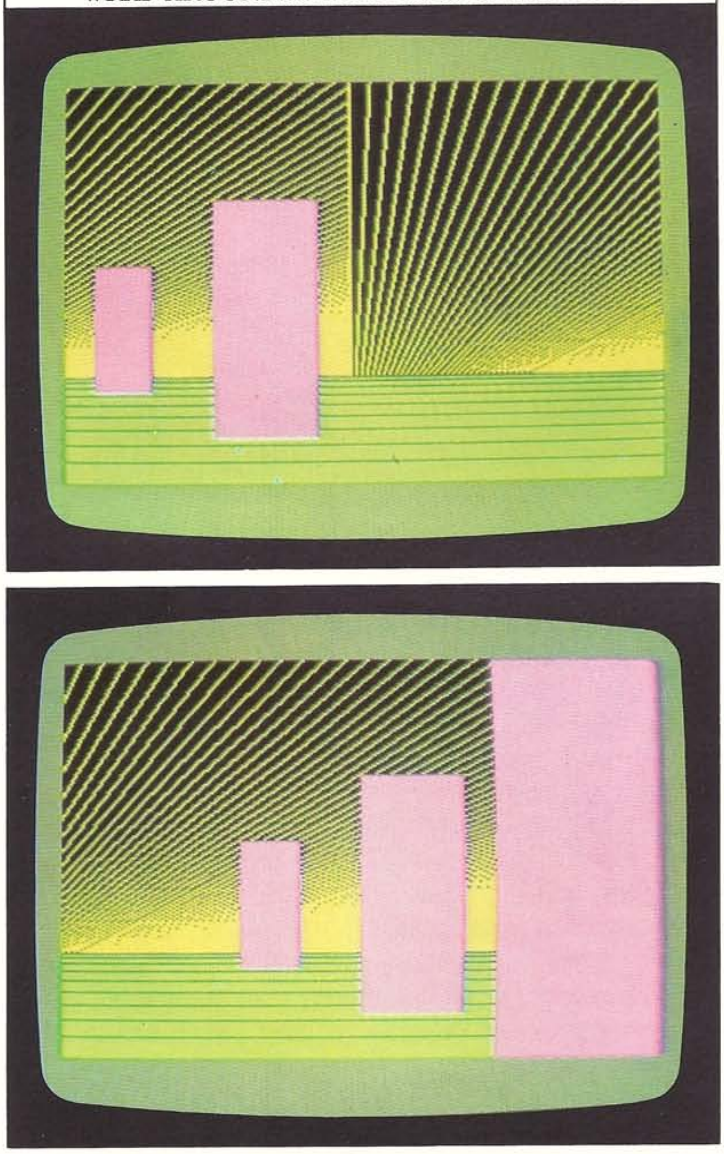
What the routine does

The routine scrolls the contents of the entire screen by one column of 8x8 pixel blocks either to the right or left. If the routine is used once it will move the display by one column only. However it can be used in an endless loop for continuous scrolling, or it can be used in a closed loop to move the display a specified amount to the left or the right. The routine will either color the area vacated by the scroll (the color can be specified), or it will allow "wrap-around" of the display.

To make a display scroll off the screen, use SYS L1,followed by a direction and a color code. The display will then scroll one 8-pixel coumn at a time to the left or right, leaving a blank colored area. This colored area is controlled by a color code. To make a display scroll left or right but wrap-around the screen, use SYS L1, followed by a direction (coded by 0 or 1) and then the color code 0. The scroll routine can be used to move a display from side to side behind stationary sprites, and if it is used with the RND function, the screen can be made to scroll unpredictably. It can also be used to gradually remove one display from the screen to make way for another instead of simply clearing the screen with the clear-and-color routine.

The routine uses standard Commodore color combination codes. For details, see the chart on page 63.

WRAP-AROUND/BLANK-COLOR SCROLLING



SYNTAX AND PARAMETERS

SYS L1,D,C	
D	Direction of scroll (0=right, 1=left).
C	Color of vacated area (0=wrap around, 1-255 = standard color combinations).

ROUTINE LISTING

```
5400 IF PEEK(51689)=32 THEN 5430
5410 SYS A3,5440 : FOR C=51679 TO 52168
5420 READ B : POKE C,B : NEXT C
5430 L1=51689
5440 DATA 0,0,0,0,0,0,0
5450 DATA 0,0,0,32,40,192,192
5460 DATA 0,208,3,76,213,202,169
5470 DATA 0,141,16,192,141,17,192
5480 DATA 32,40,192,140,223,201,192
5490 DATA 0,240,16,140,224,201,160

5500 DATA 7,169,0,153,225,201,136
5510 DATA 16,250,76,69,202,140,8
5520 DATA 192,140,26,192,140,9,192
5530 DATA 140,27,192,140,13,192,173
5540 DATA 16,192,141,12,192,141,28
5550 DATA 192,32,0,193,169,225,133
5560 DATA 251,169,201,133,252,32,191
5570 DATA 203,32,82,193,160,0,177
5580 DATA 251,141,224,201,169,38,141
5590 DATA 14,192,173,16,192,141,12

5600 DATA 192,141,28,192,173,17,192
5610 DATA 141,13,192,169,8,141,8
5620 DATA 192,141,26,192,169,0,141
5630 DATA 9,192,141,27,192,32,0
5640 DATA 193,56,165,253,233,8,133
5650 DATA 251,165,254,233,0,133,252
5660 DATA 32,191,203,24,165,253,133
5670 DATA 251,105,8,133,253,165,254
5680 DATA 133,252,105,0,133,254,172
5690 DATA 14,192,136,140,14,192,16

5700 DATA 227,169,225,133,253,169,201
5710 DATA 133,254,32,191,203,32,82
5720 DATA 193,56,165,251,233,1,133
5730 DATA 253,165,252,233,0,133,254
5740 DATA 160,0,177,251,145,253,200
5750 DATA 192,39,208,247,173,224,201
5760 DATA 145,253,24,173,16,192,105
5770 DATA 8,141,16,192,201,200,240
5780 DATA 6,172,223,201,76,1,202
5790 DATA 96,169,0,141,16,192,141

5800 DATA 17,192,32,40,192,140,223
5810 DATA 201,192,0,240,16,140,224
5820 DATA 201,160,7,169,0,153,225
5830 DATA 201,136,16,250,76,46,203
5840 DATA 160,1,140,9,192,140,27
5850 DATA 192,160,56,140,8,192,140
5860 DATA 26,192,172,16,192,140,12
5870 DATA 192,140,28,192,172,17,192
5880 DATA 140,13,192,32,0,193,169
5890 DATA 225,133,251,169,201,133,252

5900 DATA 32,191,203,32,82,193,160
5910 DATA 0,177,251,141,224,201,169
5920 DATA 38,141,14,192,169,0,141
5930 DATA 26,192,141,27,192,173,16
5940 DATA 192,141,12,192,141,28,192
5950 DATA 173,17,192,141,13,192,169
5960 DATA 48,141,8,192,169,1,141
5970 DATA 9,192,32,0,193,24,165
5980 DATA 253,105,8,133,251,165,254
5990 DATA 105,0,133,252,32,191,203

6000 DATA 56,165,253,133,251,233,8
6010 DATA 133,253,165,254,133,252,233
6020 DATA 0,133,254,172,14,192,136
6030 DATA 140,14,192,16,227,169,225
6040 DATA 133,253,169,201,133,254,32
6050 DATA 191,203,32,82,193,24,165
6060 DATA 251,105,1,133,253,165,252
6070 DATA 105,0,133,254,160,38,177
6080 DATA 251,145,253,136,16,249,200
6090 DATA 173,224,201,145,251,24,173
6100 DATA 16,192,105,8,141,16,192
6110 DATA 201,200,240,6,172,223,201
6120 DATA 76,227,202,96,160,7,177
6130 DATA 253,145,251,136,16,249,96
```

GRAPHICS EDITOR 1

So far, all the graphics you have produced have been in the form of specific programs to produce specific pictures. The program on the next six pages lets you create pictures directly on the keyboard. It gives you instant access to all the routines so far, and it also provides two graphics cursors, an optional color grid and a facility for saving your displays on tape or disk.

How to key in the graphics editor

You can build up the graphics editor in easy stages so that each part can be tested as you key it in. The program uses graphics routines in all the blocks from A-L, so you must have these in memory before you start keying in.

Although the editor is written in six consecutive parts, do not try to assemble it using the merge routine. If you do, it will not work, because its lines are not always built up in numerical order.

Producing the cursors

Part 1 of the editor listing generates two cursors. The large cross is called the main cursor, and this is controlled by the usual cursor keys, moving one pixel at a time. A second cursor can also be made to appear by pressing the M key. This cursor cannot be moved by the cursor keys.

In general, to use the graphics editor, you need to mark one or sometimes two points with the cursors. For operations that need two points to be marked, the second or marker cursor is used. If you move the main cursor after pressing the M key, the marker cursor will stay in the main cursor's original position. Pressing M a second time unites the cursors again.

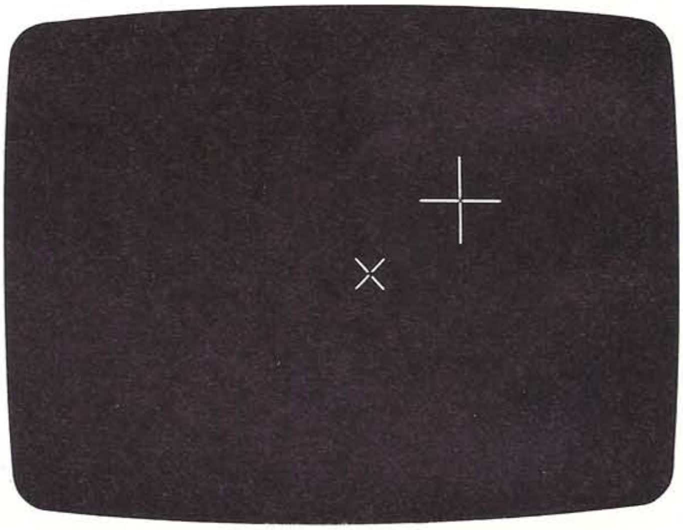
Type in part 1 of the editor and check that the cursors appear before continuing.

GRAPHICS EDITOR PART 1

```
10010 KOL=16 : D=0 : POKE 650,128
10020 SYS A1 : SYS B1,16 : SYS F1,0
10030 CX=160 : CY=100 : MX=0 : MY=220
10040 GOSUB 20000 : GOSUB 20100
10050 GET A$
10060 IF A$="" THEN D=0 : GOTO 10050
10070 KE=ASC(A$) : D=D+1
10080 IF KE<>29 THEN 10120
10090 GOSUB 20000 : CX=CX+D
10100 IF CX>319 THEN CX=319 : D=0
10110 GOSUB 20000 : GOTO 10050
10120 IF KE<>157 THEN 10160
10130 GOSUB 20000 : CX=CX-D
10140 IF CX<0 THEN CX=0 : D=0
10150 GOSUB 20000 : GOTO 10050
10160 IF KE<>145 THEN 10200
10170 GOSUB 20000 : CY=CY-D
10180 IF CY<0 THEN CY=0 : D=0
10190 GOSUB 20000 : GOTO 10050
10200 IF KE<>17 THEN D=0 : GOTO 10240
10210 GOSUB 20000 : CY=CY+D
10220 IF CY>199 THEN CY=199 : D=0

READY.
```

GRAPHICS EDITOR PART 1 (CONTD.)

```
LIST 10230-

10230 GOSUB 20000 : GOTO 10050
10240 IF KE<>77 THEN 10270
10250 GOSUB 20100 : MX=CX : MY=CY
10260 GOSUB 20100 : GOTO 10050
10270 GOTO 10270
20000 SYS F1,1
20010 SYS C1,CX-20,CY
20020 SYS D1,CX+20,CY
20030 SYS C1,CX,CY-20
20040 SYS D1,CX,CY+20
20050 SYS F1,0 : RETURN
20100 SYS F1,1
20110 SYS C1,MX-7,MY-7
20120 SYS D1,MX+7,MY+7
20130 SYS C1,MX+7,MY-7
20140 SYS D1,MX-7,MY+7
20150 SYS F1,0 : RETURN
READY.
```



The editor commands

Part 2 of the editor provides three graphics facilities — plotting, drawing and flood-filling.

To plot a point, move the main cursor to the required position, press D and the point will be plotted. The marker cursor will be updated so that it is in the same position as the main cursor.

Drawing lines is equally easy. The program draws a line from the marker cursor to the main cursor when you press L. You can draw lines from the existing marker cursor position or from a new one, specified by pressing M. After you have pressed L, the line appears, and the marker cursor is united with the main cursor. This enables you to draw a long sequence of lines quickly.

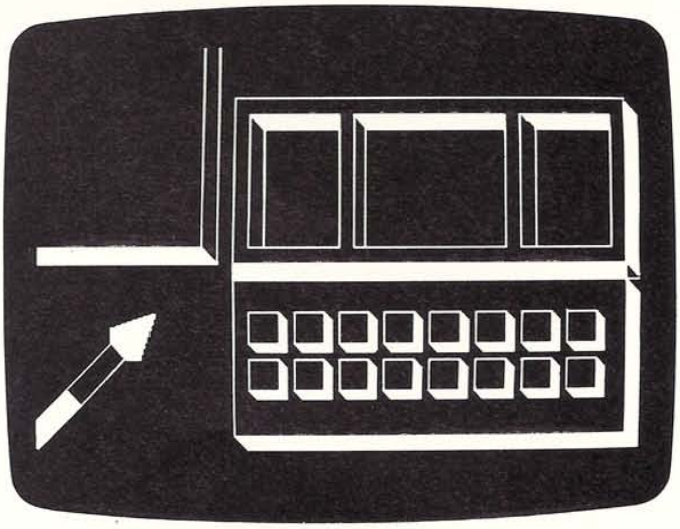
Flood-filling areas is simple. Just move the main cursor to the place where you want the flood-fill to start and press F.

If you add the part 2 listing to part 1, you will be able to test out these three facilities. If your combined parts 1 and 2 seem to run properly, make a copy on tape or disk so that if you introduce a bug later, you don't run the risk of losing all your work so far. If your program doesn't work, check that you have all the machine-code routines in memory. Press BREAK and then SYS A2 to identify problem lines when running.

GRAPHICS EDITOR PART 2

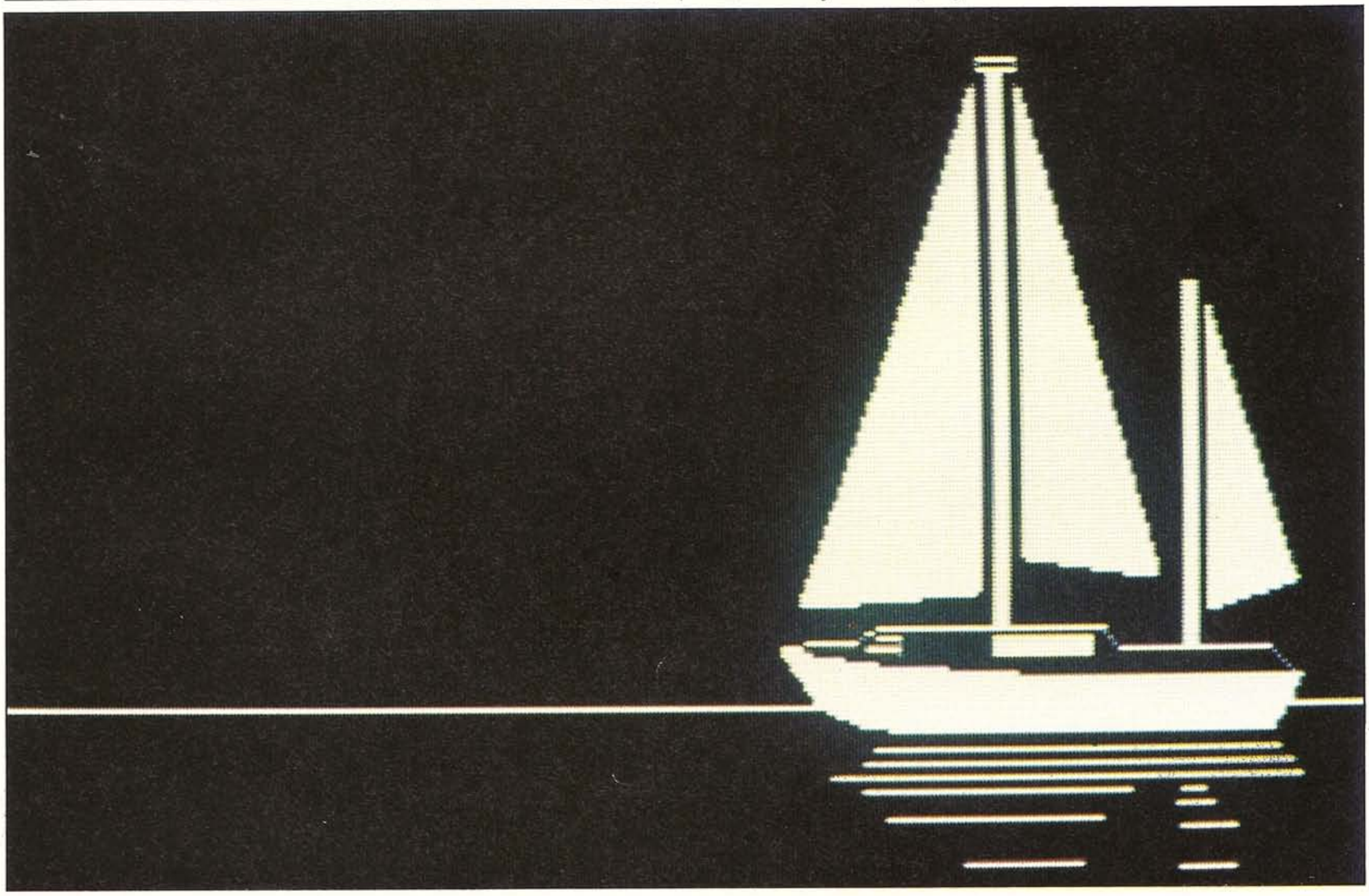
```
LIST
10270 IF KE<>68 THEN 10320
10280 GOSUB 20000 : GOSUB 20100
10290 SYS C1,CX,CY : MX=CX
10300 MY=CY : GOSUB 20100
10310 GOSUB 20000 : GOTO 10050
10320 IF KE<>76 THEN 10370
10330 GOSUB 20000 : GOSUB 20100
10340 SYS C1,MX,MY : SYS D1,CX,CY
10350 MX=CX : MY=CY : GOSUB 20100
10360 GOSUB 20000 : GOTO 10050
10370 IF KE<>70 THEN 10420
10380 GOSUB 20000 : GOSUB 20100
10390 SYS G1,CX,CY : MX=CX
10400 MY=CY : GOSUB 20100
10410 GOSUB 20000 : GOTO 10050
10420 GOTO 10420
READY.
```

DISPLAY USING PARTS 1 AND 2



The graphics editor program makes coloring very easy. But before you turn over and find out how to do this, try producing some designs in black and white first. It's always easiest to draw first and color later.

On page 49 you will find a list of keys used by the editor. If you press an unused key when the final program is running, it will ignore it. Until then, press only the keys which the program uses, or the editor may halt and your display will be lost.



GRAPHICS EDITOR 2

You can add a coloring facility to your graphics editor by keying in part 3 of the program. This allows you to set up colors for all or part of the screen. It works in two stages. First you decide what color combination you want to use, and then you decide which area of the screen you want to appear in these colors.

To select a color combination, first press I followed by a standard Commodore color control code from 0-255 (see the chart on page 63) and then by RETURN. Don't wait for a prompt after you press I as the program does not produce one. Now position the marker cursor at the top-left of the area you want to color, and move the main cursor to the bottom right. When you press C, the area will be colored by the block-color routine. Remember that coloring is a two-stage process — first color selection, then positioning.

Because color is dealt with in 8x8 pixel blocks, you will probably find that the colored area is slightly larger than you specified because the colors are set to the next boundary up. If you take a black and white design like the one shown on the previous page, it's easy to start adding some colors to it.

Using the on-screen grid

When you use the coloring facility, you may find that it's difficult to know exactly where the boundaries of the 8x8 pixel blocks are. You can overcome this problem with the part 4 of the editor. After typing it in, run the program, and press the G key. You'll find that a complete color grid is overprinted on the screen. Pressing G a second time makes the grid vanish, leaving your design exactly as it was before.

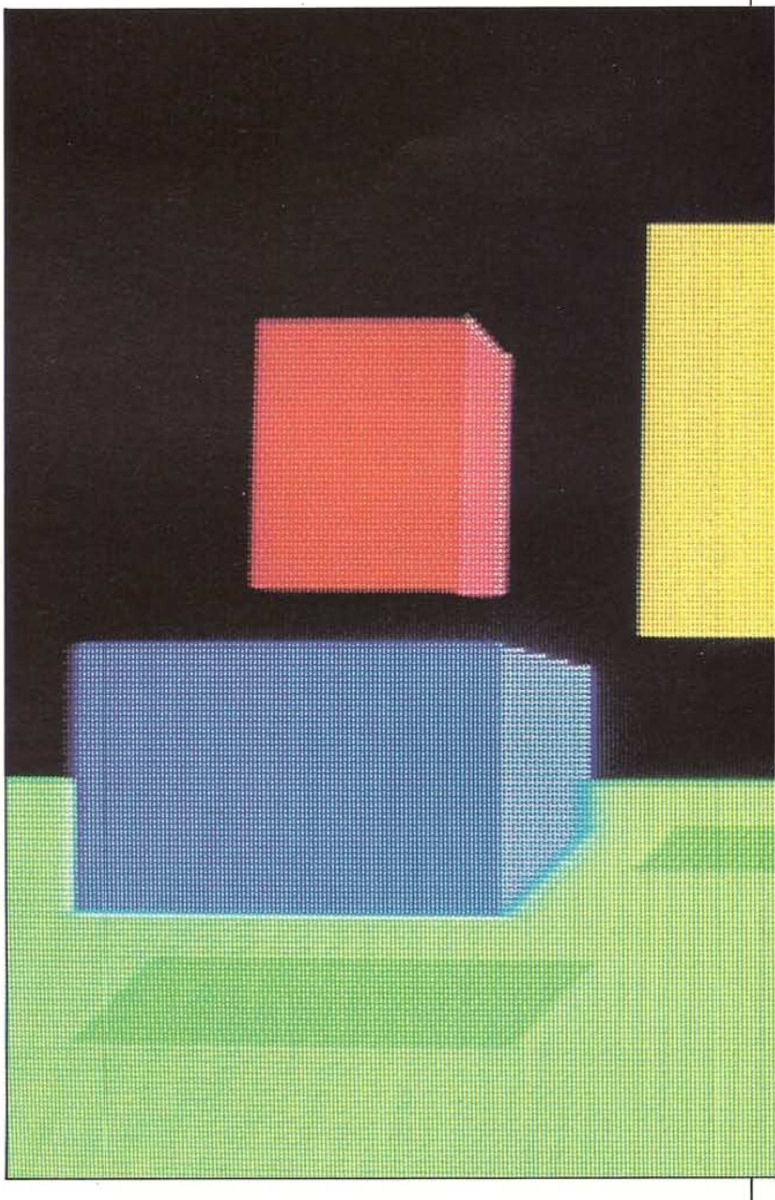
When you use this facility, it is important that you don't draw any objects or fill any areas while the grid is on the screen.

GRAPHICS EDITOR PART 3

```
10420 IF KE<>73 THEN 10510
10430 TK=0
10440 GET A$ : IF A$="" THEN 10440
10450 A=ASC(A$) : IF A=13 THEN 10490
10460 IF A<48 THEN 10440
10470 IF A>57 THEN 10440
10480 TK=TK*10+A-48 : GOTO 10440
10490 IF TK<0 OR TK>255 THEN 10050
10500 KOL=TK : GOTO 10050
10510 IF KE<>67 THEN 10630
10520 IF MX>CX THEN 10050
10530 IF MY>CY THEN 10050
10540 LX=MX AND 504 : LY=MY AND 248
10550 UX=CX AND 504 : UY=CY AND 248
10560 FOR X=LX TO UX STEP 8
10570 FOR Y=LY TO UY STEP 8
10580 SYS B2,X,Y,KOL
10590 NEXT Y : NEXT X
10600 GOSUB 20100 : MX=CX
10610 MY=CY : GOSUB 20100
10620 GOTO 10050
10630 GOTO 10630

READY.
```

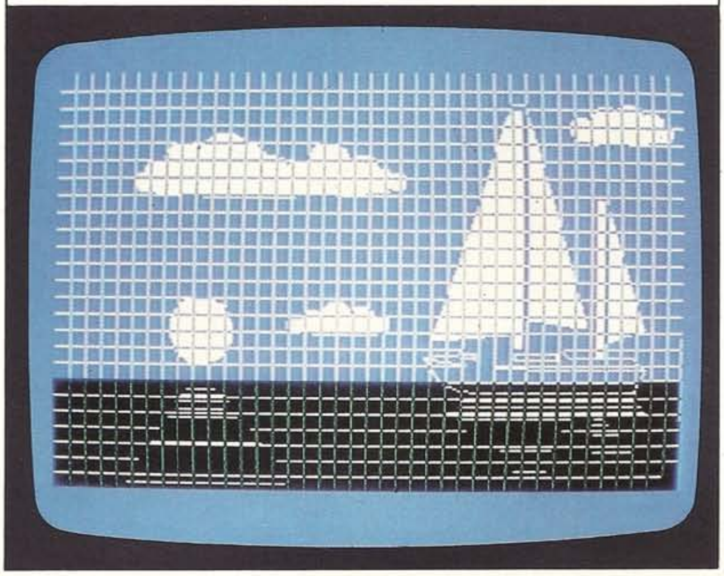
GRAPHICS EDITOR PART 4

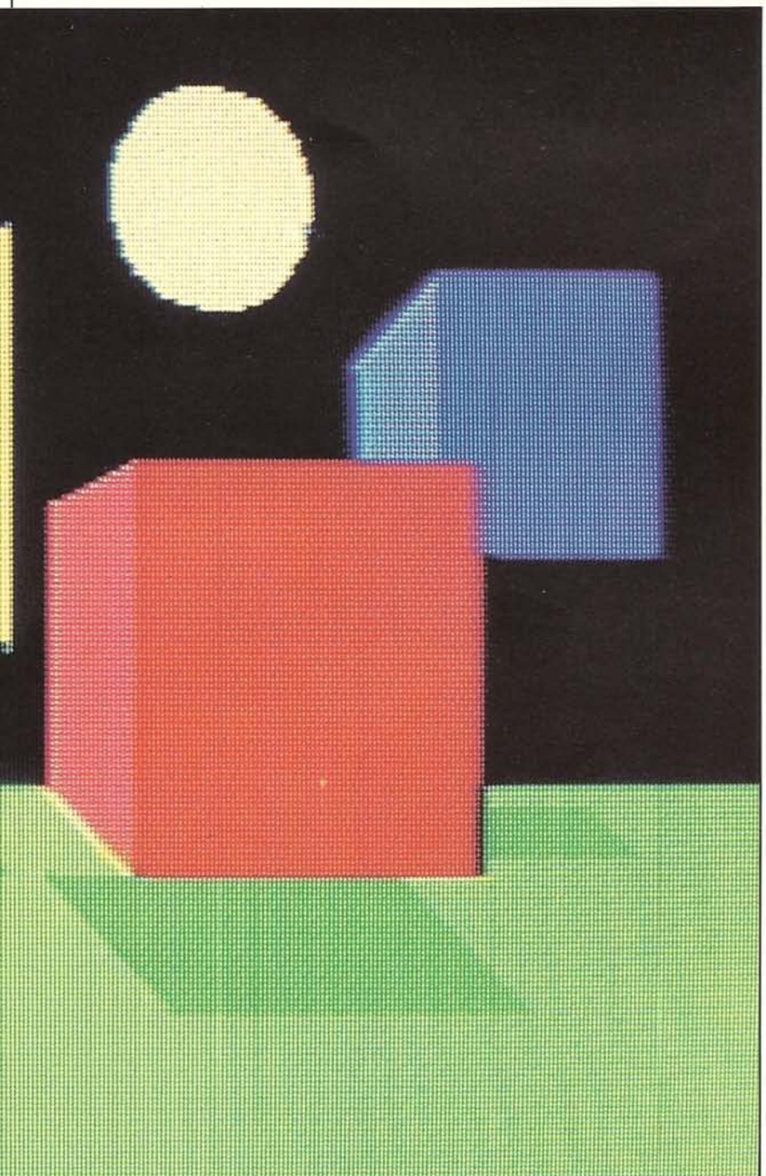
```
10630 IF KE<>71 THEN 10720
10640 GOSUB 20000 : GOSUB 20100
10650 SYS F1,1
10660 FOR X=8 TO 312 STEP 8
10670 SYS C1,X,0 : SYS D1,X,199
10680 NEXT X : FOR Y=8 TO 192 STEP 8
10690 SYS C1,0,Y : SYS D1,319,Y
10700 NEXT Y : GOSUB 20100
10710 GOSUB 20000 : GOTO 10050
10720 IF KE<>80 THEN 10840
10730 TP=0
10740 GET A$ : IF A$="" THEN 10740
10750 A=ASC(A$) : IF A=13 THEN 10800
10760 IF A=81 THEN 10050
10770 IF A<48 OR A>57 THEN 10740
10780 TP=TP*10+A-48 : GOTO 10740
10790 IF TP<0 OR TP>255 THEN 10050
10800 GOSUB 20000 : GOSUB 20100
10810 SYS J1,CX,CY,TP
10820 MX=CX : MY=CY : GOSUB 20100
10830 GOSUB 20000 : GOTO 10050
10840 GOTO 10840
READY.
```



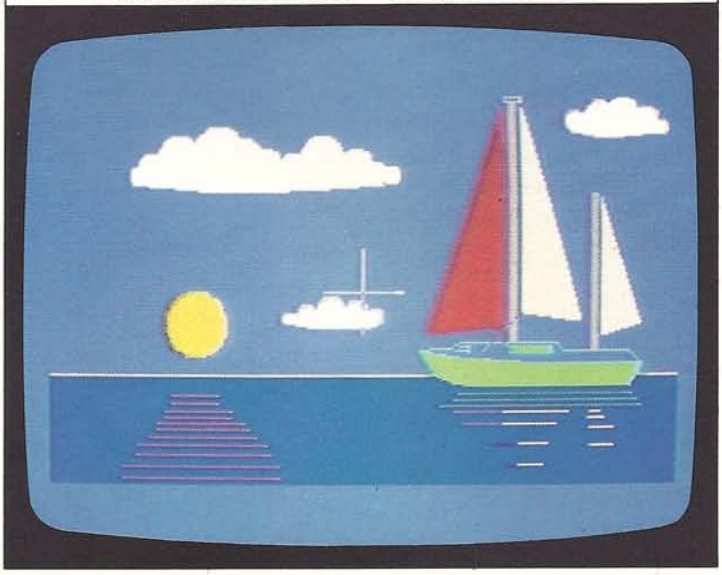
The next two displays show how the grid looks when in use. After the colors have been applied, the grid is removed and the design left intact.

GRID IN USE





COLORED DISPLAY — GRID REMOVED



Pattern-filling, circles and text

Part 4 of the editor provides a pattern-filling facility. To fill an area with a pattern, position the cursor at the top of the area, and then press the P key. Then enter a character number from 0-255 and press RETURN. To fill with a defined character, use the ROM-copy and define-character routines in the direct mode after stopping the program with the RUN/STOP and RESTORE keys when the cursors appear. Then simply re-run the program.

Part 5 of the editor allows you to draw circles and print text. To draw a circle, move the marker cursor to the center of the circle. The main cursor is then used to mark any position on the circumference. Press O, and the circle appears. To write text on the screen, move the main cursor to the start position, press T and then begin writing. If you make a mistake, you can use the INST/DEL key to erase it. Press RETURN when you get to the end of a line, or when you have finished.

GRAPHICS EDITOR PART 5

```
10840 IF KE<>79 THEN 10900
10850 X=ABS(MX-CX) : Y=ABS(MY-CY)
10860 R=INT(SQR(X*X+Y*Y))
10870 GOSUB 20000 : GOSUB 20100
10880 SYS E1,MX,MY,R : GOSUB 20100
10890 GOSUB 20000 : GOTO 10050
10900 IF KE<>84 THEN 11050
10910 GOSUB 20000 : GOSUB 20100
10920 T=0 : MX=CX : SYS H1
10930 GET A$ : IF A$="" THEN 10930
10940 IF A$<>CHR$(20) THEN 10980
10950 IF T=0 THEN 10930
10960 CX=CX-8 : SYS H2,CX,CY," "
10970 T=T-1 : GOTO 10930
10980 IF A$<>CHR$(13) THEN 11020
10990 IF CY+8<200 THEN CY=CY+8
11000 MY=CY : CX=MX : GOSUB 20100
11010 GOSUB 20000 : GOTO 10050
11020 SYS H2,CX,CY,A$
11030 IF CX<312 THEN CX=CX+8 : T=T+1
11040 GOTO 10930
11050 GOTO 11050
READY.
```

GRAPHICS EDITOR 3

By this stage, you have probably produced some designs which really needed some tidying up, but so far you haven't been able to erase anything, except by overprinting with something else. You can use the text facility to erase anything that you have drawn. To do this, move the cursors to the top left of the area you want to erase. Press T to switch to text and then press the space bar. This will erase one 8x8 square at a time.

Alternatively, you can block-color any area with black on black. However, you should remember that this won't erase any designs, but will just hide them. Changing the colors again will make the design visible once more.

How to store and retrieve your pictures

Having drawn a picture, you can make the editor save it on tape or disk. Part 6 of the program adds a short display section at the beginning of the program which asks you for some details. Your displays will be stored as files, each with a filename. The computer needs to know three things — the kind of storage system you are using, the name of the "LOAD" file (the one to be taken from storage and put on the screen) and the name of the "SAVE" file (the display to be sent to storage, in other words the one you are about to create).

After you have keyed in part 6 of the listing, your graphics editor is complete. You will find that the program now starts by asking three questions. You must answer these before the program will continue:

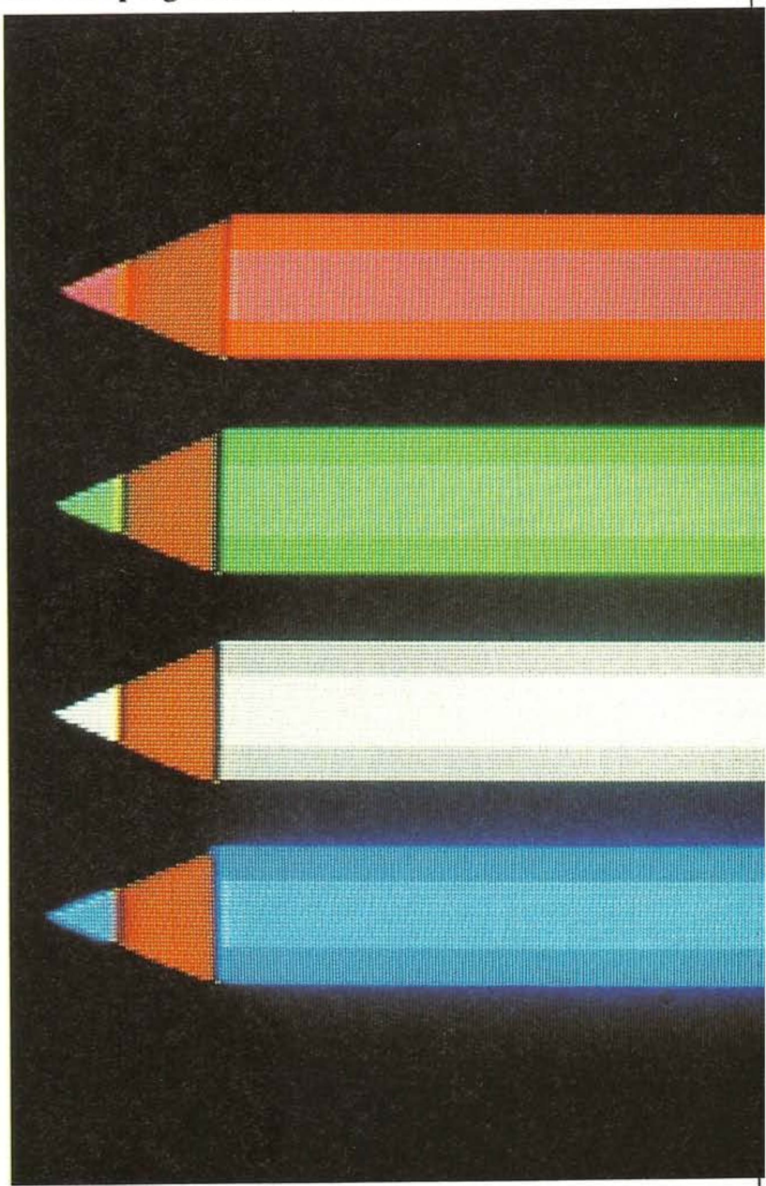
1/2 ?
LOAD ?
SAVE ?

The first means "tape or disk storage?". Key in 1 for tape, 2 for disk. The second question asks which file you want the program to LOAD, and the third which file you

GRAPHICS EDITOR PART 6

```
10000 GOTO 11310
11050 IF KE<>87 THEN 11170
11060 GOSUB 20000 : GOSUB 20100 : G=1
11070 GOSUB 20200 : IF DEV=1 THEN 11090
11080 OPEN 1,8,2,"0:"+FO$+",S,W" : GOTO
11100
11090 OPEN 1,1,1,FO$
11100 FOR C=8192 TO 16191
11110 PRINT#1,CHR$(PEEK(C));
11120 NEXT C
11130 FOR C=6144 TO 7167
11140 PRINT#1,CHR$(PEEK(C));
11150 NEXT C : CLOSE 1 : GOSUB 20100
11160 GOSUB 20000 : GOTO 10050
11170 IF KE<>82 THEN 10050
11180 GOSUB 20000 : GOSUB 20100
11190 IF DEV=1 THEN 11210
11200 OPEN 1,8,2,"0:"+FI$+",S,R" : GOTO
11220
11210 OPEN 1,1,0,FI$
11220 FOR C=8192 TO 16191
11230 GET#1,A$

READY.
```

```
11240 POKE C,ASC(A$+CHR$(0))
11250 NEXT C
11260 FOR C=6144 TO 7167 : GET #1,A$
11270 POKE C,ASC(A$+CHR$(0)) : NEXT C
11280 CLOSE 1 : G=0
11290 GOSUB 20200 : GOSUB 20100
11300 GOSUB 20000 : GOTO 10050
11310 PRINT CHR$(147)
11320 INPUT "1/2 ";A$
11330 PRINT
11340 DEV=0
11350 IF A$="2" THEN DEV=1
11360 PRINT
11370 PRINT "LOAD ";
11380 INPUT FI$
11390 PRINT : PRINT
11400 PRINT "SAVE ";
11410 INPUT FO$
11420 GOTO 10010
20200 FOR C=1024 TO 2047
20210 IF G=1 THEN POKE 5120+C,PEEK(C)
20220 IF G=0 THEN POKE C,PEEK(5120+C)
20230 NEXT C : RETURN
READY.
```



want to SAVE. You don't have to LOAD or SAVE anything, but in case you want do, the program needs to know the filenames before it can continue.

When you have answered the questions and created a display, pressing the W key will make the computer store it. Pressing the R key will make the computer retrieve the LOAD file and show it on the screen. You do not need to press RETURN for either. Storage and retrival take a number of minutes as there is a lot of information involved, so don't be impatient if initially nothing seems to happen.

Once either operation is complete, the program continues to run from where it left off.

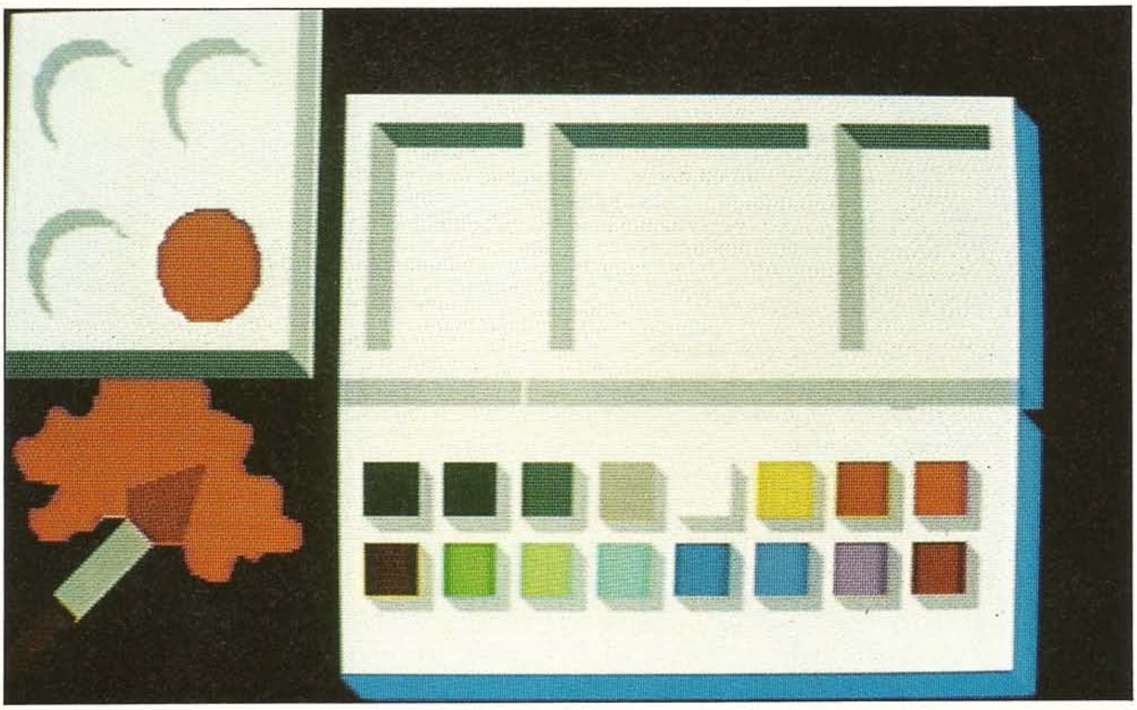
Choosing filenames

If you are using disks, it's important to remember that once you have decided on two filenames, you can't change them except by starting the program again. So if you call the current SAVE file "DISPLAY3", for example, and you already have a DISPLAY3 on disk, you won't be able to SAVE your current display.

GRAPHICS EDITOR CONTROL KEYS

The Graphics Editor uses the keys listed below. When the program is complete, any other letter keys will be ignored. Color changes set by key I are only visible when put into effect with key C.

Key	Function
C	Block-color
D	Plot point
F	Flood-fill
G	Print/erase color grid
I	Set color combination (enter color code and follow with RETURN)
L	Draw line
M	Position marker cursor
O	Draw circle
P	Pattern-fill (follow with character number)
R	LOAD stored display
W	SAVE current display



TURTLE GRAPHICS

Suppose you were asked to write a program that would draw six squares, each rotated at an angle of 60 degrees to the next. How would you go about it? You could either draw out a design on a high-resolution grid and read off the coordinates needed, or, if you were mathematically-minded, you could write a program that calculated the coordinates itself. However, either method would take quite a time.

Shape programs in BASIC are often rather cumbersome. However, LOGO, another computer language developed in the 1970s, tackles the task of drawing shapes far more effectively. The part of the LOGO language that has become of greatest interest to micro owners concerns the "turtle" — an imaginary animal that produces complex shapes by following easy-to-use instructions. With the help of the routines in block M on the opposite page, you can generate some fascinating turtle graphics on your Commodore, using not LOGO but BASIC and machine code.

How turtle graphics work

With turtle graphics, the commands that control movement are like the ones you would use in giving directions. Here for example is a sequence of turtle instructions written in a BASIC framework:

```
FOR N=1 TO 4
FORWARD 50
RIGHT 90
NEXT N
```

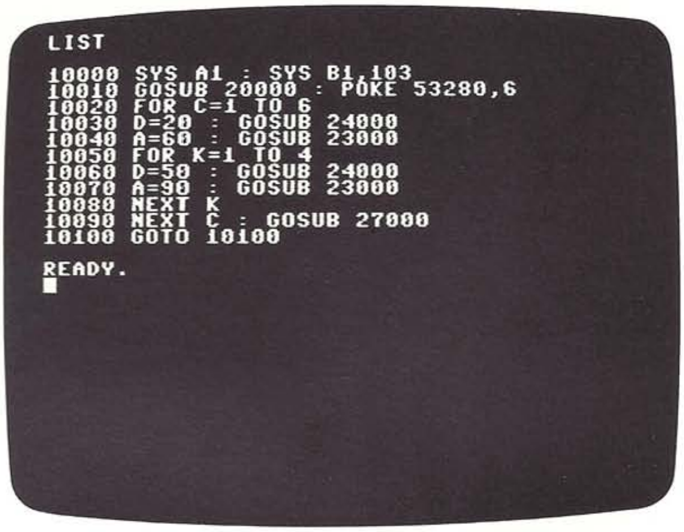
This would move the turtle around a square with sides of length 50 units. FORWARD makes the turtle move in the direction it is facing for the specified distance — in this case 50 units. RIGHT makes it turn right, in this case making an angle of 90 degrees. Because this is repeated four times the turtle traces out a square-shaped path.

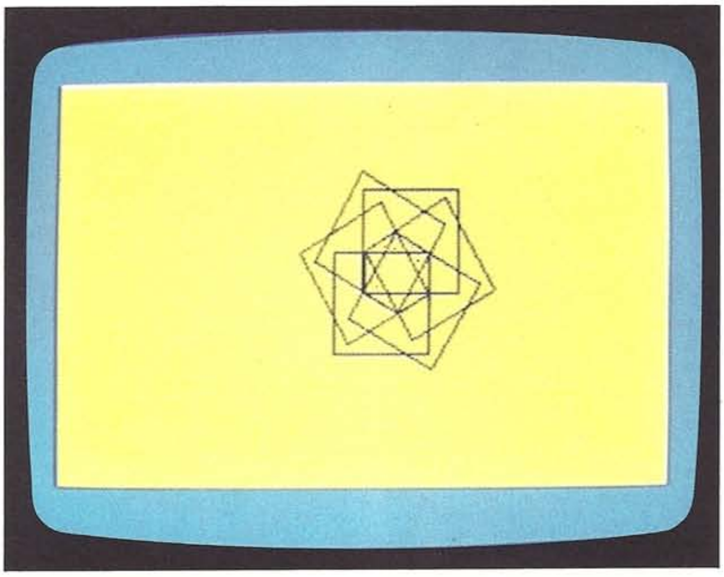
The orientation of any shape the turtle produces depends on its initial direction. So if you start it pointing vertically upward and then make it draw a square and finally turn through 60 degrees, you can simply repeat this set of instructions to produce a nest of squares. Programming shapes this way is easy.

Turtle routines for the Commodore

How can you make your Commodore understand an instruction like FORWARD or RIGHT? The answer, as you can see in the following program, is to call a BASIC subroutine instead, one which does exactly the same thing as the turtle command. Block M contains eight separate turtle routines, each written in ordinary BASIC, and each of which in turn calls one or more of the machine-code graphics routines.

NEST OF SQUARES WITH TURTLE GRAPHICS





What the routines do

In block M, the first turtle routine, starting at line 20000, sets the turtle's initial position at the center of the screen, the initial direction vertically upward, and selects a "pen down" option, so that drawing will begin as soon as the turtle is moved. All the angles in these routines are measured in degrees clockwise from the positive horizontal axis, and all the distances are measured in pixels.

The second turtle routine, starting at line 21000, draws the shape that represents the turtle itself. The erase routine from block F is used so that it can both draw and erase the turtle.

The third routine, at line 22000, turns the turtle left. To use it, you set the variable A to the angle through which you want to turn and then call the routine with GOSUB. For example, to turn the turtle left by 30 degrees, you would use the following line:

A=30 : GOSUB 22000

The routine to turn right starts at line 23000, and it is called in exactly the same way. The routine at line 24000 is the one for FORWARD. You just state the distance you want the turtle to travel by setting the variable D, and then by calling the routine with GOSUB 24000. For example, to move forward 50 pixels, you would use the line:

D=50 : GOSUB 24000

Similarly, to move the turtle backward, you would set D to the distance and call the routine at line 25000.

The final two routines are at lines 26000 and 27000. These carry out the pen-up and pen-down options. Neither of these two routines needs any parameters.

How to try the routines out

You can try out the turtle routines yourself. First, load blocks A-D and F, then key in your turtle program starting at line 10000, and finally add block M. Read the details in the block M panel carefully so that you understand how to operate the routines before you key them in. You will find a wide range of turtle graphics demonstration programs on the following eight pages.

The turtle routines must be keyed in or loaded after all the required machine-code routines and main program have been entered. If you do not follow this order, your turtle programs will not work.

TURTLE ROUTINES LISTING

```
20000 XI=160 : YI=100
20010 AI=270 : P=1
20020 :
21000 SYS F1,1
21010 CA=COS((AI-90)*π/180)
21020 SA=SIN((AI-90)*π/180)
21030 SYS C1,XI,YI
21040 SYS D1,-3*CA+XI,-3*SA+YI
21050 SYS D1,-6*SA+XI,6*CA+YI
21060 SYS D1,3*CA+XI,3*SA+YI

21070 SYS D1,XI,YI
21080 SYS F1,0 : RETURN
21090 :
22000 IF P=1 THEN GOSUB 21000
22010 AI=AI-A
22020 IF AI<0 THEN AI=AI+360
22030 IF P=1 THEN GOSUB 21000
22040 RETURN
22050 :
23000 IF P=1 THEN GOSUB 21000

23010 AI=AI+A
23020 IF AI>360 THEN AI=AI-360
23030 IF P=1 THEN GOSUB 21000
23040 RETURN
23050 :
24000 XO=D*COS(AI*π/180)+XI
24010 YO=D*SIN(AI*π/180)+YI
24020 IF P=1 THEN 24040
24030 XI=XO : YI=YO : RETURN
24040 GOSUB 21000

24050 SYS C1,XI,YI
24060 XI=XO : YI=YO
24070 SYS D1,XI,YI
24080 GOTO 21000
24090 :
25000 D=-D : GOTO 24000
25010 :
26000 IF P=1 THEN RETURN
26010 P=1 : GOTO 21000
26020 :
27000 IF P=0 THEN RETURN
27010 P=0 : GOTO 21000
```

BLOCK M

TURTLE GRAPHICS routines

What the routines do

The routines in this block enable the Commodore to set up a mobile turtle, and then move it through relative angles and distances, rather than through absolute ones. For example, the turtle can be programmed to move forward 50 pixels and then turn 90 degrees to its right regardless of the position and direction it started in. This block does not itself program machine code. The eight subroutines in it are written in simple BASIC using GOSUB and RETURN. However, each of these BASIC subroutines works by calling one or more of the machine-code graphics routines in blocks A-D and F.

How to use the turtle graphics routines

To use turtle graphics, first load graphics routine blocks A-D and F (or the complete set if you have it). Then add your turtle program giving it line numbers between 10000 and 19999, and then finally add block M. It is very important that you assemble a turtle program in this order. All turtle programs must begin either with the turtle initialization routine, if you want the turtle to start in the middle of the screen, or the turtle shape routine, if you want to make it start somewhere else.

SYNTAX AND PARAMETERS

To set up or move the turtle in a program, first decide which turtle routine you need. Then key in an angle or distances if the routine requires them, separated by colons, then GOSUB and the line number which calls the routine.

Turtle intialization GOSUB 20000
Turtle shape XI= : YI= : AI= : P= : GOSUB 21000
Turn left A= : GOSUB 22000
Turn right A= : GOSUB 23000
Forward D= : GOSUB 24000
Backward D= : GOSUB 25000
Pen up GOSUB 26000
Pen down GOSUB 27000

XI,YI	(**Turtle initialization** only) Initial turtle horizontal and vertical coordinates (ranges 0-319 and 0-199). Key in XI= then the horizontal coordinate, and YI= then the vertical coordinate.
AI	(**Turtle initialization** only) Initial angle at which the turtle is to point, measured in degrees clockwise from the positive horizontal axis (range 0-360). Key in AI= followed by the angle.
P	(**Turtle initialization** only) Pen up (0) or pen down (1)
A	(**Turn left** and **turn right** only) Angle through which the turtle is to turn, measured in degrees (no range limit). Key in A= followed by the angle.
D	(**Forward** and **backward** only) Distance in pixels through which the turtle is to travel (no range limit, although some values may produce off-screen displays). Key in D= followed by the distance.

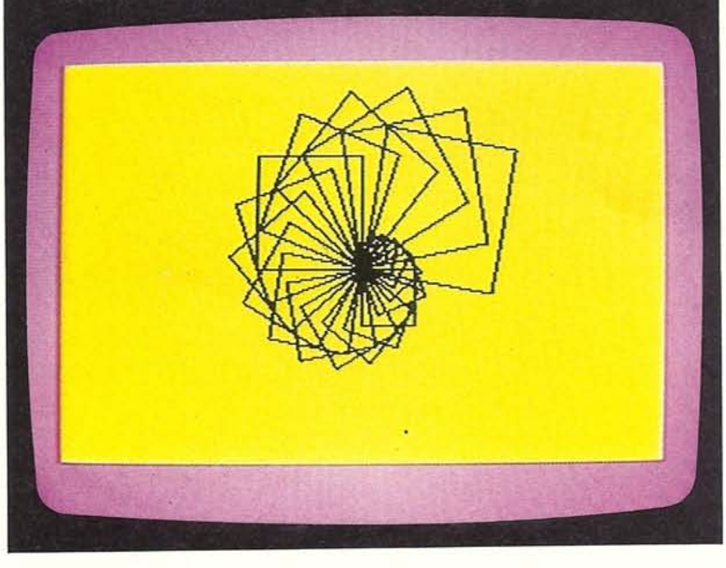
TURTLE SHAPES 1

So far, you have seen how a turtle graphics program can be used to move through fixed angles and distances. But it's possible to write a turtle graphics program so that the turtle's movements are controlled by variables instead of by set figures. The program can then start by asking for values for these variables and so produce results which you can specify from the keyboard.

The first program on these two pages works in this way. It draws a nest of 24 squares in a way very similar to the one you saw on page 50. However, this time the results are much more interesting. The size of successive squares changes, and you can control the initial length of the sides and also the STEP size by which they increase each time a square is drawn. To try the program out, make sure that you have routine blocks A-D and F in memory, add the program listing, and finally add block M. After typing RUN, key in two parameters and watch the display unfold.

VARIABLE TURTLE SQUARES (parameters 1,3)

```
LIST

10000 PRINT CHR$(147)
10010 INPUT "D,S";D,S
10020 SYS A1 : SYS B1,7
10030 POKE 53280,4 : GOSUB 20000
10040 FOR C=1 TO 24 : A=90
10050 FOR K=1 TO 4
10060 GOSUB 24000
10070 GOSUB 22000
10080 NEXT K
10090 A=20 : D=D+S
10100 GOSUB 23000
10110 NEXT C
10120 GOSUB 27000
10130 GOTO 10130

READY.
█
```

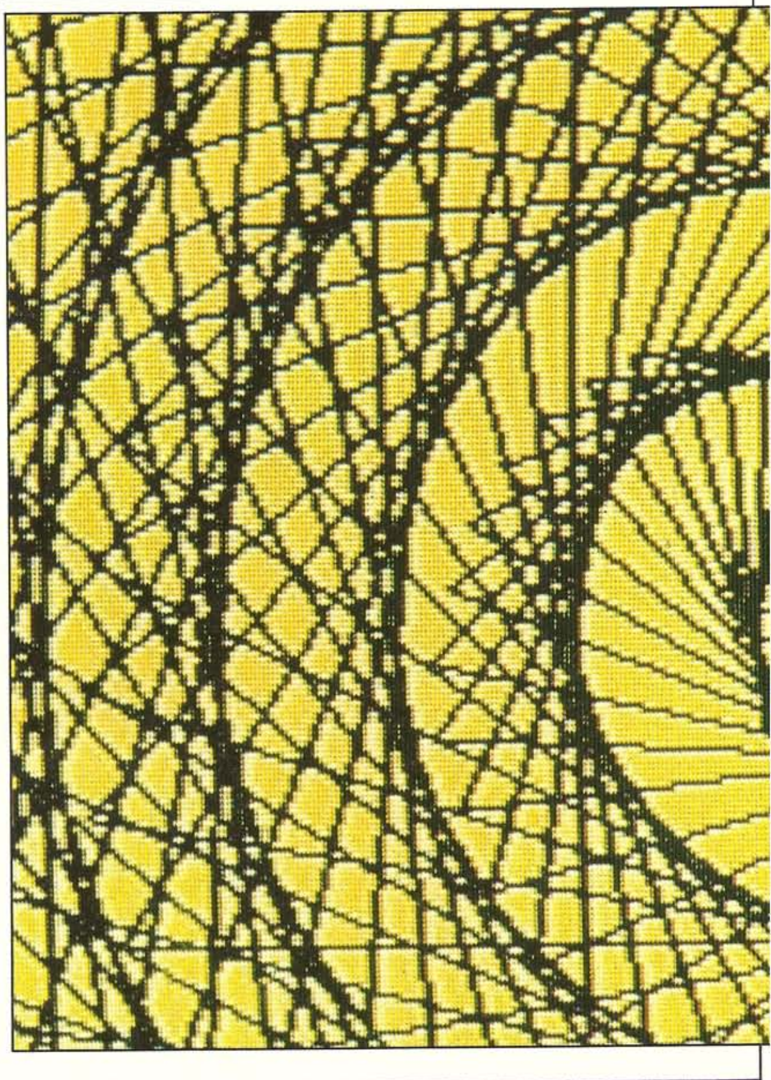


Multi-shape programs

Some turtle graphics programs will produce a huge range of quite different results. The next program is a good example.

MULTI-SHAPE PROGRAM

```
LIST

10000 PRINT CHR$(147)
10010 INPUT "A,D";A,D
10020 SYS A1 : SYS B1,7
10030 GOSUB 20000 : POKE 53280,5
10040 FOR C=1 TO 200
10050 GOSUB 24000
10060 GOSUB 23000
10070 D=D+3
10080 NEXT C
10090 GOSUB 27000
10100 GOTO 10100

READY.
█
```



The program lets you INPUT two parameters. These are the angle through which the turtle turns after it has drawn a line, and the increase in the forward distance with each successive line. The program is controlled by a loop so that the turtle stops after 60 lines.

The small displays on the right show the effect of three different pairs of parameters. The photographs show the displays before the program has reached completion. The large display below is the complete display produced by using the parameters 123 and 1.

With this program, it's very difficult to predict how a shape will turn out, but this is the fun of turtle graphics — you can create some fascinating displays by experimenting. The three displays below show the results you should see if you try entering parameters 123 and 1, 144 and 5, and 170 and 10.

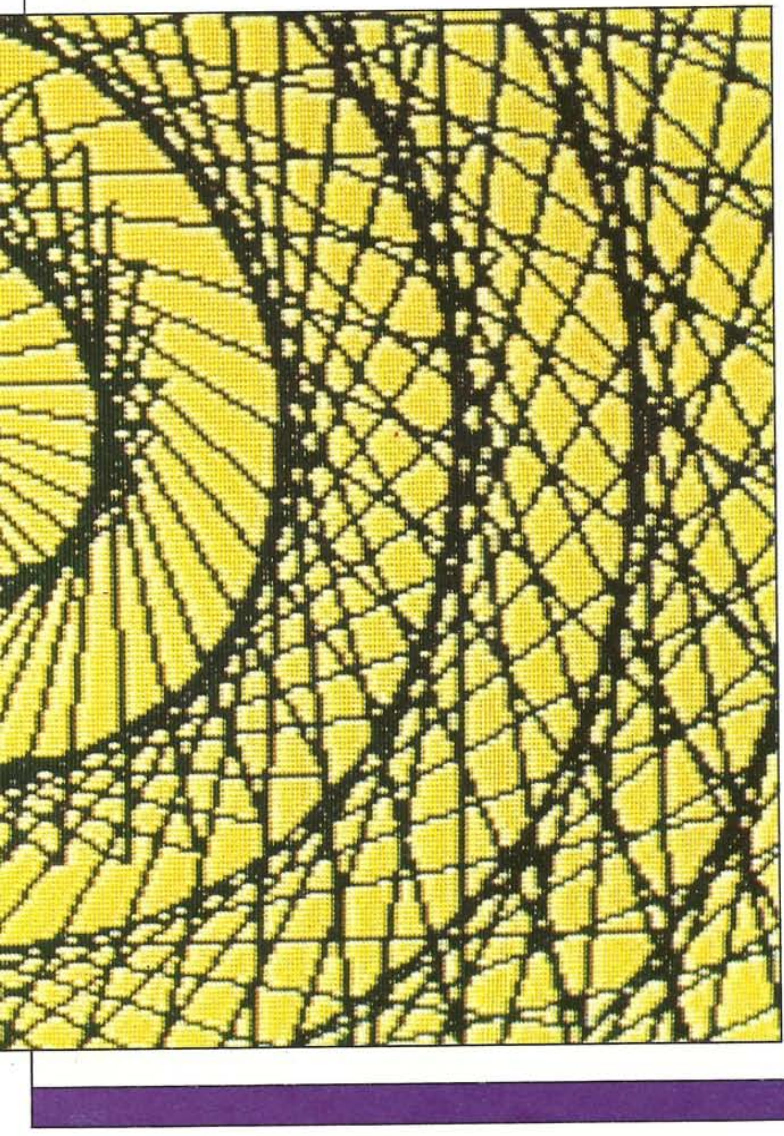
MULTI-SHAPE PROGRAM

04:15

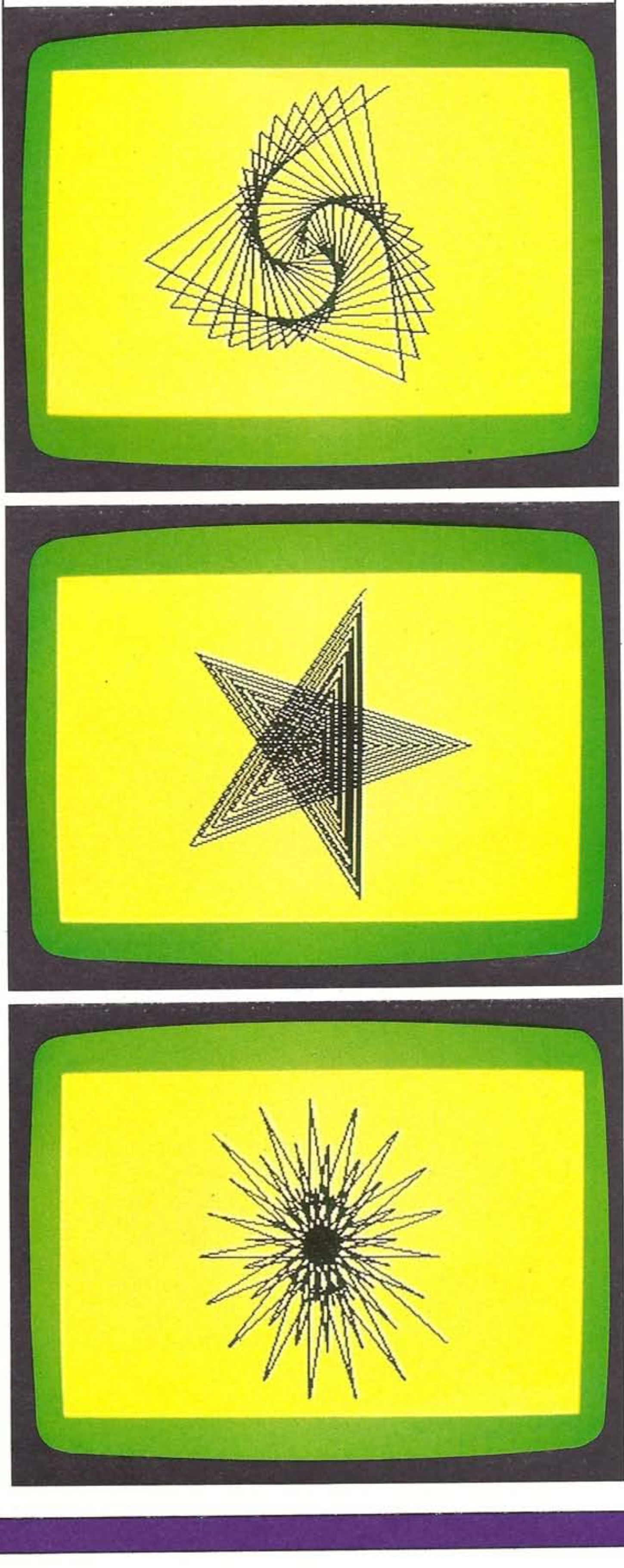
How the program works
The program asks for two parameters. The angle through which the turtle turns remains constant, while the distance it travels between turns increases.
Lines 10050 and 10060 make the turtle move forward then turn right.
Line 10070 increases the forward distance that the turtle moves on every loop.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
B Clear-and-color	13
C Plot	15
D Draw	17
F Erase	25



MULTI-SHAPE DISPLAYS



TURTLE SHAPES 2

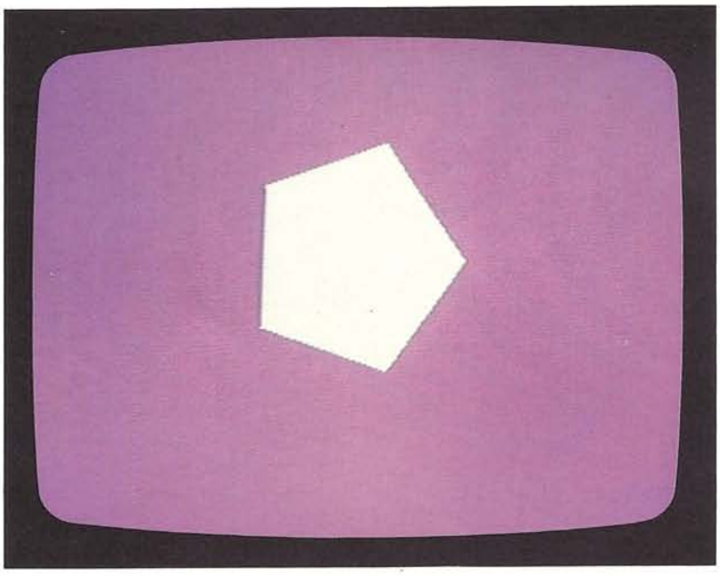
There are several kinds of objects that turtle graphics are particularly good at displaying. The simplest are "closed" shapes — ones that start and finish at the same point. You see this with the first of the following programs, which draws polygons.

This program asks you to input two parameters and then draws a closed regular shape and fills it. The first of the two values you need to enter specifies the length of the perimeter, that is the total length of the edges of the shape. The second value specifies the number of sides the shape will have.

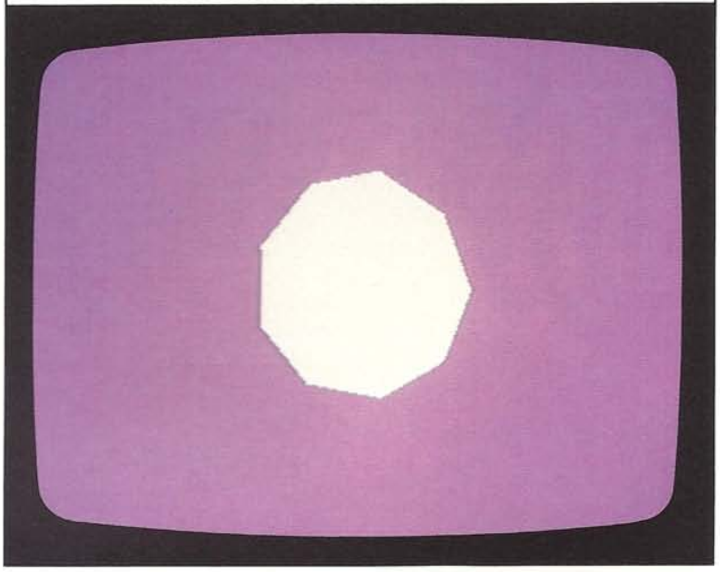
Each of the following three displays uses a perimeter value of 350 with numbers of sides 5, 9 and 15 respectively. To run the program, load routine blocks A-D and F, add the listing below, and then add the turtle routines in block M.

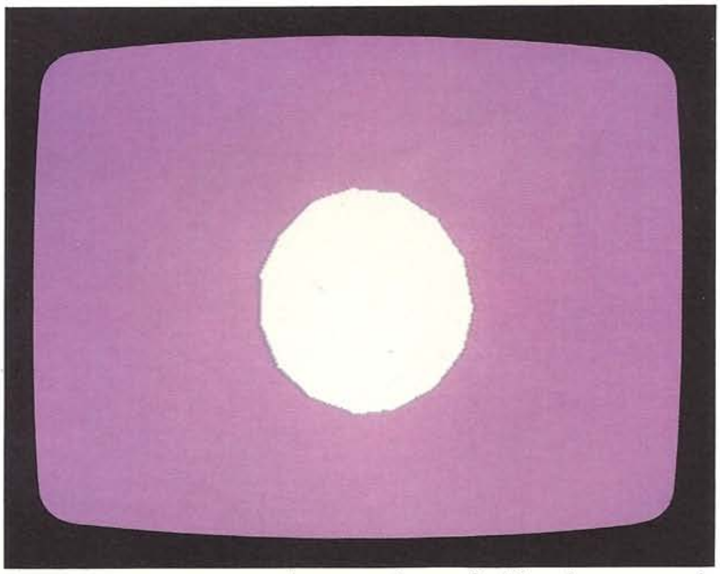
Remember that you must add the program and the machine-code and turtle routines in the correct order.

POLYGON PROGRAM



POLYGON PROGRAM DISPLAYS





As you can see, as the number of sides increases it becomes more and more difficult to distinguish the shape produced from a circle. In fact, the high-resolution circle and arc routines on page 21 draw shapes in just this way — as a sequence of short lines.

One final point to notice about this program is that it does not use the default initial position for the turtle. It does not call the turtle initialization routine starting at line 20000 but instead sets up its own initial values in line 10040 so that the turtle starts to the left of the screen's center.

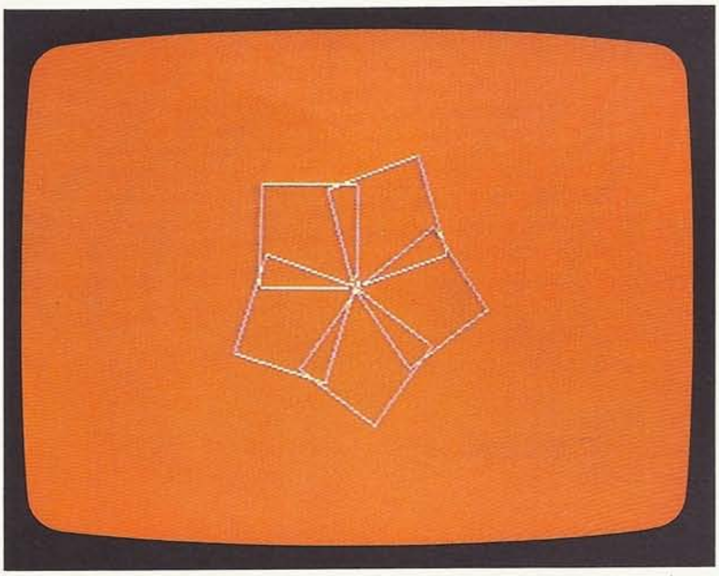
Using multiple loops

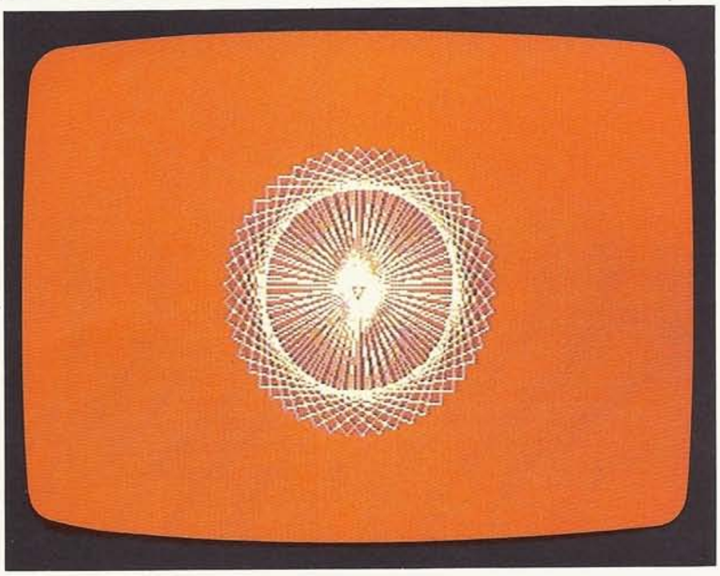
Now you know how to display a closed shape, there are several ways in which a series of closed shapes can be combined to produce interesting effects. For example, you can display a variable number of shapes so that they all touch at one corner, as the next program shows.

Again, this program needs to be added to blocks A-D and F, and then followed by block M. The displays below shows the results you should see with parameters 5 and then 50.

MULTIPLE CLOSED SHAPES PROGRAM

```
LIST

10000 PRINT CHR$(147)
10010 INPUT "N";N
10020 SYS A1 : SYS B1,18
10030 POKE 53280,2 : GOSUB 20000
10040 D=50
10050 FOR C=1 TO N : A=90
10060 FOR K=1 TO 4
10070 GOSUB 24000
10080 GOSUB 22000
10090 NEXT K
10100 A=360/N
10110 GOSUB 23000
10120 NEXT C
10130 GOTO 10130

READY.
```

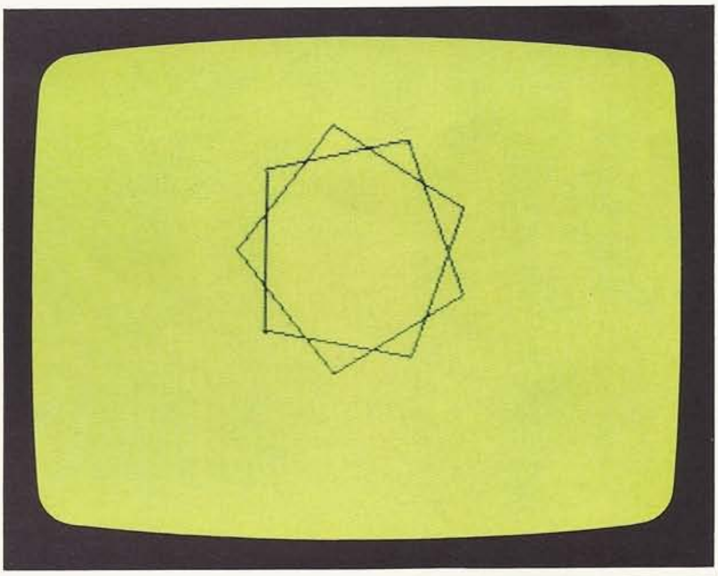


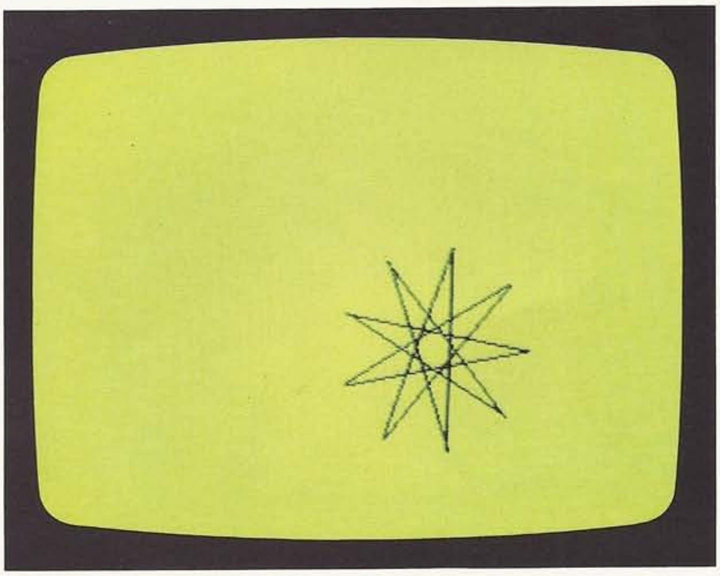


As a final example of closed shapes, try the following program. The displays below are produced by parameter values of 80,80 and 100,160. As usual, it needs routine blocks A-D and F.

CLOSED SHAPES PROGRAM

```
LIST

10000 PRINT CHR$(147)
10010 INPUT "D,A";D,A
10020 SYS A1 : SYS B1,5
10030 POKE 53280,5
10040 XI=110 : YI=120 : AI=270
10050 P=1 : GOSUB 21000
10060 L=0
10070 L=L+1
10080 M=360*L/A
10090 N=INT(M)
10100 IF ABS(N-M)>0.001 THEN 10070
10110 FOR C=1 TO N
10120 GOSUB 24000
10130 GOSUB 23000
10140 NEXT C
10150 GOSUB 27000
10160 GOTO 10160

READY.
```





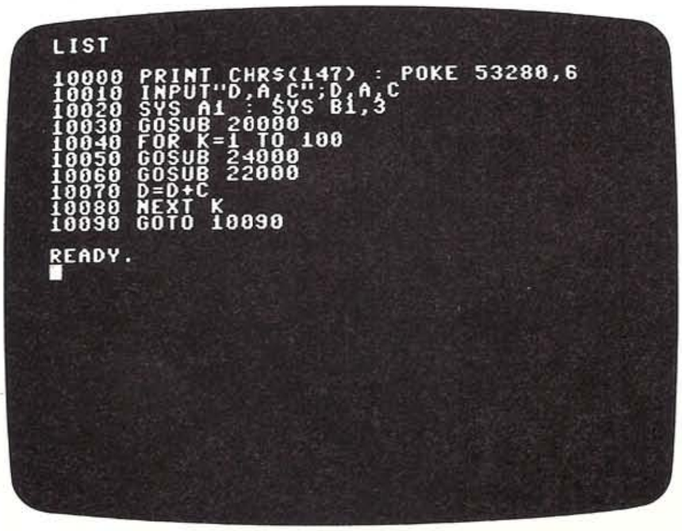
TURTLE SPIRALS

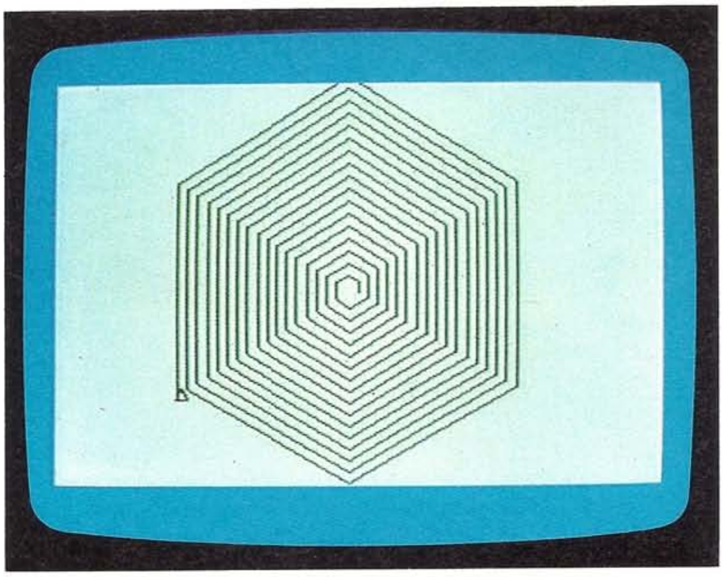
One display which turtle graphics makes simple is the spiral. In general, to draw a spiral you need to move the turtle forward repeatedly by some increasing amount, and rotate it through a fixed angle.

You can develop this approach with turtle graphics so that several parameters are used. The following program shows you one way of doing this — it draws a spiral which you can specify. When the program is run, it asks you to enter three parameter values. These are, in order, the length of the initial move forward, the angle through which to turn, and the step size for the increase in length. The following sequence of displays was produced with this program. The two small displays are produced by parameters 5,60,1 and 5,65,1.

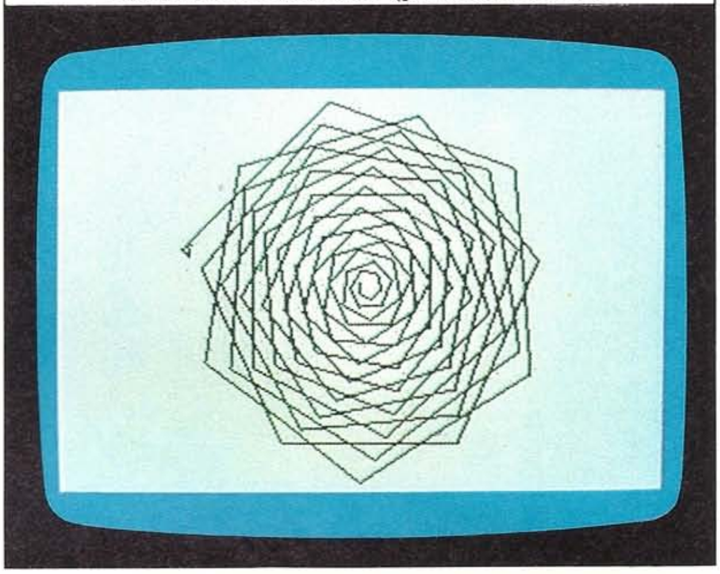
To test the program, load the graphics routines in blocks A-D and F, add the listing below, and then add the turtle routines in block M. Again, remember that the order in which you do this must be right.

POLYSPIRAL PROGRAM


```
LIST

10000 PRINT CHR$(147) : POKE 53280,6
10010 INPUT"D,A,C";D,A,C
10020 SYS A1 : SYS B1,3
10030 GOSUB 20000
10040 FOR K=1 TO 100
10050 GOSUB 24000
10060 GOSUB 22000
10070 D=D+C
10080 NEXT K
10090 GOTO 10090

READY.
```



POLYSPIRAL DISPLAY (parameters 5,65,1)



POLYSPIRAL PROGRAM

01:40

How the program works
The program accepts three parameter values — a distance through which the turtle will move forward, an angle through which it will then turn, and an increase which is added to the distance after each turn. The large display here is created by parameters 5,61,1.

Line 10010
asks for the three parameters.

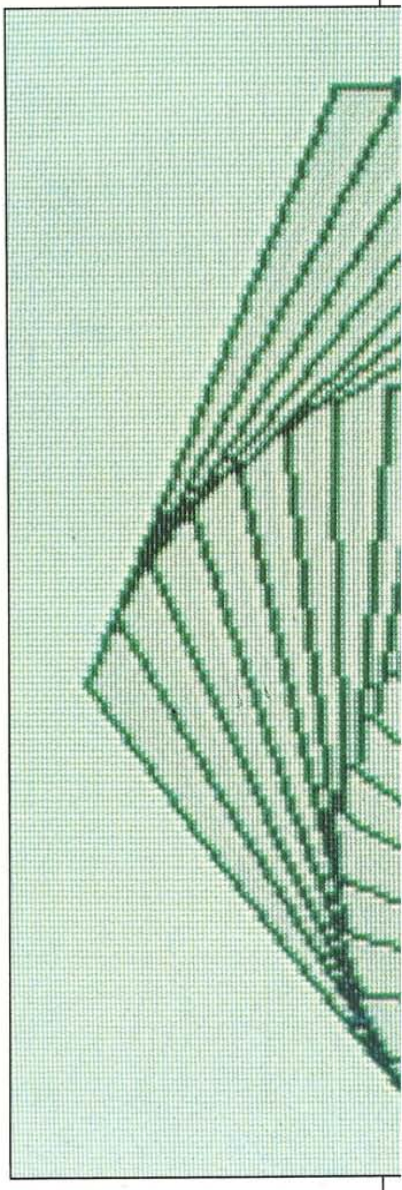
Line 10030
initializes the turtle, making it start at the center of the screen.

Lines 10040-10080
form a loop which makes the turtle move forward and turn.

Line 10070 increases the distance by the number selected at the beginning of the program.

ROUTINES USED BY THIS PROGRAM

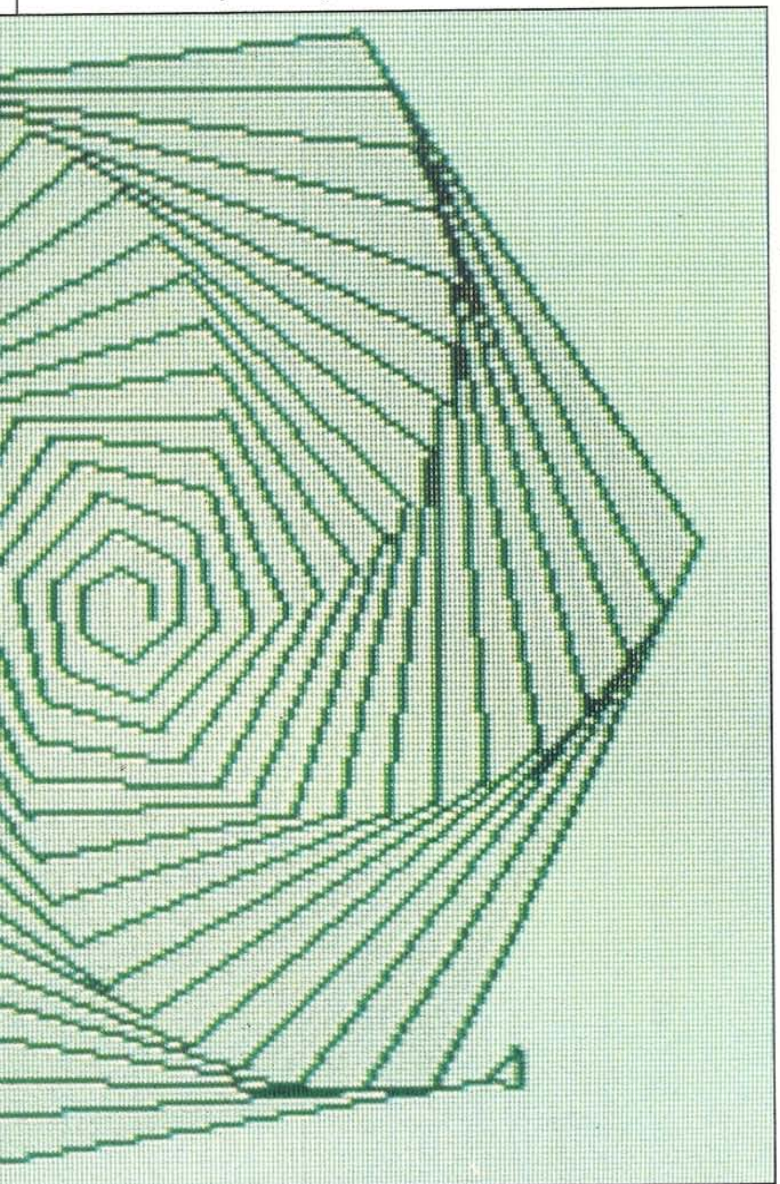
Block Routine(s)	Page
A High-resolution	11
B Clear-and-color	13
C Plot	15
D Draw	17
F Erase	25



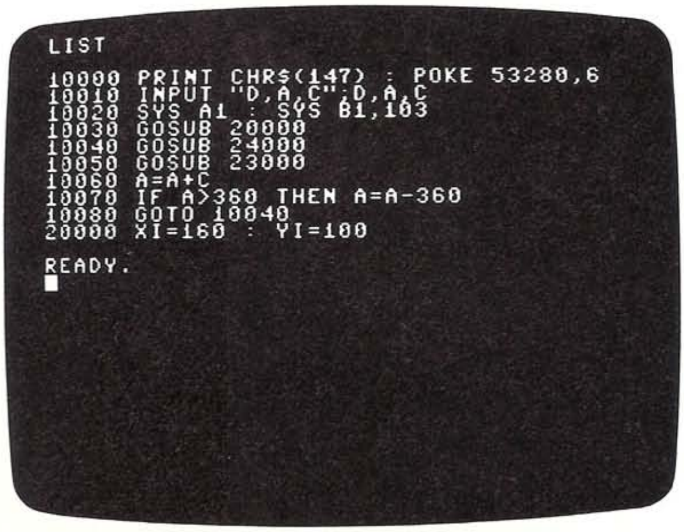
The next program also creates a family of spiral shapes and again requires you to specify three parameters. However, this time the results are very different. The parameters for this program are the length of the turtle movements, the initial angle through which the turtle rotates, and the amount by which this angle is increased on each pass through the loop.

Increasing the angle instead of the distance completely alters the display. This simple program produces a family of spiral patterns whose shapes are very difficult to predict from the input parameters. The curves are best described as "inward spirals".

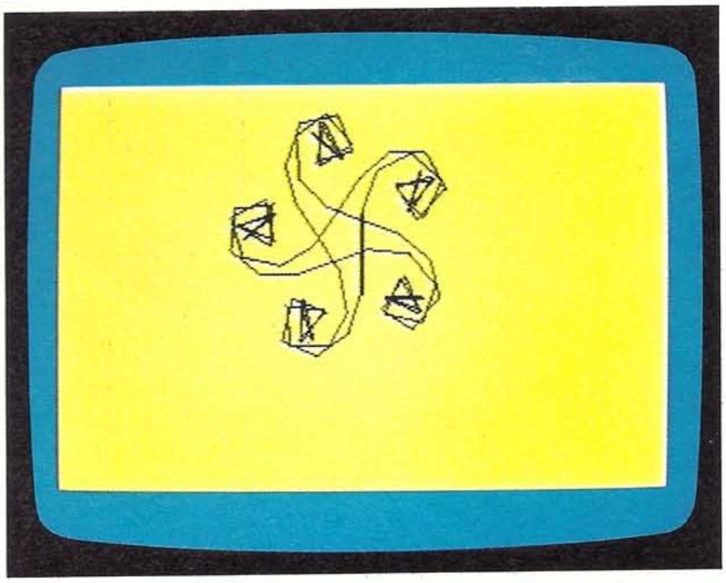
You will find that many combinations of parameters make the turtle wander off the screen after a few turns. You may have to wait hours to find out if it ever returns! The trick is to find an angle and an angle increase value which together make the turtle track back over a small area. If you do find a combination that keeps the turtle within the screen boundaries, you will probably see a pattern like the ones below. These shapes both have rotational symmetry. The first is produced by parameters 20,2,20 and the second by 10,1,78. The first has five-fold symmetry and the second three-fold symmetry. It produces an endless variety of simple or complex patterns on this theme.

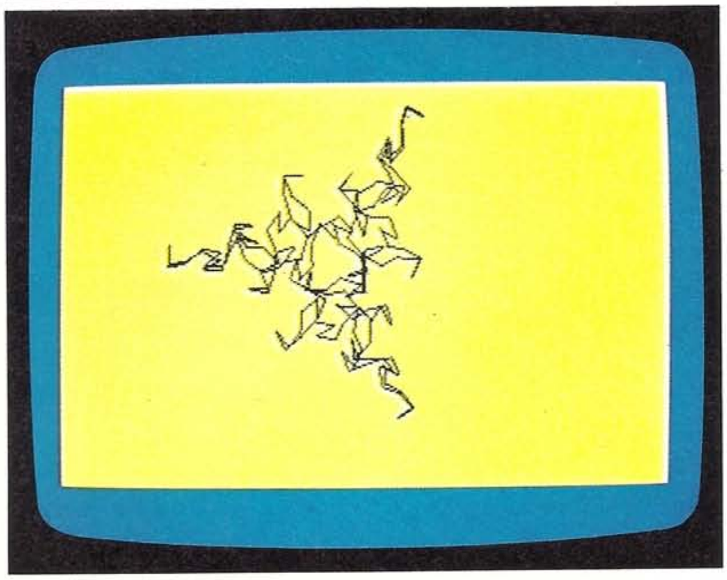


INWARD SPIRAL PROGRAM



```
LIST

10000 PRINT CHR$(147) : POKE 53280,6
10010 INPUT "D,A,C";D,A,C
10020 SYS A1 : SYS B1,103
10030 GOSUB 20000
10040 GOSUB 24000
10050 GOSUB 23000
10060 A=A+C
10070 IF A>360 THEN A=A-360
10080 GOTO 10040
20000 XI=160 : YI=100

READY.
```





TURTLE PATTERNS

One technique that can be very useful in producing turtle displays is that of creating patterns inside subroutines. What this means is that you develop a shape or pattern and write it so that it can be called as a subroutine. This shape can be as simple or as complicated as you like — as long as it can be produced by the turtle routines.

Once you have created the subroutine, you can make the turtle move across the screen, drawing the shape at different orientations or even different sizes as it goes. Because you can nest subroutines by having GOSUBs within other GOSUBs, you can draw patterns within patterns, so that a fairly straightforward original design ends up by producing a very detailed pattern on the screen.

The three programs on these two pages are all related, being based on the same simple shape. The first program produces just the shape. The second program repeats the shape four times in a specific way, and the third program repeats the whole of the shape created by the second, again in a specific way.

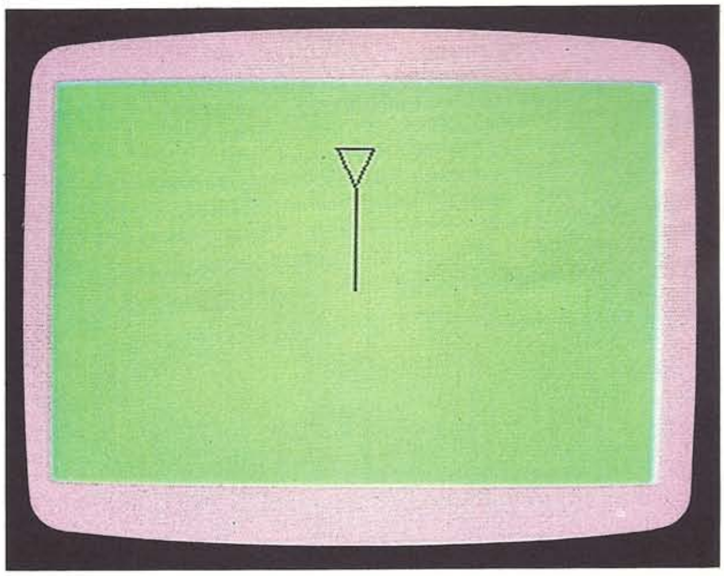
To try out the first program, load routine blocks A-D and F, key in the program and then add block M. You can then adapt the main program as shown here.

How to repeat a simple shape

The first program draws the fundamental shape. It's a design like a games bat. The part of the program that draws the bat is contained in a subroutine starting at line 15000. It is now fairly straightforward to write another subroutine that builds up a simple square pattern using the previously created bat subroutine. This gives you the second program. Finally, you can write a third program that calls this new subroutine several times to repeat the design. The result is a complex new display.

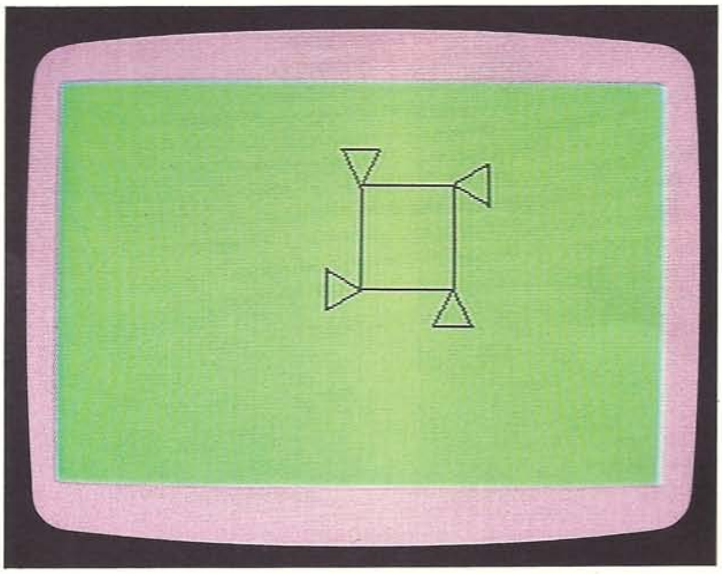
SIMPLE TURTLE DESIGN

```
LIST

10000 SYS A1 : SYS B1,5
10010 GOSUB 20000 : POKE 53280,4
10020 GOSUB 15000
10030 GOSUB 27000 : GOTO 10030
14999 :
15000 D=50 : GOSUB 24000
15010 A=30 : GOSUB 22000
15020 FOR C=1 TO 3
15030 D=20 : GOSUB 24000
15040 A=120 : GOSUB 23000
15050 NEXT C
15060 RETURN
20000 XI=160 : YI=100

READY.
```



FIRST TURTLE PATTERN

```
LIST

10000 SYS A1 : SYS B1,5
10010 GOSUB 20000 : POKE 53280,4
10020 GOSUB 12000
10030 GOSUB 27000 : GOTO 10030
11999 :
12000 FOR K=1 TO 4
12010 GOSUB 15000
12020 A=120 : GOSUB 23000
12030 NEXT K : RETURN
14999 :
15000 D=50 : GOSUB 24000
15010 A=30 : GOSUB 22000
15020 FOR C=1 TO 3
15030 D=20 : GOSUB 24000
15040 A=120 : GOSUB 23000
15050 NEXT C
15060 RETURN
20000 XI=160 : YI=100

READY.
```



FINAL TURTLE PATTERN

```
10000 SYS A1 : SYS B1,5
10010 GOSUB 20000 : POKE 53280,4
10020 FOR KK=1 TO 8
10030 GOSUB 12000
10040 A=135 : GOSUB 23000
10050 NEXT KK
10060 GOSUB 27000 : GOTO 10060
11999 :
12000 FOR K=1 TO 4
12010 GOSUB 15000
12020 A=120 : GOSUB 23000
12030 NEXT K : RETURN
14999 :
15000 D=50 : GOSUB 24000
15010 A=30 : GOSUB 22000
15020 FOR C=1 TO 3
15030 D=20 : GOSUB 24000
15040 A=120 : GOSUB 23000
15050 NEXT C
15060 RETURN
20000 XI=160 : YI=100

READY.
```

Writing shapes as subroutines

You can use subroutines to build up a library of shapes for use in turtle graphics. By nesting the subroutines, you can add shapes together, although there is a limit to the depth of nesting that the Commodore can handle.

TURTLE PATTERN PROGRAM

02:25

How the program works
The program produces a simple design, repeats it four times in a square, and then repeats this square eight times.

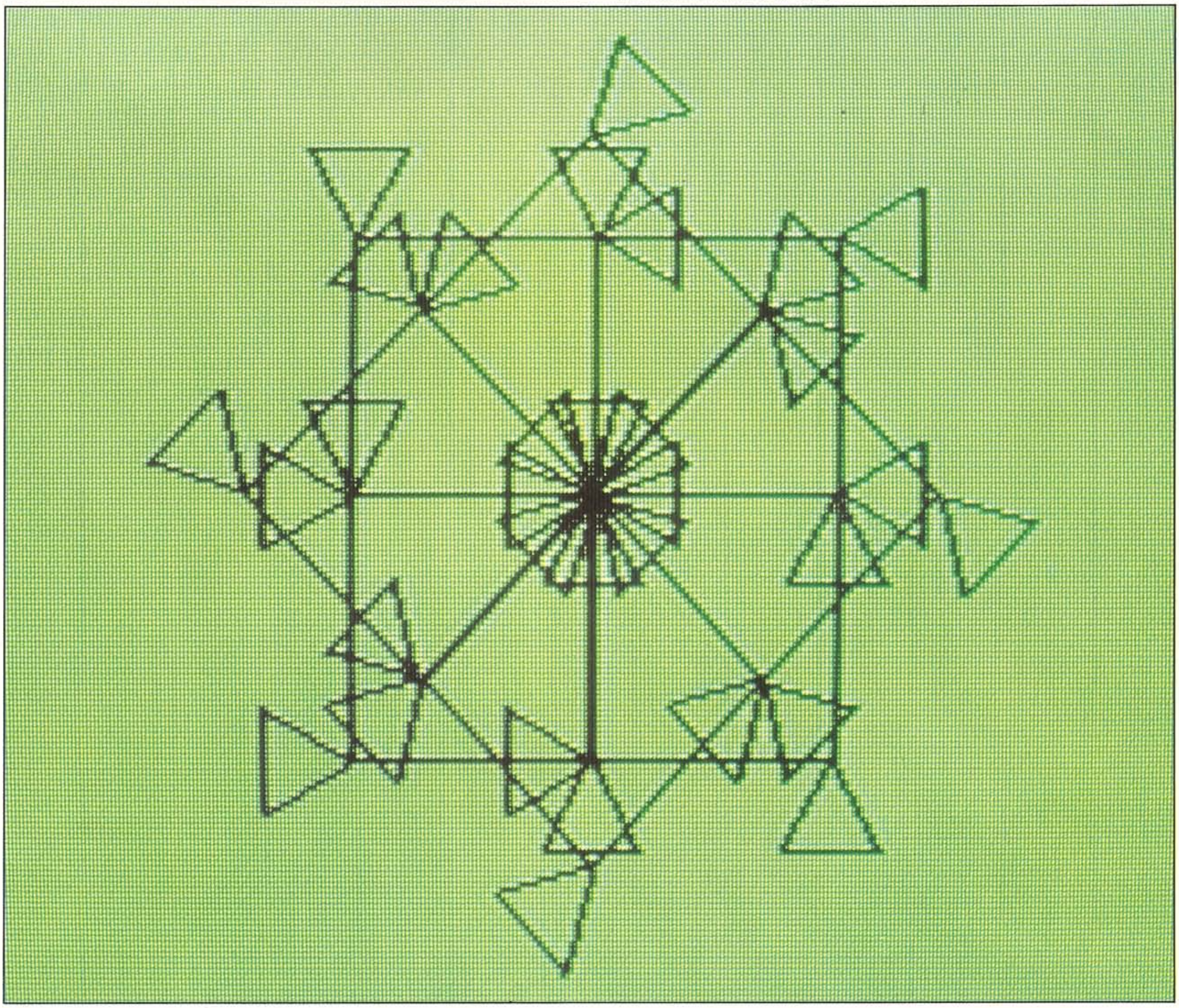
Lines 10020-10050 call the subroutine at line 12000, moving the turtle on each time.

Lines 12000-12030 call the subroutine at line 15000.

Lines 15000-15060 produce the bat.

ROUTINES USED BY THIS PROGRAM

Block Routine(s)	Page
A High-resolution	11
B Clear-and-color	13
C Plot	15
D Draw	17
F Erase	25



HIGH-RESOLUTION GRID

The two grids on this page allow you to work out coordinates for use with the machine-code graphics routines, and also enable you to work out row bit totals for use with the define-character routine.

Screen grid

The main grid below shows the coordinates of points that lie on the visible screen area. When you use the graphics routines, you are not limited to coordinates within this range. Most of the routines will accept parameter values that extend outside this area, either positive or negative, as long as the total range of co-ordinates does not exceed the Commodore's integer handling capacity. In reality, this restricts most co-ordinates to a total range of about 32,000.

This capability means that you can produce designs that are only partially on-screen. For example, if you key in SYS E1,160,800,700 you will see a very shallow arc on the screen — the only visible part of a complete circle which the computer has calculated.

Although invisible parts of displays are supported in the computer's memory, the scroll routine cannot be used to move hidden parts of a display.

Character grid

You can use this grid to work out row bit totals for inclusion with the define-character routine. Pencil in your design and then add up the row totals.

SINGLE CHARACTER GRID

Bit values

Row bit totals	1	2	4	8	16	32	64	128
0								
1								
2								
3								
4								
5								
6								
7								

HIGH-RESOLUTION SCREEN GRID

	0	8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152	160	168	176	184	192	200	208	216	224	232	240	248	256	264	272	280	288	296	304	312
0																																								
8																																								
16																																								
24																																								
32																																								
40																																								
48																																								
56																																								
64																																								
72																																								
80																																								
88																																								
96																																								
104																																								
112																																								
120																																								
128																																								
136																																								
144																																								
152																																								
160																																								
168																																								
176																																								
184																																								
192																																								

ERROR TRAPPING

Even if you are an experienced micro user, it is easy to make mistakes when keying in programs. If you have been very careful (and lucky) when using this book, you might have managed to key in all the machine-code DATA numbers without making any mistakes. However, the chances are that one or two simple errors will have crept into your copy of the routines. Depending on where these errors occur, you may not experience any problems for quite a time. Then some time later, you will call on a routine which you have not previously used, and an error in it will cause your program to go wrong in some way—perhaps giving a different problem every time you run it. There are no error reports generated when using machine code, so how do you know where you have gone wrong?

The Checksum program

If you transfer information from one medium to another, you need some way of checking that the transfer has been accurate. A simple but effective way of doing this with numerical information is to add together all the numbers to produce a "checksum". You can then compare the two checksums—before and after transfer.

The program at the bottom of this page lets you check your routines using this method. It will go to a specified machine-code routine and add together all the DATA numbers it contains to produce a checksum which it then compares with its own built-in list.

How to use the Checksum program

To use this program, you must first enter block A and run it, and then merge the Checksum program with this block. Now if you run the combined program and block A, it will ask you the question A-L?, requesting you to name a routine. If you enter A, and should confirm that block A is correct by displaying DATA OK. If block A is incorrect, it should display DATA ERROR. Any other message output or printed by BASIC as a result of an error indicates that one or more of the DATA statements in block A is at fault. You now need to find and correct the error. When your first block of machine code passes the checksum test, it is then safe to go on and use the program to test the other blocks. You must check block A first because the Checksum program uses the restore routine. You will also find the merge routine useful for adding the Checksum program onto the end of the routines you want to test.

Having combined and tested block A with the program, you can then test any other block with it. Just load the block or blocks you want to test, and then merge the Checksum listing with them. Any block which is incorrect will give a DATA ERROR report when the Checksum tests it. The chances of an incorrect listing coincidentally producing the correct checksum total is negligible, so, if you do get a DATA ERROR message, it is almost certain that your listing is not correct.

How to avoid errors

When you are keying in the routines, it is a good idea to SAVE each routine before you run it for the first time, rather than testing it first and then storing it. Because the routines produce machine code, errors can make them disrupt themselves. For example, this may lead to a listing becoming corrupted by other characters that are thrown up on the screen, locking the computer into a state that only disconnection will break.

Secondly, if you key in the routines but they seem to be ineffective, try jumping over the first line of the first routine with GOTO. The routines may be ignored if their first DATA number accidentally happens to be in memory already.

CHECKSUM PROGRAM

```
10000 PRINT CHR$(147)
10010 INPUT "A-L";A$
10020 IF A$<"A" OR A$>"L" THEN 10000
10030 A=ASC(A$)-64
10040 ON A GOTO 10080,10100,10120
10050 ON A-3 GOTO 10140,10160,10180
10060 ON A-6 GOTO 10200,10220,10240
10070 ON A-9 GOTO 10260,10280,10300
10080 L=0 : C=22319 : B=253
10090 GOSUB 10360 : GOTO 10320
10100 L=650 : C=34257 : B=275
10110 GOSUB 10360 : GOTO 10320
10120 L=1240 : C=8606 : B=76
10130 GOSUB 10360 : GOTO 10320
10140 L=1540 : C=46426 : B=389
10150 GOSUB 10360 : GOTO 10320
10160 L=2540 : C=39981 : B=358
10170 GOSUB 10360 : GOTO 10320
10180 L=3240 : C=2067 : B=19
10190 GOSUB 10360 : GOTO 10320
10200 L=3540 : C=47612 : B=393

READY.
```

```
LIST 10210-

10210 GOSUB 10360 : GOTO 10320
10220 L=4240 : C=26764 : B=203
10230 GOSUB 10360 : GOTO 10320
10240 L=4640 : C=6326 : B=50
10250 GOSUB 10360 : GOTO 10320
10260 L=4840 : C=27057 : B=217
10270 GOSUB 10360 : GOTO 10320
10280 L=5240 : C=8484 : B=63
10290 GOSUB 10360 : GOTO 10320
10300 L=5440 : C=65736 : B=490
10310 GOSUB 10360
10320 IF T=C THEN 10340
10330 PRINT "DATA ERROR" : END
10340 PRINT "DATA OK" : PRINT
10350 GOTO 10010
10360 RESTORE : IF L<>0 THEN SYS A3,L
10370 T=0 : FOR K=1 TO B : READ D
10380 T=T+D : NEXT K : RETURN
READY.
```

ROUTINES CHECKLIST

The main table on these two pages gives you details of all the machine-code graphics routines that are featured in this book. Block M, the turtle graphics routine block, does not appear here because it only uses ordinary BASIC. The chart enables you to look up the information you need to use the machine-code routines in your own programs.

Syntax

When you are using any routine, it is important to use the correct syntax. Each routine is called from a main BASIC program by the command SYS, followed by the variable which identifies the routine (B1, H2 and so on) and then by parameters, if the routine requires them. Remember to separate all this information by commas as shown in the chart.

Because the routines are activated by variables which stand for five-figure memory addresses, it is very important that you do not use the same variables to represent any other values in your programs. For example, calling two sets of coordinates A1,B1 and A2,B2 could make a program crash. This is because these variables are already used by routine blocks A and B to signify the addresses of four routines.

Block	Page	Title	Syntax	Parameters	
A	11	High-resolution	SYS A1	None	
A	11	Low-resolution	SYS A2	None	
A	11	Restore	SYS A3,N	N	program line number
A	11	Rescue	SYS 49271	None	
A	11	Merge	SYS 49297, A$ [,8]	A$	filename
B	13	Clear-and-color	SYS B1,C	C	color code
B	13	Block-color	SYS B2,X,Y,C	X,Y C	block coordinates color code
C	15	Plot	SYS C1,X,Y	X,Y	point coordinates
D	17	Draw	SYS D1,X,Y	X,Y	line-end coordinates
E	21	Circle	SYS E1,X,Y,R	X,Y R	center coordinates radius length
E	21	Arc	SYS E2,X,Y,R,P,Q	X,Y R P Q	center coordinates radius length starting angle finishing angle
F	25	Erase	SYS F1,N	N	off/on
G	27	Flood-fill	SYS G1,X,Y	X,Y	start coordinates
H	31	ROM-copy	SYS H1	None	
H	31	Text	SYS H2,X,Y,A$	X,Y A$	start coordinates text
I	33	Define-character	SYS 11,C,X1-X8	C X1-X8	character code row bit totals
J	35	Pattern-fill	SYS J1,X,Y,C,	X,Y C	start coordinates filling character code
K	39	Block-copy	SYS K1,X,Y,A,B	X,Y A,B	origin block coordinates destination block coordinates
L	43	Scroll	SYS L1,D,C,	D C	direction color code

Parameters

The chart shows what parameters need to be specified for each routine, and what the range limits for each parameter are. Routines which perform specific operations like switching from low to high resolution do not require any parameters.

Parameter ranges

The parameter ranges in the chart indicate values that will give results fully or partially-on screen. Some of the routines actually accept parameters that give results completely off the screen. Although in these cases you cannot see the resulting display, the computer will remember the "invisible" coordinates. This means that in turtle graphics, for example, a program may produce a display that disappears off the screen, to reappear again later.

When you use the machine-code graphics routines, the screen acts as a window which allows you to look at just a very small part of the theoretical display area. Most of the routines which actually produce graphics will accept any coordinates that lie within the Commodore's integer handling range. That means that the coordinates can reach nearly 32,000. The total display area is therefore about 100 screens wide and 160 screens deep! Only one sixteen-thousandth of this is visible screen.

Parameter Ranges	Address	Checksum
—	49273	22319
—	49254	
any line number	49209	
—	49271	
any current filename	49297	
0-255	49559	34257
0-319 and 0-199 0-255	49634	
0-319 and 0-199	49712	8606
0-319 and 0-199	49792	46426
0-319 and 0-199 any value	50202	39981
0-319 and 0-199 any value any value any value	50225	
0=off, 1=on	50560	2067
0-319 and 0-199	50694	47612
—	51104	
0-319 and 0-199 any text	51167	26764
0-255 0-255 each	51328	6326
0-319 and 0-199 0-255	51394	27057
0-319 and 0-199 0-319 and 0-199	51616	8484
1=left, 0=right 0-255	51689	65736

Address

The start address for each routine shows where its machine code begins in memory. Each start address is represented by a variable. Start address 49273, for example, which is the beginning of the machine-code routine which makes the screen switch to high resolution, is represented by A1. To activate the high-resolution routine, you could either type SYS A1 or SYS 49273.

Checksum

These figures are the ones used by the Checksum program on page 61 to test if the total of a routine's DATA numbers, as keyed in, is correct. This gives a simple way of checking a listing that uses machine-code routines. When machine-code instructions are being carried out, faults will not generate BASIC error reports, making it difficult to track down bugs. Full details appear on page 61.

COMMODORE COLOR CODES

Combinations of foreground and background colors are coded by a single number from 0 to 255. To select any foreground and background color combination, add together the two numbers shown. The resulting color code can then be used with the clear-and-color, block-color or scroll routines.

Color	As foreground	As background
Black	0	0
White	16	1
Red	32	2
Cyan	48	3
Purple	64	4
Green	80	5
Blue	96	6
Yellow	112	7
Orange	128	8
Brown	144	9
Light red	160	10
Dark gray	176	11
Medium gray	192	12
Light green	208	13
Light blue	224	14
Light gray	240	15

INDEX

Main entries are given in **bold type.**

Acknowledgments

Dorling Kindersley would like to thank all those who helped in the preparation of this book especially Hugh Schermuly (design), James Burnie and Roger Cornes (program checking), Fred Gill (proofreading), and Richard Bird (indexing).

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, a turtle graphics pattern generator, and a wide variety of demonstration programs, will help you open up the full potential of the Commodore 64 – 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 the Commodore 64.

“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

ISBN 0-86318-087-6

9 780863 180873