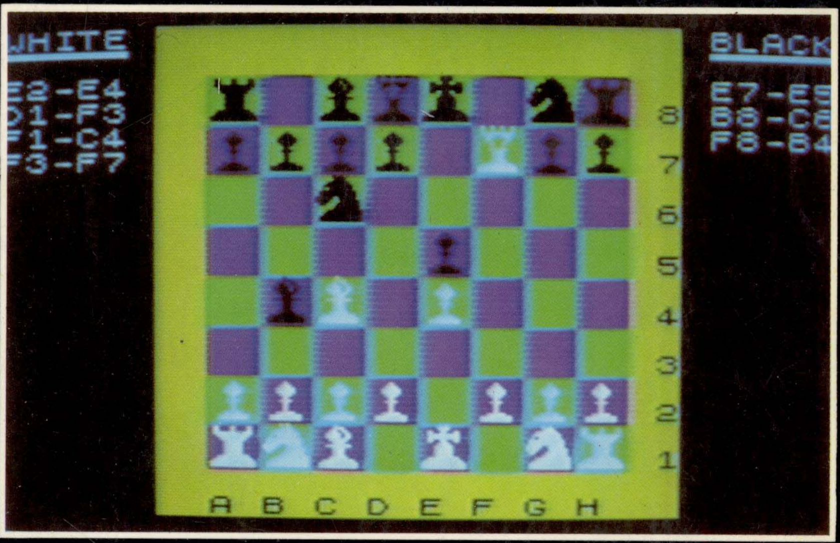
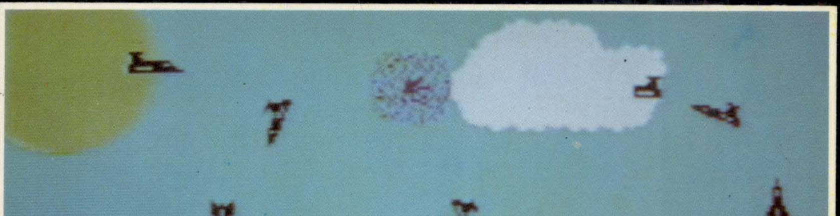
Advanced Graphics with the Sinclair ZX Spectrum





I. O. Angell and B. J. Jones

Advanced Graphics with the Sinclair ZX Spectrum

Ian O. Angell and Brian J. Jones

Department of Statistics and Computer Science,
Royal Holloway College,
University of London,
Egham, Surrey

M

MACMILLAN PRESS
LONDON

First published 1983
Reprinted (with corrections) 1983

Published by
THE MACMILLAN PRESS LTD
London and Basingstoke
Companies and representatives
throughout the world

Printed in Great Britain at
The Camelot Press Ltd,
Southampton

ISBN 0 333 35050 2 (book)
35051 0 (cassette)

Preface

With the rapid advance of computer technology has come a substantial reduction in the price of computer hardware. In the coming years the price of peripheral devices will also tumble. This means that users with a limited budget, who previously had access only to the most elementary computing devices, will soon be able to afford the most sophisticated computers. They will also be able to escape from the limitation of tabular numerical output and buy microprocessor attachments for television monitors or inexpensive special-purpose colour graphics devices. Sinclair computers have always led the field in this respect. Software, however, does not appear to be getting cheaper.

Because of the enormous capital expenditure required to set up graphical output in the past, both in machines and software, the subject of computer graphics has been the preserve of large research groups. This inaccessibility has led to a mystique growing up around the subject and it has achieved a false reputation for difficulty. This book is an attempt to lay the ghost of complexity; it will also show that complicated (and hence expensive) software packages, which are naturally of great value in research organisations, need not frighten away the average computer user. For most purposes these packages are unnecessary. This book, as well as being an introduction to computer graphics, may be considered a (very inexpensive) software package: it is a lot cheaper than commercially available packages! Naturally, because of this fundamental approach, users have to achieve a reasonable understanding of their graphics device before pictures, other than those provided, may be drawn. This need not be a disadvantage; the amount of groundwork required will be seen to be very limited. As a direct result, the knowledge of the user grows along with the package and he is far less likely to misinterpret any of the graphical routines. References are given and relevant further reading material is recommended in order to expand the reader's horizons in the subject.

It is assumed that the reader has an elementary knowledge of Cartesian coordinate geometry (the authors recommend the books detailed in Cohn (1961), Coxeter (1974), and McCrae (1953) – see References), and also of the BASIC programming language (see the Spectrum BASIC Handbook (Vickers (1982) and Hurley (1982)). Many interesting programming exercises are proposed, and these should raise the standard of the reader's BASIC expertise. BASIC is a universally popular language, available (in various guises) on all types of microcomputer, so the programs may be easily adjusted to run on machines other than the Spectrum: it is also a good medium for transmitting the algorithms used in computer graphics, enabling readers to translate these ideas readily into any other computer language of their choice.

The concepts necessary for the study of computer graphics are organised as a combination of theory and worked examples; these are introduced as and when needed in the natural progression of the subject. Some program listings form part of the examples and these should not be considered just as algorithms that describe solutions to fundamental graphical problems, but also as a computer graphics software package in BASIC, or simply as programs to draw patterns. Alongside the examples are a series of exercises that expand these ideas. The practical problems implicit in programming the various concepts of computer graphics are often more a source of difficulty to the student than the concepts themselves. Therefore it is essential that readers implement many of the program listings given in the book in order to understand the algorithms, as well as attempt a large number of exercises. As an extra learning aid, a companion audio-cassette tape is being made available; this contains most of the larger program listings given in this book. If readers are nervous of the mathematics, they should run the programs first before studying the theory.

This approach to the subject has been used with great success in teaching computer graphics to undergraduates and postgraduates at Royal Holloway College. Quickly producing apparently complex pictures results in the positive feedback of enthusiastic interest. The ability to construct pictures on line-drawing and colour interactive graphics screens makes a long-lasting impression on students; and the step-by-step approach brings them very quickly to the level of very sophisticated computer graphics. That level is outside the scope of this book, but where necessary readers will find relevant references to guide them into the more advanced topics.

This book is aimed at those who are competent BASIC programmers but complete beginners in graphics. It contains the elementary ideas and basic information about pixel and two-dimensional graphics, which must be mastered before attempting the more involved ideas of character and three-dimensional graphics. This is followed by a section relating to character graphics and the display of data (in line drawings and colour) – probably the most important non-specialised, commercial use of computer graphics. Later chapters introduce the reader to the geometry of three-dimensional space, and to a variety of projections of this space on to the two-dimensional space of graphics devices. The related problems of hidden lines and hidden surfaces, as well as the construction of complex three-dimensional objects, are dealt with in detail. Finally, we return to advanced ideas in BASIC programming and a large worked example of a video game.

Graphics is one of the most rapidly expanding areas of computer science. It is being used more and more in the fields of Computer Aided Design (CAD), Computer Assisted Management (CAM) and Computer Assisted Learning (CAL). At one time it was only the big corporations such as aircraft and automobile manufacturers that used these techniques, but now most companies are realising the potential and financial savings of these ideas. What is more, not only is computer graphics profitable, it's fun! The Sinclair Spectrum is an ideal machine on which to learn the basics of computer graphics, and an excellent springboard to the most sophisticated (and expensive) graphics devices.

We hope this book will display some of the excitement and enthusiasm for computer graphics experienced by us, our colleagues and students. To demonstrate just how useful computer drawings are for illustrating books and pamphlets, all the pictures here were drawn by computer specifically for this book.

Contents

Introduction

This book may be read at a number of different levels. Firstly, it can be considered as a recipe book of graphics programs for those who simply want to draw complex pictures with their Spectrum. We naturally hope that the reader, having drawn these figures, will be inspired to delve deeper into the book in order to understand how and why the programs were constructed. Secondly, some of the programs may be used as a package to produce and label data diagrams (pie-charts, histograms and graphs) for business and laboratory use. Finally, and the main reason for writing the book, it is an introductory text to computer graphics, which leads the reader from the elementary notions of the subject through to such advanced topics as character graphics, construction of three-dimensional objects and hidden line (and surface) algorithms.

The complex programs later in the book are much too involved to be given as a single listing. Furthermore we will see a great deal of repetition in the use of elementary algorithms. Therefore we use the *top down* or *modular* approach in writing and explaining programs. The solution to each major graphics problem is conceived as a series of solutions to subproblems. These subproblems may be further broken down into a set of problems to be solved (*modules*). These modules will be programmed in the form of BASIC subroutines. Each is given an identifier (in lower case characters) and will solve a particular subtask. Then the totality of submodules combine to solve the required graphics problem. Because the program listings are used to represent algorithms for the solution of these subtasks, we decided in general not to use statements like GO SUB 6000. We prefer instead to assign the subroutine identifier to the address value at the beginning of the routine (for example, LET scene3 = 6000) and thus we can write statements like GO SUB scene3. We use lower case for subroutine identifiers (and groupings of routines in the text) only: all other program variables will be in upper case to avoid confusion.

Spectrum BASIC does not have the facility of passing parameters into routines. Values of input parameters have to be set in assignment statements outside the routine, and the names of output parameters must be known if sensible use is to be made of the routine. This can be rather inconvenient if you are using someone else's package of routines. It is essential that users know the names of the input and output parameters; therefore in our routines we use the REMarks IN: (to identify the INput parameters) and OUT: (for the OUTput parameters). We number our programs so that all program statements are on lines ending in 0, and REMarks on lines ending in 1 to 9 (except the naming of routines). The IN: and OUT: REMarks follow directly the naming REMark on lines ending in 1 and 2 respectively. Also the cassette tape listings of programs use character codes to highlight and colour various REMarks (see chapter 13). In cases where we think that the word REM detracts from readability of a line we use these codes to make it invisible. We have minimised the REMarks on the cassette so that we can pack the maximum amount of program listing on to the tape. It is a good idea to expand these listings by adding the complete REMarks, and SAVE them on your own tapes.

For those who want only to run our programs, we give a list of complete programs at the end of each chapter together with suitable data values. In fact it is a good idea for all, including the serious readers, to SAVE the routines on tape before approaching each chapter. They can then LOAD, MERGE and RUN the programs as they occur in the text. The cassette tape available to accompany the text contains all the larger listings in the book, as well as BYTE data for diagrams and character sets used in later programs (which would otherwise have to be constructed by readers themselves, a rather time-consuming process). Our routines were written for the 48K Spectrum: if you have a 16K machine you should read appendix A and note the changes that need to be made.

As an example of what to expect, we give below the program required to draw figure I.1, a line drawing of a body of revolution in which all the hidden lines have been suppressed. This will work on both types of machine.

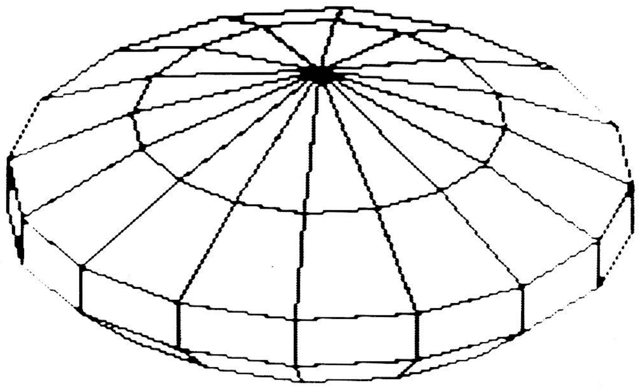


Figure I.1

The program requires the MERG(E)ing of listings 2.1 ('start'), 2.2 (two functions FN X and FN Y), 2.3 ('setorigin'), 2.4 ('moveto') and 3.3 ('clip' and 'lineto'). This combination of routines will be called 'lib1', and it was designed for drawing line figures on the television screen.

To 'lib1' must be added listings 3.4 ('angle'), 8.1 ('mult3' and 'idR3'), 8.2 ('tran3'), 8.3 ('scale3'), 8.4 ('rot3'), 9.1 ('look3') and 9.2 ('main program').

Routines, which when combined we call 'lib3', are used for transforming and observing objects in three-dimensional space.

We need also listing 10.3 ('revbod') as well as the 'scene3' routine given in listing I.1 below.

Listing I.1

```
6000 REM scene3/flying saucer
6010 DIM X(12): DIM Y(12)
6020 DIM S(6): DIM T(6)
6030 DIM A(4,4): DIM B(4,4): DIM R(4,4)
6040 DATA 0,3, 3,2, 5,1, 5,0, 4,-1, 0,-3
6050 RESTORE scene3
6060 LET revbod = 6500
6069 REM create object.
6070 LET NUMV = 5
6080 INPUT "NUMBER OF HORIZONTAL LINES",NUMH
6090 INPUT "ANGLE PHI ";PHI
6100 FOR I = 1 TO NUMV + 1: READ S(I),T(I): NEXT I
6109 REM position the observer.
6110 GO SUB idR3: GO SUB look3
6129 REM draw object.
6120 GO SUB revbod
6130 RETURN
```

Figure I.1 requires the data HORIZ = 12, VERT = 8, EX = 1, EY = 2, EZ = 3, DX = 0, DY = 0, DZ = 0, NUMH = 16 and PHI = 0. Each value has to be typed in individually on request by the machine. The picture will take about 5 minutes to draw, so be patient. Run the program with different data values. What happens if HORIZ = 6 and VERT = 4, and the other values stay the same? Set HORIZ = 15, VERT = 10, EX = 1, EY = –2, EZ = 3, DX = 1, DY = 0 and DZ = 0. Try NUMH = 20, PHI = 0.1. You will have to read up to and including chapter 10 to understand the details of what is happening.

This example illustrates the reasoning behind the layout of this book. Assuming that you are a fast typist, or that you have bought the accompanying tape, then a relatively complex three-dimensional picture can be constructed very quickly with a minimum of fuss. Even one-finger typists (like the authors) will have little difficulty in implementing this and the other programs, before they go on to study the book in detail.

We hope that this example will inspire you to implement all the programs in this book, to try most of the examples, and then to go on to draw your very own computer graphics pictures.

Now you can read the rest of our book and we wish you many happy hours with your Spectrum.

1 Graphics Operations of the ZX Spectrum

Throughout the course of this book we will be assuming that the reader is reasonably familiar with the BASIC programming language on the ZX Spectrum. In this chapter, however, we shall be looking at some of the BASIC commands – those concerned wholly or partly with graphics. With a series of example programs and simple exercises we shall examine and explore the Spectrum's capabilities. In the chapters that follow we shall use this knowledge to develop a sound understanding, both practical and mathematical, of computer graphics.

Initially we shall consider the hardware and software facilities available for producing pictures. All microcomputers that produce television pictures generate their graphical display using RASTER SCAN technology. This is also true of most of the newer commercial mini and main-frame computers. An area of memory is reserved to hold the display information for the screen and this is examined, bit by bit, as the electron beam sweeps across the raster screen. The display is composed of points, each of which is represented by a single bit (a binary on/off switch) in the memory. In the simplest case the beam is switched on for a short period each time a binary on is found, thus producing a point of light on the screen.

PAPER and INK

On the Spectrum we are given two commands; these directly control the way that the points are displayed. This affects the picture, which is made up of INK dots (binary ons) on a PAPER background (binary offs). The commands, named PAPER and INK (naturally), are called by using the name followed by a number N $(0 \leqslant N \leqslant 9)$.

PAPER N sets the background colour of the picture. After this statement is executed, all newly generated binary offs in the memory will be displayed in colour N (that is, until another PAPER command is executed).

INK N sets the points of light corresponding to binary ons to colour N in a similar way.

The number N, when in the range 0 to 7, represents the colour printed above the corresponding numeric key on the keyboard. If N is 8, then the colour previously set for an area is used. If N is 9, then the colour of PAPER/INK is set to either black or white and will contrast with the other INK/PAPER colour currently in use. In general, black INK on white PAPER is clearest, as is obvious from any book, and this is the normal setting the for Spectrum.

Display File

This type of picture is referred to as a *memory-mapped* display since it corresponds directly to the contents of an area of memory. On the Spectrum this part of the memory is known as the *display file* and starts at location 16384. A simple exploration of how the display is affected by changing the contents of the memory can be made with a program such as listing 1.1.

Listing 1.1

```
10 LET CORNER = 16384
20 LET VALUE = 137
30 POKE CORNER,VALUE
40 STOP
```

This program uses POKE to store a VALUE (entered as a decimal) in the first location of the display file. This location holds the information for the top left-hand CORNER of the screen. Since each location, or *byte*, contains eight binary bits, the first eight points on the display are affected. These change to show a pattern equivalent to the binary representation of the VALUE: in this case 10001001.

Exercise 1.1
(i) Experiment with different VALUEs and change the program either to,
  (a) use BIN (binary) representation for VALUE, or to
  (b) use a FOR. . .NEXT loop to change VALUE.
(ii) Use the PAPER and INK commands to change the background and foreground colours and then re-run the program to see what difference this makes.

BORDER

When the PAPER colour is changed it soon becomes obvious that we cannot write on the whole of the screen. A BORDER is left around the edge of the PAPER to avoid the distortion at the edge of the screen suffered by all television displays. The colour of this BORDER can be changed, in a similar way to the PAPER and INK colours, by the command

BORDER N

where N is in the range 0 to 7 and indicates the new BORDER colour.

Character Blocks

A complete picture can be built up by storing various VALUEs at locations in the display memory in a similar way to listing 1.1. For instance, we could store the eight VALUEs 0, 98, 148, 136, 136, 136, 148, 98 in the display-file locations that represent the start of eight consecutive lines on the screen (see listing 1.2). We see the pattern of INK dots corresponding to the 'ones' shown in figure 1.1.

	128	64	32	16	8	4	2	1	
00000000 =									= 0
01100010 =		64 +	32				+ 2		= 98
10010100 =	128			+ 16		+ 4			= 148
10001000 =	128				+ 8				= 136
10001000 =	128				+ 8				= 136
10001000 =	128				+ 8				= 136
10010100 =	128			+ 16		+ 4			= 148
01100010 =		64 +	32				+ 2		= 98

Figure 1.1

This is the way in which characters are defined (and redefined) on the Spectrum, but we shall leave further investigation of this until chapter 5. Nevertheless it does illustrate that a picture, even as small as this, takes time to prepare and requires a comparatively complicated program to produce the display.

Listing 1.2

```
10 LET CORNER = 16384
20 LET LINE = 256
30 DATA 0,98,148,136,136,136,148,98
40 FOR I = 0 TO 7
50 LET MEMORY = CORNER + I*LINE
60 READ VALUE
70 POKE MEMORY,VALUE
80 NEXT I
90 STOP
```

PLOT and DRAW

We have seen how the screen display can be changed by storing different values in the display file. But there are over six thousand locations in the display file and changing each of these individually would be quite tedious. We obviously need a more effective method of changing the display.

BASIC provides us with graphics commands to deal with this problem, the simplest of which are PLOT and DRAW. All the graphics commands treat the display as a grid of 256 points horizontally by 176 points vertically (45056 in total). These points are known as *pixels* and are individually identified by a pair of integers. The graphics commands help in constructing pictures by allowing us to control a *graphics pen*, which is initially positioned over pixel (0, 0). We can now explain these commands.

PLOT X, Y moves our pen to pixel (X, Y) and plots an INK point there.

DRAW X,Y draws a line from our pen's current position to the point, X pixels away horizontally and Y pixels away vertically. If X is negative, the point will be to the left and if X is positive, it will be to the right. Similarly if Y is negative, the point will be below our old position, or if Y is positive, above.

After the execution of these commands, the pen remains over the last pixel to be visited, awaiting the next command. Before examining the other more advanced graphics commands, we shall first see what is possible using only lines and/or points.

We are now in a position to draw large-scale pictures on the screen. For instance, we can draw a box around that area of screen available for graphics (listing 1.3).

Listing 1.3

```
 10 PLOT 0,175
 20 PLOT 255,175
 30 PLOT 255,0
 40 PLOT 0,0
 50 IF INKEY$ <>"" THEN GO TO 50
 60 IF INKEY$ = "" THEN GO TO 60
 70 DRAW 0,175
 80 DRAW  255,0
 90 DRAW 0,-175
100 DRAW -255,0
110 STOP
```

This program first PLOTs points at the corners of the PAPER; it then waits until a key is pressed before joining them up by DRAWing lines around the boundary of the PAPER. On comparing the PLOT and DRAW commands we see that there is an important difference in the way they work: the PLOT command uses the *absolute* pixel coordinates, whereas the DRAW command uses the *relative* positions of the points. This means that, in order to draw a line segment between two pixel points on the screen, it is first necessary to use PLOT to move the graphics pen to the point at one end of a line segment, then work out the position of the second end point relative to the first, before finally the line may be DRAWn. Note that in listing 1.3 all the points are decided before the program is run. In general, points are more likely to be INPUT, READ or calculated while the program is running.

Exercise 1.2

Write a program that calculates the position of lines to draw a grid. DRAW them using two FOR. . .NEXT loops (one for horizontal lines, the other for vertical lines).