TEACH YOURSELF

Computer Graphics

John Lansdown

COMPUTER GRAPHICS

John Lansdown

TEACH YOURSELF BOOKS
Hodder and Stoughton

First published 1987

British Library Cataloguing in Publication Data

Lansdown, John
Computer graphics. (Teach Yourself Books).
1. Computer graphics 2. Microcomputers—
Programming
I. Title
006.6′86 T385

ISBN 0 340 40819 7

Printed in Great Britain for
Hodder and Stoughton Educational,
a division of Hodder and Stoughton Ltd,
Mill Road, Dunton Green, Sevenoaks, Kent,
by Richard Clay Ltd, Bungay, Suffolk.
Photoset by Rowland Phototypesetting Ltd,
Bury St Edmunds, Suffolk

COMPUTER GRAPHICS

John Lansdown is an internationally recognised computing expert who has specialised in computer graphics since the 1960s. He has created the graphics effects for three feature films and over 100 TV advertisements and documentaries, and has held lecturing posts at the Royal College of Art and St Martins School of Art, London. He is currently Chairman of System Simulation Ltd in London.

TEACH YOURSELF BOOKS

Contents

1

Introduction

The importance of computer graphics

Probably the most striking and unexpected development in information technology in recent years is the use of computers to create pictures of all sorts. Many feature films now include computer graphics and computer animation as a matter of course, and hardly a day goes by without some new computer-produced imagery appearing on our television screens. Often these pictures are of such quality and realism that we are unaware of the part that computers played in their making. A whole new industry of designers, programmers, research workers and manufacturers has grown up to feed the demand, and some of the most powerful computers in the world are now being used in the process.

But drawings are not being created just for their own sakes. In business, computer graphics is being used to help clarify trends, to illustrate statistics (Figure 1.1), and to aid executives in their decision-making. In architecture and engineering, drawings have always played a significant part in conveying design ideas to those who have to build structures and other artifacts. It is not surprising, therefore, that computer-aided design (CAD) systems having impressive graphics facilities (Figure 1.2) are in widespread use. In science and medicine, computer graphics is being used more and more to assist in the understanding of complex phenomena. Even in home computing, graphics – some of it of excellent quality – plays a vital part in games and teaching programs of all types. All in all, then, there is now virtually no area of human endeavour that is untouched by the impact of computer graphics.

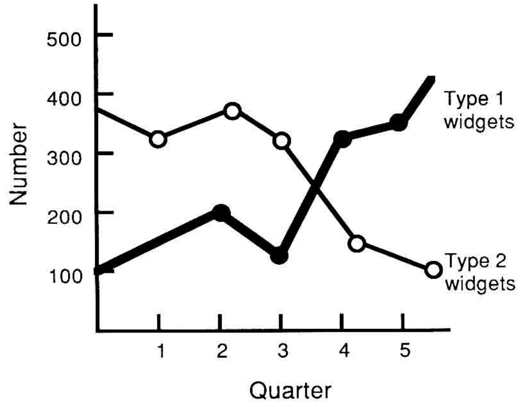

Figure 1.1 Statistical graphics

Drawings, diagrams and pictures help increase our understanding of all sorts of information. Often, ideas can be made clearer and more convincing if we can see them in graphic form – but not everyone has the necessary physical skills to prepare good drawings. Computer graphics helps us to bypass the need for these skills and allows us to concentrate on communicating information.

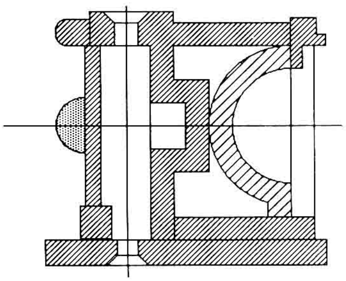

Figure 1.2 CAD example

The aim of this book

The aim of this book is to introduce those already experienced in general computing to some of the equipment and software techniques that are being used for making drawings by computer. After reading the text, coding the subroutines listed, and working through the exercises, you should have a good grounding in the basics of the subject. Depending on the hardware you have available, you will be able to use the techniques outlined here to make drawings of some interest and complexity. But the subject of computer graphics is developing rapidly. To keep up to date with the latest thinking and to increase your graphics programming skills you will need to read journal articles and other books. You will be better able to understand these more advanced texts if you work carefully through this volume.

The subroutine listings

One of the first things you will notice as you go through the text from Chapter 7 onwards is that the listings of subroutines shown in the tables are not in a programming language known to you. They are, in fact, written in what is called a *pseudo-code.* A pseudo-code is something like a computer language, but one which cannot run on a computer in the form given. We use a pseudo-code in order to write procedures that people can understand and, with a little effort, can convert into the computer language of their choice. We sometimes sacrifice efficiency and speed of running in order to help make things more understandable, and you should take this into account when you do your coding. Appendix 2 shows how this inefficiency has been dealt with in examples from various languages. You should consult this Appendix before embarking on any programming.

Again in an attempt to communicate methods more clearly, we take a special liberty with the concept of equality. Computers do not store real numbers in an exact form, so testing two reals to see if they are equal can sometimes give misleading results. In coding the subroutines you should note this, and you will usually find it safer to change statements of the form:

```
If A = B   Then . . .
```

to

```
If Abs (A − B) < E   Then . . .
```

where E is some small real number such as 0.0001.

Our pseudo-code resembles a structured BASIC and is something like Pascal, Comal or Algol. It ought to be readily understandable by those with even limited experience in programming in these languages. If you are going to program the subroutines in a structured language supporting If-Then-Else statements, dummy parameters in subroutine calls, and long variable names, you should have little difficulty. However, standard BASIC does not support these things, so extra care must be taken if you are going to work in that language.

Perhaps the biggest difficulty with standard BASIC (though it is often a useful facility for some purposes) is that all variables are global and, if set to a number in a subroutine, take those values in the rest of the program. This makes it difficult to incorporate pre-written subroutines into your programs, because they may have variable names which clash with ones you have already used. Two possible ways of minimising this problem (and some others) are given in Appendix 2.

The structure of this book

The text divides logically into two sections: Chapters 1 to 5 deal with general principles and hardware considerations; the remaining chapters deal with graphics programming. In the first section, there are only a few exercises. In the second, however, exercises abound and you should make special effort to do these. When you work alone you can monitor your performance, both in understanding and programming, only by self-testing. Fortunately, with computer graphics, you can usually see directly whether you are right or wrong but, in order to arrive at a position where drawings are possible, considerable preliminary work has to be done. The exercises (and examples) will help in this. Persevere with your efforts. You will be rewarded.

2

Graphics Systems

Like a general purpose computer system, a graphics system has three fundamental parts:

1 **hardware:** the equipment itself.
2 **software:** the instructions used to govern the workings of the hardware.
3 **documentation:** the descriptive information on the hardware and software given to enable us to exploit the capabilities of the system to the full.

Although it is possible to create some form of drawing with any computer, in a graphics system the three fundamental parts are interrelated and arranged in a way which allows the easy production of pictures of all sorts.

The basic hardware

A graphics system employs four types of equipment (Figure 2.1):

1 A **processing unit** which carries out the computation; it is this item that we usually call the 'computer' and, in personal computers and an increasing percentage of professional machines, the unit is a *microprocessor*.
2 **Storage** in which programs and data are stored until required; this is sometimes known as secondary or external storage.
3 **Input equipment** which we need in order to present graphical information to the computer; a keyboard is the most usual input

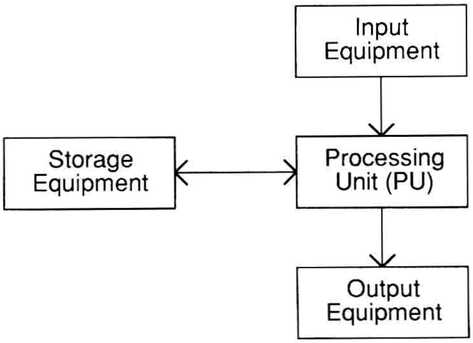

Figure 2.1 Basic configuration

device to be found in any system but a wide variety of other devices is also available.

4 **Output equipment** which the computer needs in order to present information to us; a visual display unit (VDU) capable of displaying some form of line drawing is the most widespread output device.

In some personal computers and many desk-top systems, the processor, storage (either tape or disc units), keyboard and VDU are supplied by the manufacturer as one single unit; in others, they are separate items which are connected by external cables – allowing purchasers to assemble a system which is tailor-made to their particular needs. In almost all cases where serious graphic work is to be carried out, more than one item of input and output equipment will form part of the system. You must be warned, however, that it is not always a simple matter to interconnect hardware from different manufacturers.

Sometimes, the input and output devices themselves also contain processing units. These are used to enable the devices to perform their tasks more efficiently and quickly by taking over some of the work of the central processing unit. Such devices give rise to a configuration shown diagramatically in Figure 2.2 and in these cases we talk of the input and output devices as *intelligent terminals*.

In some large-scale professional graphic systems, the processing and storage is sometimes divided into main and satellite sections as in Figure 2.3, with the satellite part catering for the purely graphic

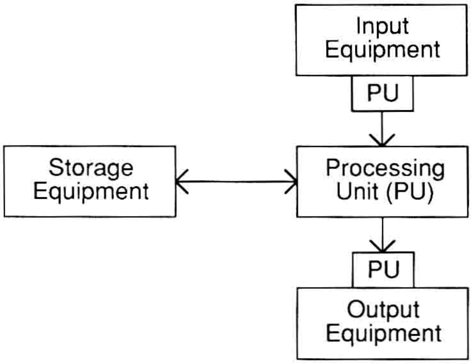

Figure 2.2 Intelligent terminal configuration

computation, interaction and display, and the main part catering for the more general work and databases. With this configuration, the main computer might serve more than one satellite which, in turn, may have multiple input and output devices.

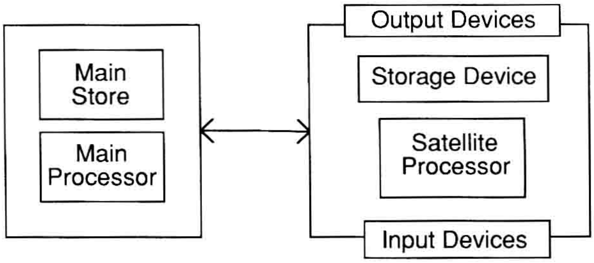

Figure 2.3 Satellite computer configuration

The basic software

In addition to these items of hardware, two elements of software are also fundamental:

1 An operating system.
2 A language system.

The operating system

The operating system is a special program which controls the operation of the equipment as a whole. It schedules the work and ensures that any input, output, computation and storage is carried out in a correct and orderly fashion. In many personal computers, the operating system often tends to be a fairly simple item of software, limited in scope and allowing perhaps only the listing, saving, recalling and running of programs. In many professional machines (particularly those that support many users at once), the operating system is often extremely sophisticated and is sometimes the largest program the machine has to obey.

The language system

The language system allows us to create programs reasonably quickly and easily by permitting program writing in a style somewhat closer to human language than the abstract code used by the computer itself. Many personal computers support only one programming language – namely BASIC. Others, especially professional machines, have systems which can cope with a number of languages of which Fortran, Pascal and C are probably the most common for scientific and graphic purposes.

Limitations of standard programming languages

The basic units of hardware and software just described are usually provided by manufacturers as general-purpose graphics packages; purchasers are expected to employ these to write application programs to suit their individual needs. As none of the well-known computer languages were originally designed for graphics, manufacturers often supply versions of the languages enhanced to include appropriate graphics commands such as DRAW, MOVETO, CIRCLE and so on. Understandably though, as manufacturers wish to exploit the special features of their own machines, there is very little standardisation in these enhancements. As a rule, therefore, graphics programs written for one type of machine cannot be run on other types of machine without modification. This lack of standardisation not only makes program portability difficult to achieve but also presents problems in books such as this where, in order not to be too abstract, instructions have inevitably to be orientated towards particular machines and languages. We have attempted, however, to make our examples as general as possible. Urgent moves are afoot in the computer industry to bring about a measure of standardisation – at least in the sorts of graphic facilities provided. However, these efforts are likely to take a little time to influence the production of machines and programming languages. We have attempted to anticipate the acceptance of graphics standards by creating our programming examples roughly within the framework of current thinking in this area.

It is essential to realise that all the elements in a system are important and that defects in any one of these will limit the performance of the whole. The quality and types of drawings that can be created and the degree of graphical interaction possible depends on the available hardware and software. The extent to which you can properly exploit the full potential of the system depends on the quality and comprehensiveness of the documentation.

Graphic styles

Different types of system and, particularly, different forms of output device give rise to three different computer graphic styles:

1 Character or mosaic graphics.
2 Calligraphic or vector graphics.
3 Raster or pixel graphics.

Character or mosaic graphics

Here, as in Figure 2.4, drawings are assembled mosaic-fashion from a special set of graphics characters either supplied with the machine or designed by the users themselves. The characters are accessed

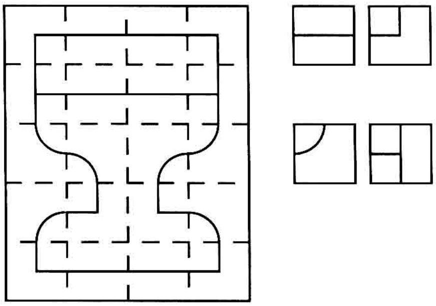

Figure 2.4 Mosaic type drawing showing four elements used

from the keyboard as if they were alphanumerics. With careful design and ingenuity, some interesting pictures can be made in this way, but drawing with characters is a very limiting style suitable only to such things as simple diagrams, videotext and video-game figures. Some personal computers can draw only in character graphics.

Calligraphic or vector graphics

Here, drawings are made up from lines (often just in one colour). This style, which is the one most often used for plotting onto paper, can produce very accurate and complex drawings and is generally the one favoured for engineering and drafting applications. It is not easy to produce areas of solid colour by vector graphics, so this sytle is not used when shaded drawing or a high degree of realism is sought. VDU vector systems are expensive to produce hence, except when plotting on paper, personal computers do not use the calligraphic style.

Raster or pixel graphics

In this case, drawings are made up of arrays of closely-spaced dots, called *pixels* (short for 'picture elements'), which allow either lines or areas to be delineated in various colours. Because of the hardware simplification raster graphics makes possible, this style is rapidly becoming the most widespread, but the dotted nature of the drawings creates problems of accuracy and resolution which have to be specially dealt with by software (or, sometimes, hardware) techniques. The most visible manifestation of these problems is that diagonal lines have a 'staircased' appearance which can be disturbing. Most personal computer systems use this style of drawing.

In general, only vector and raster graphics are dealt with in this book.

Graphic tasks

We use graphic systems in order to facilitate graphical input and output. For input, we need support for three tasks:

1 Interaction with the system.
2 Setting-up and editing drawings and graphic text.

3 Converting existing drawings into machine form: a process known as *digitising*.

For output, we need support for two tasks:

1 Making drawings.
2 Creating graphical menus or symbols to assist in interaction.

Except in the case of some expensive equipment where special processors are incorporated, graphics computers differ from general purpose ones in respect of their input and output devices and the way these are supported by the software. For most applications, input and output are of equal importance, but it is probably true to say that those who sell graphics systems tend to stress only the quality of the output facilities of their product. Of course, an impressive output can only be demonstrated if the system is capable of providing it so, in that sense, the salespersons are not being misleading. Users soon find, however, that properly designed input facilities which are well-supported by comprehensive software are just as important if the graphics potential of the system is to be exploited to the full.

Graphics primitives

In order to allow us to carry out graphics tasks, systems come equipped with their own particular sets of drawing instructions, or *primitives* as they are sometimes called. Some systems, especially the more expensive ones, have comprehensive sets consisting of dozens of primitives to draw such things as lines, polygons, circles, arcs and other figures; to change line styles; to fill-in areas with solid colours or hatching; to erase or display portions of a drawing; to rotate, move or reflect figures, and so on – all with single commands. Other systems have much more limited sets of primitives, perhaps allowing the display only of single lines in one style and colour. Systems differ not only on the output tasks they facilitate but on the input tasks too; some will have no special input primitives, others will cater for input from all sorts of sources.

To assist us in dealing with these differing capabilities, we will assume that the system we use has only six primitives: one to tell the system that we want to use graphics, four for output and one for input. We assume that the (x, y) coordinates given to the primitives lie within the ranges permitted by our drawing surface (Figure 2.5). Anything outside these ranges will give rise to an error condition. Our primitives are:

1 **GRAPHICS(n),** which clears the screen and puts the system into graphics mode if n = 1 and out of it (without clearing the screen) if n = 0. Any graphics instructions used when n = 0 (or if n is undefined) are ignored.
2 **MOVE(x, y),** which puts the drawing head (electronic beam or pen) in the position on the drawing surface defined by the coordinates (x, y) without making any mark or trace.
3 **DRAW(x, y),** which moves the drawing head from its present position, wherever that is, to a new position (x, y) leaving its own version of a straight line as it goes.
4 **TEXT(x, y, message),** which prints the string defined in message starting at point (x, y).
5 **COLOUR(n),** which sets the drawing colour to the hue (n). When n = 0, the colour is set to the same as that of the background so that anything drawn is effectively invisible. The object of having COLOUR(0) is to allow us to erase existing lines by drawing over them with the background colour.
6 **ACCEPT(flag, x, y),** which will take a pair of (x, y) coordinates from an input device and set the flag parameter to a number depending on some action performed by the user (such as pressing a button).

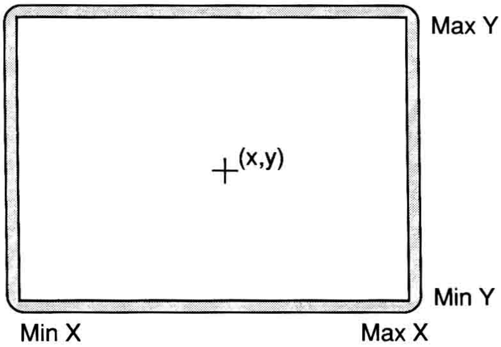

Figure 2.5 All points must lie within the drawing surface (i.e. in the range Min X, Max X horizontally and Min Y, Max Y vertically)