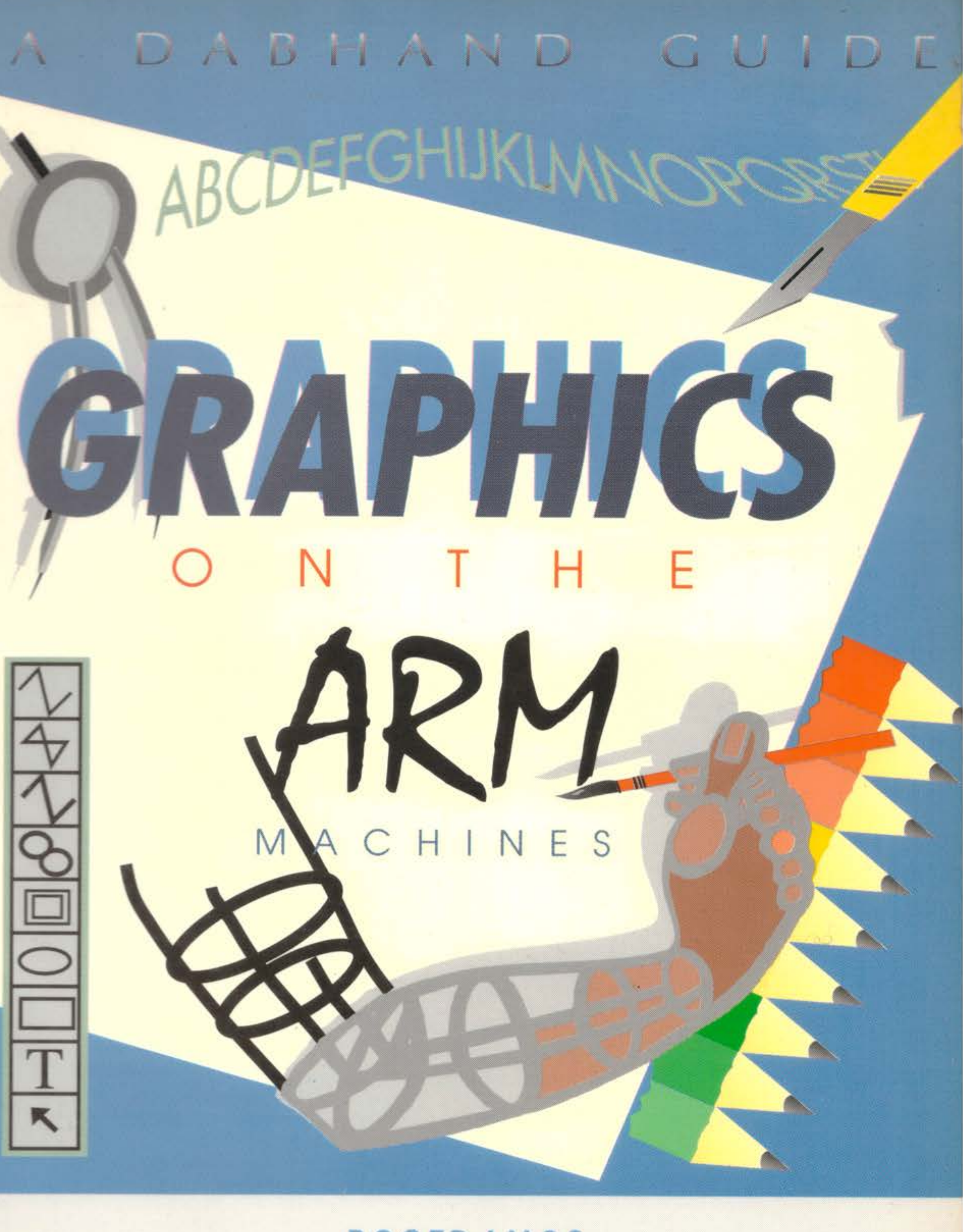
A DABHAND GUIDE

GRAPHICS ON THE ARM MACHINES



ROGER AMOS

DABS PRESS

[redacted]

A DABHAND GUIDE

GRAPHICS ON THE ARM MACHINES

ROGER AMOS

DABS PRESS

Stephen Murphy

©Roger Amos 1993
ISBN 1 870336 13 5
First Edition January 1993
First printed in the UK January 1993

Editor: Anne Rooney
Typesetting: Roger Amos
Cover Design: Mark Nuttall

Published by: Dabs Press Publishing, PO Box 48, Prestwich, Manchester. M25 7HN

Printed and bound in the UK by Ashford Colour Press, Gosport Hampshire. PO12 4DT.

Contents

Preface

Suddenly, four years after the launch of the Acorn Archimedes in 1988, there is a new surge of interest in graphics using the Acorn 32-bit machines (which now include the A3000, A3010, A3020, A4000, A5000 and the A4 portable). This is evidenced not only in the recent introduction of a regular graphics page in the magazine *BBC Acorn User*, but also in a spate of graphics software and support hardware.

It is not as if the original Archimedes was unsuitable for graphics. Indeed Clares were soon on the scene with sophisticated painting, ray tracing and animation packages that made full use of the machine's 256-colour capability. With RISC OS came *Draw* on which many users learned the massive potential of vector graphics and object-based art—and some found themselves baffled by its sophistication. Also with RISC OS came the sprite editor *Paint*, not intended as an art program but perfectly capable of being used as one, and making possible pixel-based art in 16-colour and even 2-colour screen modes.

The explosion of interest coincided with the launch of RISC OS 3 in late 1991. Perhaps it was the limitations of the 256-colour palette that had held back the development of graphics on the ARM Machines. To keep the cost of the machine within the budget of the education establishments, Acorn's principal market, the video controller chip is limited to a fixed set of 256 colours or any 2, 4 or 16 from a palette of 4096 colours—for the computer artist this is either a severe restriction or a challenge, depending on viewpoint. But RISC OS 3 overcame the limitations by its support for dithering which simulates hundreds of colours in 16-colour modes and thousands in 256-colour modes.

Probably under the stimulus to improve upon Acorn's *Draw*, Jonathan Marten's *DrawPlus* was launched in 1991 followed by *Vector* in 1992, both adding valuable new features and convenience to those of the original. And, inspired by the drawing packages used on the PC and Macintosh machines by professional designers, Computer Concepts developed *ArtWorks* released in Autumn 1992.

Another stimulus has been the increasing availability of support hardware. Graphics created on Acorn machines can now be output to colour ink-jet printers capable of quality close to that produced by printing presses. Professional colour scanners with resolutions of up to 1200 dots per inch can now be used, albeit with limitations, to convert photographs and other documents to ARM graphics. Pictures can also be imported from video sources using video digitisers, some of which cost well under £100. And if the limitations of the computer's monitor system are more than you can bear, for a few hundred pounds you can add a board to your machine which allows your monitor to display all of the 16,777,216 colours recognised by object-based software.

This book provides an overview of the graphics capability of the ARM machines and tries to give a grounding in the fundamental principles underlying it. I believe that, if you understand just what the system can—and cannot—do, then your natural creativity will devise all sorts of fascinating new applications for it. Starting with vector graphics and moving on to pixel graphics, the book reviews some of the software available, much of it very modestly priced.

Clearly you need some artistic ability to make the most of graphics software. Perhaps artistic insight is even more valuable. This book is not a handbook of artistic techniques. But even those whose talent is modest will find that much graphics software will compensate for what is lacking. The knowledge that you can instantly undo an unfortunate operation is a source of comfort—and an encouragement to experiment with techniques previously untried.

For inspiration, you will find the work of some talented artists in the colour plate section at the centre of the book. I am very grateful to those who have allowed me to reproduce their work. Their names (except for some who are anonymous) and the software they used is noted in the captions.

I am indebted to the software suppliers who generously supplied products for review or who gave of their time; their constructive criticism and suggestions proved invaluable. Their names and addresses are given in Appendix 5. Nearly every company approached proved only too willing to cooperate. I am only sorry that limitations on time and energy (and on the length of this book!) made it impractical to approach every supplier. I am well aware that there is plenty more excellent graphics software available that has not been described here and so the omission of any particular package from this book must on no account be regarded as significant.

I must also set on record my thanks to the Editor of *RISC User* magazine, the staff of RISC Developments and to Dave and Clare Atherton of Dabs Press for their continuing support and encouragement. And special thanks are due to my nephew Simon Haslam for allowing me to pick his brains regarding the intricacies of ARM code and user access to RISC OS's internal routines, and for his patient reading of the manuscript and helpful suggestions.

Roger Amos,

Rugby, November 1992

1 Understanding Graphics

What are graphics?

In computer jargon the term *graphics* generally denotes any images that appear on the monitor screen or in printouts with the exception of text using the computer's or the printer's resident typefaces. So pictures, lines and other shapes are all graphics while normal text is not.

That, however, is misleading. The word *graphics* comes from the Greek γραφω which means *I write*. Strictly, then, graphics should embrace text and, indeed, the visual aspect of the entire content of the monitor display or printout. A *graphic designer*, after all, is responsible for the overall appearance of pages in books, magazines and newspapers; his province includes the text as well as rules, tables and illustrations.

Similarly in the Acorn RISC-based computers the distinction between text and graphics is not always clear. The RISC OS 3 operating system includes the outline font families Corpus, Homerton and Trinity in ROM; they are permanently resident in the machine's memory and many applications display text in these attractive typefaces instead of the rather uninteresting 'system font'. Moreover, the characters in these fonts are stored as outline drawings (vector graphics); most printers print them as graphics and some software accesses them as graphics rather than as text allowing them to be manipulated in exactly the same way as other vector graphics. The distinction between text and graphics is tenuous.

This book is concerned principally with graphics in the more generally accepted sense: images such as pictures and line diagrams. But there will be plenty of references to text where this is relevant to the understanding of graphics.

Types of graphics

There are two principal systems of computer graphics: *vector* (or *line*) graphics and *pixel* (or *bitmap*) graphics. These two systems are represented in the RISC OS applications suite by *Draw* and *Paint* respectively. The differences between them are a consequence partly of the two quite independent ways in which the screen is organised in Acorn computers and partly of the form in which the graphics are stored.

Mathematical and abstract graphics (described in Chapter 15) form a third system.

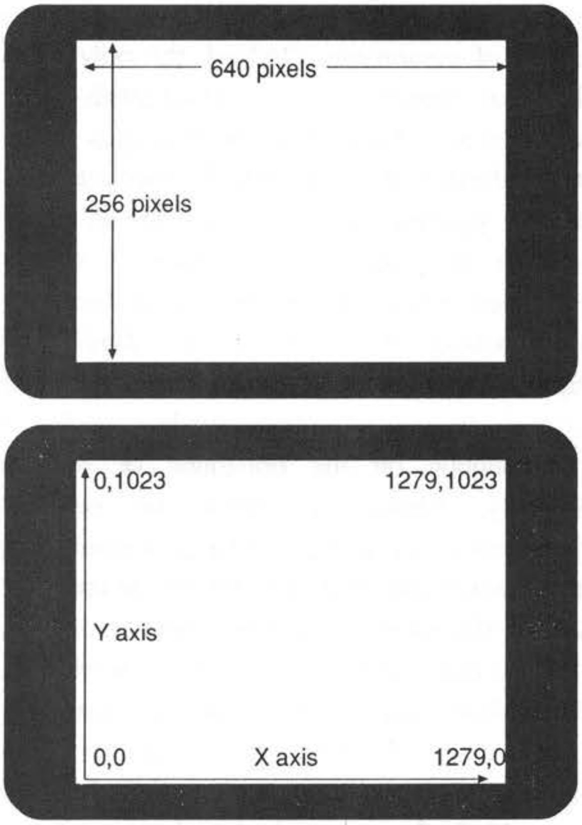


Figure 1.1—Organisation of the screen (assuming mode 0, 8, 12 or 15) as (above) pixels and (below) co-ordinates

Figure 1.1 illustrates a monitor display in stylised form. Graphics (and text) are confined to an active area which occupies all of the screen except for the border. The active part of the screen consists of an array of tiny rectangular dots called *picture elements* or *pixels*, each of which contains a single colour. Everything that appears on the screen, text or graphics, is visible only because the colours of its pixels have been made different from those of the surroundings.

Because the monitor display consists entirely of rectangular pixels, inevitably most graphics are a matter of compromise. Indeed, the only items that the screen can reproduce with strict accuracy are straight lines or rectangles which are perfectly vertical or perfectly horizontal and whose dimensions correspond to an exact number of pixels. Oblique lines, if closely examined, will always appear jagged, since they are inevitably composed of a series of miniature zigzags in which each element is horizontal or vertical; this is most noticeable when the line is only slightly off the horizontal or vertical. Similarly, curves can never be perfectly represented in a rectangular matrix of pixels. Rug and tapestry makers face a very similar problem! Clearly the smaller the pixels used—that is to say, the higher the resolution—the less obvious these shortcomings will appear. The same, of course, applies to printouts: those produced on a dot matrix printer at 144 dots per inch will inevitably look more crude than those produced on a laser printer at 300 dpi, while at 600 dpi curves appear smooth and gradual, even under a magnifying glass. See Figure 1.2.

The number of columns of pixels across the screen, the number of rows of pixels down the screen, the shape and size of the pixels and the maximum number of different colours that may be displayed simultaneously on the screen are all a function of the *screen mode*. RISC OS incorporates a wide range of screen modes (47 of them in RISC OS 3) which cater for most requirements and the range can be further

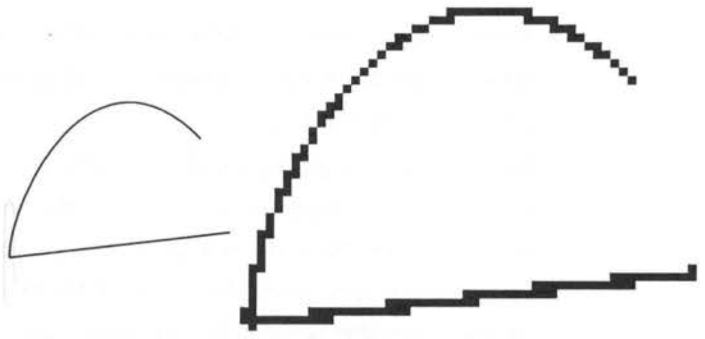


Figure 1.2—The art of compromise: on the left an oblique line and a curve as reproduced by a laser printer at 600 dots per inch; on the right the same graphics as shown on a low-resolution (mode 13) screen magnified by 4. This demonstrates that in computer graphics there is, strictly speaking, no such thing as either an oblique line or a curve

extended using software. For the present we shall confine our considerations to 12 of the most commonly used modes which are integral to RISC OS. Modes 0, 8, 12 and 15 are all 640 pixels wide and 256 pixels deep; they allow 2, 4, 16 and 256 colours respectively. In these modes the pixels are oblong, their height being double their width; consequently graphics in these modes have a horizontal resolution that is twice as sharp as the vertical resolution. Modes 18, 19, 20 and 21 are also 640 pixels wide and again offer 2, 4, 16 and 256 colours respectively. They are, however, 512 pixels deep and their pixels are square, giving equally fine resolution along both axes. To use these modes you must have a multisync monitor. If you have the monitor supplied with the A5000 machine, you will probably most often use modes 25 to 28 which are similar to modes 18 to 21, but only 480 pixels deep; the pixels are square and the display occupies nearly the full width of the monitor screen. This contrasts with the other modes which leave a substantial border on either side of the active display area.

Since the numbers of rows and columns of pixels on the screen vary depending on the screen mode in use, these do not provide a consistent medium for specifying the sizes or locations of graphics on the screen. For example, an image that was defined as 20 pixels high and located 100 pixels from the bottom of the screen would double its height and move up the screen if you changed from mode 20 to mode 12.

For this reason RISC OS offers an alternative convention which measures and positions graphics on the screen with greater consistency. This convention maps the screen using a co-ordinate system calibrated in 'OS units'. By default the origin is at the bottom left-hand corner of the screen, although it can be repositioned, with the ORIGIN statement in BASIC, for example. Most screen modes are 1280 OS units wide and 1024 deep, so the top right-hand corner of the screen is at 1279,1023. The VGA modes (25 to 28), however, are slightly smaller at 1280 × 960. So graphics created in, say, mode 20 may be too tall to fit on a VGA screen, as owners upgrading to the A5000 machine soon discover! Some other modes cover a larger graphics area: mode 31, for instance, is 1600 × 1200 OS units. In this book, for simplicity, we shall assume that all screens are 1280 × 1024 OS units unless otherwise stated.

Any point on the screen (or even outside its perimeter but in the same plane) can be specified by its x (horizontal) axis and y (vertical) axis co-ordinates. You may be familiar with BASIC's DRAW and PLOT facilities which place graphics on the screen using this system.

Since in the multisync modes 18 to 21 the active area of the screen measures 1280 × 1024 OS units and the screen consists of 640 × 512 pixels, it follows that the OS units are exactly half the width and height of the pixels. Perhaps future Acorn machines supporting even higher-resolution monitors will offer screen modes having 1280 × 1024 pixels in which the pixels and graphics co-ordinates will correspond exactly.

Vector graphics

In vector graphics the image on the monitor screen or in the printout consists of lines and filled shapes which the computer has plotted from co-ordinate data. Since the screen's co-ordinate system, as we have seen, is largely independent of the screen mode, vector graphics are also independent of screen mode or printer resolution.

The artwork in vector graphics consists of one or more 'objects' each made up of one or more lines. The lines in an object may form closed paths such as rectangles or circles; areas enclosed by lines may be assigned their own fill colour. Since each object is effectively an independent drawing which can be moved, deleted or edited usually without affecting other objects in the artwork, vector graphics packages are sometimes termed 'object-based' packages. (Most object-based drawing packages handle other kinds of object besides vector graphics.)

As you compose the first object in your drawing, the computer stores a block of data describing it in a reserved area of memory called the data *stack*. As you create further objects in your artwork, the computer adds blocks of data describing them above the existing stack contents. Consequently the sequence of the blocks of data that build up in the stack initially follows the order in which the objects were created. However, both the data in the blocks and the sequence of the blocks in the stack may be edited subsequently, as we shall see.

This data stack is the very core of vector graphics and other object-based applications, including desktop publishing (DTP). It is this stack that is saved when you store your artwork on disc. And whenever the application needs to redraw the picture, *e.g.* after editing, after reloading work or clearing some other window that temporarily obscured your artwork, the computer reads through the data in the stack and from them redraws your picture, object by object.

The presence of the stack confers some remarkable properties on object-based graphics. Consider what happens, for instance, when redrawing two objects which overlap. The one that comes lower (earlier) in the stack is drawn in the normal way, but soon afterwards the other object is also drawn, partially overwriting the first one. So, in the screen display or printout, the object whose definition comes later in the stack will appear to stand in front of the earlier one. The stack, then, determines the *order* of the objects from back to front and this provides what is almost a third dimension in the artwork.

For instance, if you use vector graphics to create a conventional picture of, say, a countryside scene, it is simplest to draw the background first, and then items in the middle distance and finally items in the foreground. The last item drawn will be at the top of the stack and therefore will appear to stand in front of all other objects on the screen or in the printout. However, you need not feel constrained by this. Most packages provide facilities to edit the sequence of the data blocks in the stack. In *Draw* these facilities are somewhat rudimentary, allowing you only to send an object (or group of objects) to the front (*i.e.* the top of the stack) or to the back (*i.e.* the bottom of the stack). With careful repeated use, however, these facilities allow you to rearrange the objects in the artwork in any order from front to back. Many other packages such as *DrawPlus* and *Vector* also allow you to step an object backwards or forwards through the stack until it is at the required 'height'.

But, although objects created later may overwrite earlier ones, they do not destroy them. Even if an object is completely hidden—for example, if you draw the sun and then draw a cloud which completely obscures it (see Figure 1.3)—the data

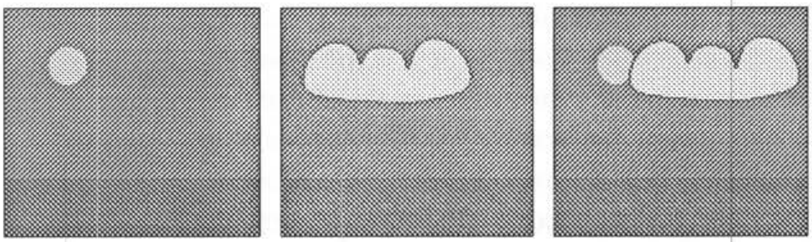


Figure 1.3—In vector graphics hidden objects are not lost irretrievably

that describe the hidden sun are still present in the stack. So, if you subsequently move either object or if you delete the offending cloud, the sun will reappear.

Another fascinating consequence of the *modus operandi* of vector graphics packages is the comparative ease with which you can correct any part of your drawing with which you are dissatisfied. If you draw a line and then find it is the wrong thickness, the wrong length, the wrong colour or simply in the wrong place, you can very easily amend it, or even delete it and start it again. You needn't worry about restoring whatever detail was behind it; the software will take care of that automatically as it redraws its way through the data stack. This is, of course, a boon to the artist with limited talent, or to the perfectionist!

Moreover, it opens the field of computer graphics to folk who are deprived of other means of creative visual expression. A friend of mine who is a very gifted engineer lost almost all use of his hands in a tragic accident. Subsequently he has been unable to use a pen or any conventional drawing implements. He cannot use a mouse, but when his company purchased *Corel Draw* (which has a similar specification to *ArtWorks*) to help in the creation of promotional documentation he found that he could control the software quite satisfactorily using a trackerball with foot switches duplicating the three buttons. Today this means of turning his ideas to hard copy is making a significant contribution to his company's work on the very frontiers of technology. At a less sophisticated level, similar hardware and graphics software give hitherto unimaginable creative opportunities to people suffering many kinds of handicap.

Ironically some criticise vector graphics for generally lacking detail, while others point out that, in contrast with pixel graphics where the detail is of necessity limited by the pixel size, vector graphics offer *infinite* detail. Both sides are only partly correct. In vector graphics you can have as much detail as you wish, but adding copious minute detail is extremely tedious. Consequently vector graphics often use simple fill colours causing them to resemble drawings or cartoons rather than paintings or photographs. More sophisticated packages such as *ArtWorks*, however, provide graded colour fills, making possible very realistic representations of three-dimensional objects. *Chameleon 2* adds this facility to *Draw* and similar packages while the grade facilities in *Vector* and the RISC OS 3 version of *Draw* also provide this kind of effect. See colour plates 1 and 3 for examples.

But vector graphics do not offer *infinite* detail; nothing in this world is infinite! In vector graphics the resolution is determined by the units in which the application stores its co-ordinate data; in *Draw* that unit is 1/46,080 inch. The reason for this strange-sounding figure is given in the next chapter. Obviously, to make full use of such fine resolution you must either work at a far higher magnification than the regular zoom facility allows or enter your co-ordinate data in numeric format. It may appear as though infinite detail is possible, though, as you can magnify part of a vector-graphics drawing by as much as a thousand times without the limitations of resolution becoming obvious—this stands in marked contrast with pixel graphics which are intended for display at one particular size and whose pixel structure is disturbingly obvious at magnification factors as low as four. (Contrast Figure 1.4 with Figure 1.6.)

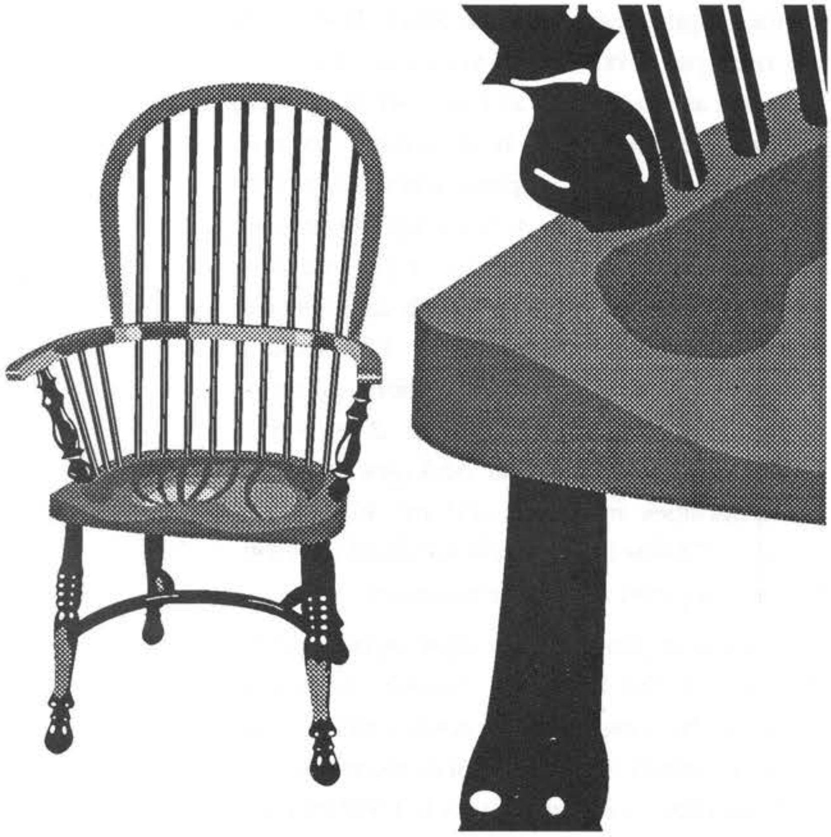


Figure 1.4—This stickback chair drawn in *Draw* ably demonstrates that vector graphics can be magnified without any adverse effects other than revealing the artist's shortcomings! The view on the left is at 45% of original size and that on the right is at 270%

Indeed the ease with which vector graphics can be magnified is another of its strengths. To make your drawing (or part of it) *n* times its present size it is only necessary for the computer to find the relevant co-ordinates and line thicknesses in the data stack and to multiply them by *n* (see Figure 1.5), a very simple task for a computer.

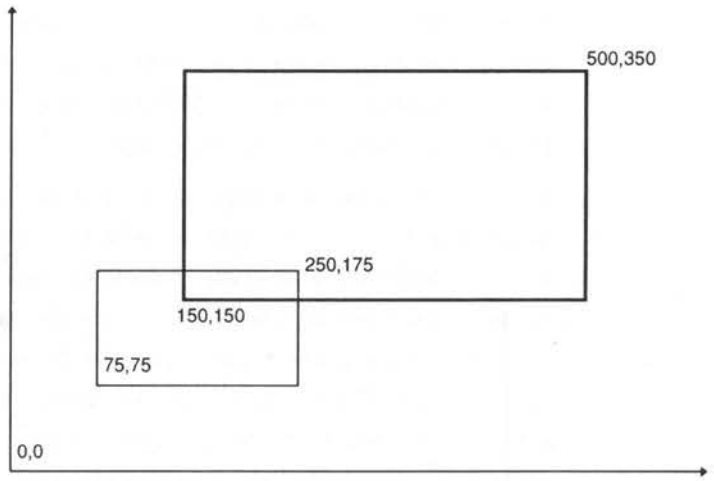


Figure 1.5—In vector graphics magnifying a drawing only involves multiplying the co-ordinates and the line thicknesses by the magnification factor (2 in this example)

When the new values are in place, the artwork is redrawn at its new size. Of course, if *n* is less than 1, your graphics are made smaller. (This flexibility is the *raison d'être* for outline font systems; one set of character outlines will provide all possible font sizes.) Moreover, you can apply separate magnification factors to the x and y axes, allowing your graphics to be stretched or squashed. And if *n* is made negative, the image is flipped about the axis or axes concerned. Indeed magnification provides one useful technique for handling all those fiddly details too small to handle at even the highest zoom setting: magnify the object by a really substantial factor, say 100, allowing you to work on it with room to breathe, and when you have finished magnify it by 0.01 to restore it to its proper size. And if you ever need to hide a drawing, try magnifying it by 0.001. It can then be concealed conveniently behind a full stop in some text, recalling classical espionage techniques! Magnifying it by 1000 will restore it to its original condition, although any text present may now be incorrectly sized.

There is of course a penalty to pay for all these fascinating features and that penalty is in *time*. Vector graphics, by definition, must be plotted on the screen line by line and even for the ARM that takes appreciable time, especially if curves are involved. For example, on an ARM2-fitted machine in mode 20 *Draw* takes exactly 10 seconds to plot one of my drawings consisting of 38 kilobytes of pure vector graphics, mostly curves. An ARM3 reduces the time to about 2 seconds. To use such complex vector graphics as the basis of a fast-action arcade game would clearly be totally impractical!

Pixel graphics

Pixel graphics consist of two-dimensional arrays of pixels. Normally these pixels correspond to the pixels which make up the monitor display, although they need not always do so.

These two-dimensional arrays of pixels are commonly called *sprites*. Sometimes a sprite is an entire screen, but it can be any size, larger or smaller than a screen. When you save your artwork, the file created simply represents the colours in the sprite, pixel by pixel, together with some other vital data such as the sprite's dimensions, the screen mode in which it is intended for display and sometimes information about the colours used. The data in the file are of course sufficient to reproduce the image, but generally contain no details whatever concerning the stages by which the image was created.

This leads immediately to one significant contrast with vector graphics. In pixel graphics if you overlay part of your image with new matter, the overwritten material is lost irretrievably—unless you have a copy of it saved somewhere else. Similarly, if you erase some detail that you decide is unwanted, you may then find you have an unsightly gap in the background which needs to be filled in.

Pixel graphics packages include routines for creating lines and shapes in a variety of colours, just as vector graphics packages do. But they also contain a diversity of facilities which have no counterpart in vector graphics packages. These are routines concerned with individual pixels rather than larger areas of the screen. Examples are pixel editors, which allow you to change the colour of individual pixels, and 'spraygun' facilities which allow you to change pseudo-random groups of pixels in a given area to a chosen colour.

This facility to address individual pixels meant that the pure colours offered by the system palette could be supplemented by additional colour effects using 'dither patterns', *i.e.* alternating the colours of adjacent pixels in a chequerboard pattern, long before this facility became available in vector graphics packages on the ARM machines. In high-resolution graphics modes, individual pixels are barely visible, and on the screen a dither pattern will give a convincing impression of an intermediate colour. By combining dither patterns and spraygun effects you can produce subtle gradations of shading, allowing the creation of pictures with 'photographic' realism.

Detail is of course limited to the size of the pixels. Each piece of pixel artwork is ideally intended for display in one particular screen mode, although RISC OS contains sophisticated routines that allow sprites to be displayed in 'foreign' screen modes—even those which use pixels of different shapes or sizes—while retaining their original proportions. As mentioned earlier, using a zoom facility to magnify the image does not reveal additional detail—it simply makes the pixel structure of the graphics all the more obvious, as Figure 1.6 shows.

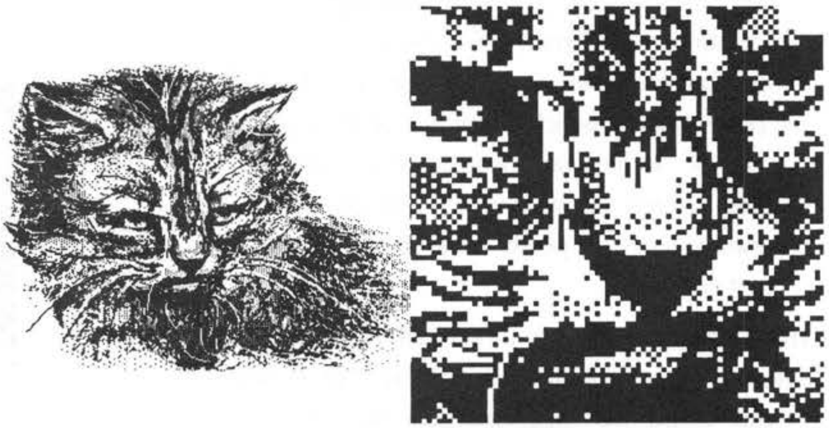


Figure 1.6—(left) Pixel graphics make fine detail comparatively easy, but (right) magnification even by a factor as low as 4 causes unpleasant effects. This sprite is from a collection of clip art available from Orion Computers

But when displayed in its intended mode, especially if this is a high-definition mode such as mode 12 or mode 15, the level of detail in a sprite is quite acceptable; indeed the picture quality is similar to that of a standard TV or video picture. And since this level of detail can be created with comparative ease, most of the more artistic graphics software packages use pixel-graphics techniques.

Another advantage of pixel graphics is speed. RISC OS includes a comprehensive library of routines for handling and manipulating sprites and these are easily accessible to the user. They work so fast that even BASIC applications on an ARM2 machine can throw a succession of sprites on the screen fast enough to provide smooth, life-like animation.

Applications of graphics—an overview

Each of the two graphics systems has its strengths and its weaknesses. These are summarised in Table 1.1. Not surprisingly, the

Table 1.1—Comparison of Vector Graphics and Pixel Graphics

	Vector graphics	**Pixel graphics**
Speed of plotting	slow	fast
Ease of adding detail	poor	good
Ease of editing	excellent	poor
Resolution	almost unlimited	limited by pixel size
Applications	DTP, graphic design, engineering drawings and schematics, 3D drawings, originals for animation	computer art, fast arcade-style games, animation

fundamental differences between them have led to their finding divergent uses.

Vector graphics are ideal for all applications which demand accuracy and unlimited restrictions on editing. Engineering and architectural drawings, electronic circuit diagrams (Figure 1.7) and process plant

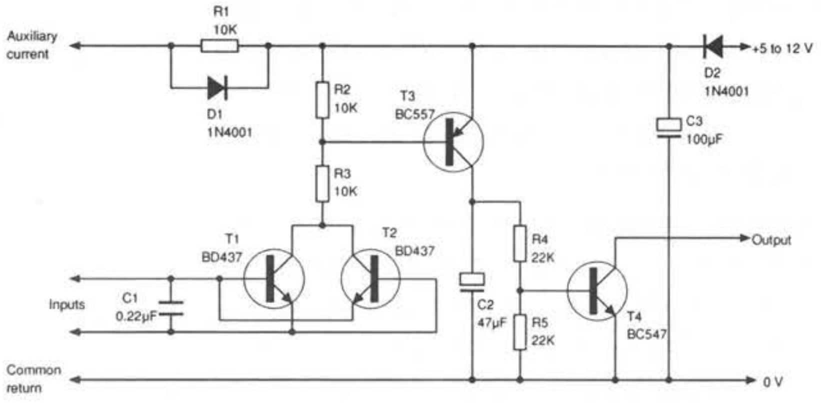


Figure 1.7—Vector graphics are ideal for electronic circuit diagrams and other forms of technical drawing. This example was produced in a few minutes using *Draw* and the *Cir-Kit* set of symbols from Wiseword Software

schematics can all be produced to very high accuracy and can be edited easily whenever the design is amended.

Because vector graphics can build up images from separate overlapping components they are suitable for many applications. Most of the vector graphics programs for the ARM machines are educational, but some are used for design in industry and commerce. The *smArt* 'linked graphics system' from 4Mation is an example. Schoolchildren can experiment with a car that can be given various body styles (sports coupé, hatchback or saloon), various door styles and wheel trims; a fashion collection (Figure 1.8) allows potential fashion designers to dress mannequins in garments and wigs offering almost limitless permutations of style and colour, and a heraldry collection allows crests to be built up from a wide choice of heraldic motifs.

All serious desktop publishing (DTP) packages use the same object-based philosophy, even if their graphics facilities are limited to displaying and printing rather than creating or editing graphics. It is not hard to see why. The introduction of a new object, such as a picture, over the top of an area of text must not cause the

Figure 1.8 — 4Mation's 'fashion collection' allows the user to examine the effects of different combinations of a wide range of garments and hairstyles

text to be lost; the text must be preserved and—normally—reformatted so that it flows around the newly introduced object. This is achieved using the typical object-based technique of keeping a copy of the text (in ASCII format) in the data stack.

A multiplicity of graphic design packages, ranging from the comparatively simple *FontFX* to the sophisticated *Fontasy*, import characters from the outline font system as vector graphics. They can then perform an almost unlimited variety of fascinating manipulations and transformations on the graphic objects. The ability to take any object (each character is an object) or any group of objects and move it, rotate it, squash it or expand it—and then put it back as it was if not satisfied—is all grist to the mill for graphic designers.

Vector graphics can be created from mathematical models. This is the basis of so-called three-dimensional drawing packages such as *Euclid*. These generate drawings of three-dimensional objects such as houses, cars or items of furniture as viewed from any angle, distance and elevation specified by the user.

Although it was suggested earlier that vector graphics are too slow for use in animations, they do nevertheless have many roles in the creation of animations. Since the artist can move objects around, concealing and revealing other objects in the background, vector graphics provide a highly versatile computer equivalent of the paper cut-out animation system. Although vector graphics are used to generate the original images that will be strung together as an animation, those originals are usually converted to sprites to facilitate their rapid handling. This is analogous to photographing the frames set up with paper cut-outs.

There are, however, some animation systems which use vector graphics in real time, although the subjects depicted must of necessity be kept simple. These are *interactive* animations of which the most obvious examples are flight simulators (Figure 1.9). In these the size and

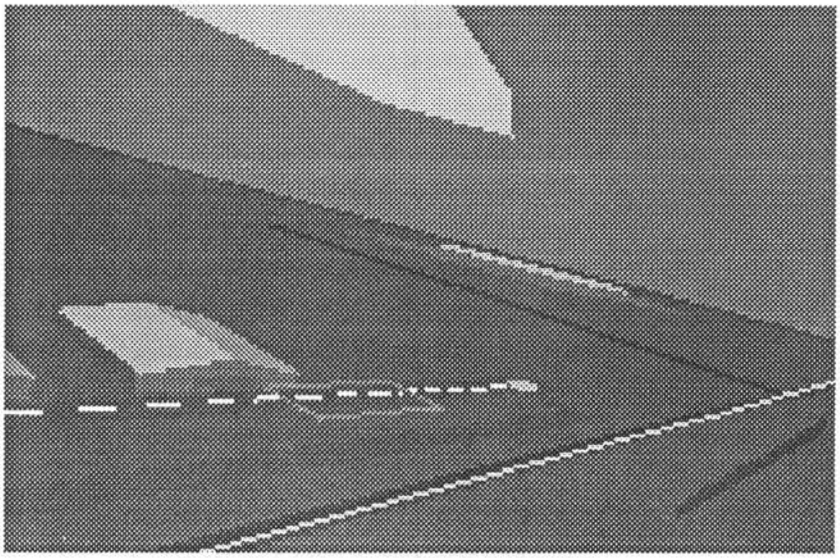


Figure 1.9—A view from the cockpit in the flight-simulator game *Chocks Away Special Missions* from the Fourth Dimension. Why are there are no curves in the scene, not even in the clouds? Because the animation uses vector graphics in real time and curves would slow the process unacceptably. The image appears coarse because it was grabbed from the screen as a sprite and the application—also for reasons of speed—uses mode 13, only 320 x 256 pixels

viewpoint of the roads, fields and buildings visible from the cockpit depend entirely on the location, altitude and orientation of the simulated aircraft and this is of course under the user's control! The subjects are stored as mathematical models and their images are generated in real time as vector graphics. The level of detail of these graphics in some applications depends on the processing power available; an ARM3 can draw more in a given time than an ARM2 can.

'Virtual reality', which has attracted much attention in the media, uses the same principles; it is a 'simulated world' which the operator views generally using a personal stereoscopic display. This senses movements of his head, giving the user the impression that he has entered the virtual world. It has serious uses—for example an architect can take a client on a simulated tour of a proposed building before it has left the drawing board and psychologists are researching its uses in diagnosing and treating such disorders as schizophrenia. It also has leisure uses in highly sophisticated computer games, especially fantasy adventures, often employing several players. Domark's *3D Construction Kit* describes itself as a virtual-reality system for the ARM machines, although it uses the regular monitor and interfaces rather than a stereoscopic viewer.

Since pixel graphics offer fine and direct control of detail with a resolution of one pixel, they are the basis of many computer art packages. Many of those for the ARM machines are restricted to mode 15 which offers a palette of 256 colours with effectively thousands more colours available as dither patterns. Stunningly lifelike pictures can be built up by patient users having an artistic bent; reproductions of the old masters sometimes even hint at the texture of the oil paint with which the original was created!

The artwork used in most fast-action arcade games uses exactly the same principles. The level of detail available is staggering, the picture quality being similar to that obtainable from a video cassette recorder. Indeed some recent games include scenes that were actually 'filmed' using a video camera and converted to graphics using a video digitiser. Material from photographs can also be converted to pixel graphics using a scanner and much image enhancement software is available which can sharpen pixel graphics giving more realistic displays both on screen and in printouts.

The use of mathematical models was mentioned in connection with three-dimensional vector graphics and interactive animations. They also find application in pixel graphics. Given a mathematical model of a scene including details of the light sources present, suitable software can recreate the image that would be captured by a camera at a known location. This technique is known as *ray tracing* since it effectively examines the rays arriving at the viewing point and traces them back to each light source, taking into account reflections from the various surfaces in the scene. The process is time-consuming but does result in stunningly beautiful pixel graphics of imaginary objects. Several examples are included in the colour plate section.

Animation involves throwing a series of images on to the screen in rapid succession. The need for both speed and detail dictates that most animation is pixel-graphics based. This leads to the problem of storage space. Smooth, lifelike animation demands at least 12.5 frames per second. If your animation uses whole mode 15 screens, you will be able to store only four such frames on an 800K disc, giving an animated sequence lasting less than a third of a second! However, by a combination of such techniques as file compression, the use of less demanding screen modes and the repeated use of small sprites it is possible to build up interesting animated sequences lasting several minutes which will still fit on an 800K disc. These techniques will be described later.

Conversion between graphics types

To convert vector graphics to pixel graphics is simple. The computer performs this conversion every time it sends vector graphics to the screen or to the printer (unless it is a PostScript printer). To convert a vector graphics image (or part of it) to a sprite, you simply set the required screen mode, put the graphics on the screen and then 'grab' the appropriate part of screen memory. The 'Get screen area' or 'Snapshot' facility in *Paint* is ideal for this. If you don't have sufficient memory to allow your vector graphics software and *Paint* to multi-task, you can always use the *SCREENSAVE command or a screen grabbing utility and load the resulting sprite into *Paint* for editing later.

Conversion in the opposite direction—from pixel graphics to vector graphics—is rather more problematic. You might be excused for thinking it is totally impossible, but there are software packages that perform this conversion, albeit with limitations. Generally known, understandably enough, as 'tracing packages', they work by detecting the edges of the coloured areas in the sprite and constructing best-fit approximation outlines which are then translated into vector graphics data format. Many sprites—especially those containing an abundance of pixel-sized detail—are unsuitable for this conversion. The best results are from sprites containing simple blocks of pure colour. The items depicted in the image are converted on an 'as seen' basis and not necessarily as you or I would convert them. Thus in a picture of the sun rising over the horizon the sun is usually interpreted as an orange semi-circle and not as a full circle half concealed below the horizon. Moreover the order of the resulting objects in the stack might not necessarily reflect the spatial relationships of the subjects depicted, although most packages do succeed in achieving a fairly sensible order. These packages are considered in more detail in Chapter 12.

2 Understanding Vector Graphics

Although most vector graphics software—and certainly *Draw*—is very easy to use, even by folk who have no knowledge of how the applications work, some understanding of the fundamentals will help any user to get better results. It will also suggest all sorts of fascinating techniques and areas for fruitful experimentation.

This chapter considers some of those fundamentals. It does not purport to be a user manual for *Draw* since many of *Draw's* features are not covered. But it does describe the general features of vector graphics software, taking its examples so far as possible from *Draw* since all users of RISC OS have access to it and *Draw* ably exemplifies this kind of application. Where *Draw* is unusual, reference will be made to other applications as well.

Path objects

First, a little bit about terminology. *Draw* refers to vector graphics as *path objects.* This is to distinguish them from three other types of object (sprites, text lines and text areas) which it allows you to incorporate in your artwork. In this chapter we are concerned mainly with path objects and text objects. Path objects are so called because they follow the idealised version of the path taken by the pointer during the creation (and any subsequent editing) of the object. Paths may be *open*, that is they begin in one place and end elsewhere, or they may be *closed*, that is they end on the same point where they began and therefore form a continuous circuit as in a rectangle or an ellipse. They may be divided up into *sub-paths* by *moves*; a number of open or closed sub-paths may still constitute one path object as we shall see.

In common with most other packages *Draw* recognises three types of segment: straight lines (which *Draw* calls simply *lines*), curved lines (which *Draw* calls simply *curves*) and *moves* which represent a break in the line, moving the pointer elsewhere; path objects can consist of a mixture of lines, curves and moves, but there are some restrictions on the use of moves. More on this later.

Segments and points

In geometry classes at school you may have been taught that a straight line is the shortest distance between two points. In vector graphics that is still true, but the two points are just as important as the line between them. Indeed there would be no line at all without the two points because it is the co-ordinates of these two points that are stored to mark the start and end positions of the line. All vector graphics by definition consist of vectors, often called *segments*, which may be straight lines, curved lines or moves that do not draw a line at all. All segments have a point at each end, and a point may be shared by two adjacent segments. The points at each end of a segment are sometimes called *endpoints* to distinguish them from another kind of point which we shall meet shortly.

Although the points themselves are invisible in the finished graphics, during drawing or editing they are normally represented on the screen as solid squares (some packages sometimes use other shapes); the co-ordinate position is the centre of the square. A point may be displayed in a different colour to indicate that it is selected and therefore amenable to various editing operations. A selected point can be dragged (using Adjust) to a new location with the pointer, this resulting of course in the repositioning of the segment(s) of which it is part. The co-ordinate data in the stack are updated automatically.

Lines

To draw straight lines is simple, although it varies in detail from package to package. In *Draw* you click on the uppermost icon in the toolbox—see Figure 2.1—or choose Line in the Enter menu; the pointer changes to a cross-hairs; you now move it to where your first line is to start. Now

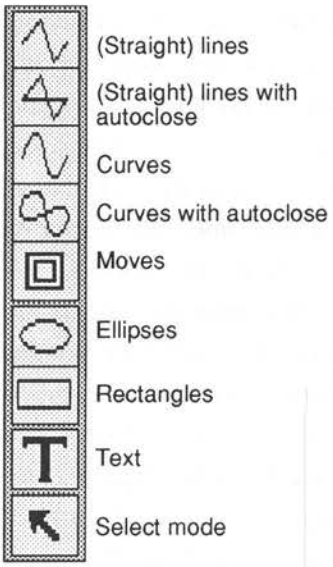


Figure 2.1—the toolbox icons in *Draw*

click Select; this places your first endpoint and starts the line; the line is shown in red and is mobile. The start of the line remains on the point and its end follows the pointer as you move it around.

When the pointer is positioned where you want the line to end, click Select again. This creates the second endpoint and fixes your line which now changes to pale grey. It also provisionally starts a new line from that point; you can continue adding more lines, end to end, *ad infinitum*—subject, of course, to the availability of computer memory, but as each extra segment adds only 12 bytes to the data stack, you will probably have room for as many of them as you could wish!

Incidentally, pressing Delete will delete the last segment and pressing Delete again will delete the previous segment; if you keep on pressing

Delete, eventually you will delete the point with which you began the object. Pressing Escape during object creation will cause the entire unfinished object to be lost. In RISC OS 3, pressing F8 will undo your last action; if your last action was the deletion of an object, F8 will restore it, provided the 'undo buffer' is big enough to hold its definition.

When you have finished your object, click Adjust. This returns the pointer to its normal shape but leaves the object in an editable state with the lines shown in grey and the points in blue. When you click Adjust on a point, you select it and it changes to red; you also select the segment which *ends* on that point and it too turns red. Clicking Menu provides a range of editing options available on the selected line or point; there is more on this later. You can move the selected point around by dragging it with Adjust; for one-pixel movement hold Adjust on the point and press the arrowed cursor key for the direction in which you wish to move it.

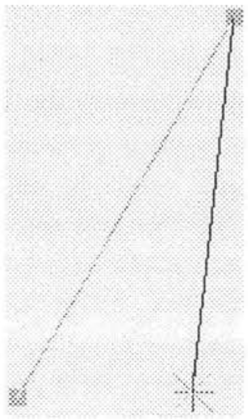


Figure 2.2—line drawing in *Draw*

When you are satisfied that your points are correctly placed, click Select again. This finishes the object; all points disappear and the lines assume the thickness, colour and any other style attributes that you selected (more on these later). It also provisionally starts a new path object with the cross-hairs ready to position its first point. See Figure 2.2.

Curves

Curves are created in a similar way to lines. Click on the third-from-top icon in *Draw's* toolbox or select Curve in the Enter menu. When you click on Adjust to indicate that you have finished placing points, you will notice two differences from lines: firstly the curves have automatically followed the smoothest possible route between the points and secondly that each curve has not *two* points, but *four*. It has an *endpoint* at each end just as a straight line has and these are shared with adjoining segments (if there are any). But it also has two *control points*, one near each end. These are shown in orange (they don't change colour) and are linked to the endpoints by a grey line which we shall call the *link*.

This curve system (known as the *Bezier* curve system) is widely used in graphics. A straight line, we reminded ourselves earlier, is the shortest distance between two points. A curved line could therefore be defined as a line between two points that does not take the shortest route. Indeed, the more curved you make it, the longer the distance you would travel if you followed it from one endpoint to the other. Bezier curves mathematically define curved lines in a very simple and economical manner; they only require the computer to store the coordinates of the two control points as well as the two endpoints.

While the object is editable, the control points can be dragged with Adjust just as the endpoints can. Clicking adjust on a control point does not change its colour, but instead selects the endpoint to which it is linked. Incidentally whenever you drag an endpoint, you will find that any control points linked to it move an equal distance in the same direction. This is useful, since the overall shape of the line at that point is preserved.

There are two aspects of a control point's effect on the curvature of the curve to which it relates.

Firstly, it controls the *direction* in which the curve is heading at the endpoint. This it controls in the simplest possible manner: the link forms a tangent to the curve at the endpoint. In other words at the endpoint where the link meets the curve itself, they are pointing in precisely the same direction. See Figure 2.3.

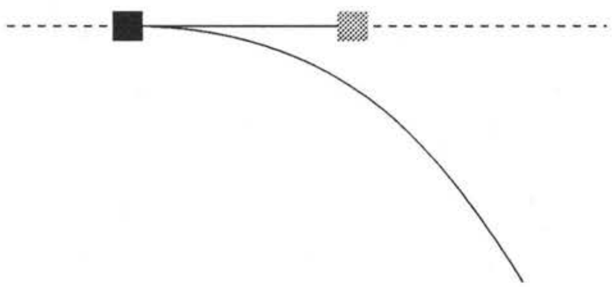


Figure 2.3—the link is a tangent to the curve at its endpoint

Secondly, the length of the link controls the length of the curve. Extend the link and you will extend the curve, so making it more tortuous; shorten the link and you will shorten the curve and thereby make it straighter. There is an irony in this, since extending the link will reduce the tightness of the curve by giving it a greater distance in which to change direction; the curve will more gradually veer away from the direction of the link and towards its destination. If the link passes close enough to the other endpoint, the curve may even kink in order to take in the length that you have added. On the other hand, if you keep the link very short, the curve will bend very rapidly into whatever direction is needed to take it to the other endpoint. If all this seems complicated, don't worry. A brief 'play' and you will find that it becomes intuitive. See Figure 2.4.

Auto-close

The second and fourth icons in the toolbox

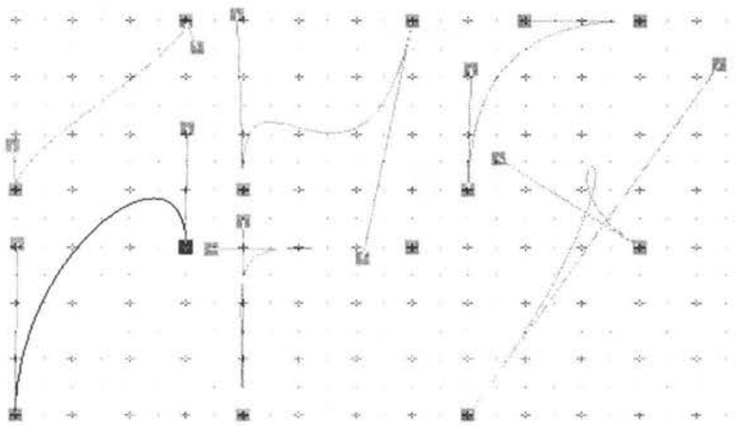


Figure 2.4—how the control points in Bezier curves control the flow of the curves

produce more straight lines and curves respectively, with one difference—auto-close. The principal difference is that when you click on Adjust (or double-click Select) to indicate that you have finished plotting points, an extra line or curve will be drawn from your last endpoint to the first point in the path (or in the the last sub-path if your object has included any moves). This closes the path (or sub-path) and makes your object (or part of it) a continuous circuit. You can draw a triangle very quickly, for instance, by clicking on the second icon and then clicking select where you want the three angles and finally clicking on Adjust.

Incidentally, the sixth and seventh icons in the toolbox are short-cuts for producing accurate ellipses (including circles) and rectangles (including squares) respectively. Both are plotted using just two points. For ellipses your first click locates the centre and your second is a corner of the boundary box, *i.e.* it represents both the horizontal and vertical limits of the object. For rectangles the two presses locate diametrically opposite corners of the figure. The objects produced are conventional closed-path objects with points and they can be edited in the normal way; they are identical to objects which you could have drawn point by point but which would probably have taken you rather longer.

Moves

Moves allow you to break off a path and resume it elsewhere. Click on the fifth icon from the top in the toolbox; move the pointer to the starting point of the next line or curve and click Select to place your point; you will find that *Draw* has automatically reselected lines or curves, whichever was previously in force. In the finished object there will be an apparent 'gap' in the path where you introduced the move; no line will appear. Sequences of lines and curves separated by moves are sometimes called *sub-paths.*

There are several restrictions on the use of moves:

(i) A path object cannot begin or end with a move, since either would be futile. Clicking on the move icon has no effect unless you are have one or more segments already laid down (you are in the middle of creating a path object). And if having drawn a few segments you click on the move icon, place the pointer, click Select again to fix the end of the move and then click on Adjust to finish your object, you will find that both the move and the last point have disappeared.

(ii) One move cannot follow immediately after another; that would be futile as well.

A number of seemingly extraordinary things happen when moves and auto-close are used together. You may even be tempted to think you have discovered a bug in *Draw*, but in fact there are good sound reasons behind all that *Draw* does; when you understand the rules, you can turn them to your advantage to create all manner of fascinating effects. The rules governing moves and auto-close are:

(iii) A move cannot form part of a closed path (such as a triangle, rectangle or an ellipse).

(iv) If you click on the move icon while auto-close is selected, the auto-close routine will be applied to the segments you have just drawn, making them into a closed sub-path. But the object itself remains open; you can move the pointer and resume drawing.

(v) If you select auto-close in an open-path object that includes one or more moves, the final segment will close to the start of the latest sub-path and not to the starting point of the object.

(vi) If when editing a closed-path object you change a line or a curve to a move (most software does not allow this!) the result will be two closed paths separated by the move.

Multiple closed-path objects

If you bear these rules in mind, *Draw's* behaviour becomes predictable and very flexible. Applying rule (iv) for instance, if you select auto-close lines and draw four points and then select move, auto-close will be applied immediately to the four points you have created and the path between them will be closed to form a quadrilateral, just as if you had clicked Adjust instead of selecting move. The difference is that the object itself is not closed; you still have the cross-hairs pointer and when you next click select you will start another closed path which is still part of the original object. And you may continue *ad infinitum*, creating as many closed paths as you like (subject to memory availability!) all within one object, all linked (or should I say separated?) by moves. Making all your closed paths part of one object in this way, instead of making each an independent path object, saves memory and storage space; it is also drawn more quickly.

Of course one of these closed paths may overlap another. It may partially overlap it or it may be drawn entirely enclosed by it or enclosing another one, or both. Now this in itself is not particularly interesting unless you choose to give the object a fill colour (more on fill colours later). With *Draw's* default setting of the strangely named winding rule (more on this later too) the fill colour will not be applied in the overlapping area which will remain transparent. So, if one closed path totally encloses another (provided it is part of the same object) the inner path behaves as though it were a hole or window in the surrounding outline through which you can see any other objects that are behind. This is a very useful technique in creating animations or other special effects; you can put windows in a car or a house through which you can see what is inside or on the other side (see Figure 2.11).

The facility also makes alternating patterns such as chequerboards very quick and easy (Figure 2.5). They can be constructed as series of rectangular sub-paths that overlap each other; the fill colour (black) is applied only in areas covered by one but not two rectangles.

Moves in edits

Draw's editing facilities allow you to change a line or a curve in a finished path object to a move, provided of course that the segment concerned is not the first or last in the path or adjacent to an existing move—to comply with rules (i) and (ii) above. Remarkably, as outlined

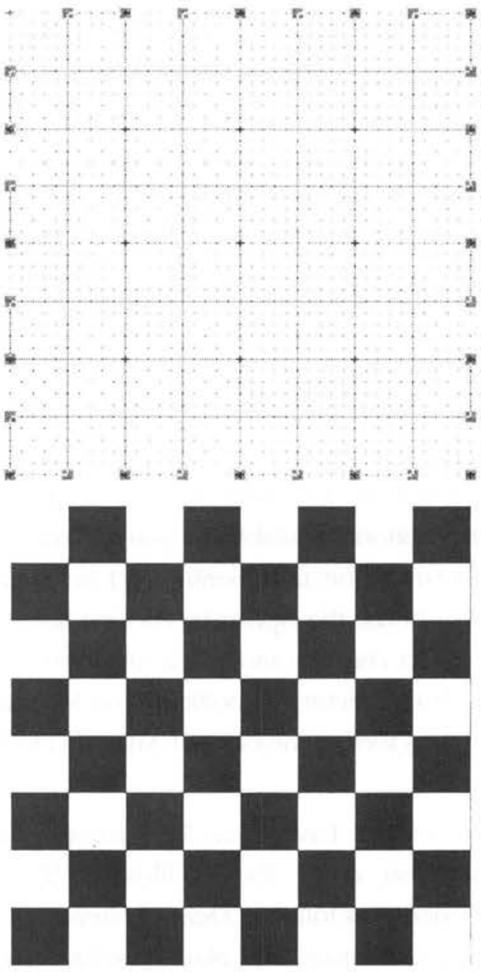


Figure 2.5—chequerboard type patterns are simply constructed using multiple-closed-path objects. Despite its appearance this object consists of four horizontal and four vertical rectangular sub-paths which overlap alternately

in rule (vi) above, you are allowed to change a line or a curve to a move even when it forms part of a closed path. Since by rule (iii) a move cannot form part of a closed path, we are confronted by a paradox and indeed *DrawPlus* and *Vector* disallow such an edit. In fact Draw resolves the paradox in a thoroughly sensible if subtle manner. Nevertheless the results of this operation have confused experienced users: even the writers of edition 2 of the *Archimedes User Guide* (page 114) failed to understand what was happening.

Try the following. Select auto-close line and create a rectangle; we'll name its four successive points A, B, C and D in the best geometry-class tradition (Figure 2.6a). Click Adjust on D; this selects point D and the line CD; click Menu. One of the options available to you on the editing menu (more on this menu later) is Change to move. Choose this option by clicking Select on it. The object changes and you now appear to have three lines meeting at point A: you have a new line CA as well as the original AB and DA (Figure 2.6b).

How can you have three lines meeting at one point? You can't; it's an illusion. What has happened is as follows. *Draw* behaves as though you selected move after placing point C and so it autocloses C to A, creating the new line CA. The move takes the pointer to the next point on the stack which is D. On the stack it finds the line DA which was drawn when you originally autoclosed the rectangle. It regards this as the next line in the object and draws it. But, remember rule (vi) above: when you edit a move into a closed-path object, *two* closed path objects result. So it now adds a closing line back from A to D. Despite appearances you have *two* coincident lines between A and D! You can prove this by clicking on D to select the line AD.

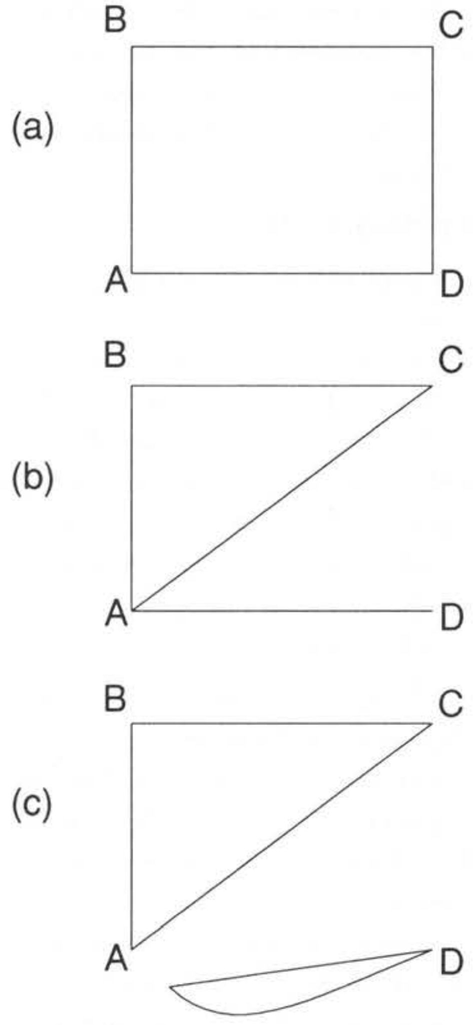


Figure 2.6—the seemingly bizarre effect of editing line CD in the closed path ABCD to a move. In (b) three lines appear to meet at point A. In fact the line AD is two superimposed lines forming a two-line closed path as demonstrated by changing one to a curve (c). Moreover, its point A is a separate point from the original, as demonstrated by dragging it.

Change it to a curve and drag a control point to separate the two coincident lines. And if you drag point A, you will find that this is in fact a separate point from your original starting point A; it just happened to be positioned directly over it (Figure 2.6c).

Editing path objects

Of course it goes without saying that you can use the toolbox (or the Enter menu) to switch between (straight) lines, curves and moves whenever you wish as you draw your object, subject to the restrictions concerning the use of moves outlined above. But you can also edit a finished path object, deleting or inserting segments and changing lines to curves and *vice versa* and either to moves (if appropriate) after you have finished drawing it.

You enter editing mode automatically after you have clicked Adjust to finish the object you are creating. You can also edit a finished object either by selecting the object (see later on select mode) and clicking on the Edit option in the select menu or by clicking on any of the line or curve icons in the toolbox and then clicking Adjust over one of the points in the object you wish to edit. You cannot edit an object if it forms part of a group; you must ungroup it first (see later on the Select menu).

When an object is editable its lines and curves are displayed in pale grey, the endpoints in blue and control points in orange; a selected line appears in red and a selected endpoint in red. These colours may appear different if displayed against a coloured, *i.e.* non-white, background.

If a line appears to be invisible and is displayed in yellow when selected, then you have two lines coincident.

While the object is editable, you can select points by clicking Adjust on them. You can move the selected point by dragging it with Adjust or by pressing the arrowed cursor keys while pressing Adjust. The line or curve which ends at that point is also selected. Obviously, if you select the starting point of an open-path object, no lines or curves will be selected. You will not be able to select or edit a move which ends on a closed sub-path.

While a point is selected, clicking menu will produce a menu of operations which may be applied to the point or segment selected. Some of the options will be greyed out, *i.e.* not selectable, because they are not applicable.

Change to curve, Change to line, Change to move

At least one of these options will be greyed out since it would be pointless to change the selected segment to whatever kind of segment it already is. And you will be unable to change the selected segment to a move if it is the first or last segment in the object or if it is adjacent to a move. If you change a line or curve to a move in a closed path, remember that you will end up with *two* closed sub-paths linked by the move.

Add point

This adds a new point half way along the selected segment, converting it to two segments of the same type (line or curve) as the original. You cannot add a point to a move since this would create two adjacent moves; you must change it to a line or a curve first. You can then change one of the daughter segments back to a move if you wish.

Delete segment

This deletes the selected segment and endpoint and joins the free segment on one side to the free point on the other. If the segment concerned is the last in the object and the previous segment was a move, then the move and its point are deleted as well.

Flatten join

This is only available in RISC OS 3 versions of *Draw*. It adjusts the two control points linked to the selected point so that they are exactly opposite each other; that is, the two tangents form a continuous smooth transition, 'flattening' the curve.

Open path

This is only available if the selected segment is part of a closed path. The closed path is opened at its starting point, whether or not this was the selected segment. You cannot open a closed path at any point except its starting point. This operation creates a new finishing point for the previously closed path, near the starting point.

Close path

This is only available if the selected segment is part of an open path. It is the reverse of the Open path option; it removes the final point of the path and ends it instead on the starting point. Unlike the auto-close option, it does not create a new segment to join the last point to the first.

Close path may be applied to an object consisting of a single curve, so that it curves round and ends on its starting point. Oddly, it may even be applied to an object consisting of a single (straight) line, producing a line having zero length that begins and ends on the same point.

Enter coordinate

This displays the current co-ordinates of the selected point and allows you to enter new co-ordinates. It also allows you to choose between Imperial and metric measurements (inches and centimetres). All values entered are rounded to the nearest multiple of 1/46,080 inch (or its metric equivalent), this being the length of the internal unit in which *Draw* works. We shall look at this unit in more detail and use Enter coordinate later in this chapter.

Snap to grid

In RISC OS 3 versions of *Draw* this facility, provided in the Select menu, is duplicated in the (path) Edit menu. It forces all points in the object on to the nearest grid points.

Styles

When you have finished editing your object, click select anywhere in the *Draw* window to see how the finished object looks. The object is now redrawn in the chosen *style.*

Ultimately, each path object consists of a number of lines, curves and moves and a number of enclosed areas. Each object is allowed one set of *style* attributes. These are line width, line colour, fill colour, line pattern, join style, start cap style, end cap style and winding rule. So, if you have ever been tempted to think of a line as a fairly humble sort of beast, just a path between two points, now is the time to think again. In graphics even straight lines can become exceedingly complex. In *Draw*, whose vector graphics facilities are fairly basic, lines can have a mind-boggingly vast range of properties, as evidenced in that it takes at least 108 bytes of data to describe a path object consisting of a straight line between two points.

Only one setting of each style may be used in each path object, so you cannot have some black lines and some red ones within the same object; nor can you have some lines 1 pt wide and some 4 pts wide. If you want to vary styles in this way you must create several separate path objects; you can always 'group' them afterwards so that they are treated as though they were one object.

Let's consider the styles of a line as exemplified in *Draw*. When any of the path object tools are selected, the new path object being created will be given the style settings currently set in the top half of the Style menu selected from the application's main menu. If you wish to change the style of an object that has already been finished, you must first select it (see later) and make your choice from the style menu while the object remains selected. Note that while an object is selected, the Style menu always indicates the styles currently in force in it; this may be useful if you have forgotten, say, the line thickness used in a certain object and now you wish to create another like it.

Line width

Firstly a line has a *width*. Since this is generally a comparatively small measurement it is expressed in points (pts). The width menu offers a choice of useful values and a writable field is provided allowing you to enter values not offered as standard. Fractions of pts are acceptable; remember that *Draw* converts all measurements to multiples of 1/640 pt, even though the width menu displays widths to two decimal places only. When an object is magnified, the width of its lines (if finite) as well as its co-ordinate values is multiplied by the magnification factor to keep the proportions consistent. You can check this for yourself by drawing a few lines with the width set to 0.25 pt and then selecting the object and magnifying it by 0.5. Leaving the object selected, the width menu will now show the value 0.12; line width itself has been subjected to the magnification process.

When a line has a finite width, it is assumed to be balanced evenly on either side of the infinitesimally wide path that links the co-ordinates of the two end points.

Thin lines

The first item in the width menu and the one that is selected by default happens to be the only exception to the rule concerning the magnification of line widths. It is simply listed as *thin*. In vector graphics a *thin line* has special properties. Its absolute width varies depending on the output device; it is always reproduced as the thinnest line which the output device, be it a screen or a printer, can handle. In other words, it is always one pixel wide, irrespective of the pixel size. The width of a thin line is not affected by zooming or magnification. If you zoom up from 1:1 to 8:1, the whole object will appear much bigger of course, but if composed of thin lines, they will appear the same width as they did at 1:1. And if you zoom out to 1:8, the thin lines will still be the same thickness even though the image of the object is much smaller.

This leads to several important practical considerations that you should bear in mind whenever you use thin lines.

Firstly, because their absolute width depends on the vagaries of the output device, beware of using *visible* thin lines as a part of any artwork that must be viewed at several different sizes. (They are fine in artwork when invisible, *e.g.* having no line colour, but used to bound an area of fill colour.) If the thin line itself is part of the detail, it will change the proportions of that detail when the graphics size is changed.

Secondly, the thin line that looks substantial enough on the screen (where it is 1/90 inch wide) may be almost invisible when reproduced on paper using a 600 dots per inch laser printer (where it is 1/600 inch wide). I once used some vector graphics clip art which incorporated an abundance of very fine detail and was horrified to find that in the printout all that intricate detail had disappeared! On investigation I found that the detail consisted of thin lines in white against a black background; in my laser printer the toner had simply 'filled in' the 1/600 inch wide gaps that the lines should have left. With the line width changed to 0.5 pt (4 pixels wide at 600 dpi) the printout was perfectly satisfactory.

On another occasion before I appreciated the subtleties of thin lines, I designed a newsletter in *Draw* whose layout included some thin lines, this time black against the white background. I used the PostScript printer driver, sending the output to a disc file which was taken to a commercial printing shop for professional film setting. On the films that were returned my thin lines had vanished altogether! The film setter had a resolution of 1200 dots per inch and I imagine that the resulting 1/1200-inch-wide lines were just too thin to be seen, perhaps too thin even for the resolution of the film!

Ironically, just as a thin line can cause problems through its propensity to disappear (a function of the mechanics of the printing process), so thin lines are sometimes used to *stop* details from disappearing.

Try the following in *Draw*. Select the rectangle icon, set line colour to None but select a good bold fill colour like black, red or green. Now draw a rectangle that is at least an inch wide but only two or three pixels high on the screen (at 1:1), so it's short and wide. This rectangle represents an important detail in some graphics that are going to be displayed at various magnifications, some very small. So, let's see how it looks at 1/10 of its present size. Select it and magnify it by 0.1. You should find that it has disappeared altogether! This is not surprising when you consider that its new height to the nearest pixel is 0. Now magnify it by 10 to restore it to its original size. It will reappear, just as it was when you drew it. Select the line icon, set line width to thin and line colour the same as the fill colour used in the rectangle. Now draw a single line across the width of the rectangle, about half way up. Of course, when finished you won't see this line at all, since it blends into the rectangle behind it. Select both objects and group them and then magnify by 0.1 as before. This time a thin red object will be visible. It is in fact the thin line, being reproduced one pixel wide as *Draw* is bound always to do with thin lines. At least *something* is now visible to represent your important piece of detail at low magnifications. The thin line is said to have *hinted* your piece of detail. The Acorn outline font system employs hinting using thin lines in this way to ensure that parts of characters are always represented at even the smallest typesizes (provided of course that the font in use has been properly hinted!).

Line colour

The line colour is, of course, the colour of the line itself. The line colour dialogue box in *Draw* allows you to enter a line colour that is 'None'. When this is selected the line itself will be transparent or invisible. It will still be there and can be edited in the normal way, including being changed to a 'proper' colour.

Why should anyone want an invisible line? There are in fact many reasons. Often the line will be required only as the boundary of a shape having a fill colour, for example a green rectangle. You could produce a green rectangle by making the line colour the same as the fill colour, but then you would also have to take the line width into consideration. Also, if you are creating a series of objects each having a different fill colour, every time you changed fill colour, you would need to change line colour as well. It is far easier to leave the line colour transparent. You may also find transparent line colours and fill colours useful if you are creating an animation with a package such as *Tween*. *Tween* produces animations from *Draw* files and one of its useful features is that it will produce intermediate frames from 'key' frames. But the key frames must contain the same objects in the same order in the stack; only their positions, sizes, proportions and colours may vary. So if you want an object suddenly to appear out of nowhere, one way is to make it initially transparent and at the required frame give it real colour. You can make it vanish again by the reverse process.

Some users find the colour dialogue boxes in *Draw* confusing, since they offer a choice of 16 colours plus transparent in any screen mode (even in the 256 colour modes). They also appear to offer a means of fine tuning the colours that are present, by dragging red, green and blue sliders as in the RISC OS palette, except that 256 levels of red, green and blue are accessible. And yet, when you try using those sliders in a 16-colour mode in RISC OS 2, no new colours are produced. (In RISC OS 3 some new colours do appear.) So what is happening?

You must for a moment return to the fundamental principles of vector graphics. In chapter 1 the very second sentence I wrote about vector graphics is that its very nature makes it 'independent of screen mode or printer resolution'. The limitation to 16 or 256 colours is a function of the screen mode and therefore irrelevant in vector graphics. Just as *Draw* stores its co-ordinate data in units that are far too small for any practical output device to resolve, so it also stores its colour information in greater detail than much hardware can handle. In the data stack *Draw* stores each line colour (and each fill colour) using neither the Desktop nor the RISC OS colour systems but by allocating one complete byte to each of the primary colours red, green and blue—plus a transparency flag. In other words *Draw* supports full 24-bit colour giving you theoretically a choice of 256 × 256 × 256 = 16,777,216 different colours, plus the transparent option. So you can manipulate those sliders with confidence knowing that in the data stack the objects you create will be allocated whatever subtle shades you have specified. With most hardware, however, RISC OS is limited at present to a maximum of 256 colours and so, when reproducing objects having colours that do not correspond to any of those available in the current palette, it will substitute whatever colour in the current palette is the closest approximation.

RISC OS 3 represents an advance on RISC OS 2 in that it can use dither patterns (chequerboard-type patterns of pixels using two or three palette colours) as though they were palette colours. In this way *Draw* can simulate hundreds of colours in 16-colour modes and thousands of colours in 256-colour modes. Of course, if you are fortunate enough to own a video enhancer that implements 24-bit colour or a colour printer that handles true 24-bit colour, then you should be able to view your graphics creations in their full glory.

When selecting colours using *Draw's* colour dialogue boxes, you must remember to click on the OK icon or your selection will be ignored.

Fill colour

The fill colour is the colour enclosed by the object (but this is affected by the winding rule described later). Objects do not have to be closed paths for the fill colour to become visible, since a virtual line between the start point and end point delimits the filled area on an open-path object. Even an object consisting of a single straight line is allocated a fill colour, although it will not be seen since a single straight line does

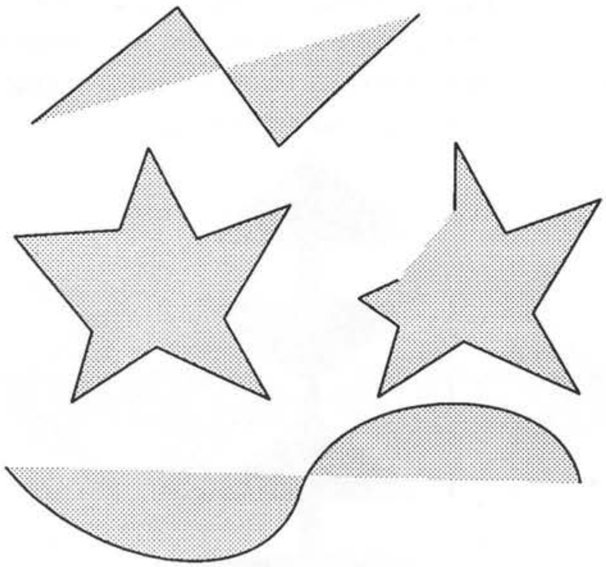


Figure 2.7—in *Draw* both open-path and closed path objects can have fill colours. In open-path objects the fill colour is delimited by a virtual line between the starting point and the end point

not enclose an area. Two straight lines or a single curve can enclose a filled area; on the open side the colour fill will extend to a virtual straight line between the object's endpoints (see Figure 2.7).

The dialogue box is exactly similar to the line colour dialogue box, supporting 24-bit colour plus a transparency flag labelled 'None'. If the fill colour is transparent, the background colour or other objects behind the present one will of course be visible through the object.

Pattern

Beside normal solid lines, a line can have a pattern such as dotted, dashed, dot-and-dashed. These patterns are useful in graphic design for borders and edges, for example in coupons to be cut out. A less obvious use is exemplified in Figure 2.8 which is a reproduction of a traditional African pattern; giving the solid shapes dotted lines in the fill colour simulates a jagged edge effect representing weaving.

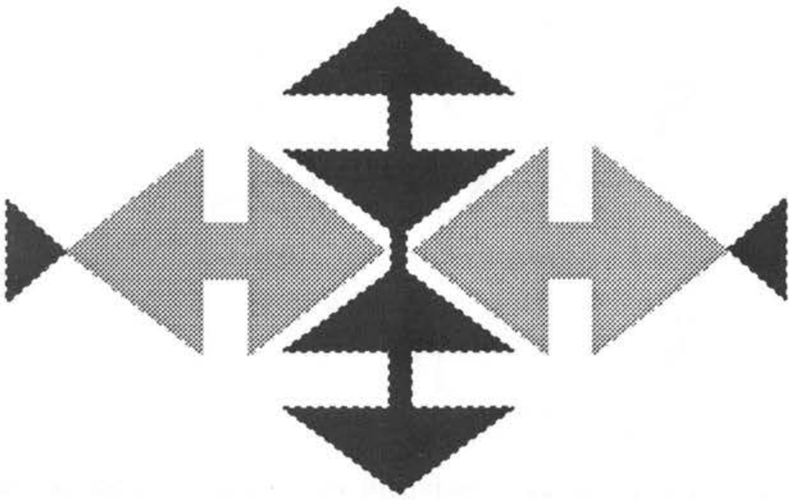


Figure 2.8—in this reproduction of a traditional African pattern dotted lines have been used to simulate weaving

Join

In vector graphics most segments form part of an object consisting of many such segments linked end to end with points shared between them. Usually a point marks the spot where there is an angle between the segments (if there were no angle there would be probably no reason for the point to exist). These angles are called *joins* and *Draw* offers a choice of three styles: mitred, round and bevelled, although it is only in segments that are more than a few pts wide that the differences become apparent (Figure 2.9).

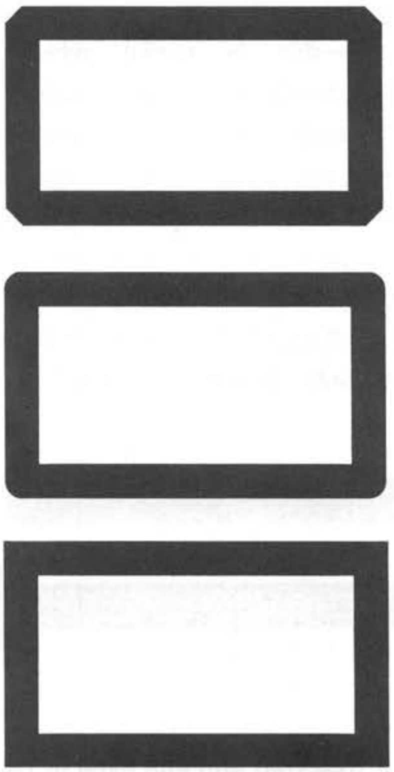


Figure 2.9—three join styles offered in *Draw*: from top to bottom bevelled, rounded and mitred. Line width is 16 pt.

Start cap, End cap

As was mentioned above, most segments in vector graphics are joined on to other segments; often objects consist of closed paths such as rectangles, triangles and ellipses. In these all segments are joined to other segments at both ends and there are no 'loose ends'. But loose ends can and sometimes do occur and are called *caps*. They occur at the start and end of open-path objects and also at the starts and ends of sub-paths. *Draw* offers a choice of style for these. *Draw* distinguishes between the cap at the starting endpoint and that at the finishing endpoint, so you may select a different style at each end—this is useful when drawing arrows!—although the same choice of styles is offered for each. The difference between the butt and the square styles is simply that additional length is added to the square end (Figure 2.10). In the triangular caps, you may specify the width and height; the units are the line width. As with joins, the effect of these styles will usually be apparent only if the lines have appreciable width.

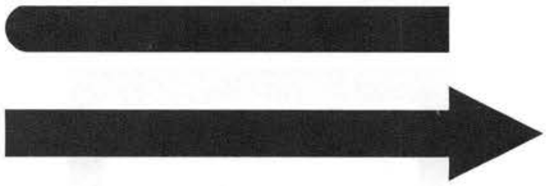


Figure 2.10—Start and end caps in *Draw*. Top: rounded (left) and butt (right). Bottom: square (left) and triangular with default width of × 1 and height × 2 (right)

Winding rule

This curiously named style causes much confusion, even amongst experienced graphics software users. The winding rule determines whether an enclosed area is filled with the currently selected fill colour or left transparent—in other words, it determines what is reckoned to be inside. (If the fill colour transparency flag is set, clearly the problem does not arise.) Winding rule is of interest where there is a loop in the boundary or in multiple-closed-path objects where one closed path is wholly or partly enclosed within another.

In *Draw* as in most vector graphics software there is a choice of two winding rules. The default is the 'even/odd' winding rule, which is probably the more useful of the two. Imagine that you have entered into the world of graphics that you are creating and that you propose to walk from outside the boundary of the object into the enclosed area whose fill colour—or lack of it—we are considering. According to the even/odd winding rule, the area is filled only if on your journey into it from outside the object you crossed an *odd* number of paths. If you crossed an even number of paths, the area is left transparent, ignoring any fill colour selected. So a hole in the centre of the object remains transparent; this may be useful when drawing buildings or vehicles having windows through which you can see other objects behind (Figure 2.11). Or when drawing letters of the alphabet which contain holes, such as 'O' and 'b'. But an island in that hole would be filled.

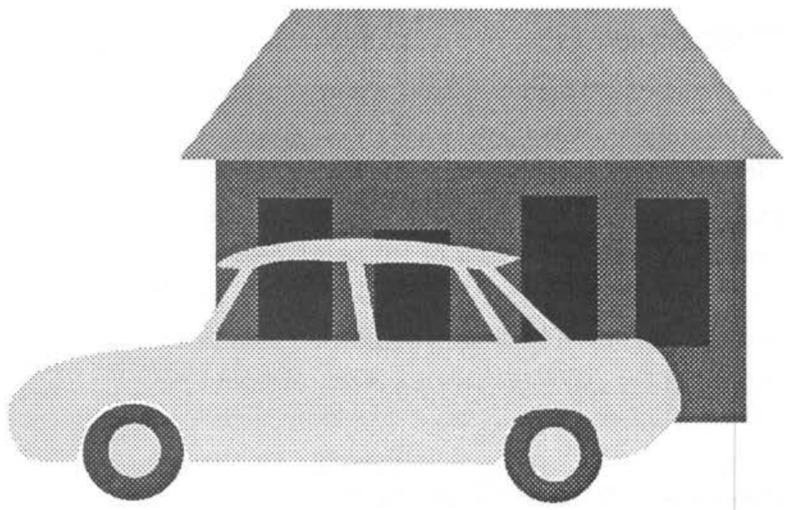


Figure 2.11—careful use of the winding rule allows you to have transparent holes or windows within objects—as in the car body

The other winding rule is called 'non-zero'. This rather suggests that if the number of paths enclosing the area is any other than zero, that area is filled. And when you use it, so it first appears to be. But it is a little more complicated than that (and this is where the 'winding' comes in). It all depends on the *direction* (clockwise or anticlockwise) in which you created those paths enclosing the area. If all were made in the same direction, the number of paths is added up; since it must always be greater than zero, the area is always filled. But if any were made in the opposite direction, you must subtract them from the number in the first direction. And if the number of paths in each direction is equal, they cancel out and the effective number is—zero. So the area is not filled, but left transparent.

Operations on whole objects

So far we have considered the creation and editing of path objects in *Draw.* These are operations generally involving parts of objects such as points and individual segments. But there is also a wide range of operations that can be applied to whole objects or groups of objects. First you must enter *select mode* by clicking on the lowest icon in the toolbox or by clicking on the entry Select in the main menu or by clicking on the entry Select all in the Select menu. When in select mode, you must select the object (or objects) concerned.

Selecting and deselecting objects

There are four ways an object can become selected:

(i) If you wish to select one object only, move the pointer over that object and click Select. Any objects that were previously selected will be deselected.

(ii) If you wish to select an additional object when one or more objects are already selected, move the pointer over the object concerned and click Adjust. That object will now be selected in addition to those that were previously selected.

(iii) If you wish to select *all* the objects in the drawing, click Menu, choose the Select menu and click on Select all.

(iv) If you wish to select all the objects in a certain area of the drawing, move the pointer to a nearby position where it is not over any object, hold down Select and drag out a box'. All objects overlapping the box when you release Select become selected. Any objects that were previously selected and which are wholly outside the box are deselected. Alternatively you may drag out your box with Adjust. As before this will select all objects overlapping the box, but any other objects that were previously selected will remain selected.

Objects that have been selected are distinguished by the appearance around their edges of a dotted red boundary box having two 'handles' on its right-hand edge. The significance of these will be explained shortly.

Objects can be deselected in the following ways:

(i) To deselect just one object, move the pointer over it and click Adjust. If other objects were previously selected, they will remain selected.

(ii) Click Select over another object. That object becomes selected and all others are deselected.

(iii) Click Select with the pointer not over any object. All objects are deselected.

(iv) On the Select menu click on Clear. This deselects all objects.

(v) On the toolbox click on a different icon so that you leave Select mode. This deselects all objects.

If you have many objects overlapping it can sometimes be very difficult to select the particular object you want. A useful trick is to use the box method (iv) to select all objects in the vicinity and then click Adjust on all the unwanted objects to deselect them until only the one you wish to select remains selected. But sometimes you may have no option but to drag a collection of selected objects away from the rest of the drawing and pull them apart in order to reach and select the one needing attention.

So let us see what you can do to the objects you have selected.

Moving objects

You can *move* selected objects anywhere in the drawing. Place the pointer in the red dotted boundary box and hold down Select. The colour of the dotted boundary will change; moving the mouse will now drag the entire object; release Select when it is in the required position. If other objects are also selected, they too will be moved an equal distance in the same direction.

Scaling objects

You can *scale* selected objects, *i.e.* change their size. Move the pointer into the lower of the two handles on the right-hand edge of the dotted red boundary. Hold down Select and drag the object's bottom right-hand corner to its chosen new position. The top left-hand corner will remain in its original location so that the object's height and width are changed. If the object is a path object or a sprite, it can be inverted or mirror-imaged by dragging its bottom over its top or its right-hand edge over its left-hand edge. If other objects are selected, they too will be scaled in equal proportion. This operation is inevitably crude. It is fine where you wish to resize an object to exactly fit in a certain area. But if you want to magnify an object by a precise factor, for example, if you want to double its height or reduce both height and width by a factor of exactly 5, you should use the magnification facility described below.

Rotating objects

You can rotate path objects. If you are using the version of *Draw* provided with RISC OS 3 you can also rotate text objects and sprites, but not text areas. Move the pointer into the upper of the two of handles on the right-hand edge of the dotted red boundary. When you drag this handle with Select, you will find that the selected object rotates as though it were pivoted at its mid-point. If other objects are selected, they too will be rotated through an equal angle.

This operation, like the last described, is inevitably imprecise. It is fine where you wish to rotate the object so that it points roughly in a certain direction. If you want to rotate an object through a precise angle, such as 60 degrees, you should use the Rotate menu option described below.

Saving selected objects

While one or more objects are selected, they may be saved to a file quite independently of the whole drawing. From the main menu choose Save and then Selection. The resulting file may be saved to disc or into another *Draw* window or on to the *Draw* icon to create a new window containing only the selection. It may also be saved into another compatible application such as a picture frame in a DTP package.

Changing object style

As mentioned earlier, while an object is selected, you can change its style characteristics if it is a path object or a text object. Changes selected from the Style menu, such as new line widths, line colours or fill colours, will be applied to all selected path objects.

The Select menu

The following operations are accessed from the Select sub-menu of the main menu. Some options may be unavailable and will be shown in pale grey type. If no objects are selected, normally only the top item, Select all, will be available.

Select all

This causes all objects in the drawing to become selected. It is useful if you wish to shift the entire contents of the drawing, perhaps if it was overlapping the printable area of the page, for example.

Clear

This causes all objects in the drawing to become deselected.

Copy

This creates a copy of each selected item, slightly offset from the original. After the operation, the copies are selected, not the originals. If you click on the menu with Adjust, the menu will remain open allowing you to repeat the operation and build up large numbers of identical objects!

It is not widely known that the copy facility can be used to copy objects between windows. When the objects that you wish to copy have been selected in one window, move the pointer into the destination window and open the Select menu. Just the Select all and the Copy options will be available; clicking on Copy will copy the selected objects from the other window.

Note that the copied objects are fully independent of the original; subsequent edits to either are not reflected in the other. (Some packages such as *Vector* offer another object copy facility called *replication* in which the copies are clones of the original and cannot be edited; if the original is edited, however, the changes are reflected in the copies. *Draw* does not offer replication.)

Delete

This removes all selected objects from the drawing and is the only way of erasing unwanted objects. After Select all, of course, it deletes all objects, leaving you with an empty window. No warning is given, so use this facility with care. In RISC OS 3 the Undo option (F8) will restore accidentally deleted objects, provided the Undo buffer is large enough to contain their definitions.

Front

This moves selected objects to the top of the stack, *i.e.* to the front of the drawing so that they obscure any other objects which they overlap.

Back

This moves selected objects to the bottom of the stack, *i.e.* to the back of the drawing so that they are obscured by any other objects which they overlap.

Group

This is only available if more than one object is selected. It groups the selected objects so that they are subsequently treated as though they were one object with a common boundary box.

It is a useful means of ensuring that groups of objects stay together.

In the grouping operation the group is always moved to the top of the stack, that is, to the front of the drawing. If you wish the group to stay at some depth in the stack, you must subsequently select all the objects that need to be in front of the group and apply the Front option.

Ungroup

This reverses the effect of the last described operation, separating a previously grouped object into its constituent objects. In RISC OS 2 it is only available if a single grouped object is selected. In RISC OS 3 any number of selected groups may be ungrouped simultaneously. The ungrouped objects retain their places in the stack.

Edit

If a single path object is selected, the option enters edit mode so that the object can be edited, *e.g.* its points and control points become visible and can be adjusted. It is useful if you have had difficulty in entering edit mode from enter mode. In RISC OS 3 you are subsequently returned to select mode with the edited object selected.

If you are using the version of *Draw* supplied with RISC OS 3, you can also edit the contents of a single text object. A dialogue box will appear containing the wording of the text object (reproduced in the system font). You can move the caret using the cursor keys and delete or insert characters at any point. Pressing Return will close the dialogue box and the contents of the text object will be updated.

Snap-to-grid

This only applies if the grid is displayed and grid lock is active (select the Grid menu from the main menu). In RISC OS 2 it forces each point in the selected path objects to snap on to their nearest grid points, providing a useful means of ensuring that vertical and horizontal lines are true. In RISC OS 3 it forces the bottom left-hand corner of the object's boundary box on to its nearest grid point, moving the object as a whole but leaving the relative positions of the points in the object unaffected. The identically named option in the Edit menu forces points on to the nearest grid points.

Justify

This only applies to groups of objects. It provides a means whereby the constituent objects in each group can be neatly aligned. Horizontal alignment is by left-hand edges, right-hand edges or centre line. Vertical justification is by top edge, bottom edge or centre line.

The following five items are only present in the Select menu in the version of *Draw* supplied with RISC OS 2. In versions supplied with RISC OS 3 they are in a separate menu, the Transform menu, accessed from *Draw's* main menu.

Rotate

This rotates the selected objects through a specified number of degrees anticlockwise. For clockwise rotation insert a minus sign before the figure or subtract the required angle from 360. Objects rotate about their mid-point. This option allows more precise rotation than dragging the upper handle on the right-hand side of a selected object's boundary box. Text objects and sprites can only be rotated if you are using RISC OS 3.

X scale, Y scale

These options scale the selected object(s) by an exact figure rather than the inexact amount obtained by dragging the lower handle. The figure is in proportion to the current size, so entering 2.0 will double that dimension and entering 0.5 will halve it. The X-scale relates to the horizontal axis and the Y-scale to the vertical. Line thickness is unaffected.

Line scale

This scales the line width in the selected path objects. Scaling is relative to current line width and thin lines are unaffected.

Magnify

This combines X scale, Y scale and Line scale into one operation, the same magnification factor being applied to all co-ordinates and finite line widths.

If you are using RISC OS 3 you also have the following three items available from the Select menu.

Interpolate, Grade

To use either of these facilities you must have two similar path objects selected and grouped. They must be *similar* in that they have the same number of points and segments and the same sequence of segment types. Both operations create a number of new objects having characteristics intermediate between the two 'parent' objects. The new objects have their size, shape, orientation, location, line width, width and height of triangle start and end caps, line colours and fill colours intermediate between those of the two parent objects. See Figure 2.12.

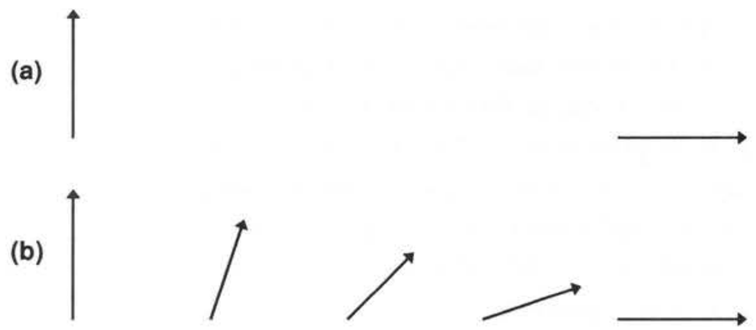


Figure 2.12—operation of the Interpolate facility in *Draw*: (a) shows two open-path objects, grouped; (b) is what happens when they are interpolated (4 gradings)

In practice the only difference between Interpolate and Grade is that Interpolate produces one extra object and this lacks a fill colour, although this will only become apparent when working with closed paths having fill colours. In the Interpolate option, the number that you enter is the number of interpolations or gradations required, *i.e.* one less than the total of objects with which you wish to finish. See Figure 2.13.

These features can be used (with some limitations) to perform 'inbetweening' in animations, that is the creation of intermediate frames. They can also be used to create three-dimensional graphs and highly intricate patterns and to add graded colour fills to drawings: you copy the object to be filled, reduce its size,

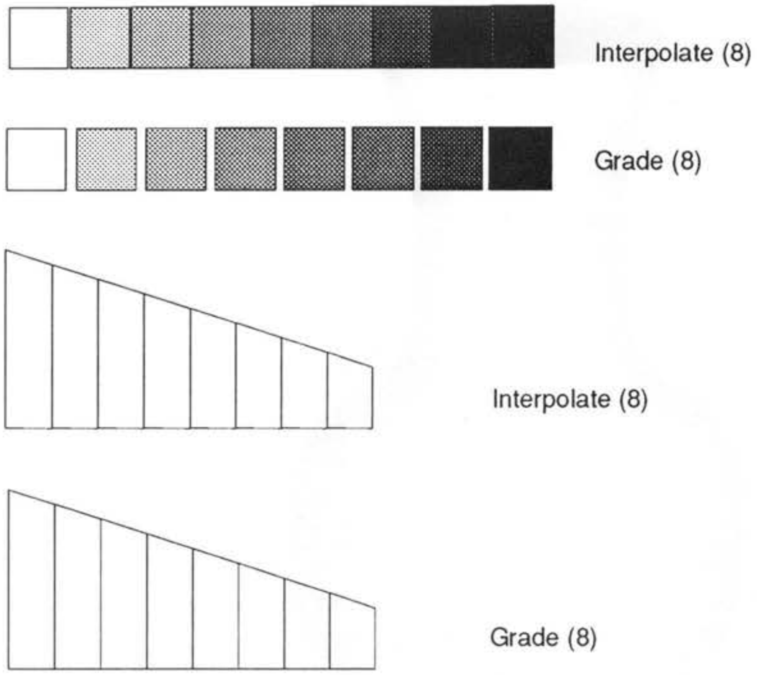


Figure 2.13—the differences between interpolation and grading depend on the nature of the objects involved: with closed-path objects (above) the number of new objects is different; with open-path objects (below) there is in fact no difference

change its fill colour and use the grade facility. Figure 2.14 gives an example.

If you have an early A5000 machine which incorporates version 0.76 of *Draw*, you should beware of a bug in the Interpolate/Grade routines. If the parent objects include open paths or sub-paths, each interpolated object is drawn twice, the two drawings exactly overlying each other, new lines being drawn to join one of the new objects to the next in the series on one side and the other new object to the next object on the other side. Probably this bug will not cause much trouble, since the facility is more useful when applied to closed paths and closed sub-

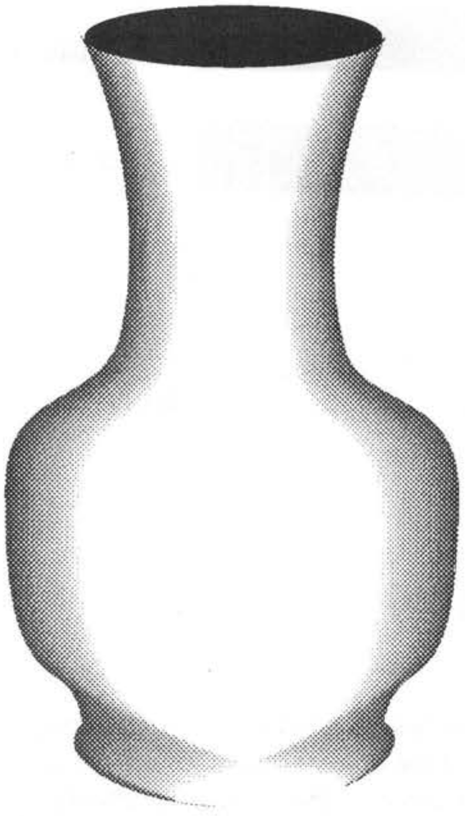


Figure 2.14—An example of the kind of visual effect that can be obtained very easily using the Grade facility in the RISC OS 3 version of *Draw.* A smaller white copy of the original dark outline has been placed over it and interpolated to give a total of 16 objects

paths and with these Grading and Interpolation proceed correctly.

During the interpolation/grading process, however, the closed paths are opened, albeit imperceptibly. Although this is quite normal in this type of operation, it has a consequence that becomes apparent if you should subsequently decide to isolate the original parent objects, edit them and then repeat the interpolation/grading

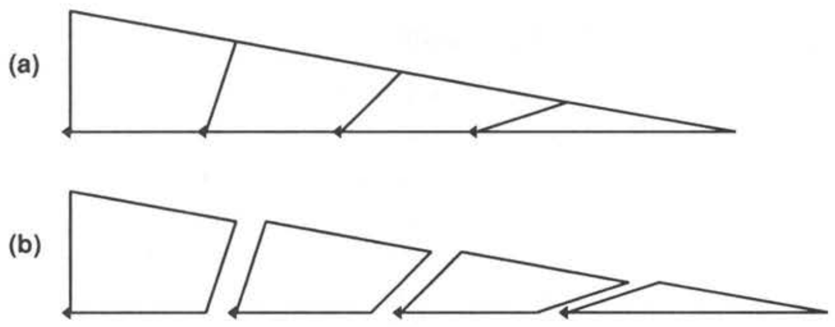


Figure 2.15—the effects of a "bug" in version 0.76 of *Draw* on the interpolation shown in Figure 2.12. (a) shows the drawing after interpolation and (b) shows constituent objects separated

process. Unless you remember to close their paths first, the bug will come into action and give messy results.

Convert to Path

This is only available if one or more text objects is selected. Although there is no immediate difference in the appearance of the object, except that the anti-aliasing disappears, it has been converted to a group of path objects, each character being a separate path object. This conversion is not reversible and the former text object can no longer be edited as a text object. Text that has been converted to vector graphics in this way is, however, suitable for various operations that cannot be applied to text. For instance, the characters can be separated and moved about individually; they can be given separate line and fill colours and their line thickness can be edited. This kind of font manipulation is considered in more detail in Chapter 4.

Sprite objects

Sprites (pixel graphics) can be imported into *Draw* windows where they form sprite objects. They cannot be created or edited in *Draw*—those are the functions of its companion application *Paint* (see Chapter 8). But they can be moved, scaled and flipped about either axis in the same way as path objects. In RISC OS 3, but not RISC OS 2, they can also be rotated.

Text handling

Although this book is primarily concerned with graphics, most drawings incorporate text as labels and titles. *Draw* recognises two totally different types of object which handle text: text objects and text areas, sometimes called text column objects. They have little in common except that they both display text using the installed font manager and fonts.

Text areas are really beyond the scope of this book. The text is prepared in a separate text editor such as *Edit* and dragged into the *Draw* window; embedded commands in the text file allow changes of style and format wherever required. Indeed it offers many of the facilities of desktop publishing, but without the convenience. The process and the commands are described briefly in application manuals and in more detail in my book *Budget DTP on the Acorn Archimedes* (Dabs Press, 1992).

Text Objects

Text objects, in contrast, are ideal for providing titling, labels and annotations in graphics. A text object consists of a single line of text, uniform in style. It is created using the text entry facility in *Draw*, either by clicking on the T icon in the toolbox or by selecting the Text option in the Entry menu.

Position the pointer where you want your text to start and click Select; the pointer changes to a caret. If necessary you can reposition the caret by moving the pointer elsewhere and clicking Select again. You can now enter text at the keyboard. As you type, the characters appear on the screen at the position of the caret which moves to the right (just as it does in *Edit* and DTP applications). No editing facilities are provided apart from the Delete key.

To finish your text object press Return or click Select. If you press Return the caret moves to a new position below the start of the object you have just finished. This is useful if you need to build a column of neatly aligned text entries. If you click Select, the caret moves to the current pointer position. But beware! If you press Escape while creating a text object, your unfinished text object is lost irretrievably. You also lose an unfinished text object if you click on the Select icon in RISC OS 2, but not in RISC OS 3.

Text object styles

At any time while entering text you can change any aspect of its style. Click Menu and select Style. The lower half of the style menu applies to text objects and offers you a choice of font name, font size, font height, text colour and background colour. Any text you have already typed will be rewritten in the new style.

Font name

This leads to a menu of available fonts, the first item always being *system font*, the font normally used by the computer in, for instance, directory displays. This is followed by an alphabetical list of the outline fonts present in the Fonts directory when *Draw* was started up. *Draw* version 0.44 (supplied with RISC OS 2) can only handle a maximum of 63 outline fonts; if your Fonts directory contains more than this, only the first 63 (in alphabetical order) will be displayed.

Font size

This font size is actually the font *width*. The menu offers the following font sizes: 8, 10, 12, 14 and 20 pt. A writable icon allows you to enter any other size you may wish and is by default set to 6.40 pt. Fractional point sizes are permitted; the outline font manager rounds fractional sizes to the nearest 1/16 pt. If you enter a value in the writable icon, you must still click on it to select it.

Font height

The font height menu is the same as the font size menu but the default size is 12.80 pt. This is because the default font, the system font, is normally used with its height set to double its width, giving it the proportions with which we are most familiar from mode 0, 12 and 15 screens. Whenever you choose a font size, the font height is automatically set to the same value. If you wish to use a font at a height different from its width, you must subsequently amend the height value. This allows you to enter text that is extended (wider than natural) or condensed (narrower than natural).

Text colour, Background colour

These call up colour selection boxes which are like the line colour and fill colour boxes already described in every respect. The text background colour is white by default. If you are creating a text object over a background having a different colour, such as a graphic with red as its fill colour, you should set the text background colour to correspond with this background. This is for the purpose of anti-aliasing (for which see Chapter 3). If you omit to do this, on the screen the characters will appear to have faint white outlines. This faint outline, however, does not appear in printouts, since anti-aliasing is not applied in printouts.

As with path objects, you cannot mix styles in one text object, so you cannot have some words in italics and some in Roman. To create such a line of mixed-style text you must either use a text area or use two or more text objects, which can be moved carefully together and then grouped and justified.

Editing text objects

If you have RISC OS 2, you will be unable to change the wording in a finished text object. So, if you spot a mistake, you can only correct it by deleting the entire text object and creating a new one. If you have RISC OS 3, you can edit the wording by selecting the object and choosing the Edit option from the Select menu. A separate writable field appears in which you edit the text using the cursor keys. When you press Return the contents of the path object are updated.

Text objects can be subjected to most of the operations on whole objects outlined above. They can be moved and scaled; when you scale a text object the width and height of the constituent characters are automatically adjusted to fill the new outline. You can easily create very condensed (narrow) or expanded (wide) characters in this way. This makes text objects very useful in titles since they can be scaled to fill the space available. Text objects can be rotated if you have RISC OS 3.

If you re-load a previously saved *Draw* file in which a text object uses a font that has subsequently been deleted from Fonts, the affected text object will be displayed in the system font.

Internal, print, screen and pointer resolution

As we have seen in this chapter, vector graphics consist of lines and shapes plotted on the monitor screen or on the printer's 'virtual screen' from co-ordinate data stored in the data stack. The very high precision of this data is one factor which makes vector graphics so useful.

You may wonder why we should be so concerned about precision in computer graphics, when—as Figure 1.2 revealed—much of it represents compromise. This is because vector graphics is not usually an end in itself but a means to an end. You may use it simply to create pretty pictures on your monitor screen if you wish, but most often it forms a part of some serious task in industry or commerce. And accurate measurement is crucial in most of the everyday applications of vector graphics. You may be producing an engineering drawing for a vital component in the aerospace industry. Or you may be laying out a display advertisement to fill a page in a glossy magazine. In either of those applications the *size* of the finished artwork is critical. Note that measurements in graphics always relate to the size of the final *printout* (assuming that the printout scale setting, if any is provided, is 100%) and not that of the monitor display. This is, of course, because the size of the monitor display is affected by the monitor type and adjustment and also the setting of the zoom facility. (With a 14-inch monitor in mode 20 at a zoom factor of 1:1 the screen display is generally about 20% larger than life.)

While it is true that vector graphics are plotted on the monitor screen using the screen co-ordinate system, the software itself is not confined to the OS units. Most applications use 'internal units' in the stack to store the dimensions of the objects in the artwork. The user, however, is able to enter measurements in more familiar units such as inches or centimetres, the computer converting these to the application's internal units whenever necessary and, of course, converting these in turn to OS units when plotting. All these conversions are 'transparent', that is, as a user you need not be aware that they are happening. Indeed, internal units are only mentioned here because they determine the limit of resolution of the graphics—the shortest line that can be drawn or the narrowest line width—and this in turn affects the software's versatility.

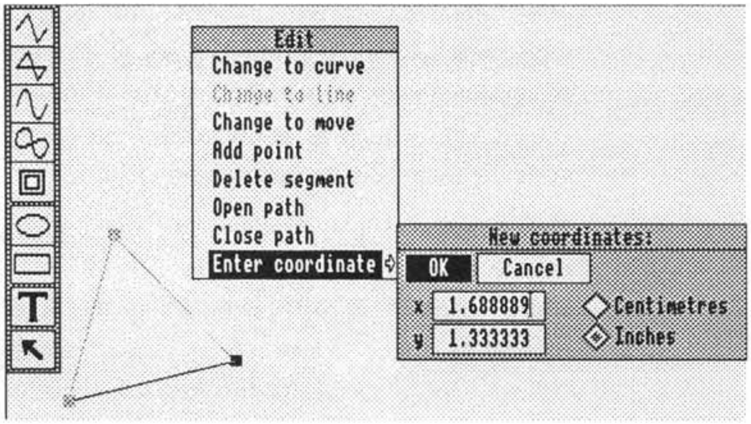
The *resolution* of the software's internal data normally far exceeds that of any practical output device (the monitor, plotter or printer). As was mentioned in Chapter 1 *Draw* uses an internal resolution of 1/46,080 (=0.000 021 701 389) inch. Why this seemingly strange figure? Two factors make this a particularly convenient unit for computer graphics. Firstly, in the most commonly used screen modes (12, 15, 20 and 21) the pixels are nominally 1/90 inch wide, there are 640 such pixels across the screen and 1280 OS units across the screen, making each OS unit nominally equal to 1/180 inch. Now 1/46,080 = 1/180 × 1/256. So, to convert internal units to screen co-ordinates, the computer must divide by 256; this is one of the easiest and fastest calculations for a 32-bit microprocessor, simply a right shift of 8 bits! And speed is critical since this division must be performed numerous times in plotting each line. Secondly, very small measurements in graphics commonly use the point (pt), a unit which originated in the print industry; 1 pt is commonly taken as equal to 1/72 inch. And 1/46,080 inch is a convenient fraction of 1 pt: it is 1/640 pt, *i.e.* 1/640 × 1/72 inch. This is useful when working with font sizes and line widths which are normally measured in pts.

A consequence of this very high internal resolution is that there is always a dramatic loss of accuracy at the interfaces with the outside world. For example, in *Draw* on a page 8 inches wide, there are theoretically 368,640 positions across the page at which you could place an object. However, a 300 dots per inch printer recognises only 2400 such positions across the page and even professional filmsetting equipment capable of 2400 dots per inch would offer only 19,200 positions across the page. So, whenever artwork is printed, there is a process of approximation in which a spread of internal co-ordinate values is rounded to the nearest available location on the output device.

The monitor display involves an even greater degree of approximation. In modes 12, 15, 20 and 21 each of the 640 pixels across the screen represents (at a zoom of 1:1) 512 possible different locations along the horizontal axis in *Draw*.

Now because RISC OS is a WIMP (windows, icons, menus and pointer) environment, the number of pixels per inch on the screen is even more significant than you might at first suppose. Although you may consider the screen as primarily an *output* device, the one which gives you your first glimpse of the stunning graphics you are creating, it is also an integral part of the graphics *input* system. Most drawing operations take place at the *pointer*. And no matter how carefully you manipulate the mouse or trackerball or whatever other graphical input device you favour, the pointer's resolution is limited to whole pixels. This must be so since it, like everything else in the screen display, is composed of pixels. So the screen resolution of 90 dots per inch is also, by definition, the resolution of the pointer, your drawing device.

You can check this very simply if you wish. Start up *Draw* with a fresh window and draw one or two random lines. Ensure that the zoom is at its default setting of 1:1. Click adjust on one of the points; this will select it, *i.e.* turn it red. Now click menu and move right on the item, Enter



x 1.688889
y 1.333333
Centimetres [✓ Inches]

Figure 2.16—the Enter coordinate facility in *Draw*

coordinate'. A dialogue box will appear displaying the current co-ordinates of the selected point as stored in the data stack. Typical figures might be x=1.688 889 and y=1.133 333 (inches)—see Figure 2.16. Make a note of the co-ordinate values and close the dialogue box by clicking on the OK icon. Now, let's move that point one pixel up the screen. The easiest way to do this is to hold down adjust and press the arrowed cursor-up key; you will see the screen update with the point one pixel higher than before. Now check those co-ordinates again; you should find that y is now 0.011 111 greater than previously. So you moved the point a total of 0.011 111 inches up its y axis, *i.e.* one pixel = 1/90 inch.

If you wish, you can repeat the experiment dragging the point by very carefully manipulating your mouse or trackerball until the screen updates. But the result will be exactly the same. You may indeed find that you cannot get such small moves from your mouse. If so, you should use the *Configure* application to set your mouse to its slowest or second-slowest speed; this is really essential for graphics work.

Since the pointer only offers a means of accessing one in every 512 locations (along one axis), how can you make use of the software's very fine precision? There are several ways:

(i) The zoom facility in most applications allows you to enlarge the screen display—in *Draw* the maximum enlargement is 8:1 but many graphics packages allow greater enlargements (*e.g. Vector* allows up to 20:1). Zooming only affects the screen display; it does not affect the data in the stack describing the artwork. It is therefore a fundamentally different operation from magnification which changes the size, *i.e.* the co-ordinate data stored in the stack, of objects in the artwork.

Zooming in not only allows you to see more detail, but also gives you greater precision than at a scale of 1:1. If you repeat the demonstration described earlier, this time using a zoom setting of 8:1, you will get very different results. A one pixel move upwards from y=1.133 333 now takes you to 1.134 722, a move of just 0.001 389 inch. This is, as we should expect, 1/720 inch, *i.e.* 1/8 of 1/90 inch.

(ii) If the zoom facility is still not sufficiently precise, you may consider temporarily magnifying the object on which you are working. The magnification facility, as explained in the previous chapter, multiplies all co-ordinates and line thicknesses by the chosen magnification factor; it therefore changes the size of the object concerned. So when you have finished work on the magnified object, you must remember to restore it to its original size.

Obviously magnifying by, say, 100 will allow you to work on the object with 100 times the precision that was previously available. So in *Draw* with the zoom on 8:1 *and* the object magnified by 100 each pixel of pointer movement which previously represented 0.001 389 inch ought to become 0.000 014 inch when the object is subsequently magnified by 0.01 to restore it to its original size. In practice, however, this would exceed the limits of resolution; the final co-ordinate values would therefore be rounded to the nearest multiple of 1/46,080 inch.

(iii) The Enter coordinate dialogue box in *Draw*, as its name suggests, allows you to enter co-ordinates as raw data from the keyboard. You can place a caret in the field and edit it as you wish. Indeed you can enter as many significant figures as you wish, but *Draw* will always round off your entry to the nearest multiple of 1/46,080 inch (or its metric equivalent). So, if you enter a co-ordinate value as 3.14159265 inch, don't be surprised if you subsequently discover that it has become 3.141590 inch.

(iv) For most purposes you will probably not need anywhere near as fine a resolution as 1/46,080 inch. Probably what is more important for you is that lines are properly straight, equal distances are consistent and items are neatly aligned. For this reason most packages provide a *grid.* The grid is a rectangular or triangular pattern of accurately positioned dots which may be optionally reproduced on the screen but which is not reproduced in printouts. It can be used for measurement or location purposes as you draw. The isometric (triangular) pattern is helpful in producing perspective-free represent-ations of three-dimensional objects. Most importantly, a grid lock facility forces any points, *i.e.* starts and ends of lines, that you draw on to the nearest grid point. In *Draw* the grid interval can be any specified Imperial or metric measurement. For example, the *Cir-Kit* set of electronic circuit diagram symbols is designed for use in *Draw* on a 1/16 inch grid and this makes it possible to create circuit diagrams very rapidly indeed, since the symbols automatically centre themselves neatly over the conductors. Clearly, when objects are locked on to a grid in this way, the Enter coordinate facility will confirm that the points are always correctly positioned, especially if the grid units are a convenient multiple of 1/46,080 inch.

3

Outline Fonts and the Font Editor

You may wonder what a chapter on the Acorn outline font system and the font editor application, *FontEd*, is doing in a book on graphics. It is included for three reasons:

(i) Since text plays a vital rôle in most graphic design, some understanding of the capabilities of the outline font system is essential for a grasp of graphics in the widest sense.

(ii) Much of the following two chapters will be concerned with the graphical manipulation of text imported as graphics from the outline font system. Therefore some understanding of its principles will be helpful.

(iii) The font editor, *FontEd*, provides an interesting example of established ARM vector graphics software that contrasts with *Draw*. It is, of course, a highly specialised application intended only for the creation and editing of fonts for the outline font system, but it can also be used to export characters to *Draw* and compatible applications as vector graphics.

This chapter does not purport to be a user manual for *FontEd* nor to give any kind of grounding in the fundamentals of typeface design which is a highly intricate and specialised art. It briefly describes the main features of the application in the hope that you may find it a useful adjunct to your graphics software. And if you ever need to edit a font, you may find this chapter useful. First we need to consider the outline font system in general.

The Acorn outline font system

Outline font systems store the shapes of text characters as vector graphics, *i.e.* as outlines. The reason for the popularity of outline font systems in computers and laser printers, as we saw in Chapter 1, lies in the ease with which vector graphics can be magnified or reduced. One set of outlines provides every possible type size, the computer (or printer) simply multiplying the stored co-ordinate values by a constant appropriate to the required size. This results in beautifully smooth curves and oblique strokes at even the largest sizes, a delightful contrast from the jagged edges obtained when reproducing bitmapped characters at large sizes (see Figure

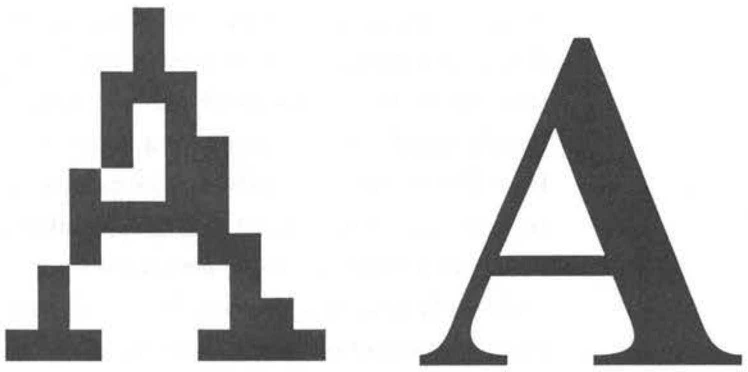


Figure 3.1—the reason for outline font systems! Both characters are 180pt Trinity Bold 'A's, that on the left obtained by enlarging a 12pt bitmap and that on the right using the outline font system in the normal manner

3.1). Moreover, with some limitations, condensed (narrow) and expanded (wide) characters can be created by using different constants for the vertical and horizontal axes. Users of RISC OS 3 will find that text can be easily rotated and oblique (inclined) print can be obtained from Roman (upright) fonts by specifying the required degree of obliqueness.

In the Acorn outline font system your fonts must normally be stored in an application directory named Fonts and RISC OS must know where to find this before it can run any applications that use outline fonts. If you have a hard disc, it is most convenient to locate Fonts along with System in the hard disc's root directory so that it is seen at start-up. RISC OS 3 has its own ROM-based resources filing system which includes not only the principal applications (including *Draw*, *Edit* and *Paint*) but also a fonts directory containing the outline font families Corpus, Homerton and Trinity. To edit these you will need a version of the font editor later than 0.27. If you edited these fonts, however, you could not save them back into ROM; to use them you would need to save them in a separate disc-based Fonts directory. RISC OS 3 supports such a separate directory to hold disc-based outline font collections and if the same font names appear in both ROM and disc, the disc-based versions are used in preference. If you wish to use both the original ROM-based fonts and your disc-based edited versions, you must therefore save these with different names. The versions of *Draw* and *Edit* supplied with RISC OS 3—unlike earlier versions—impose no restriction on the numbers of outline fonts that are present.

Most fonts come in families containing a Roman, a bold, an italic and a bold italic variety, although the names used for these varieties vary somewhat from font to font. For example, the term *oblique* is used in place of *italic* where the design is simply a leaning version of the Roman. Within Fonts each font family is stored in a subdirectory having the generic font name such as Trinity, Corpus, Paladin or Vogue. Each font family directory normally contains one subdirectory for the medium or Roman varieties and one for the bold varieties. Each of these subdirectories may contain a further subdirectory for the italic varieties. Each font itself consists of two files, one called Outlines which contains the graphics and one called IntMetrics which contains data concerning the dimensions of the characters. Figure 3.2 illustrates the organisation of the Acorn Trinity font, which is typical, containing a total of eight files in five directories.

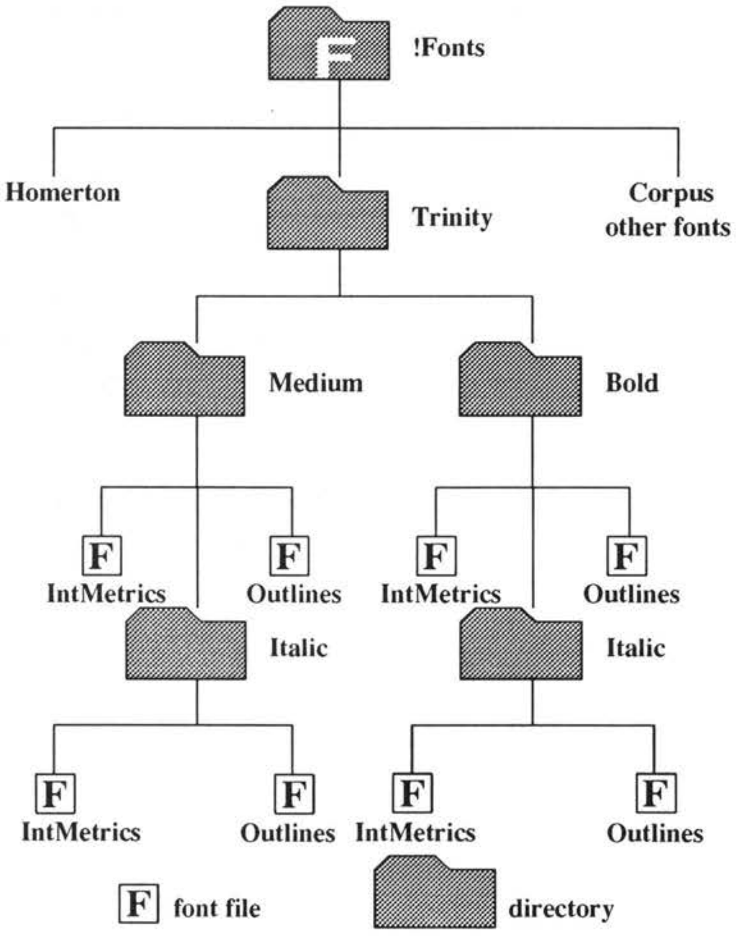


Figure 3.2—typical font file organisation in the Acorn outline font system

Oddly enough, when an application uses an outline font for pure text purposes, it does not plot the characters on the screen in the way that vector graphics are normally plotted. It would take the computer far too long to plot the thousands of characters in a well filled screen of, say, 12pt Paladin, especially bearing in mind that nearly all of the characters include curves. To speed up the plotting of text, the process is split into two stages, each character being plotted only once. The plotted image is of course a sprite, effectively a little block of screen. As characters are plotted, the resulting sprites are stored in a reserved area of memory called the *font cache*. Whenever the computer needs another example of a character that is already in the font cache, it copies it straight from the cache; it does not need to plot it again. Sprite plotting is a much faster operation than vector plotting as we have seen. The font cache tries to hold a copy of each character in every font you are using, so if you are using text in lots of different sizes this will take up more space in the font cache. When the font cache is full, characters will be overwritten when you need another size or style. Consequently, the size of the font cache is critical in applications that make extensive use outline fonts. It should be at least 64 Kbytes and, if you are using a font-intensive application (such as DTP) and have the memory to spare, 256 Kbytes is a more suitable size.

Anti-aliasing

There is one other important point about font caching. Under most normal conditions (using a font size no more than 18 pt and a 16- or 256-colour screen mode in use) the font manager converts the outline data not to two-colour (typically black on white) sprite images, but to eight colours (black, white and six intermediate greys). You may wonder why it should do this, when the original outline was of a single-colour character. The answer is simple—it dramatically improves the quality of the screen display.

We saw at the start of chapter 1 (Figure 1.1) that in computer graphics there are no such things as true curves or true oblique lines. Lines that are slightly oblique are reproduced as a series of unsightly steps—a phenomenon known as *aliasing*. A more attractive screen display is possible by judicious use of intermediate shades in the plot, a technique called, appropriately enough, *anti-aliasing*. For instance, in plotting a black oblique or curved line on a white background, a pixel which is completely covered is obviously made black and one which is completely missed is made white. But a pixel 80% of whose area is covered is made dark grey, the closest available approximation to 80% black, while a pixel only 20% covered is made pale grey, the closest approximation to 20% black.

This technique *appears* to give text on the screen much smoother outlines than those reproduced without anti-aliasing. It is an optical illusion, but a very effective one, as anyone can discover by switching to a two-colour screen mode (in which anti-aliasing is of course impossible). Figure 3.3 shows the mode-28 screen pixel structure of some 12 pt Trinity Medium text enlarged 12 times, clearly revealing the anti-aliasing.

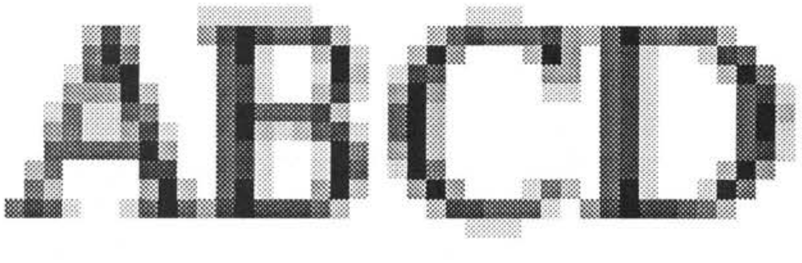


Figure 3.3—anti-aliased 12pt text enlarged 12 times to show the use of intermediate shades

Anti-aliasing is not appropriate to hard-copy printing, since most printers can print in only one colour.

The font editor

The outline font editor application, *Fonted* is available from Acorn. Its proper use demands care. It is very easy to create fonts badly; I have examined many fonts from public domain sources and not yet have I seen one which has correct scaffolding and hinting. Of course, fonts from reputable sources such as Acorn, Computer Concepts, the Electronic Font Foundry and RISC Developments (except their public domain selections!) are correctly hinted and scaffolded.

FontEd is installed on the icon bar in the normal way and follows the conventions for RISC OS applications with one important exception: it does not issue any warning if you click on the close icon of the font index window. The window is closed immediately and edits made since the font was last saved are lost irretrievably. So it is even more important than with other applications that you save your work regularly. I have lost precious work through inadvertently clicking on the close icon in mistake for the adjacent back icon.

As with other RISC OS applications you may work on more than one font at a time; if you have two or more fonts loaded, you can transfer characters between them, a very useful facility. But beware! On a 1-Mbyte machine with two fonts loaded, especially in Mode 20, you may run out of memory. And when the font editor runs out of memory, it simply crashes, sometimes without warning, edits since the last save being lost. So check the available memory (on the Task Display) regularly and, once again, save your work frequently. If the available memory is running low, change to a less demanding screen mode (such as mode 19) and reduce the font cache to zero. Ironically, *FontEd* does not use the font cache.

A further word of warning. When drawing or editing characters, for absolute accuracy you must use a zoom factor of at least 2:1 if in a multisync mode (such as mode 20) or at least 4:1 in a non-multisync mode (such as mode 12). If you use a lower zoom factor (such as the default of 1:1) the screen display, and therefore the mouse, will have a working resolution that is more coarse than the design grid of the font editor itself. This will lead to such anomalies as lines that appear horizontal on the screen proving to be slightly inclined or of strokes that appear on screen to have the same length proving to have noticeably different lengths when the character is printed. In font creation and editing it is absolutely essential to pay attention to detail. It is an activity in which your sins very quickly find you out!

There are four stages to font creation. The experienced font designer may be able to combine the first three stages into one operation, but it is best to consider them in sequence:

(i) drawing the outlines of the characters (and setting their widths)

(ii) adding the hinting lines

(iii) adding the scaffolds

(iv) creating the composite characters (such as the accented letters).

A font can be used if it contains only the outlines and width values of the characters, but the results will be disappointing, especially at smaller printout sizes or coarser print resolutions. There are many public domain and shareware fonts in circulation that are unscaffolded and unhinted; some have been electronically converted from fonts that originated on other computer systems.

If you have not used the font editor before, you are strongly advised to load a font from a reputable source and carefully examine the structure of the characters.

Operations on whole fonts

To load an existing font, either drag its Outlines icon on to the *FontEd* icon or double-click on its Outlines icon. If your version of *FontEd* is earlier than version 0.27 you must also drag the IntMetrics file icon into the opened font window. The grid in the font window will begin to fill up with anti-aliased images of the characters in the font; this is a 'background' task and you can proceed with any operation without waiting for the font display to be completed. The computer will beep when the display is finished. Figure 3.4 shows a typical complete font index window.

To create a new font from scratch, simply click select on the *FontEd* icon. This opens a new font window which is, of course, empty.

Clicking Menu with the pointer in the font window leads to the following menu of operations which affect whole fonts:

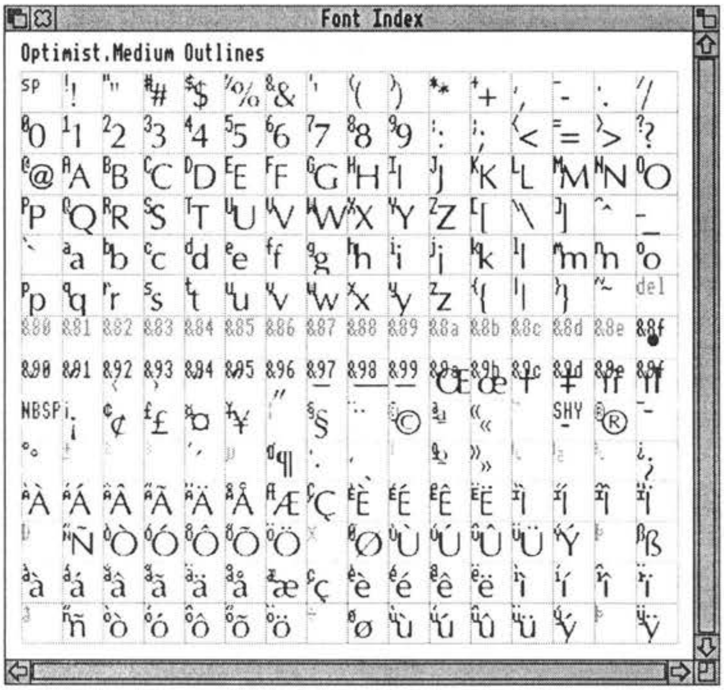
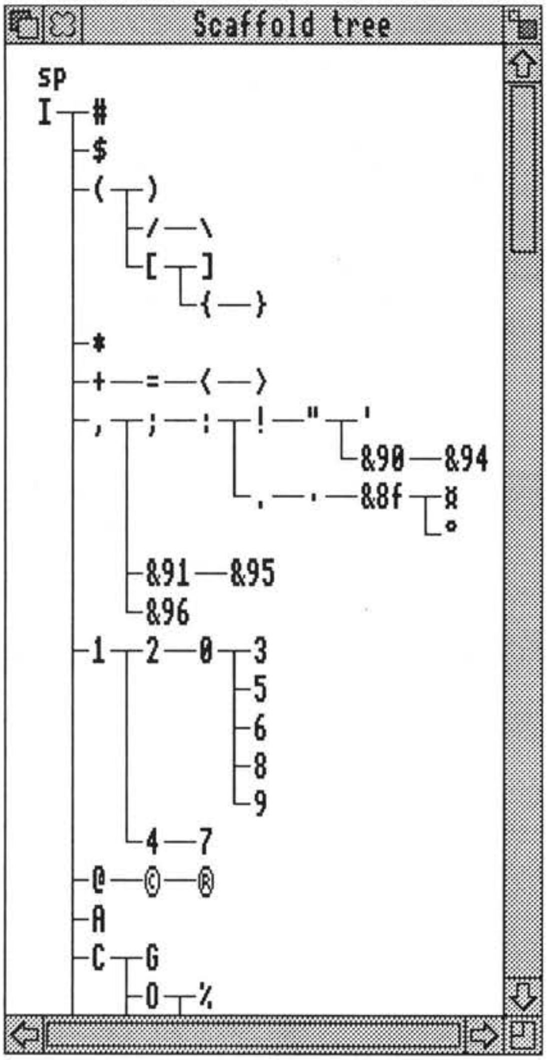


Figure 3.4—the font index display for Optimist.Medium from Wiseword Software

Redisplay—this simply causes the display of characters in the font window to be redrawn. It is only effective after a change of screen mode. So if you loaded the font while in, say, mode 12 and subsequently changed to mode 20, you should use this option to obtain a higher-resolution rendering of the characters.

Show tree—this produces a long scrolling window listing the characters currently in the font in the order of the priority of their scaffold designs. The top of a typical scaffold tree display is shown in Figure 3.5. This may make little sense to you if you have not had experience of font design, including scaffolding. The scaffolds are sets of lines which do not appear when the



```
sp
I┬#
 ├$
 ├(┬)
 │ ├/—\
 │ └[┬]
 │   └{—}
 ├*
 ├+—=—<—>
 ├,┬;—:┬!—"┬'
 │ │     │   └&90—&94
 │ │     └.—·—&8f┬¤
 │ │             └°
 │ ├&91—&95
 │ └&96
 ├1┬2—0┬3
 │ │   ├5
 │ │   ├6
 │ │   ├8
 │ │   └9
 │ └4—7
 ├@—©—®
 ├A
 ├C┬G
 │ ├O┬%
```

Figure 3.5—the top of the scaffold tree display for Optimist.Medium shows that all scaffolds are derived ultimately from that of the 'I'

character is printed, but which help consistency by ensuring that similar characters have uniform dimensions. Most often the first character to be given a scaffold is the capital 'I', whereupon this character appears at the top of the tree. Its scaffold lines are inherited by many other characters, modified as necessary, and these modified scaffolds are in turn inherited by other characters, so that each character in the font has a scaffold pattern that can be traced back eventually to that of the 'I'. The Show tree facility is very useful when scaffolding fonts as it shows at a glance which characters have not yet been given scaffolds. When applying scaffolding to an unscaffolded font, it can also be helpful to study the tree for a similar font that has been scaffolded.

Alter—this leads to a submenu of four items:

Font name—use this to change the name of the font you are editing or to set the name of the new font you are creating. If you are starting a new font the default setting will be <Untitled>. The font name must specify the filing path between Fonts and the Outlines and IntMetrics files created by the font editor. So if you are creating a font to be named, say, Arc Light, you should enter the font name as Arc.Light. You must, of course, ensure that your *Fonts* directory contains a directory named *Arc* and that this in turn contains a directory named *Light* in which your Outlines and IntMetrics files will be saved.

Design size—this is the scale of the outline drawings stored in the Outlines file. Clearly, the larger the scale, the smaller the item being portrayed. So, if for instance you load a font having a design size of 500, such as Trinity Medium, and drag a character out of the font index into a *Draw* window, the character will be become a path object equivalent in size to 80 pt text. If you repeat the exercise with a font having a design size of 1000, the size of the resulting path object will be only 40 pt. You can use the design size facility to re-size an entire finished font if its size proves to be inaccurate. But you should not change the size value in the midst of drawing characters. If you find that the design size is wrong after you have begun to design characters, leave the setting at its incorrect value and finish the font. Then, when the font is finished, adjust the design size setting until the size is correct. If you design some characters, then change the design size, and then design more, you may end up with a font having inconsistent character sizes.

Skeleton—this allows you to enter a pixel size for the skeleton display. It is recommended that you leave it at the default setting of 0.

Format—this option, available from version 0.27 onwards, offers you a choice of two file formats called version 6 and version 7. Until documentation concerning the advantages of version 7 becomes available, it is safer to leave the setting on the default, version 6.

Save—use this to save the font files you have created or edited. If the font has been previously loaded or saved and you wish to save it using the existing names and directory structure, you need only click on Outlines and Metrics to the right of the Save option. To save in different directories or on a back-up disc, you will need to drag the two file icons into the appropriate directory display. If the font has not been previously saved, you *must* delete the offered 'FontFile' for the outlines file and rename it *Outlines.* Drag both icons into the appropriate directory display. The directory path must coincide with the font name specified in the font editor.

Make bitmap—besides producing outline font files, the font editor will also produce bitmap font files from your outlines. You must of course specify the required font size in points and the output device resolution in dots per inch. Once specified, a new font index is produced showing the bitmapped characters. The resulting bitmap font can be saved as a file and the characters can be examined with a number of options. For most purposes this facility is likely to be of academic importance only. It does have a practical use for the font designer, however, since it ably demonstrates the beneficial effects of hinting and scaffolding by providing a representation of the font's appearance when printed at the specified size and dot density.

Debug—this is for development purposes only and is unlikely to be of interest to general users.

Operations on whole characters

To examine a character

There are two options available if you wish to look more closely at an individual character from the font index. Move the pointer over the character concerned and double click Adjust. This opens a 'full character' window in which the character is displayed in black against a white background, these of course being the colours most frequently used in print. From version 0.27 onwards the full character window can be scaled by dragging the Adjust size icon (in its bottom right-hand corner) allowing you to see how the character will appear at various sizes.

To examine the structure of a character you must open its 'skeleton window' by double clicking select over the character's index entry. This is the window in which characters are created and edited. It shows the character as an outline complete with hinting lines and scaffolds, if present. The zoom facility is similar to that in *Draw* but allows enlargement or reduction by factors up to 1000:1 or 1:1000 in version 0.27.

The skeleton window can also be opened by clicking Select in the full character window. The full character window can also be opened from the skeleton window by clicking Menu and selecting the Full char option. All windows are closed in the normal way. Both skeleton and full character windows are shown in Figure 3.6.

You can click over the character in the Scaffold Tree window as an alternative to the font window if you wish.

To copy a character

To copy one character into another character's slot in the index, simply move the pointer over the character to be copied, hold down Select and drag the character until it is exactly over the destination character's position, then release Select. An exact copy of the original character complete with points, hinting and scaffold lines will be made in the new character position. Note that any character previously present in the destination position will be overwritten and lost irrevocably. No warning is given.

The copy of the character is quite independent of

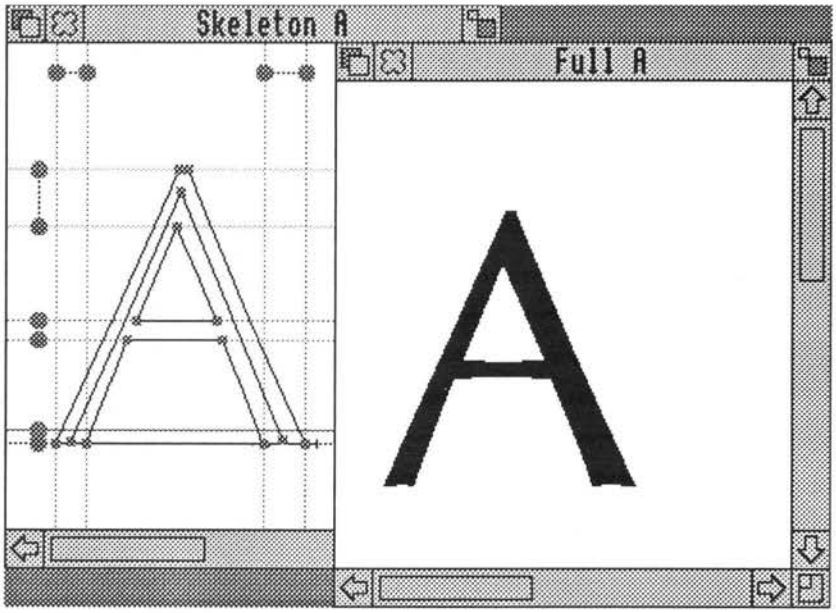


Figure 3.6—skeleton and full character windows from Optimist.Medium

the original; subsequent editing of either will not affect the other. Indeed this facility provides a useful shortcut in font creation when creating characters that are similar to existing ones, such as the 'F' from the 'E' or the 'Q' from the 'O'. It is also useful for making a temporary copy of a character that is being edited; if the edit proves unsuccessful, the copy (stored in a little used location such as &80) can be copied back to the original.

If you have loaded more than one font this operation can be used to copy characters from one font to the other. If the two fonts have different design sizes, the character being copied will be rescaled automatically by the appropriate factor. You can also drag a character into a directory display; it will be saved as a *Draw* file named DrawFile. You can also drag it into a

Draw window where it assumes the form of a conventional path object. Note that when you drag a character into a directory viewer or into a *Draw* window, scaffolding and hinting lines are removed. If you drag a compound character such as an accented letter into *Draw*, the separate components will all be one path in *Draw*, even though they originated as separate characters in *FontEd*.

You cannot transfer a path object directly from *Draw* to *FontEd*. An application called *D2Font* available from David Pilling which converts path objects to outline fonts is discussed later.

There is another way to copy a character; see below.

To merge two or more characters

If you hold down Shift while you drag one character into another's location, you will again obtain a copy of the original in the destination location, but this copy is different in several important respects from that obtained without Shift. Firstly, the copy is an outline only; it has no points or scaffold lines and cannot be edited. It is an example of *replication*—it is a clone of its source character and will reflect any subsequent changes to the source character. Secondly, copying with Shift does *not* overwrite any character already in the destination location. Any character previously present will still be present, the outline of the copied character being superimposed over it. But note that if these outlines overlap, the overlapping area will appear transparent when applications reproduce the character on screen and in printouts. (In other words, the two outlines constitute one object and the outline font system follows the even/odd winding rule.)

This Shift-drag process is often used to merge two or more characters, *e.g.* to add an accent to a letter. If you hold down Shift as you drag the letter component into the destination 'slot' and again as you drag the accent component character into the same 'slot', both components of your compound character will be outline-only clones and this is fine for most accented letters. Alternatively, one component may use a conventional copy without Shift, while the other alone is Shift-dragged. This is useful where the accented character is slightly different from the non-accented form and therefore needs editing. For instance, in the accented forms of the letter 'i' the dot must be eliminated, viz: 'ı'. So, copy 'i' into the 'ı' location in the normal manner, delete the curves that make up the dot and then Shift-drag the undotted character into the other locations where it is needed. Finally Shift-drag the accents into the appropriate locations. Examples are shown in Figure 3.7.

You can change the positions of the 'clone' components in a composite character. Move the pointer into the skeleton window and click select. Each time you click Select a different outline will become 'mobile', although this is not immediately apparent on the screen. The mobile component can be moved using the cursor keys; faster movement is obtained by using the cursor keys in conjunction with Ctrl. Whenever you create a composite character, you should *always* move at least one of the components, even if

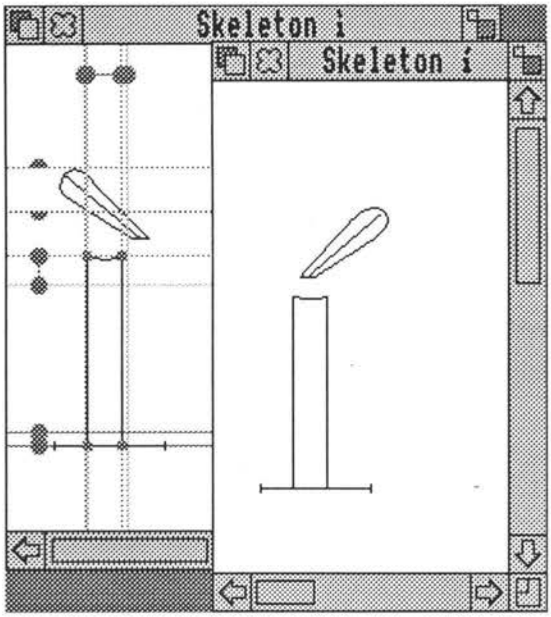


Figure 3.7—skeleton windows from Optimist.Medium reveal that in 'ì' the main body of the character is constructed conventionally (it has points and scaffolds) while the accent, since it has no points or scaffolds, is a replication of a separate character; 'í', in contrast, consists of two merged replications

only to return it to its original position. Failure to move at least one component leads to incomplete data in the IntMetrics file and this can cause a multiplicity of malfunctions when the font is subsequently used by applications. (The most common symptom is the screen display refusing to display the affected characters or, sometimes, any characters from the affected font.)

You can superimpose many characters by Shift-dragging them in this way. But there are some limitations. You cannot Shift-drag a character that consists only of Shift-dragged outlines; this has no effect. (But you can copy such a character in the normal manner.) If you Shift-drag a character that consists partly of conventional drawing and partly of Shift-dragged outlines, only the outline of the conventional drawings will be transferred. You cannot Shift-drag a character onto itself, nor can you Shift-drag the same character more than once into the same location; the second Shift-drag deletes the first. So you cannot create the colon (':') by Shift-dragging the full-stop twice, although you can create it by conventionally copying the full stop and then Shift-dragging the full stop into the same location. You could, of course, create the semicolon (';') by Shift-dragging the comma and the full stop into its location. In this way you not only save yourself much wasteful duplication of effort, but you also ensure that common features are absolutely consistent.

You cannot delete a cloned component of a compound character. If you make a mistake and Shift-drag the wrong character into the location, your only option is to delete the affected character and start again from scratch.

To delete a character

Drag an undefined character into its location. The character previously in the location is lost irretrievably. Caution! No warning is given. If there are no undefined characters, you could drag the space or NBSP (non-break space, ASCII &A0) into the location concerned. The location will now of course contain a copy of the space complete with its width setting.

To add one character's scaffold to another character

Hold down Ctrl and drag the source character into the destination character's slot or skeleton window. Only the scaffold is affected. The two characters' outlines are preserved. This operation will not transfer scaffolds between fonts.

To compare two characters

Open one character's skeleton window and then drag the other character into it. The dragged character's outline only will appear in pale grey superimposed on the present character. To remove the superimposed character, drag a space or NBSP or any undefined character into the skeleton window. This facility is useful for comparing characters for size or height, especially when building scaffold trees. The two characters need not be from the same font.

You may sometimes find yourself in the situation of wanting to modify a character's structure while preserving its overall shape, for example, if for scaffolding purposes you wish to add an extra point into a curve, but you cannot afford to lose the line of the existing curve. Proceed as follows. Copy your original character into an unused slot, say &80. Now drag &80 into your original's skeleton window. At first it seems that nothing has happened, but when you move or delete your curved line you will find a pale copy of the original beneath it. This is of course the outline of the copy in &80. You can now add your extra point and adjust your new curves so that they follow the original line. Finally, delete the copy in &80.

From version 0.27 onwards you can also drag sprites into skeleton windows. The sprite must have been created in a two-colour mode such as mode 18. Once loaded it is enclosed in a frame with eight handles like a DTP frame; these allow it to be squashed, stretched or moved. This remarkable facility allows you to create fonts by scanning an example of the required typeface and dragging the resulting sprites into *FontEd* for use as templates around which the character outlines can be 'traced'. For more on this see below on Display in the Skeleton menu.

Character creation and editing

Drawing character outlines is broadly similar to creating path objects in *Draw* but there are several fundamental differences. All drawing and editing is done in the character's skeleton window. You will not be able to draw if a scaffold is selected, *i.e.* shown in red, (press Escape to deselect it) or if a sprite is active (turn off its handles from the Display menu—click Menu in the skeleton window).

Drawing lines

To draw a line, position the pointer where you wish the line to begin and press Select. Continue holding down Select and drag to the end of the line. Then release Select. You now have an isolated line with a point at each end. Points are shown as green squares. To draw an entire outline with many points proceed as before but wherever you wish to create a point momentarily release Select and then press it again and continue drawing. To complete the outline bring the pointer back over the starting point and release Select. The computer will beep to tell you that it has joined two lines at the current pointer position. If you have a full character window open (and this is recommended) you will see that the outline is automatically filled in solid as soon as it is complete.

Sometimes, although you heard the computer beep as you completed the outline, the display in the full character window remains an empty outline. This means that the outline is broken somewhere. You probably moved the mouse a little while your finger was off Select at one of the points so that, although the screen shows what looks like a complete outline, one of the points is in fact two separate points with an infinitesimal gap between them. To find the offending point(s) move the pointer on to each point in turn and on each press Adjust. Dwell with Adjust down for a second or two. At one point you should find that the computer beeps and the full character display goes solid. If the computer beeps but the full character display still remains stubbornly hollow, you have yet another break somewhere else in your outline and must continue your search. If your character has a hollow centre (such as the 'O' or the 'A', you will of course need to create separate outer and inner outlines with a deliberate break between them. Both outlines must be continuous. If there is a break in the inner outline while the outer is intact, the character's image in the Full character window will appear solid rather than hollow.

Moving lines

You can move any point by placing the pointer on it and dragging it while holding down Adjust.

So any line can be moved by moving the points at each end of it. There is also a means of moving whole characters or parts of characters—see below.

Curves

All lines are initially straight lines but have the potential to become curves. To change a line to a curve you must first select it. Place the pointer on the line itself, *i.e.* between the two points, and click Adjust. The line will become blue and two red control points will appear on it. *FontEd* like *Draw* uses Bezier curves, but in *FontEd* only one line may be selected at any one time and the control points are only visible on the line that is selected. Moreover, as soon as you move the pointer over one control point and press Adjust, the other control point disappears. This does in fact help the designer working on complex curves, since the multiplicity of overlapping control points that appear when a sinuous path object is taken into edit mode in *Draw* can be overwhelming. To bend your line, place the pointer on a control point and move it just as you would any other point.

To straighten a curved line, select it, press Menu and click on the second option, Straighten.

To deselect a line, click Select elsewhere or press Escape.

Moves

Moves are implicit in *FontEd*. That is to say, you do not have to create them especially. Every character is one object, even if it contains several distinct outlines. The different outlines are therefore separated by implicit moves.

Deleting lines

Select the line, press Menu and click on the first option, Delete. Deleting a line in a continuous outline does not result in a smaller continuous outline as it would in *Draw.* It leaves a gap (in effect a move) in the outline.

To add an extra point into an outline

Position the pointer over an existing point and click Select. It will seem that nothing has happened, but your original point has in fact become two points superimposed and joined by a line of zero length. If you attempt to drag the uppermost point with Adjust you will drag first one control point on the new line, next its other control point and finally the new point itself. With the new point correctly positioned, you can use the Straighten facility to straighten out the new line.

Alternatively, position the pointer over an existing point and select-drag to the required new position. A new straight line will be created within the existing outline.

To open the path of an outline

Either select any line and delete it or, if you wish to retain all the existing lines and open the path at a point, create a new line (as above) and immediately delete it in the normal manner.

To move the whole or part of a character

The font editor does provide a facility which allows a whole character or any selection of points in it to be moved an equal number of coordinate steps in any direction. This is useful for repositioning whole characters and, in conjunction with the scaffolds (see later), ensuring that similar characters (like the 'm' and the 'n' have uniform height. The facility uses the scaffold system; to access it you must enter scaffold mode. If you already have one or more scaffolds present, simply select one of them. If there are no scaffolds present in the existing character, you must temporarily introduce one, as follows. Click Menu and move the pointer over the third item, Scaffold. On the scaffold menu select the first item, New local. On the New local scaffold menu you could select any option, but the top item is most convenient, H-scaffold. This will put a bright blue horizontal line across the character. For our purposes its position is immaterial. Select the scaffold by clicking Select in the 'blip' at its left-hand end; the scaffold will change to red to show it is selected.

Now you must link the points which you want to move to the selected scaffold. To do this you either move the pointer over the point and click Select or, if there are many points which you want to move, you move the pointer to a position where it is not over any lines, hold down Select and drag out a box which embraces the points you wish to link. When the box encloses the points concerned, release Select. The points which have been linked will change to red. Clicking Select on any point will toggle its link on and off. You can unlink all the linked points by pressing function key F9. Points do not need to be on or even near to the scaffold to which they are linked.

While the points appear in red to show they are linked to the selected scaffold, you can move them using the cursor keys. Each stab on a cursor key will move all the linked points one coordinate step in that direction. Using Ctrl in conjunction with the cursor keys will give faster movement. In this way any number of points from one to an entire character can be repositioned.

If you created the scaffold simply in order to move the points, you should afterwards unlink the points using F9 and then delete the scaffold; press Menu and select the top option, Delete.

If you used an already existing scaffold line, you will have to decide carefully which points should be unlinked and possibly relinked to their original scaffolds—see later on scaffolding.

Adding hinting

Hinting lines are ordinary lines along the centre of certain strokes. Their purpose is to ensure that, when the screen or printer is reproducing the character at a very small size, *something* is drawn to represent that stroke. See the section on thin lines in Chapter 2 for further details.

Hinting lines are needed in all curved strokes and in all straight strokes which are oblique, *i.e.* neither horizontal nor vertical. They may be omitted in very bold oblique strokes, where it is almost unthinkable that the stroke could ever fail to appear. Thus the capital 'A' will require a hinting line in each of its two major strokes since both are oblique, but not in the cross stroke since that is horizontal.

A point to remember is that in characters which have a hollow centre, such as the 'O', there must be at least one break in the hinting line; it must on no account form a complete loop. If it does, the drawing will consist of three concentric outlines; by the application of the odd/even winding rule the area between the outer and central outlines would be filled as would the inner; thus the letter 'O' would appear to consist of a very thin ring closely surrounding a filled ovoid. The position of the break is immaterial, but it is best to put it at a point where the outline is thick and so not in danger of disappearing.

The Skeleton menu

Clicking Menu while the pointer is in the skeleton window leads to the following menu of useful facilities:

Delete—this deletes the currently selected line or scaffold. The option is greyed out if no line or scaffold is currently selected.

Straighten—this straightens the currently selected line. The option is greyed out if there is no line currently selected.

Scaffold—this leads to a further menu concerned exclusively with operations on scaffolds. See the section on scaffolds, below.

Display—this leads to a further menu concerned with the skeleton display. This will be considered below.

Full char—this opens a full-character window for the character concerned. The skeleton window is left open.

Width—this opens a dialogue box displaying the current width setting of the character concerned. If no width has been set, the value shown will be 0. You may edit the value as required. The width is represented in the skeleton window as a red line along the x axis.

The width unit is 1/1000 em and relates to the character reproduced at 12 pt size.

Zoom—the zoom facility is similar to that in *Draw*. The maximum zoom available in version 0.27 is 1000:1 and the minimum 1:1000. The *variable* facility automatically adjusts the zoom setting so that the entire character fits in the skeleton window. When disabled, the zoom defaults to the setting displayed.

The Display menu

The Display Menu has eight options (in version 0.27) which relate to facilities that can be optionally displayed in the skeleton window. A tick appears beside each option that has been selected. Clicking on items toggles them on and off.

Pointer—this is selected by default. When deselected the pointer disappears during drag operations. During fine work it can be useful to disable the pointer to prevent it from concealing parts of the outline.

Coords—this is deselected by default. When selected, the current x and y co-ordinates of the pointer are displayed alongside it during drag operations. This can be useful, especially when no scaffolds are present, since it helps to ensure the use of consistent heights and widths throughout a font.

Width—this is selected by default. This is the red line displaying the currently set character width along the x axis.

Char BBox, **Orig BBox**, **Font BBox**—all three bounding boxes are deselected by default. The Character bounding box, displayed as a solid red line when selected, is the hypothetical box that just encloses the character concerned. The Original bounding box, displayed in green, is initially the same as the Character bounding box, but if you edit the character and extend one of its strokes outside the Character bounding box, you will find that the Character bounding box is updated, while the Original bounding box remains the same. The Font bounding box, shown as a dashed red line, represents the extreme limits of all the characters in the font.

Bitmap, Handles—these facilities, only available from version 0.27 onwards, control the display of sprites by the font editor. The sprite to be displayed must be in a two-colour mode such as mode 0, mode 18 or mode 25. If the Bitmap option is selected, the sprite can be displayed by dragging its file icon into the skeleton window.

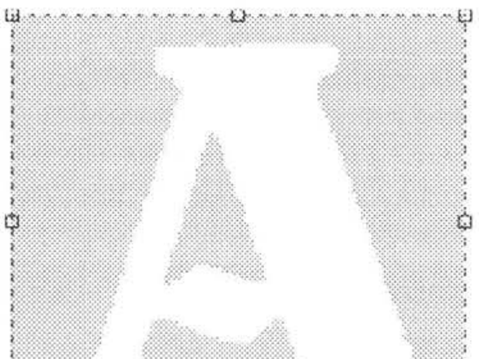


Figure 3.8—a sprite in its 'handles' as imported into the font editor

Even if it originated as a black design against a white background, for example as the output of a scanner, in *FontEd* it will be displayed in white against a pale grey background. It will disappear if the Bitmap option is disabled. If the Handles option is selected, the sprite will be enclosed in a dashed red boundary box with eight handles (see Figure 3.8). The sprite can be moved by placing the pointer inside the boundary box and dragging with Select. The sprite can be enlarged, reduced, squashed or expanded using the handles. Handles in the centre of a side move that side; those in the corners move that corner while the diagonally opposite corner remains fixed.

Scaffolding

Scaffolds are special lines which are introduced into the character design, but which do not appear in normal screen displays or printouts. Their purpose is to tidy up the display of characters so that similar strokes have consistent widths and lengths not only within each character, but throughout the font. The scaffolds which have been used in one character can be passed on to another similar character to ensure consistency. They also control the way in which curved lines are reproduced. The following account is of necessity a simplification.

Each character is allowed to use a maximum of seven vertical and seven horizontal scaffolds, numbered 1 to 7. In fact there also exist a pair of default scaffolds (one vertical and one horizontal, numbered 0). Every point in every character is always linked to two scaffolds, one vertical and one horizontal; when a point is first created it is automatically linked to the default scaffolds and is only disconnected from these when you connect it instead to another scaffold. Scaffolds in the same axis can also be linked to each other to ensure that similar characters have identical width or height.

In the horizontal axis there are three types of scaffold: the H-scaffold, the U-tangent and the D-tangent. H-scaffolds are used with horizontal or nearly horizontal strokes: in Figure 3.9 three H-scaffolds are visible on the three horizontal strokes of the character. Each H-scaffold consists of *two* parallel lines. When an H-scaffold is first introduced it appears to consist of a single line, but this is only because the two lines are co-incident. The position of the two lines is altered by dragging the circular blob at the top end of the line with Adjust (dragging with Select will

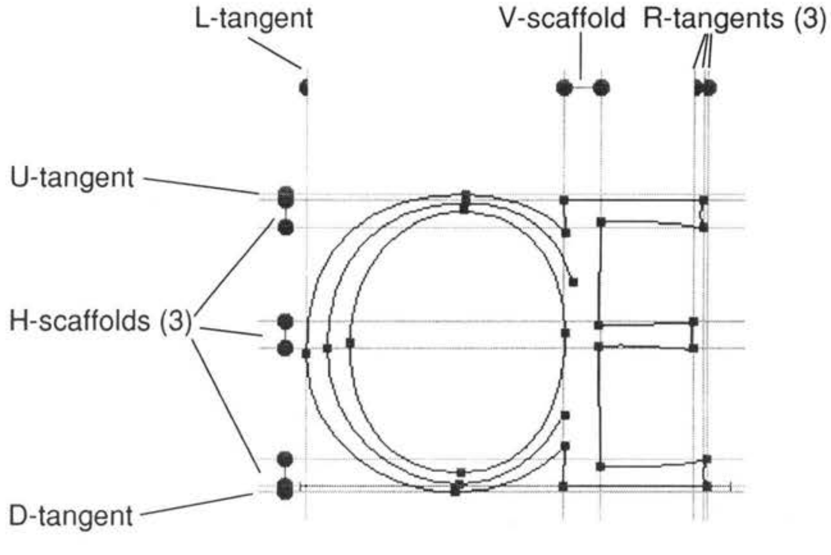


Figure 3.9—all six types of scaffold in use on a complex character from Optimist.Medium. Note the hinting line within the circular stroke

draw a line). Note that the two lines cannot be separated by more than 255 internal units. When points are linked to the scaffold (how to do this is explained later) those points will be reproduced in such a way as to give that stroke (and other similar strokes elsewhere in the font) consistent width.

U-tangents and D-tangents are used with curved strokes and, as their names suggest, they are positioned so that they just touch the outer extremity of the curve, *i.e.* they are a tangent to it, the U-tangent being at the upper extremity of the curve and the D-tangent at the lower extremity; examples of both are visible in Figure 3.9. Their purpose is partly to ensure that characters are consistently positioned throughout the font—in many fonts rounded characters such as the C, O and U sit a little below the baseline used by square characters—and partly to ensure that the curves are reproduced as smoothly as possible.

In the vertical axis there are three types of scaffold, the V-scaffold, the L-tangent and the R-tangent, which correspond in function and use to the H-scaffold, U-tangent and D-tangent respectively. The 'blobs' are at the left-hand end of the scaffolds. Again, examples of all three types are visible in Figure 3.9.

Scaffold lines are shown in pale blue if they are new, *i.e.* introduced on this character. They are shown in dark blue if they are inherited from another character. They are shown in red when selected. To select a scaffold click select on its blob. In the font index and also the scaffold tree display all characters containing the selected scaffold will be highlighted in red, enabling you to see at a glance which characters will be affected by changes to the scaffold. To deselect a scaffold either press Escape or click Shift-Select elsewhere in the skeleton window. To delete a scaffold, first select it and then choose Delete in the main menu; it is the same as deleting a line.

To link one scaffold to another, select the first scaffold and click Shift-Select on the blob of the scaffold you wish to link. This blob will change colour whenever the other scaffold is selected. To disconnect it, click Shift-Select again.

To link points to a scaffold, select the scaffold and then either click Select on the points you wish to link or drag out a box with Select. When you release Select all the points enclosed by the box will be linked. Linked points are shown in red. Clicking Select on a linked point unlinks (or 'disconnects') it. Key F9 will disconnect all points from the selected scaffold. Note that points do not have to be directly over the scaffold to which they are connected.

To introduce a new scaffold into a character in a skeleton window, move right on the Scaffold option in the main menu. This leads to a submenu in which some options will be greyed out because they are inappropriate. The first three items all lead to a further submenu offering the six different scaffold types already discussed.

Choose the first option, New local, if the scaffold is only required in the current character, *i.e.* you will not wish to pass the scaffold on to other similar characters. Choose the second option if you wish your scaffold to be passed on to other characters. Because each character may have a maximum of only seven scaffolds in each axis and because most characters inherit scaffolds from other characters, there will be many occasions when there are no free scaffolds available but there are unwanted scaffolds in the character. The Replace option allows you to select an unwanted scaffold and to delete it and replace it with a new one. Clearly, the new scaffold must be in the same axis as the original; it will be local if the original was local and global if the original was global. The operation is a short-cut method of deleting and introducing a new scaffold. If you inadvertently delete a scaffold, the next option, Undelete, restores it. The final option, Disconnect, duplicates the action of the F9 key, disconnecting all points from the currently selected scaffold.

In practice there is little difference between local and global scaffolds. Either type is passed on to other characters by character duplication or Ctrl-dragging. And if you subsequently move a scaffold line, the move will apply in every character that uses the same scaffold.

Removing scaffolds

You may occasionally wish to remove the scaffolds from a font—perhaps if it has been incorrectly scaffolded and you wish to begin again. Proceed with great care! Save the font *very* frequently, since in these circumstances crashes occur very easily. It is wisest to start with characters at the bottom of the scaffold tree and work upwards. You may find that attempts to delete scaffolds result in error messages of the type 'Incomplete scaffold link in *n*' where *n* may or may not be the character on which you are working. If it is a different character, leave the one on which you were working and attend to the other before returning to the original. It is sometimes impossible to delete a scaffold if it is connected to another scaffold, so undo all scaffold links before attempting to delete a scaffold.

D2Font

As mentioned earlier, characters can be dragged from the *FontEd* font index into directory displays or *Draw* windows, where they are treated as ordinary *Draw* files containing a single path object. *Draw* files, however, cannot be dragged into *FontEd*—it refuses to recognise them. A fascinating and inexpensive application called *D2Font* from David Pilling, however, converts *Draw* files to outline fonts.

D2Font is supplied on the same disc as *Trace* (see chapter 12) which converts sprites to *Draw* files. Also supplied on that disc are some scanned images of attractive typefaces. Clearly the user is encouraged to use *Trace* to convert these scanned images to *Draw* path objects and *D2Font* to convert those to usable outline fonts. The process is time consuming and there is much tidying for the user to do, but for those with time to spare it is all quite feasible.

D2Font's main window (Figure 3.10) is very similar to that of *FontEd*, showing each character present. Operation is simple. Out of *Draw* (or whatever) you save the path objects that make up each character, dragging their file icons into the appropriate locations in *D2Font's* index. This in itself is a long process if you have anything approaching a complete character set. Bear in mind that if the *Draw* file originated in *Trace* or a similar application, some characters such as the 'O', the 'B' and the 'P' will contain more than one outline and these may initially be separate objects; it is a wise precaution to group them to prevent them from moving relative to one another.

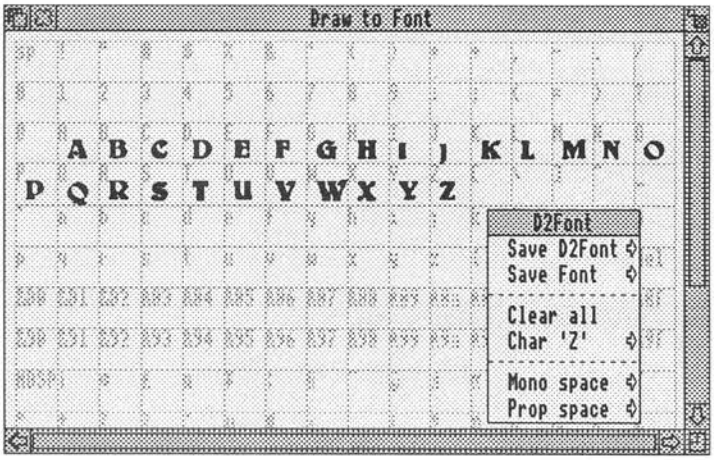


Figure 3.10—part of the *D2Font* window including the main menu. The characters visible were converted from sprites to *Draw* path objects using *Trace*. *D2Font*, *Trace* and this and other selections of scanned typefaces are supplied on the same disc from David Pilling

D2Font offers the option of saving the entire font; you enter the appropriate directory name and it saves a directory containing an Outlines and an IntMetrics file which can be used immediately, but which will almost certainly need some further editing in *FontEd*. You can also save your unfinished font in *D2Font* format. This will allow you to resume transferring designs into it later on if you do not have time to manage them all in one session. Incidentally, you can also drag individual characters out of the index as *Draw* files. They will be in the same condition as when you imported them.

Other options allow you to clear all designs and restart with an empty window, to clear individual characters and to select one of two character width options. One of these automatically sets a width that is proportional to each individual character's width and the other sets one constant width throughout the font, as in Acorn Corpus.

The only other facility offered sets the position of the character. Double-clicking in any character's compartment opens a window containing a large view of that character sitting on a red line which indicates both the character's width and its position relative to the base line. By default, *D2Font* positions all characters so that their lowest extremity is right on the base line; it has no way of recognising descenders. Consequently you will need to reposition characters having descenders such as 'g', 'p' and 'y'. You could do this later in *FontEd*, but it's easier in *D2Font*. Dragging with Select adjusts the height of the base line; dragging with Adjust changes the width.

Incidentally, there is a useful function for *D2Font* and *FontEd* that is totally unrelated to fonts. This is to merge several path objects into one. *Vector's* Merge Path and *ArtWorks'* Join Shapes (see chapter 5) menu options perform this operation which can lead to worthwhile savings of memory and redrawing time; it also allows holes or windows to be created in existing objects. Those who don't own *Vector* or *ArtWorks* but do have *D2Font* and *FontEd* may proceed as follows. Group the objects to be merged. Save the group into any character location in *D2Font*. Save the font under any name, even if it consists of just the single character. Load the font into *FontEd* and then drag the character back into *Draw* (or whatever). It will now be one path object, the formerly separate paths being joined by moves. The process works best if the initial path objects are closed paths; the font editor, after all, is used to dealing with outlines. Oddly enough, in the resulting merged object, the sub-paths are open.

Edited fonts and PostScript printers

Unlike other types of printer, a PostScript printer uses fonts stored in the printer rather than taking an image of the page from the computer. For this reason, you need a PostScript font to match each font you wish to you use. If you create or edit a font yourself using *FontEd*, clearly you will not have a matching font in your printer so you will not automatically be able to reproduce what you see on the screen. If you have RISC OS 3, however, you can download fonts to your printer from the computer. This enables you to use new or edited fonts and to reproduce them in your printout. Downloading fonts in this way uses up a lot of printer memory; you may find that you need to buy extra memory for your printer (not your computer) if you want to download many fonts. You cannot download fonts from RISC OS 2.

4 Adjuncts to *Draw*

In the wake of the original (RISC OS 2) version of *Draw*, many software packages were introduced to provide additional facilities. This chapter considers six of them, three concerned with font manipulation.

The characters in the Acorn outline font system, as we saw in Chapter 3, begin their lives as vector graphics, *i.e.* outlines, although the text that is sent to the screen or printer has normally been converted to sprites. Font manipulation packages extract the outline data from the font files and convert text back to *Draw*-compatible path objects, *i.e.* to vector graphics. In Chapter 2 we briefly met a similar text-to-path facility in the RISC OS 3 version of *Draw*.

The reason for the profusion of font manipulation software is that the font handling facilities of RISC OS 2 were somewhat limited. It is true that characters from any outline font could be reproduced at any size (to the nearest 1/16 pt), with different scaling on the vertical and horizontal axes if required, and in any text colour and background colour. Nevertheless, for graphic design this is insufficient. Even a simple drawing like that in Figure 4.1, which requires text that is both on a slant and rotated, poses difficulties. There was no way in which text could be written vertically or at angles or slanting, upside-down or backwards (*i.e.* mirror imaged). Nor could it be given a shadow or fitted around an arc or a circle. But all these effects and many more became routine with text that had been converted to graphics. Moreover, *Draw's* path objects allow separate line and fill colours

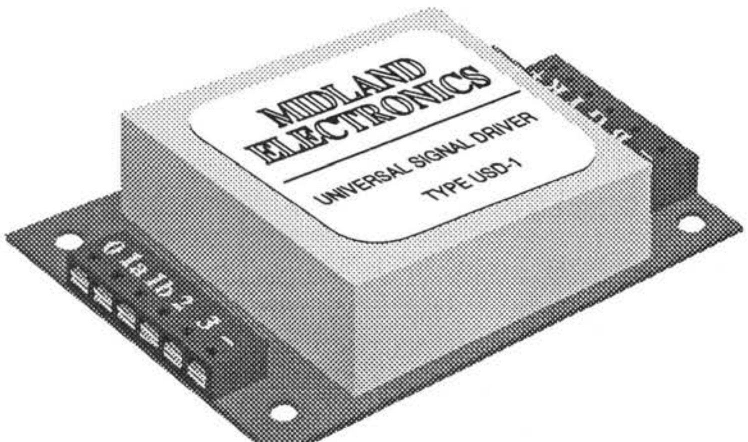


Figure 4.1—text in drawings such as this normally needs to be both rotated and slanted to match the local horizontals and verticals; the drawing was created in *Draw* and the font manipulation in *Typestudio*

and the line width is adjustable. So text could use two colours—it could even be made transparent —greatly boosting the potential for eye-catching design work. Users of desktop publishing benefited from this kind of operation for fancy titling and display effects, importing the converted text into graphics frames.

You may be tempted to think that the advent of RISC OS 3 eliminated the need for this kind of software. Not so! It is true that the enhanced font manager in RISC OS 3 can rotate and reflect outline-font-based text without converting it to vector graphics, but if your DTP software was written for RISC OS 2, it may be unable to use this facility or even to reproduce a rotated text object in an imported RISC OS 3 *Draw* file. And even if you try printing rotated text straight from *Draw* in RISC OS 3, you may find that your printer driver is unable to handle the transformation and reproduces the text horizontally. It is true that the version of *Draw* built into RISC OS 3 has its own text-to-path facility which will meet such requirements as text having different outline and fill colours. But many of the more complex manipulations like text on a circular path really demand specialist font manipulation software.

The following font-manipulation packages are considered in this chapter: *FontFX* from The Data Store; *Fontasy* from ICS (Ian Copestake Software); and *TypeStudio* from Risc Developments.

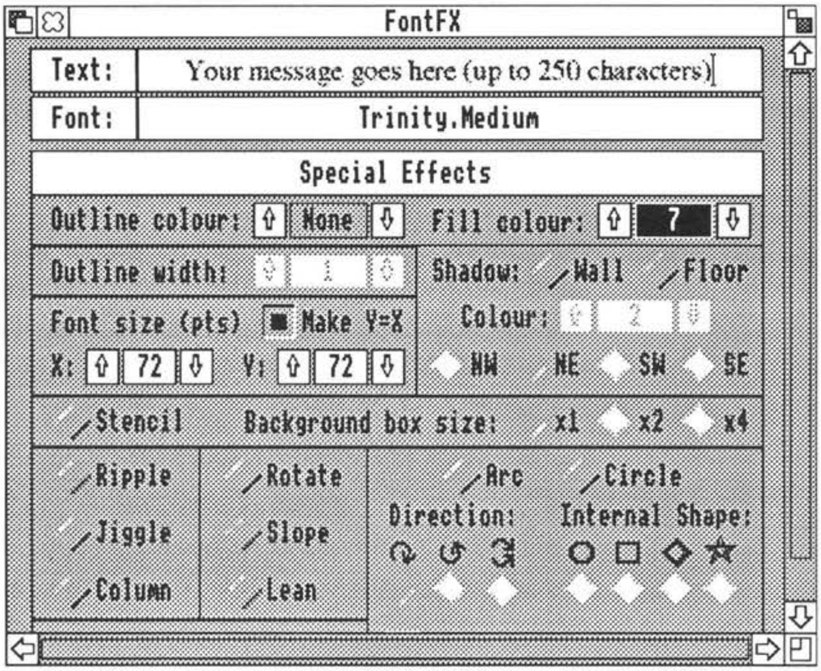
Three other specialist adjunct packages are also considered: *DrawBender* from ICS, a *Draw* file distortion utility; *Chameleon 2* from 4Mation, which, as its name suggests, is concerned with colour change; and *Placard* from ICS which facilitates the handling and printing of large drawings.

FontFX

The oldest and simplest of the font manipulation applications considered here, *FontFX* from the Data Store has developed over the years and still has much to offer. The following description is based on version 4.31.

Clicking on the icon produces a dialogue box (Figure 4.2) having a writable icon at the top in which you enter the text you wish to manipulate. The maximum permissible length is 250 characters. Your entry appears in the currently selected outline font which is named in the pane beneath; *FontFX* defaults to Trinity Medium if available. Clicking Menu anywhere within the dialogue box calls up a four-item menu, the first item being Fonts. This leads to a scrolling list of the outline fonts present; to change the currently selected font you click Select on the one you want. The other three menu entries initiate the creation of the graphics, allow you to save a file of your preferences or to reset the system defaults.

The rest of the dialogue box is taken up by the special effects which can be applied to the converted text. Outline colour and fill colour provide for separate line and fill colours in the converted text, and outline width controls the line width in the resulting path objects; understandably, it is not available if the line colour is set to transparent. Colour selection is limited to


FontFX

Text: Your message goes here (up to 250 characters)

Font: Trinity.Medium

Special Effects

Outline colour: None
Fill colour: 7

Outline width: 1

Shadow: Wall Floor

Font size (pts) Make Y=X

Colour: 2

X: 72 Y: 72

NW NE SW SE

Stencil

Background box size: x1 x2 x4

Ripple Rotate Arc Circle

Jiggle Slope Direction: Internal Shape:

Column Lean

Figure 4.2—the dialogue box in *FontFX*

16 palette colours (in all screen modes) or no colour (transparent); there is no provision for mixing non-standard colours by their RGB content, but if you want a non-standard colour it is no great problem to change this after you have transferred your converted text into *Draw*. Font size allows you to choose the size of the converted text: sizes from 1 to 999 pt are available and separate sizes may be chosen for the x and y axes.

The stencil option provides some fascinating effects which otherwise can only be achieved using the Path Merge facility in *Vector* or the Join Shapes facility in *ArtWorks*. Essentially it places the text in a box (whose size is controlled) filled with the text fill colour. The text itself retains any outline colour that was selected but its fill colour is transparent, like the holes in a stencil, and of course through it any other objects behind can be seen. A set of patterns, ideal for placing behind stencil text, is provided with *FontFX*.

In fact each letter is converted to *two* objects, one having the specified outline colour and width and the other having the depth of the box and the box colour (see Figure 4.3 This allows extra versatility when subsequently processing the stencilled text in *Draw*.

Optionally, your finished text may have a *shadow*. There is a choice of two shadow types: wall or floor. Clearly these represent the types of 'upright' or 'angled' shadow that would be thrown on a wall or a floor by your text hanging in space. A wall shadow is also sometimes called a drop shadow. The shadow colour is chosen

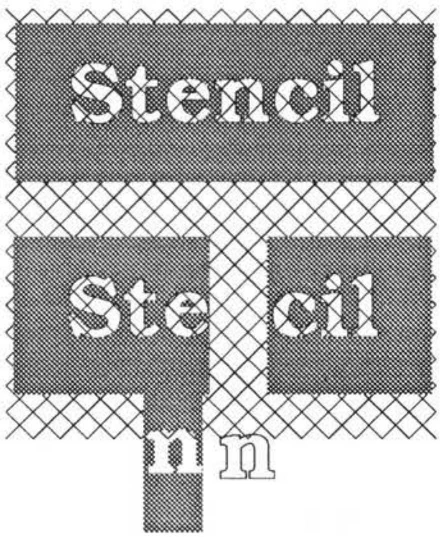


Figure 4.3—in *FontFX's* stencil effect every letter is two separate objects, one the outline and one the box

from the standard palette. There are four possible directions or offsets for each type of shadow: NW, NE, SW and SE. While these are self explanatory, the SW and SE floor shadows do of course appear to fall *in front* of the text and in this respect are similar to the reflections offered in some other packages.

Incidentally, you get some fascinating effects by making the text invisible (set both outline and fill colours to the intended background colour) but give the shadow a different colour. Shadow can be combined with any other effect including stencil. Figure 4.4 illustrates some shadow effects. Once you have imported your converted text into *Draw* you can even give the shadow its own outline colour if you wish.

Three effects are offered in which characters are displaced vertically. Column simply assembles the text as a vertical column or stack of single characters, neatly centred. Ripple mounts the characters horizontally but with gradually undulating displacement as though floating on a

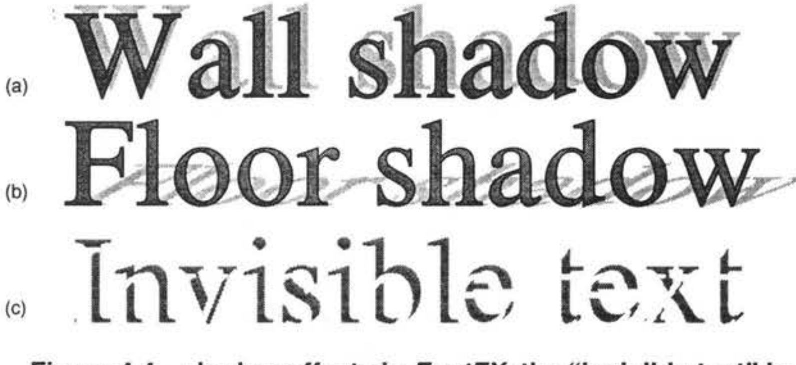


Figure 4.4—shadow effects in *FontFX*: the "invisible text" is created by making the text the same colour (white) as the background

gentle sea. Jiggle is the same effect, but the waves are shorter, indeed distinctly choppy. These effects are shown in Figure 4.5.

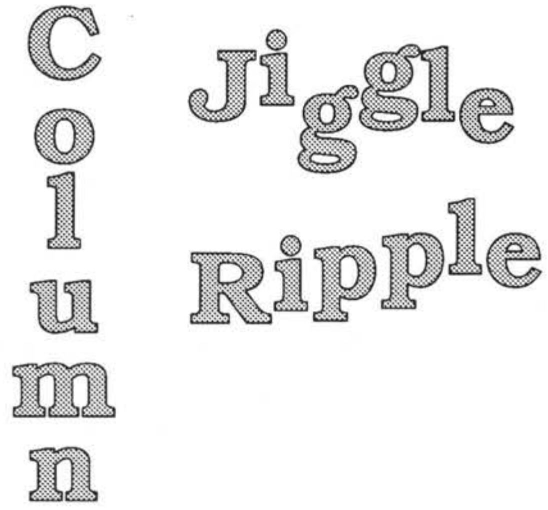


Figure 4.5—vertical displacement effects in *FontFX*

Next are a group of effects in which the characters are slewed through a specified angle; you can adjust the required angle in steps of 1 degree. Rotate simply rotates the entire converted text through the specified angle (always anticlockwise—for clockwise subtract the required angle from 360); the effect is the same as if the finished object were rotated in *Draw*. Lean leaves the horizontal strokes of the characters horizontal but rotates the vertical strokes; oblique strokes are affected *pro rata*. The text looks as if it's leaning into the wind; this allows you to create oblique effects. A combination of Lean and Rotate (the rotate must be applied later in *Draw* as you cannot combine effects within one group in *FontFX*) allows the creation of perspective effects (an example was

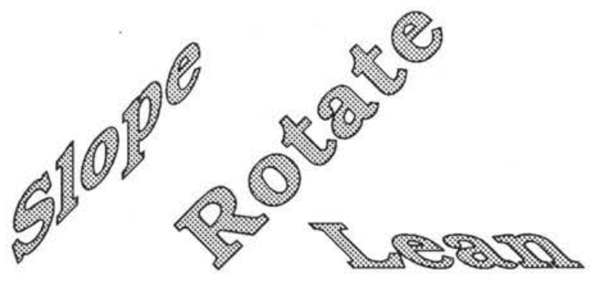


Figure 4.6–the three angled text operations in *FontFX*; the angle is 45 degrees in each example

shown in Figure 4.1). Slope is the opposite of Lean: verticals remain vertical, but horizontals are rotated, making the text look as though it's climbing a hill. Figure 4.6 shows these three effects.

The last group of effects write the text in a circle or an arc, that being part of a circle. There are options for writing the message clockwise around the top, anticlockwise around the bottom or a combination of both. Optionally you may enclose one of four shapes in the centre of the circle. And there is an option to replace spaces in the text by raised dots. Figure 4.7 gives examples. If you want more interesting effects, try one of the circular options combined with floor shadow.

When you have set up your requirements, you click on the Create option in the main menu. After a few seconds a preview window opens showing the created graphics and, usefully, giving its aspect ratio (ratio of width to height). Clicking menu over the preview window produces a save dialogue box (Figure 4.8); the graphics are exported as a file which can be dragged into an open *Draw* window, dropped

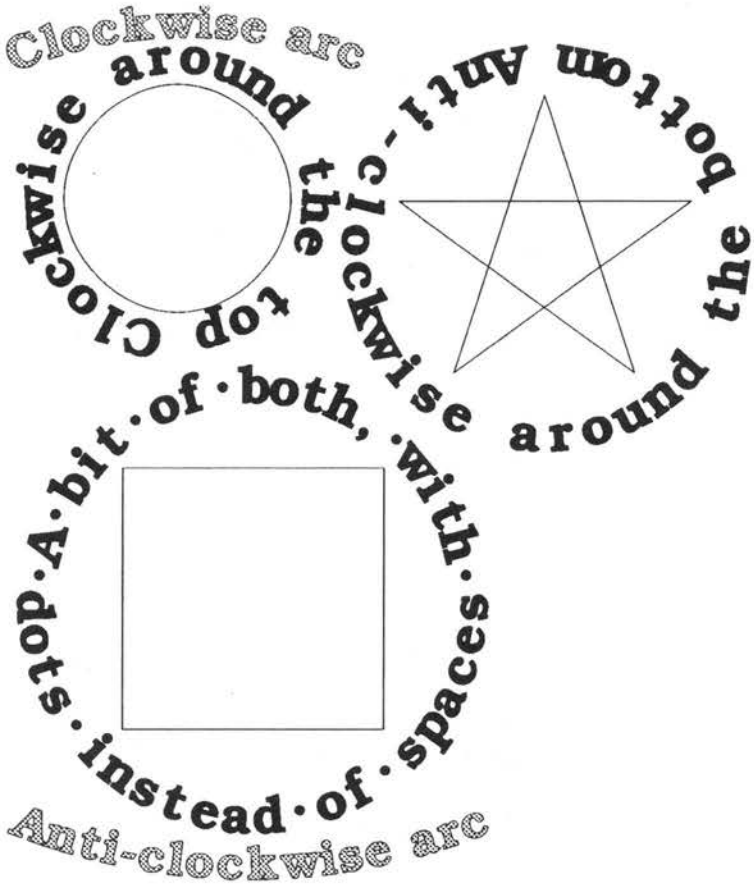


Figure 4.7—three text-in-a-circle and two arc options from *FontFX*

on to the *Draw* icon to open a new window, dragged into a graphics frame in a DTP application or saved to disc for later use.

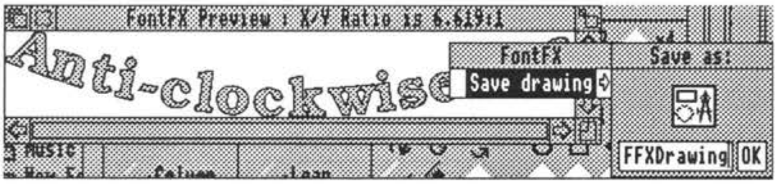
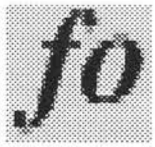


Figure 4.8—the preview window and save dialogue box in *FontFX*

One word of warning to those who are new to this kind of operation: converted text files take up far more memory space than plain text objects. Figure 4.7, for instance, is 43 KBytes long.

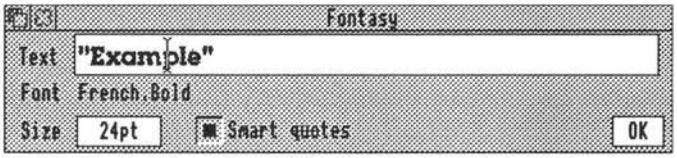
Recent versions of *FontFX* can use an effect description language. This allows you to store frequently used transformations as text files and drag them into the *FontFX* window in order to set up the specified combination of effects.

Fontasy



Fontasy from ICS offers a wider range of facilities and effects than *FontFX*. The inevitable penalty is that more thought is required to master the possibilities it offers, but it contains many delightful, sophisticated features. It is a package for the perfectionist, whether professional or enthusiast.

Clicking on the *Fontasy* icon produces the simple dialogue box shown in Figure 4.9. You enter your wording, up to 256 characters long, in the writable icon where, in the latest versions of the application, it is displayed in the currently selected font. Clicking menu in the dialogue box produces a scrolling list of the outline fonts available; click Select on the font of your choice. The currently selected font is displayed in the window; the default is Homerton Medium (if



Fontasy

Text "Example"

Font French.Bold

Size 24pt

■ Smart quotes

OK

Figure 4.9—the text-entry dialogue box in *Fontasy*

present). A writable icon allows you to enter the required font size; the default is 100 pt, but you can specify size in inches, millimetres or centimetres if you prefer. A toggle icon sets or resets a 'Smart quotes' facility. If set, this changes plain single or double quotes typed at the keyboard to the more attractive open and closed quotes in ASCII codes 144, 145, 148 and 149. When all is ready you click on the OK icon to convert the text to graphics. This opens a display window like the one shown in Figure 4.10. In fact the display window takes one of three forms: the 'original' window, the 'structure' window and the 'full' window. You can cycle between the three by clicking Adjust on the close icon.

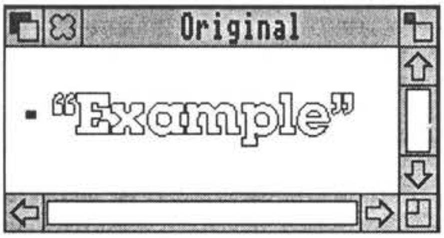


Figure 4.10—the graphics display window in *Fontasy*

The original window, as its name suggests, displays your message in the chosen font and style, but shown as an outline only. Apart from one or two text manipulations such as kerning and scaling, this display never changes. And herein lies a valuable feature of *Fontasy*. If you dislike some effect that you have tried, you can cancel it instantly by returning to the original; then you can try something else.

The structure window also shows text in outline form but reflects the various distortions and manipulations that the package offers. And the full window displays the text in its finished form with colours, shadows and other effects that have been selected.

A limited number of manipulations on individual characters is available within the original window. Click Select to select an individual character; click Adjust to add other characters to the selection or to cancel select. Selected characters, highlighted in red, can be dragged with Select; they can be scaled or reflected by dragging with Shift held down and they can be rotated by dragging with Ctrl held down. But note that a system of locks (Lock entry on the main menu) by default prevents dragging in the y-axis (vertical). The Aspect lock constrains scaling to the same scale in both x and y axes so that characters maintain their original shape. These locks may of course be cancelled to allow otherwise illegal manipulations.

Clicking Menu in any of the display windows produces a long menu of manipulations which affect the whole converted text. The first option, Line, gives the choice of line, colour, width and join styles with which we are familiar from *Draw* except that thin lines are not available. Fill gives a choice of fill colours. Colour selectors in *Fontasy* offer full 24-bit colour plus transparency.

Distort leads to a menu of three options: None, Rotate and Lean. None simply cancels any distortion already introduced and restores the text to its original form. Rotate and Lean both prompt you to enter either one angle or two angles. If you enter one, it is applied to every character; if you enter two, one will be the start value applied to the first character and the other will be applied to the last character—a gradation is applied through the text. This allows some interesting effects, especially if the angle is negative at one end and positive at the other, as in Figure 4.11. Rotation rotates every character individually, although the line of characters remains horizontal. Lean holds the horizontal strokes and rotates the verticals by a maximum of 85%. Note that in *Fontasy* all angles specify *clockwise* rotation.

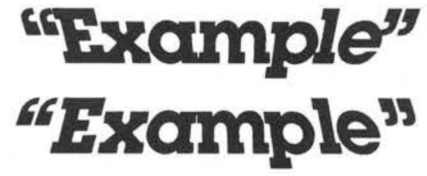


Figure 4.11—graded leans are possible in *Fontasy*

Path relates to the shape of the hypothetical path on which the characters sit. The None option turns any path effects off and restores the original. Slope simply rotates the whole line of text and the characters. To get the 'hill-climbing' effect of the similarly named effect in *FontFX* you would need to use slope in conjunction with the rotation effect to keep the characters upright. Within a submenu from Slope are two useful further options: Length prompts for a length for the slope and rescales the text to suit the specified length. If the justify toggle is set, however, the text is not scaled, but the characters are spaced out to fill the width—see Figure 4.12. The Column option gives text in a straight vertical column as in *FontFX*.

The arc facility is very versatile indeed,

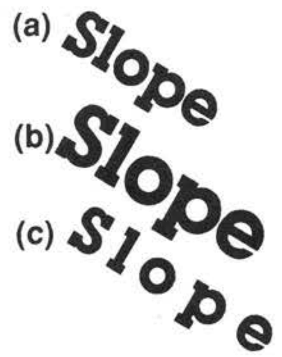


Figure 4.12—three options in *Fontasy's* Slope facility: (a) normal; (b) scaled to 30 mm length; and (c) justified to 30 mm length

effectively combining the arc and circle functions seen in *FontFX*. The dialogue box prompts for the angle, the centre and the radius of the arc. Toggles prompt for Exterior and Interior (which are mutually exclusive) and Justify. The angle is the angle subtended at the centre: entering 360 will of course give a complete circle. The radius is the distance from the arc to the centre of the imaginary circle of which it is part. You may enter an angle and allow Fontasy to calculate the radius or *vice versa*; Fontasy bases its calculations on the length of the text and the size of the selected font. Alternatively you may specify both angle and radius whereupon Fontasy will scale the text to fit the arc or justify the text to fit, if you selected the justify option. Centre is also an angle—it is the offset from vertical to the centre of the arc. If you enter 0 the arc will be at the top of the imaginary circle; if you enter 180, it will at the bottom and the text (or some of it) will be upside-down. The smaller the radius, the tighter will be the arc. Exterior and interior are the direction of the text: exterior text reads clockwise and interior text reads anticlockwise.

The Draw option allows you to import a path object from *Draw* and use this as the path along which text is fitted. A dialogue box allows a versatile choice of which part of the path to use and x-scale and y-scale values can distort the path. A reverse option runs the text backwards along the path and a justify option will space out the text to fit the length of path specified. Some examples are shown in Figure 4.13.

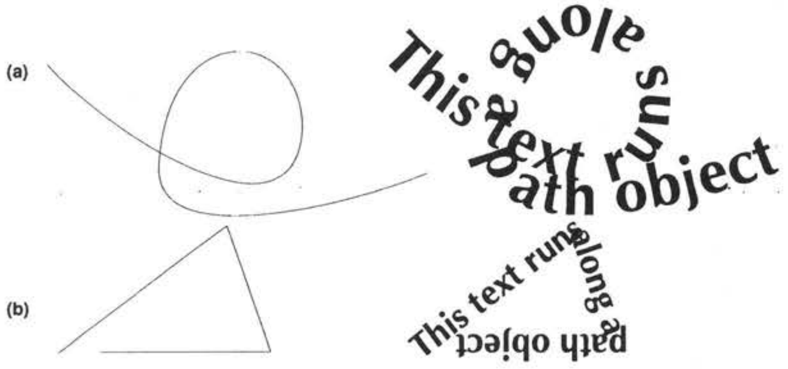

(a)
This text runs along a path object
(b)
This text runs along a path object

Figure 4.13—If you ever need your text to follow the course of a path object, *Fontasy* will make it happen. The paths used are on the left. The word breaks don't always fall conveniently at the angles; this example was carefully contrived

The shadow option allows three types of shadow and a none option that turns all shadows off. Drop shadows (wall shadows in *FontFX*) and floor shadows are both provided. Usefully in these the angle of offset and the distance between the text and its shadow can be specified; they can even be graded like the lean angles, providing some interesting effects (Figure 4.14). Block shadow gives a three-dimensional

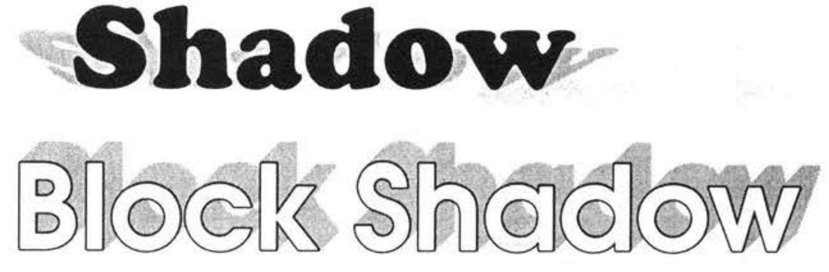


Figure 4.14—two of the shadow effects available from *Fontasy*

effect, like characters cut out of a block of thick material.

The remaining items on the main menu are concerned with scale, locks, zoom, display option and saving work. *Fontasy* will save its artwork as standard *Draw* files or as a script file, that is a special text format which also saves the menu options so that you can break off a design session and resume it later. By dropping the script file on to a different *Fontasy* window, you can force the effects described in the file on to the text in the window.

Typestudio

Typestudio from Risc Developments is in some ways more comprehensive than *Fontasy* and in other ways less so. Clicking on its icon instantly reveals its most significant difference from the two previously considered applications: it bears a striking resemblance to *Draw* (Figure 4.15); it has many of the facilities of *Draw* and to a limited extent can be used as a stand-alone package. While *FontFX* and *Fontasy* can display their graphics only in isolation, *Typestudio* allows you to see them and manipulate them in relation to other graphics, but only if they are path objects.

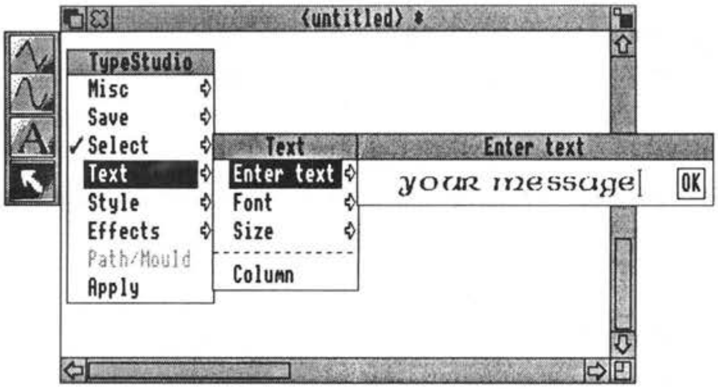


Figure 4.15—the window and main menu of *Typestudio* may seem strangely familiar...

To be precise, *Typestudio* has only a cut-down version of the vector-graphics facilities of *Draw*. It can create open-path objects consisting of lines and curves, but not moves. It cannot create closed paths except in converted text. It cannot create text objects, text entry being always converted to path objects. But this does not really matter since these facilities are provided only to allow you to to tidy up converted text and, more importantly, to create moulds and paths (more on these later) without having to load *Draw*. On a 1 MByte machine you could not have *Draw* and *Typestudio* loaded simultaneously. Other types of object are not supported. If you load a *Draw* file containing sprites, text objects or text areas, these are not displayed.

Using *Typestudio* is essentially a two-stage process. Firstly you convert your text to graphics. Secondly you perform one or more manipulations on the text-become-graphics.

From the main menu you select Text to enter your text: a dialogue box also allows you to choose your font and size; in the writable icon your text appears in the selected font. Clicking on the OK icon generates the graphics within the window. A column option, if set, forces the text into a vertical, neatly centred column.

In most respects the application closely follows *Draw's* object-based philosophy. You can select one or more objects in the normal way. A select menu like that in *Draw* allows you to group and ungroup objects, drag them, scale them, rotate them and from the Style submenu you can impose on your selection the same styles of line colour, fill colour and (out)line width that *Draw* offers. 24-bit colour (plus transparency) and thin lines are available.

The five options in the effects submenu (shadow, slant, mirror, 3-D and plinth) are a little more complex. The converted text must be selected, all preferences set and the effect ticked in the effects menu. Then you click on the A (Apply) icon in the toolbox or on the Apply option in the main menu and the effect is applied.

Very fine control of effects is provided. For instance, for shadows there is not only the same style choice as for the text itself (separate fill and outline colours and line width) but three types of shadow: besides the now familiar wall and floor shadows there is a graduated shadow (Figure

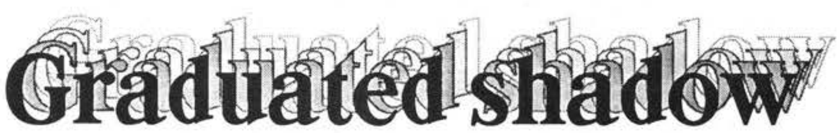


Figure 4.16—the graduated shadow facility in *Typestudio.* Note the separate line and fill colours in the shadows

4.16), effectively a series of shadows whose colour diminishes as they recede into the distance. Shadow angle and distance (height in a floor shadow) are both user-defined—the floor shadow can even be in front of the text if its angle is between 90 and 270 degrees. Slant simply leans the text, there being no limit to the lean; between 90 and 270 degrees the text is of course 'head over heels'. Mirror is effectively a variant of shadow except that the reflection is not hidden by the text; there is a choice of mirror positions (above, below, left and right), a user-definable gap between text and reflection and a full set of styles for the reflection. 3-D is similar to the block shadow in *Fontasy* with a choice of angle, distance and styles for the 'depth'. Plinth is unique to *Typestudio*. It stands each character in the centre of an identical rectangular block having user-defined characteristics. Beware! This effect strips ordinary spaces out of the resulting graphics and if you think you can use a NBSP in place of a space, you can't—it replaces the NBSP with a full stop. Other characters with codes above 128 are handled normally. Figure 4.17 shows some of the effects possible.

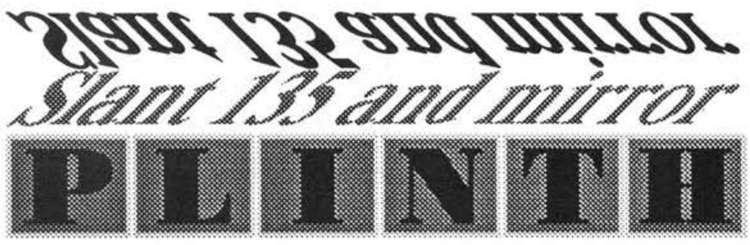


Figure 4.17—slants of over 90 degrees (top), reflections (middle) and plinths are some of the fascinating effects possible with *Typestudio*

Typestudio contains no explicit provisions for text on arcs or circles. But it does contain a text-on-a-path facility similar to that in *Fontasy.* So you draw your arc or whatever shape you wish using the drawing facilities—or import a ready-made one—select your path object in the normal way and click on the Path/Mould option in the select menu. This changes to read simply Path and a tick appears beside it to show that a path is active. The path itself is now shown in a grey dotted boundary box to indicate that it is the active path. Provision is made for text of preset size to be placed at the left, right or centre of the path or scaled to fit it or simply stretched to fit. You can apply a path retrospectively to text that has already been converted by selecting the text and clicking on the Apply icon; alternatively while the path is active any new text converted to graphics will be forced on to the path.

There is one other important facility which *Typestudio* offers: *moulds.* The term mould used in this way originated with 4Mation's *Poster* and is also used in *DrawBender.* It is a device for distorting a graphic in a controlled manner.

If you draw two path objects, select both and then click on the Mould option on the main menu, they form a special kind of group called a mould, identified by a grey boundary box. As with a path, new text converted to graphics while the path is active will be distorted into the shape of the mould and existing text that has been selected can be forced into an active mould using Apply. If the two path objects cross each other or run in different directions, amazing contortions are possible (see Figure 4.18).

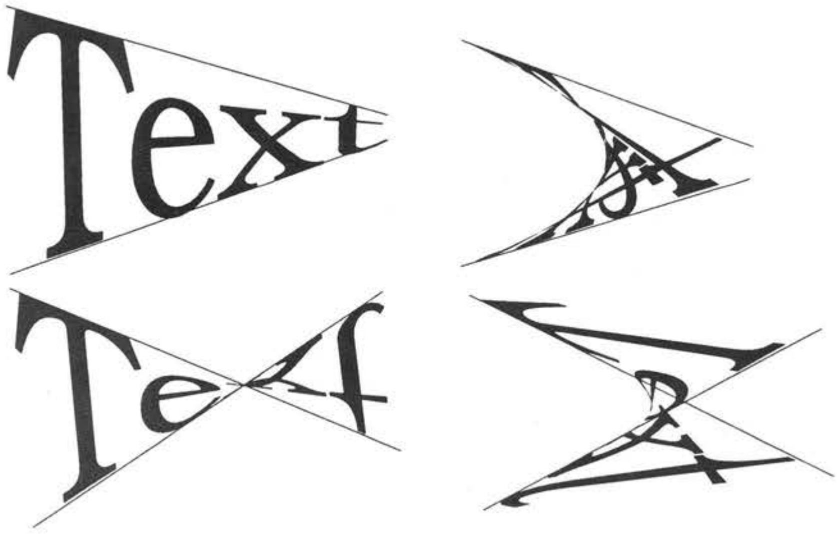


Figure 4.18—deliberate text distortion using moulds (visible) in *Typestudio*. In the two left-hand drawings the two mould lines were drawn from left to right; in the two right-hand drawings the lower line was drawn in the opposite direction

One more interesting point about *Typestudio* is that all of its special effects are applied to *graphics*, *i.e.* to path objects which may well be in the shape of text, because we took pains to convert them from text. But they can equally be applied to other path objects or groups of path objects. It can just as well add a shadow to a tree or a reflection to a bridge or, using the 3-D effect, change a square to a cube!

DrawBender

DrawBender from ICS is a path-object distortion application often used in conjunction with *Fontasy* from the same source. Used together the two applications offer similar facilities to those of *Typestudio*.

It is a very simple application to use. Clicking on the icon opens two plain windows labelled 'object' and 'mould'. Into the mould window you drag a path object which you created in *Draw* or loaded off a disc. In *DrawBender* a mould consists of a single closed-path object having at least four segments and created working clockwise. A directory full of suitable objects is supplied with the application. Four of the points in the mould must be designated as the four corners for your transformation; if the mould has more than four points you must choose (by clicking Select on them) the four to be used. To guide you a set of dots is displayed which represents the distortion of a rectangular grid that would be produced by application of the mould as currently set up—see Figure 4.19. You should choose a selection of points that keeps the dots within the mould's outline, otherwise you are likely to end up with a very tangled transformed drawing.

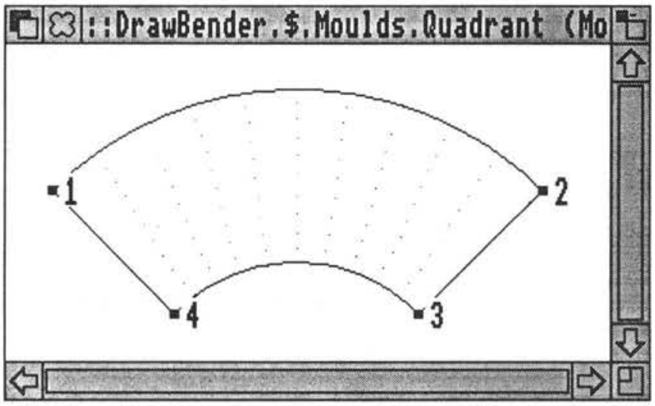


Figure 4.19—a typical mould display in *DrawBender*

Drag into the object window the path object or group of path objects which you wish to transform. It need not necessarily be the same order of magnitude in size as the mould, since the application will rescale the drawing as appropriate. Then click on Process in the mould window menu to start the transformation process. The time taken depends on the complexity of both mould and object; for complex objects it can take many minutes and there is a fast option available which produces a faster but less accurate result. The transformed object will appear in the object window and a save option allows you to save the result to disc or to another application. Figure 4.20 shows the result of a comparatively gentle transformation! Both windows have a zoom option.

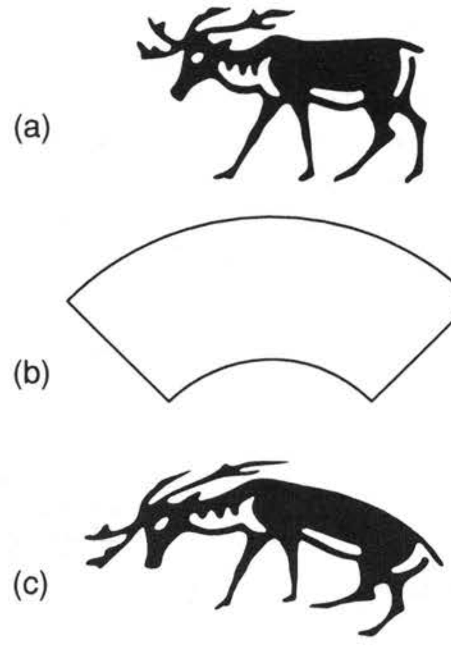


Figure 4.20—(a) a single path object and (b) a path object used as a mould, both supplied with *DrawBender*; (c) the transformed drawing obtained by processing (a) with (b)

Chameleon 2

Appropriately enough, *Chameleon* is concerned primarily with colour change. A recent major revision of this package from 4Mation, *Chameleon 2*, offers greatly enhanced facilities including the luxury of graduated colour fills. It also offers a number of other valuable facilities, including the production of CMYK separations (cyan/magenta/yellow/key separations as used in the print industry—key is black). The colour plate section in this book was produced using this facility. The version described here is *Chameleon 2* tested on an A5000.

In *Draw* it is not easy to change the colours of an object. First you must select the object (which may necessitate ungrouping it), then you call up the Style menu, select line colour, fill colour or text colour and then select either one of the 16 palette colours or manipulate the sliders until the RGB content is correct (even though you may not be able to see the resultant colour on the screen). *Chameleon* only requires you to select a colour and click the pointer over the object you wish to recolour. In practice it offers many more sophisticated options than just this. In fact, it makes possible in *Draw* files some of the image processing that is normally only possible on pixel graphics.

What *Chameleon* will *not* do is to operate interactively with *Draw*. So you cannot use it to edit the colours of artwork currently in *Draw*; you must transfer the drawing from *Draw* to *Chameleon*. Although I have referred only to *Draw* as the originating package for the drawings on which *Chameleon* operates, it does of course work equally well on *Draw*-compatible files from any source. It also accepts files from *Poster* and files in 4mation's compressed *Draw* format; it will also save files in this format. If you import files from *DrawPlus* or *Vector* it will display objects on all layers, but it will not display objects of unfamiliar types such as *Vector's* replications.

You start up *Chameleon* by dragging a file icon on to the application icon. This opens two windows, one displaying the artwork being edited and the other, shown in Figure 4.21,

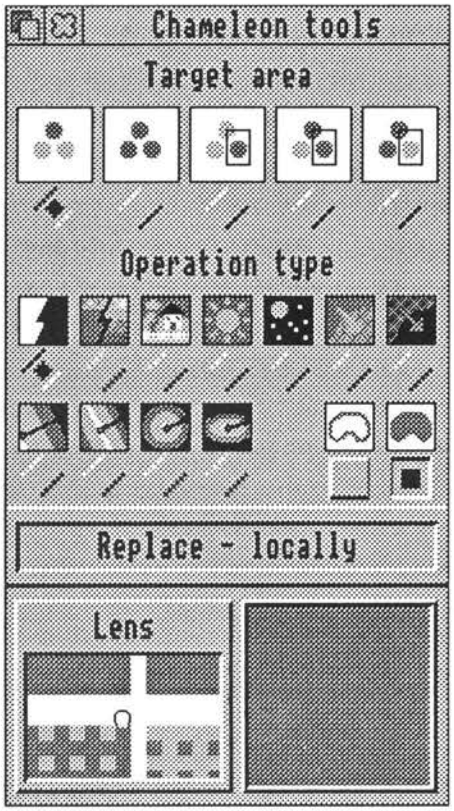


Figure 4.21—the main dialogue box in *Chameleon 2*

displaying the tools. You move the pointer in the drawing window to indicate the object you wish to edit. The 'lens' in the bottom left-hand corner of the tool window gives a magnified image, the circle in the centre identifying the pointer position with pixel accuracy.

There are five options for identifying the target object: Local affects only the object immediately beneath the pointer; Global affects all objects beneath the pointer *and* all other objects having the same colour. The other three options require you to drag a box in the drawing window. Within box affects all objects totally enclosed by the box. Beneath box affects all objects overlapping the edges of the box. Not within box is applied globally to all objects in the drawing *except* those totally enclosed by the box. It follows that there is no way you can change the colour of a single object that is wholly hidden behind another. But since such an object cannot be seen, why should its colour matter?

Chameleon disregards groups and layers. You can change the colour of an object even if it is part of a group or on an unselectable layer (see Chapter 5 for more on layers). You can also change the colour of text in a text object or a text area and you can change the colour of sprite objects in *Draw* files. Note, however, that each colour in a sprite behaves as though it were an object. If, for instance, you replace a black pixel by blue, you will find that every black pixel in the sprite has been changed to blue. You cannot change a fill colour or line colour which is currently set as transparent.

You choose the replacement colour by clicking on the Colour select icon at the bottom right of the tool window. This calls up a standard 24-bit plus transparency, three-slider colour selector. But this can also be supplemented by a 256-colour palette (like the one used in *Paint* in 256-colour modes), a 256 grey-scale palette or a 24-bit RGB colour cube (illustrated in Figure 5.6). Of course, the 256-colour palette only appears in 256 colours in 256-colour modes. The 256 grey-scale palette will appear to be in 16 shades of grey in 256-colour modes in RISC OS 2, but in RISC OS 3 dithering gives some extra shades. In RISC OS 3 the RGB colour cube appears to give almost 4096 colours in 256 colour modes, but some adjacent dither patterns are barely distinguishable. The colour cube is certainly a powerful means of selecting colours. You begin by selecting the red content from the scale beside the cube; this selects the 'slice' within which you move the pointer to the right to increase the blue content or upwards to increase green content.

You can also copy a colour off an object in a drawing by clicking Adjust over it. This makes it easy to copy one object's colours to another.

Chameleon offers far more than straight replacement of the former colour by the chosen colour. There are seven alternative colour replacement operations. To understand some of these we need to digress into the HSV (hue, saturation, value) system of colour specification which is based on the way the human eye perceives colour. The *hue* is the actual colour (wavelength), *e.g.* red or green. The *saturation* represents the amount of white that is added to the hue; the less white, the higher the saturation. The *value* (sometimes called *volume*) represents the brightness of the colour, effectively the inverse of the level of black in it.

The first operation is the simple replacement of the original colour by the one chosen. The second changes the hue of the chosen colour, but leaves the saturation and value levels unchanged. The next changes to a shade of grey (effectively removing the hue). Brighten increases the brightness (value) by reducing the black content. Darken reduces the value, increasing the black content. Weaken increases the amount of white, *i.e.* decreases the saturation. Strengthen increases the saturation by reducing the white content. In this way you can easily fine-tune the colour of any object.

If you make an unfortunate change you can restore the original by pressing F8. Up to 256 colour changes are stored in the undo buffer.

Chameleon 2 has separate icons which allow you to select whether you are working on line colours or fill colours.

One of the most exciting aspects of *Chameleon 2*, however, is its facility for 'fountain fills', otherwise called graduated fills. In these an object is given not a plain fill colour, but a range of fill colours which can suggest sheen or shading and hence texture. Careful use of fountain fills can make your *Draw* graphics look like ray-traced sprites! The option works best on screen if you have RISC OS 3 and can make use of its dithering facility. Four fountain fill options are provided. In a linear fill the object is a filled with a band of colours merging from a start colour to an end colour. The direction of the band and start and end points can be controlled by the user. A band fill is similar, but the fill extends in two directions, the start colour being in the centre and the end colour being at either side. The angle of the banding, the width of the start band and the two end points can all be controlled. The radial fill effectively provides concentric circles with a gradual range of colours. The positions and sizes of the innermost and outermost circles can be controlled. The elliptical fill is similar, but the width and height of the ellipse can be controlled.

Figure 4.22 shows the sort of effect that is made possible by fountain fills. In practice, the fountain fill consists of a series of intermediate objects having intermediate fill colours; the interpolate/grade/blend facilities in *Draw 3* and some other packages can also be used to create this kind of effect. When you transfer a drawing containing a fountain fill to another application, there may be limits on the further processing that may be applied to the fill. In *Draw* you will be able to move and scale fountain-filled objects. You will not be able to re-edit the fountain fill in *Chameleon* unless you chose an option that stores additional data in the file, but these additional data can cause problems when fountain-filled objects are processed by some other applications.

Chameleon offers the option to save not just *Draw*, compressed *Draw* and *Poster* files, but also to produce colour separations. These may be CMYK separations of material for commercial

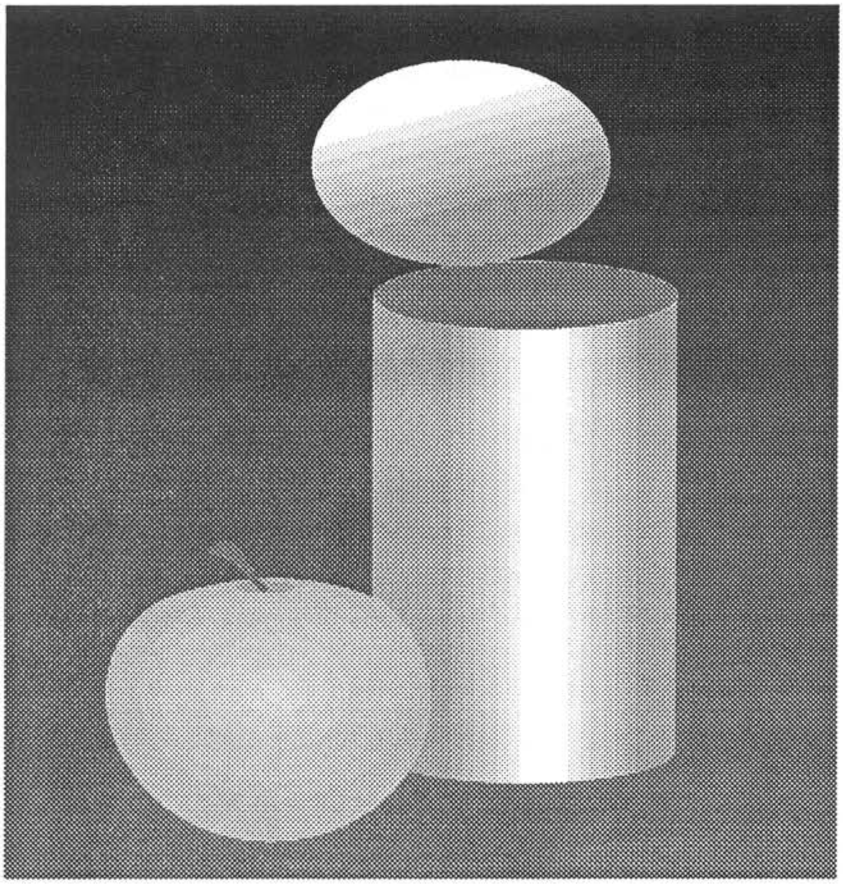


Figure 4.22—fountain fills created in *Chameleon 2*. The apple uses an elliptical fill, the tin a band fill and its lid a linear fill

full-colour printing or spot colours. In spot colours, a separate drawing is created for each colour used (maximum 64). The separations are saved as separate files within one directory. Crop marks can be automatically placed on the colour separations to aid registration.

Placard

Placard from ICS is the last of the adjunct applications considered in this chapter. *Draw* is capable of handling of drawings as large as size A0 (841 × 1189 mm), but most printers will handle only a nominal A4, some a nominal A3. Consequently the printing of very large drawings is complicated. Of course, by selecting all and grouping, you can drag the entire contents of the drawing so that each A4-size portion of it is in turn brought within the print limits window and printed. That, however, is a clumsy process and joining together the various sheets that emerge from the printer is not easy. *Placard* makes the process much easier, but it only works with *Draw* files, whether created by *Draw* itself or other applications.

First load your printer driver, since the application needs to know the size of the printable area. Then drag your *Draw* file on to *Placard's* icon. This opens a window displaying

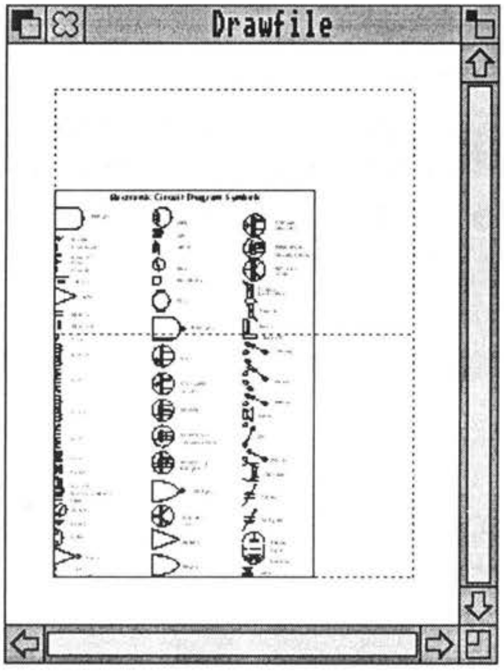


Figure 4.23—ICS's *Placard* helps to print those drawings that will not fit on one sheet

the drawing enclosed by a solid red boundary box (Figure 4.23). You will also see a number of dotted red boxes, each representing the area of a printed page. So you will see at once how many printed sheets will be needed for your drawing. You may decide on a preferred orientation, portrait or landscape, for the printouts, or you may leave it to the application to determine which orientation will use the smallest number of sheets of paper.

A zoom facility is provided so you can see the layout clearly, and a scale facility allows you to resize your drawing instantly. If, for instance, your drawing was only just over A4, you could scale it down so it fits a single A4 sheet or you could scale it up to make full use of an A3 sheet or two A4 sheets. It depends on your requirements. A margins facility allows you to reposition your drawing.

A wide range of print options includes the printing of dotted lines and page numbers as an aid to cutting printed sheets and even tabs to help in gluing the printed sheets together.

5 Advanced Vector Graphics Packages

Besides *Draw* there are now three other major vector graphics/object-based drawing packages for the ARM machines. These are *DrawPlus*, *Vector* and *ArtWorks*. All offer useful facilities not available in Acorn *Draw*. Although most have their own individual file formats, all accept standard *Draw* files and will save drawings in that format. This allows artwork from all packages (sometimes with limitations) to be viewed in *Draw* (ensuring a wide potential audience) or to be imported into DTP and other applications that accept *Draw* files. It also facilitates the exchange of drawings between applications during preparation so that the various facilities offered by different packages can all be used.

DrawPlus

Jonathan Marten's *DrawPlus* first appeared in 1991 and is available at nominal cost from many Public Domain and Shareware sources. If you have RISC OS 3 you will need version 2.12 (or later). Although inexpensive, there is nothing unprofessional about *DrawPlus*; it is a thoroughly competent and polished production —indeed many of its features have been incorporated in *Vector* (see later).

Intended as an enhanced version of *Draw* improving many of its existing facilities and offering some new ones, *DrawPlus* retains nearly all of the features of the RISC OS 2 *Draw* and this account will concentrate on the differences between the two packages. My remarks apply to version 2.12 and not necessarily to other versions. To eliminate the few negative aspects first, the New view facility and the little known provision for copying objects between windows are missing. Since the appearance of *DrawPlus*, the RISC OS 3 release of *Draw* has appeared offering further facilities such as dithering and interpolation which are not supported in *DrawPlus*. For a while I used the two applications in parallel, most work being in RISC OS 3 *Draw*, but with much passing of artwork to *DrawPlus* in order to use such facilities as its extended zoom and object order control. A new version of *DrawPlus* which will make use of the new facilities in RISC OS 3 and have some additional features is promised.

Clicking select on the *DrawPlus* icon opens two separate windows: a plain drawing window exactly like that in *Draw* with the toolbox turned off and a separate horizontal toolbox window as shown in Figure 5.1. Being a separate window it can be moved around the screen to a convenient spot. It can also be turned off (and on again) via the menus. Beneath the toolbox icons are a status line which describes the last action or selection and an indicator of the currently selected layer (more on layers later).

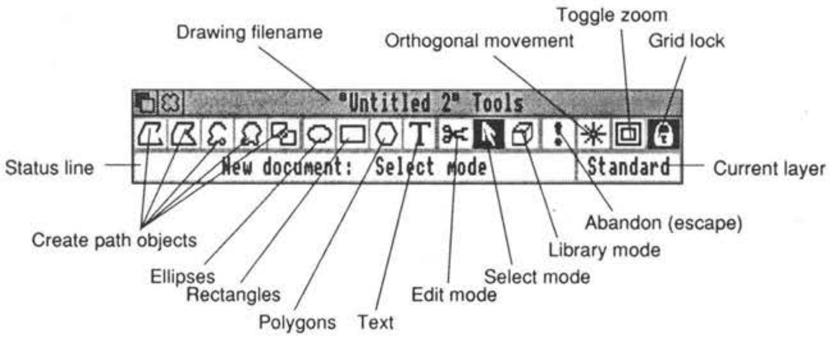


Figure 5.1—the toolbox window in *DrawPlus*. Since it is a separate window, it can be moved around the screen as required. The status line provides useful information about the last operation

The first seven icons are identical in function to the top seven icons of *Draw's* toolbox, being concerned with the creation of path objects. The eighth is an innovation: it creates finished regular polygons, the number of sides being adjustable via the Create menu; the default number is three. Path creation itself is similar to that in *Draw* but clicking Adjust deletes the last point created; you double-click Select to finish the path.

The text mode icon is as in *Draw* but you can edit text objects using the cursor keys during preparation and you can also edit finished text objects by selecting them and then entering edit mode. You cannot rotate text objects, even in RISC OS 3, but there is a built-in text-to-path facility and you can, of course, rotate text that this has converted.

Select mode is much as in *Draw* but with a few useful innovations. On entering select mode the last object drawn, edited or imported is automatically selected. Selected objects are distinguished by a solid red boundary box having a handle at each corner; any corner can be Select-dragged to rescale the object, while Adjust dragging any corner will rotate it, but only if it is a path object.

To make it easier to select objects by the 'box dragging' method, the box must encompass the centre of the object. An object that is just touched will remain unselected—this is useful when you get used to the idea. Also you can select objects by clicking Select within a millimetre of them—useful when the object is a thin line that is exactly vertical or horizontal and therefore has a boundary box that is infinitesimal in area.

The Select menu offers the familiar facilities plus locking and unlocking of objects. Locked objects can still be selected but cannot be moved, edited or deleted, although they can be copied and saved as selections. New facilities are 'forward' and 'backward' stepping of objects on the data stack—the ability to step a single object forward or backward to its required 'depth' is valuable, especially when the object concerned has become hidden because a grouping operation has brought other objects, normally behind it, to the front. The Arrange menu and the Special menu offer further operations on selected objects; these will be considered later.

Libraries

The next icon which looks like a cube is the library icon. Libraries provide an efficient means of storing and retrieving often-used graphics. Supplied with *DrawPlus* are three example libraries, one containing fancy fonts, one containing electronic circuit diagram symbols and one containing general symbols, mainly cartographic. With library mode selected, clicking Select will plot the currently selected drawing from the library at the pointer position. A Library menu includes the Show... option which opens a library window as shown in Figure 5.2. This includes a scrolling list of the titles of the items and a graphics pane in which the item currently selected (by clicking on its name) is displayed. Comprehensive facilities are

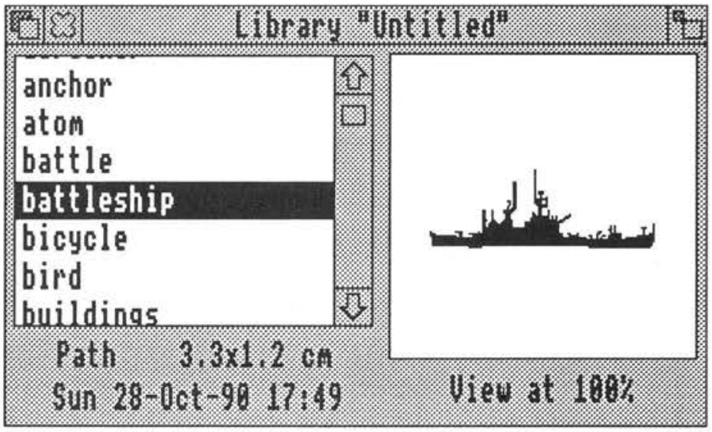


Figure 5.2—*DrawPlus'* library facility provides a convenient method of storing and retrieving frequently needed graphics

provided for changing the contents of libraries and creating new ones.

The next icon containing an exclamation mark is an Abandon facility. Clicking on it aborts the current operation and is equivalent to pressing Escape.

The final three icons are all toggle switches; that is to say, each one is a facility that is turned alternately on and off by clicking Select on it; all are independent and are used in conjunction with other operations.

The first, which looks like an eight-pointed star, is *orthogonal movement constraint.* When enabled, it constrains pointer movement to the horizontal, vertical or 45 degrees. When used in conjunction with the rectangle tool, it ensures that the rectangle is a square and with the ellipse tool it ensures that the ellipse is a circle. In path creation it is useful in ensuring that lines are exactly vertical or horizontal. And if you forget to apply it, the edit menu contains Horizontal and Vertical options that will adjust the endpoint of the selected segment so that it is truly horizontal or vertical.

The icon containing concentric squares, is a quick zoom facility. It toggles between actual size (1:1) and any pre-set zoom setting (the default is 2:1). *DrawPlus* supports zooming up to 99:1 and down to 1:99. Zooming up and down are also available using the keypresses Ctrl-Up and Ctrl-Down and a conventional zoom dialogue box is provided from the Settings menu.

Lastly, the lock icon appropriately enough turns grid lock on and off.

Layers

Layers are a facility in *DrawPlus* which may be new to those whose experience of object-based graphics has been limited to *Draw*. We shall meet them again in this chapter. The term *layer* is unfortunately deceptive since it suggests some sort of zoning of the data stack. In fact layers have nothing whatever to do with the positions of objects in the data stack, *i.e.* their order from front to back. 'Categories', 'levels' and 'overlays' have been suggested as alternative and more appropriate names, but 'layers' has become accepted.

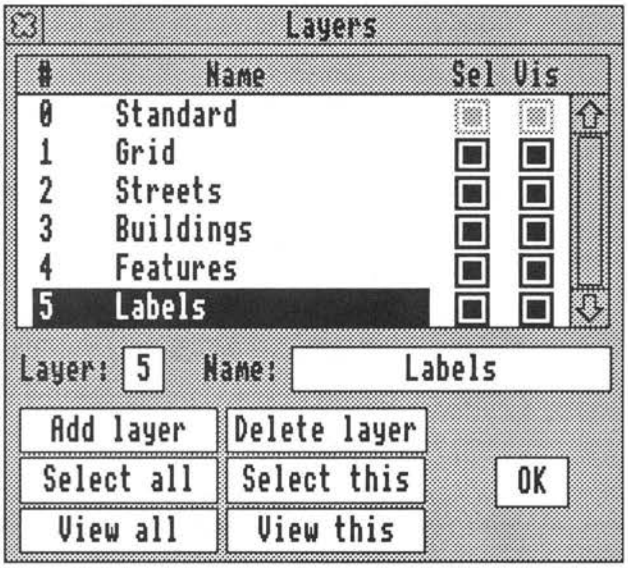
Every object in the drawing belongs to one of up to 32 layers. You may ignore the facility if you wish, whereupon all objects will automatically be in the default layer 0, named Standard. Each layer may be named or renamed by the user. For instance, if you are drawing a detailed large-scale map of a town, you might have layers named Grid, Streets, Buildings, Features and Labels. Layers are simply categories in which related or similar objects are placed. Their practical value is that whole layers may be made visible or invisible and selectable or unselectable. Incidentally, the default layer 0 is somewhat different to other layers: it is always present; it cannot be deleted or renumbered, although it can be renamed. Furthermore it is always both visible and selectable; this cannot be changed.

So, when you are satisfied with some finished objects and you wish to protect them from accidental movement, deletion or other alteration, you can assign them to a certain layer and make that layer unselectable. The objects in it will still be visible but cannot be altered by subsequent operations. They will not become selected when you are trying to select other nearby objects.

And if you should find that those objects are making it difficult to see other parts of the drawing when you are working on them later, then you can make that layer invisible. It will vanish, although still present in the stack. When you have finally finished all the objects you can restore the visibility of that layer and your artwork will be seen in its full splendour.

You will probably be able to think of practical applications for layers quite apart from their assistance in creating artwork. They add an extra dimension to drawings by providing for features to be turned on and off as required. Returning to the example of the map showing a vast range of facilities in the district covered; if all the facilities were shown superimposed, the map would be an indecipherable jumble, but by placing different kinds of facility on different layers, you can choose to make visible only those in which you are interested. 4Mation's *smArt* linked graphics system in which different objects become visible depending on the user's menu selections works in a similar way.

Layers are controlled by a Layers dialogue box (Figure 5.3) called from the Misc menu or by clicking Adjust on the current layer indicator in the toolbox window. It provides for new layers to be defined and the current layer changed. Objects may be moved to the currently selected layer by selecting them and clicking on a New



#	Name	Sel	Vis
0	Standard	▨	▨
1	Grid	▣	▣
2	Streets	▣	▣
3	Buildings	▣	▣
4	Features	▣	▣
5	Labels	▣	▣

Figure 5.3—the layers dialogue box in *DrawPlus*

layer option in the Arrange menu. Layers can also be deleted. This does not destroy the objects in them; objects from deleted layers are transferred automatically to layer 0. If a drawing containing objects in layers is saved in *DrawPlus'* 'new format' and subsequently loaded into *Draw* only the objects in layer 0 will be visible.

Incidentally, some versions of *DrawPlus* have an additional layer-type feature which is related to the order of objects in the stack. It allows you to select certain objects and allocate them as the background. They are moved to the bottom of the stack, *i.e.* the back of the drawing, and made unselectable so that they cannot be altered except by using a Clear background instruction.

The Arrange menu also controls operations on selected objects. Besides conventional scaling and rotate options, horizontal and vertical skews are available which slew the points in an object a distance depending on the slew angle selected and their distance from a centre line. It will convert rectangles to parallelograms and can turn converted Roman text to an oblique style. Horizontal and vertical reversing (mirroring) are also provided. A size/position dialogue box allows you to position objects with great accuracy. Distribute and space/pack options offer a diversity of interesting ways in which objects can be spaced out or packed together; some of these are shown in Figure 5.4.

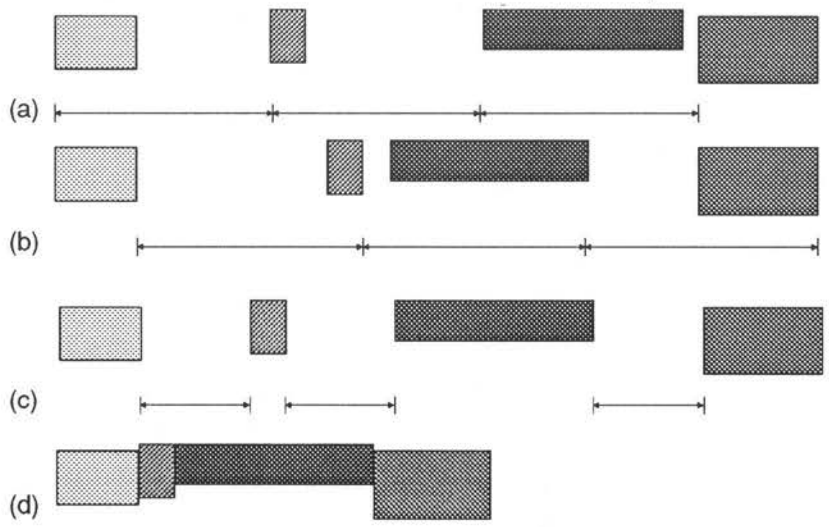


Figure 5.4—the distribution and spacing facilities in *DrawPlus*. (a and b) left and right distribute divides the space to the left or right of the rightmost or leftmost object without regard to the width of the objects themselves; (c) space separates the objects by the equally divided free space and (d) pack packs them together

The Special menu provides further operations on selected objects. Text-to-path converts text objects to path objects, making them suitable for rotation and other graphical manipulations. Explode text converts a text object to a group of single-character text objects which can then be effectively kerned or spaced using the space/pack options described above. Resize sprite returns a sprite to its natural size, that is the size it had when introduced. Bounding box draws a bounding box in the current path style around the selected object and separated from it by a user-defined distance.

Other new facilities include 10 line patterns in addition to solid lines, which can be tailored to suit your requirements by a facility in the Misc menu. The grid system is more versatile than that in the original *Draw* allowing the lock grid (the grid on to which objects are constrained) to be coarser or finer than the grid which appears on the screen. This permits the use of a fine lock grid, without the screen becoming cluttered.

When you save a drawing in the 'new format', all the settings (grid spacing, grid lock on, text and path styles, zoom setting *etc.*) in force at the time are saved with the drawing and restored when you reload the drawing. This makes it easy to resume a drawing session that was interrupted. In addition to this a Preferences facility (from the icon bar menu) allows you to choose your preferred settings for many variable features in the application and to save these in a file that will be automatically read whenever the application is started up.

All in all, a thoroughly useful application for anyone interested in graphics and, since it is available at PD/shareware prices, exceptional value for money.

Vector

If you have used *DrawPlus* you will find much in 4Mation's Vector that is familiar. This is hardly surprising since both applications were written by Jonathan Marten. Hailed as 'the enthusiast's drawing package', *Vector* combines all the features of *DrawPlus* (such as layers, libraries, bounding boxes and the comprehensive Arrange facilities) and those of the RISC OS 3 release of *Draw* (such as dithering and interpolation) with several new and fascinating facilities such as 4-point curves, path merge and split, masks and replication. The following description is based on version 1.01, most of my testing being on an A5000.

Loading a file or clicking Select on the icon bar icon opens a single window more or less like the one shown in Figure 5.5. The toolkit of 22 icons

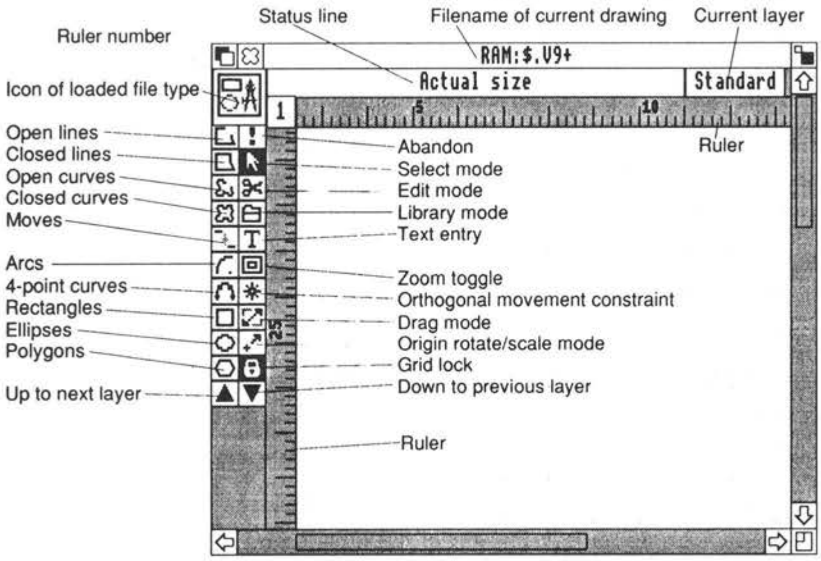


Figure 5.5—principal features of a *Vector* window; the toolbox, rulers, status line and layer name display are all optional and can be removed if not wanted

is on the top left-hand side of the window as in *Draw* and the status line and current layer indicator are at the top of the window. These features may be turned on and off from the Display menu and your preferences may be saved with the drawing.

Many of the icons in the toolkit are identical to those in *DrawPlus*. There are two new create tools, both concerned with curves. 4-point-curve provides an alternative method of producing curved segments. Its curves have four points: one at each end (so shared with any adjacent segments) and two intermediate points which, unlike Bezier control points, are actually on the path of the curve. These are placed sequentially with the pointer during path creation and so provide a more natural, almost freehand, method of creating curved paths. You can mix 4-point curves and other segment types within the same object. If you subsequently edit the 4-point curves, you find that they have become conventional Bezier curves.

The other new create tool is Arc. With Arc selected your first click defines the starting point of the arc and your second the centre of the circle of which the arc forms part; thereafter the pointer position determines the radius on which the endpoint of the arc lies. Clicking Select finishes the arc and allows you to select another tool to add lines, curves, moves or further arcs to your path object. When the object is finished, you find each arc consists of a number of conventional Bezier curve segments (one for each 90 degrees or part thereof).

There are two new toggle tools. Drag mode toggles between corner-corner drag mode and centre-corner drag mode. Corner-corner drag mode is the rough object scaling facility familiar from *Draw*: Select-dragging one corner of the object leaves the diagonally opposite corner fixed and so changes the size and/or shape of the object. In contrast, centre-corner drag mode drags the corner relative to the centre of the object and causes the opposite corner to move an equal distance in the opposite direction. It resizes the object leaving its centre at the same location.

Origin rotate/scale mode toggles between normal rotation and origin rotation. In normal rotation Adjust-dragging the handles rotates the object about its diametrically opposite corner (not its mid-point as in most packages), while origin rotation rotates the object about the drawing's origin or rescales it using the origin as point of reference. The origin is normally the bottom left-hand corner of the drawing, but an option in the Miscellaneous menu allows you to redefine the origin associated with the current ruler—and you can have up to four rulers per drawing; the rulers also control the grid. It might be useful sometimes, for instance, to have the origin in the centre of the drawing.

The two triangular icons at the bottom of the toolkit provide a quick method of changing the current layer. In other respects the layer system is identical to that of *DrawPlus*—indeed if you create a *DrawPlus* drawing with objects on several layers and save it in the 'new format', the resulting file will load into *Vector* with the layer information intact.

The library system is also very similar, *Vector* being able to use libraries created by *DrawPlus*. Even the library file icons are identical. *DrawPlus* does not recognise *Vector's* library files unless they are saved in uncompressed format.

Vector defaults to its own compressed file format which reduces most drawings to about 50% of the space they would otherwise occupy and so allows you to fit many more drawings on to a disc. You may, however, save files in standard *Draw* format. These will of course be compatible with *Draw* and other applications that use *Draw*-type files, unless they contain objects of types that are peculiar to *Vector* such as replicas and masks. *Vector* also offers a compressed *Draw* format and *Poster* format.

In RISC OS 3 sprites and text objects can be rotated. Text objects can still be edited after rotation, although they are temporarily returned to the horizontal during editing.

One of *Vector's* attractions is its excellent provision for working with colour which it appears to have inherited from *Chameleon*. Although the standard 16-colour palette with the 24-bit sliders appears at every opportunity, you can also call up a 256-colour palette (which only appears in 256 colours if a 256-colour screen mode is in use), a 256-grey scale palette or an RGB colour cube. In RISC OS 3 this produces 256 colours in 16-colour modes and theoretically 4096 colours in 256-colour modes. The latter two facilities depend on RISC OS 3's dithering option to simulate most of the shades. On multisync monitors many of the dithered shades are hardly

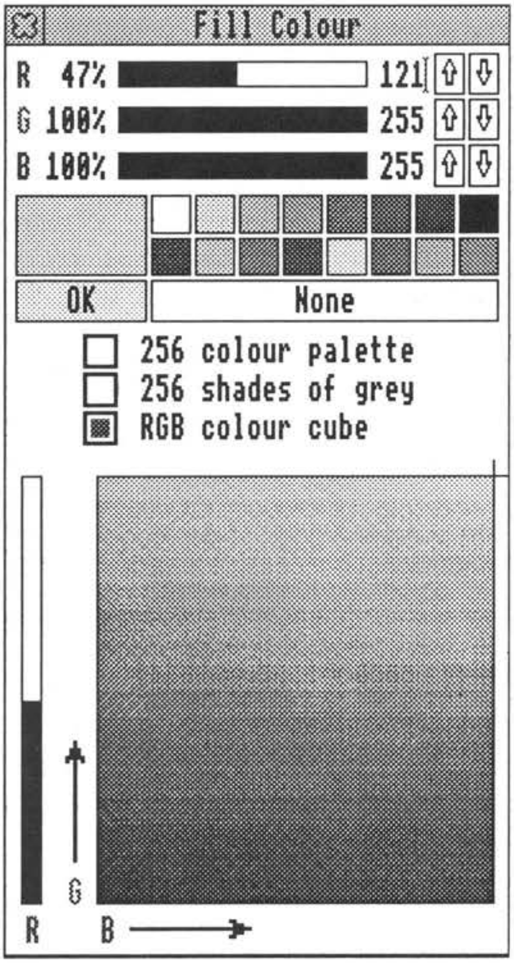


Figure 5.6—*Vector's* RGB colour cube allows easy access to 4096 colours in 256-colour modes in RISC OS 3. Most of the colours are, of course, simulated by dither patterns made up from the 256 colours of the RISC OS palette

distinguishable from true palette colours. In fact, the colour cube offers more than 4096 colours; careful moving of the pointer within any of the 256 squares in each slice of the cube will produce a range of shades in the current colour indicator above the OK icon. Another thoughtful touch is that you can save a palette with each drawing—especially useful if your drawing uses shades you have specially mixed.

The style menus for both path and text objects contain two new and useful facilities: Set as default and Apply default. If you select an object and then click on Set as default, its style characteristics will be made the default and will be applied to any new objects that you create. They can also be applied to existing objects if you select them and click on Apply default. This makes it quick and easy to transfer one object's styles to other objects.

Order facilities

Order manipulations are handled by an order dialogue box from the Select menu. This offers the same four operations as in *DrawPlus*, *i.e.* Front, Back, Forward, Backward, but also another valuable provision. If a single object is selected, its position on the stack is displayed, the first (rearmost) object being number 1. This number is writable, so you can move a single object instantly to any desired position in the stack, the objects between its old and new positions being moved one place backward or forward as necessary. This facility will be especially useful to those who use Ace Computing's *Tween* to create animations from *Draw* files.

New path utilities

In its Special menu *Vector* provides a set of path utilities that are not available in *Draw* or *DrawPlus*. These are Reverse path, Merge paths, Split subpaths and Split to Lines.

Reverse path simply reverses the polarity of the path, making the starting point the end point and *vice versa*. The effect is only noticeable if you have employed end caps or start caps since these will move to the opposite ends of the paths or subpaths. It can also affect the application of fill colours if you subsequently merge the path with other paths and apply the non-zero winding rule to the composite object.

Merge paths is a remarkably useful operation that one day will be standard on all vector graphics software. If several path objects are selected, it merges them into one path object having the style of the lowest (rearmost) of the parent objects; each former object becomes a subpath (or group of subpaths) separated by moves (unless the end point of one object and the starting point of the next were exactly superimposed). As in grouping, this operation brings the new object to the top of the stack, *i.e.* the front of the drawing.

You may wonder what benefits this seemingly obscure operation offers. There are several. Firstly, if you have a number of similar path objects in your drawing, *i.e.* all share the same style, they occupy far less memory as one combined path object than they do as separate objects, each carrying an identical full set of style data. I reduced one *Draw* file from 64K to 46K in this way. Moreover the computer will draw them more quickly, since it does not have to repeatedly read and apply the same style information. Secondly, when you merge several overlapping path objects, the winding rule comes into effect regarding the filling of areas

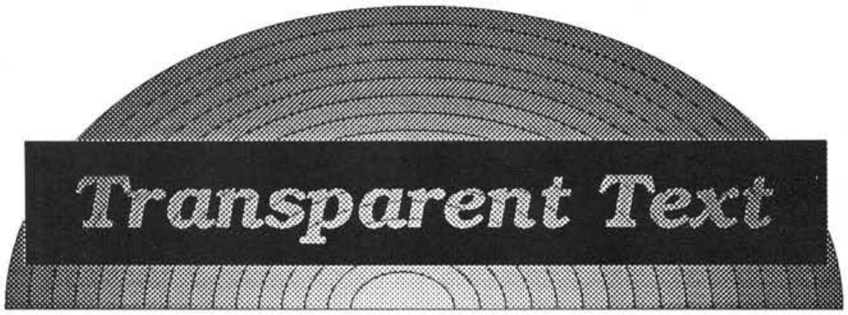


Figure 5.7—one of many text effects made possible by *Vector's* path merge facility

where subpaths overlap. This can be used to create some interesting effects such as text which, instead of having a fill colour, is transparent and through which some other design, lying behind, can be seen, as shown in Figure 5.7. Type your text and choose a suitable style (bold fonts work best); use the text-to-font facility and ungroup the transformed text. Now draw a box around the text, select everything and choose Merge path. The effect is similar to reversed-out text: by application of the even-odd winding rule the original text fill colour (black) now fills the box while the formerly black strokes of the text are transparent, like holes in a black stencil. You can of course change the black fill colour if you wish and you can give the text a different outline colour (it will also be applied to the box). Now devise your background design, drag it over the stencilled text and then send it behind. In fact you have used your text as a *mask*—we shall meet masks again shortly.

Split subpaths and Split to Lines both split a selected path object into separate objects. Split subpaths is the exact reverse of Merge paths; it splits the parent object at the moves so that each subpath becomes a separate path object. Split to Lines is more drastic: it makes every segment in the original object (except for any moves) into a separate single-segment path object.

Incidentally, you can use Split to Lines to break up a very large path object and divide it into several smaller ones; having split it, do a box-select on one area and use merge path to combine the paths in that area. Repeat until the original graphic has become as many separate paths as you require. The benefit of this is that the separate objects can have different styles.

Masks

Another interesting facility from the Special menu is the Mask facility. A *mask* is a special kind of group consisting of a path object overlying another (normally larger) object or group of objects. The two objects are selected and the Mask facility is called: the dialogue box will only allow you to create a mask from the two objects. The effect is that the larger object

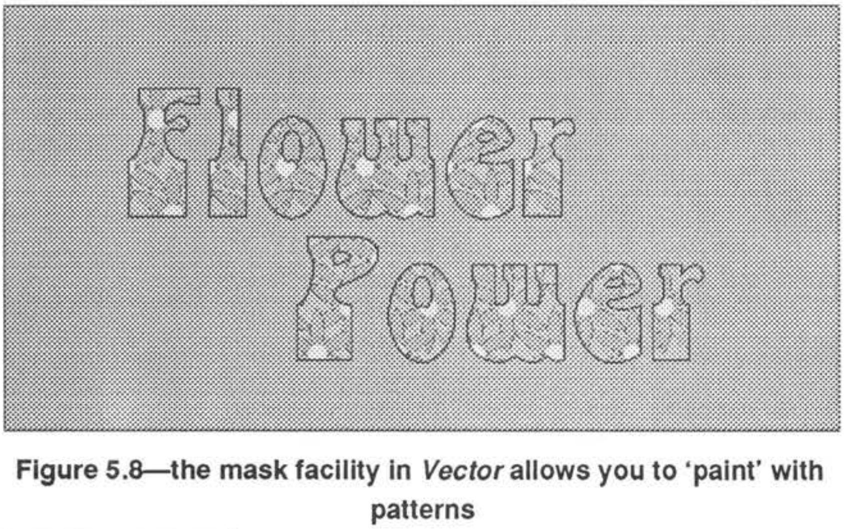


Figure 5.8—the mask facility in *Vector* allows you to 'paint' with patterns

becomes the fill colour of the path object, being visible only through those areas of the path object that would be filled according to the winding rule currently in force. If some text is converted to path, ungrouped and then path merged, it can be used as a mask over a pattern whereupon the text will appear to have been painted in that pattern (see Figure 5.8). If the Mask operation is applied to an existing mask it will 'extract' it, restoring the two original objects.

Replication

We met replication in Chapter 3 on the font editor. It is a memory-saving form of object copying which does not duplicate the full definition of the original object, but instead places on the stack a pointer to the original, so that the original is reproduced in more than one place. *Vector* offers two forms of replication. (In fact the Replicate dialogue box from the Special menu also allows an intelligent conventional copy facility which produces rows or columns of full copies at precisely controlled intervals.) A *static* replica consists of an object replicated at regular intervals in horizontal, diagonal or vertical lines or two-dimensional arrays. The result is one compound object which can be copied or scaled, but not separated or rotated. A static replica can be converted to a *dynamic* replica. This differs from a static replica in that it can be separated into its constituent replicas and these can be individually moved, scaled, copied or deleted. But they cannot be individually edited and all are technically still one object. A dynamic replica can be further converted to individual objects, allowing the drawing to be

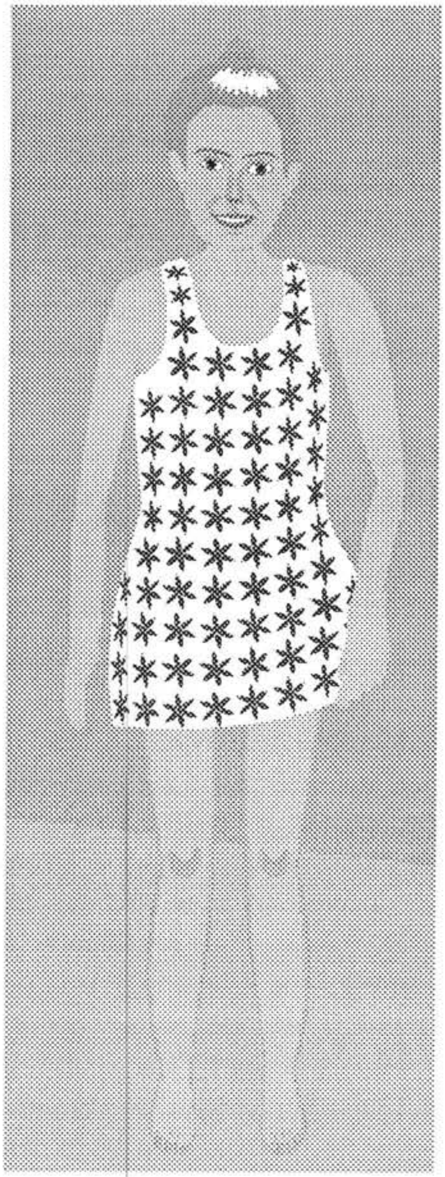


Figure 5.9—the girl's dress was drawn in *Vector* as an outline used as a mask over the pattern. The pattern is a radiated design which was then dynamically replicated. Replicas at the top and edges of the dress were rescaled to give an improved perspective

imported into *Draw* and other applications that do not support replication.

You can edit the path or style of either kind of replication. The edits are, of course, reflected in all the replicas.

It is possible to extract what is called a 'skeleton' from a replication using the Skeleton option on the Special menu. The skeleton is a set of outlines which represents in graphic form the attributes of each replica, *i.e.* its size and location. In a dynamic skeleton you can edit the individual outlines and move them around without being encumbered with the replicas themselves which might take some time to redraw. The real advantage of a skeleton is that you can use it to 'arrange' a different original. 4Mation apparently use a static skeleton to print out their floppy disc labels! When you introduce a new original to a skeleton, you have the choice of either resizing the original to fit the skeleton or resizing the skeleton to suit the original. This has many interesting possibilities where, for instance, a logo needs to be reproduced at several different sizes within a brochure or advertisement. Or the girl's dress in Figure 5.9 could be quickly redrawn with a different pattern element.

Yet another specialised type of replication which *Vector* provides is *radiation*. In radiation an object is replicated a specified number of times, each copy being superimposed on the original and rotated a specified angle about the original's centre—see Figure 5.10. The operation rapidly builds up intricate floral and other radial patterns which require remarkably little memory. As with dynamic replication, a radiated object can be converted to separate objects to allow its reproduction by *Draw* and DTP packages.

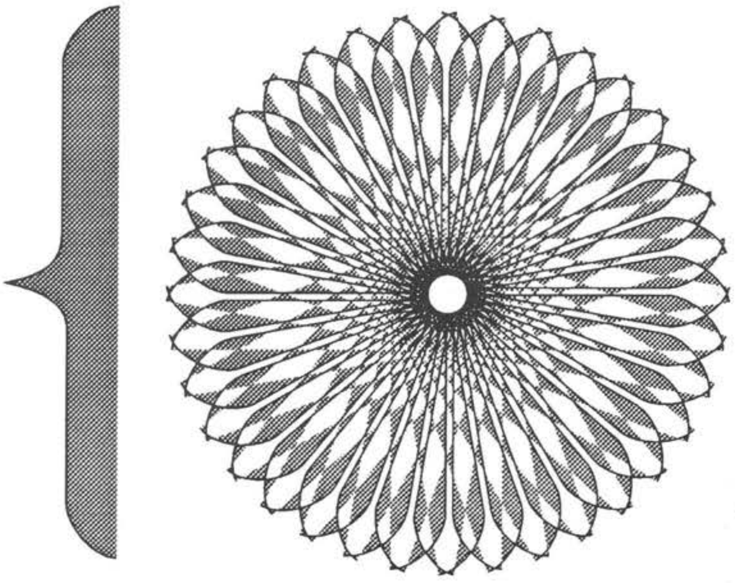
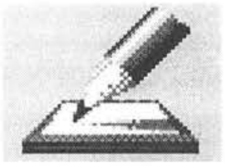


Figure 5.10—a radial pattern using the radiate facility in *Vector*. Left is a simple path object; right is a pattern made by radiating it 36 times at 10 degree angles. Finally it was converted to separate objects and path merged so that the winding rules build up a separate pattern in the fill colour

A radiated object may itself be used in a static or dynamic replication to build up a repeating floral pattern as in the example in Figure 5.9, but these take some drawing—even with an ARM3 each redrawing of the dress took about five seconds. Replication and radiation are ideal for creating repeating patterns, *e.g.* in fabric and fashion design, wallpapers and for decorated borders. The patterns can of course be placed behind masks to create a variety of interesting designs.

Vector certainly does provide the enthusiast with a powerful tool for graphic and other kinds of design, especially with RISC OS 3. There are, however, a few 'niggles'. Like *DrawPlus* it lacks the New view facility which I have found useful in *Draw*, but this omission is offset by the facility to toggle instantly between any pre-set zoom setting and actual size. For technical reasons the zoom maxima and minima are 2000% (*i.e.* 20:1) and 5% (*i.e.* 1:20) which are a little low—in *DrawPlus* I often found the most comfortable zoom setting for some intricate work to be around 40:1. There is no Undo facility, but an Undelete will restore an accidentally deleted object. Saves to floppy disc on an A5000 seem to take forever, but this is a function of the A5000's operating system; it is actually quicker to save to a RAMdisc, from the RAMdisc load the file into *Edit* and then save it out of *Edit* on to the floppy. *Vector* has many excellent features which are certain to earn it many devoted users.

ArtWorks



The largest, the most expensive and most comprehensive drawing package for the ARM machines, *ArtWorks* from Computer Concepts differs from *DrawPlus* and *Vector* in that it was not written simply as an improved or enhanced version of *Draw*. It represents a fresh start and is intended as a professional drawing and design package that will provide users of Acorn 32-bit machines with facilities that are similar to if not better than those of *Corel Draw* for IBM PC-compatibles and Aldus *Freehand* and Adobe *Illustrator* for the Apple Macintosh.

As to its capabilities, if you took most of *Vector's* facilities and added most of those offered by the adjunct software described in Chapter 4, that would give you a rough idea as to what to expect. Even so, *ArtWorks* has many facilities not available elsewhere for the ARM machines. This description is based on pre-production versions 0.820 and 0.868 in which some features had not yet reached their final production form.

Speed

Redrawing time limits productivity in most vector graphics software. For this reason Computer Concepts, who established their reputation for innovation early in the history of Acorn computers, chose to write *ArtWorks* from scratch in ARM Assembler rather than the C favoured by many ARM software writers. Consequently *ArtWorks* is very responsive—indeed on an A5000 it is claimed to redraw five times as fast as *Corel Draw* on a 486-based PC. The code for many of its tools and facilities is contained in independent modules that are loaded and identified one by one at start-up. This makes it easy to customise a package or to update a particular module.

One of the unique features of *ArtWorks* is its anti-aliased display. Judicious use of intermediate colours along oblique and curved edges makes these appear smooth, quite devoid of the 'jaggies' that often spoil computer graphics. Because this anti-aliasing, together with the blends and graded fills which the software offers, tends to slow down the redrawing, a remarkable control, named rather clumsily the 'WYSIWYG control'—WYSIWYG stands for 'What you see is what you get'—allows you to vary the quality of the display. Anti-aliasing takes effect only at the

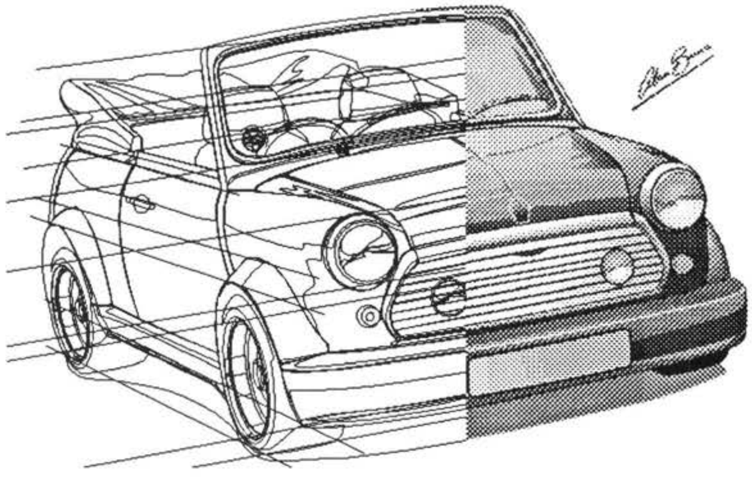


Figure 5.11—Alan Burns' widely published drawing of a Mini Cabriolet stimulated pre-launch interest in Computer Concepts' *ArtWorks*. In this composite, the left-hand side of the drawing has the WYSIWYG turned to minimum so that only outlines are reproduced, while the full anti-aliased display complete with graded colour fills is shown on the right. This illustration is from screendumps and the anti-aliasing is just visible

control's maximum setting; turning the control down causes the graded fills and then the simple colour fills to disappear; finally sprites vanish, leaving only the outlines. But at minimum setting the redraw is very fast indeed. Just above minimum is a setting where just outlines and sprites are visible—this is useful if you wish to trace around an imported sprite—a scanned image, perhaps—since you can set the required fills but can still see the original you are copying.

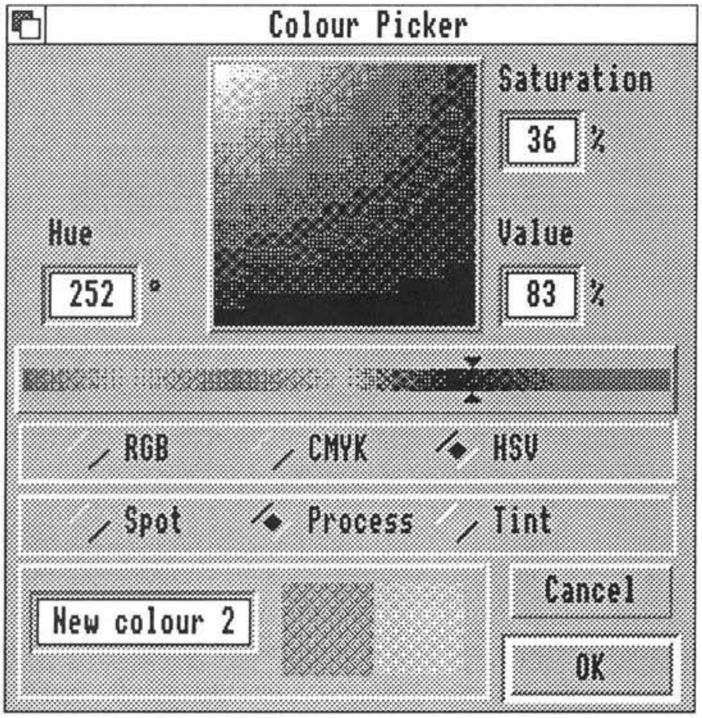
And if redraw time still bothers you, you can choose 'interruptible redraw' which designates redrawing as a background task, allowing you to get on with the task in hand.

Colour

ArtWorks incorporates its own dithering module which in conjunction with a 'primary palette' of primary colours, carefully chosen to match the output of colour printers, allows a remarkably accurate full-spectrum display in 16-colour modes, while millions of colours are available in 256-colour modes. The dithering is deliberately coarser than RISC OS 3's (4 × 4 pixel matrix compared with 2 × 2 pixel matrix) used in *Draw* and *Vector* because this allows smoother and more accurate graduated shades; it also makes many more shades available. While the colour is undeniably excellent, the dithering itself is more obtrusive and fine detail is less sharp. There is no facility to turn the dithering off.

Three colour models assist in the creation and choosing of shades. The familiar RGB system offers three sliders each calibrated 0 to 100, so that you might be excused for thinking that not all 16,777,216 colours are accessible. But in fact you can type your entries to two decimal places; indeed *ArtWorks* performs most of its internal colour calculations to 64-bit accuracy and some to 128-bit accuracy.

We met the HSV system in Chapter 4 in connection with *Chameleon 2*. *ArtWorks* offers an HSV colour cube (Figure 5.12). The hue (colour) slider calibrated 0 to 359 degrees determines which slice of the cube is opened, while the saturation and value are chosen as percentages (from 0 to 100%) within the opened slice of the cube; again fractional values are supported. The number of colours available seems vast, but remember that some are duplicated since every cube slice inevitably contains both black (value = 0%) and white (value = 100% and saturation = 0%). The HSV cube is extremely easy to use; at first attempt I matched a shade exactly within a few seconds.



Hue 252 °

Saturation 36 %

Value 83 %

[✓ HSV]

[✓ Process]

New colour 2

Figure 5.12—the HSV colour cube in *ArtWorks* provides a simple and intuitive method of editing colours, offering a choice of millions of shades

The third colour model is the CMYK system (cyan/magenta/yellow/ key) used in the printing industry. The sliders, one for each of the four 'inks', are calibrated 0 to 100% again giving access to millions of colours, but there is huge redundancy since the key (black) tends to swamp the other colours. It is important to remember that the CMYK system emulates the way in which colours are built up in four-colour printing and that in inks, the higher the percentage the *darker* the colour. Thus with cyan, magenta and yellow all at 100% (irrespective of the key setting) the resultant colour is black! This stands in utter contrast with the RGB system where having all three primaries at 100% gives white!

The CMYK facility will undoubtedly be useful to those using *ArtWorks* to prepare material for colour printing. Moreover, it is supplemented by a facility to save and print four-colour separations and also spot colour separations. (Spot colour is the use of a second colour and possibly a third colour in printing. There is not normally any attempt to mix spot colours, but tints of spot colours can be used to provide additional colour effects.)

Much colour filling in *ArtWorks* uses graduated shading facilities provided by the linear fill and radial fill tools and the blend tool, all described later. These, together with the dithering described earlier, provide gentle and natural gradations of colour, greatly enhancing the realism of drawings (as Alan Burns' work well illustrates).

Drawing with ArtWorks

Whereas both *DrawPlus* and *Vector* can be mastered quickly by anyone familiar with *Draw*, *ArtWorks* uses some terminology and conventions that are different. Consequently its use demands a little more thought and, initially, more frequent recourse to the user manual than those other applications.

Clicking Select on the icon produces an empty window and clicking in this causes a separate toolkit and a status display (Figure 5.13) to appear. These, as in *DrawPlus*, can be moved to any convenient location. There are 18 tools in the toolkit. If you don't like their arrangement, you can rearrange them by dragging with Adjust.

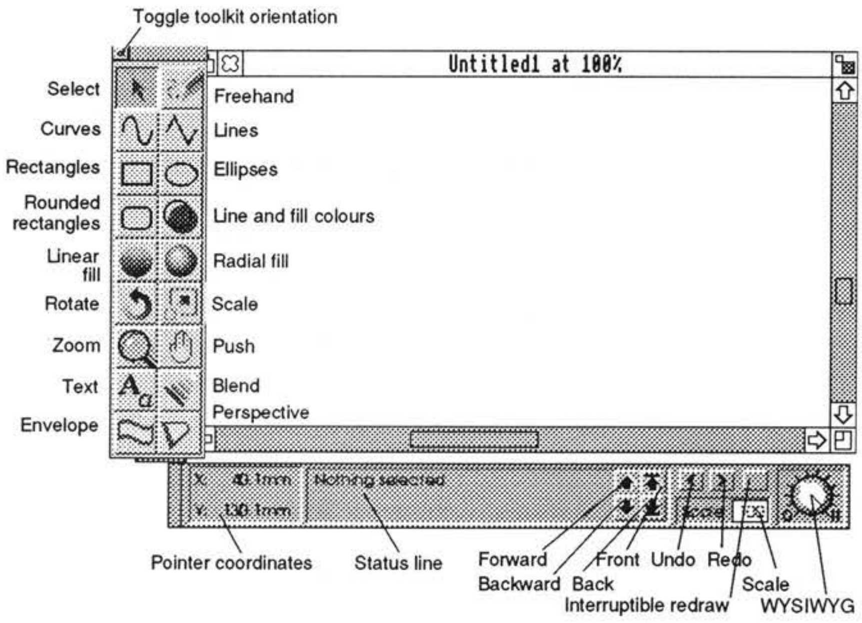


Figure 5.13—main features of a drawing screen in *ArtWorks*. The tools in the toolkit can be rearranged into any order and the toolkit can be toggled into a horizontal orientation and moved to any convenient location

Many of the tools are familiar from other packages, but some are new and some are conspicuous by their absence. Unique to *ArtWorks* is the freehand drawing tool (Figure 5.14) which allows you to drag the pointer around leaving a trail of the current line width and colour. When you finish your freehand line, ArtWorks converts it to a series of conventional

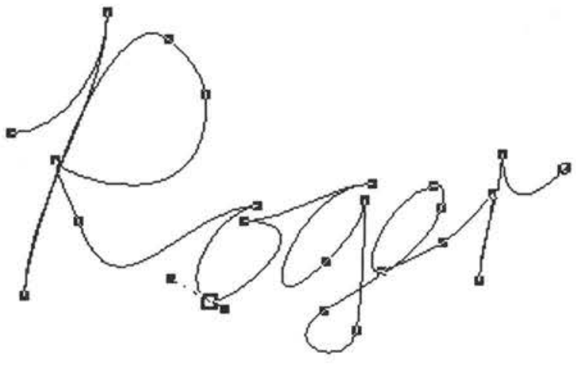


Figure 5.14—You can even sign your name using the freehand facility in *ArtWorks*. This example has been selected to reveal the endpoints of the segments. Note that the control points are only displayed on the currently selected point

Bezier curves. You can even adjust the accuracy with which *ArtWorks* performs this conversion.

There is no explicit autoclose facility. To close a path you drag its endpoint on to its start point. This is one of several respects in which *ArtWorks* follows conventions similar to those of *FontEd*.

Closed paths are called 'shapes' and only shapes can have a fill colour. You *can* assign a fill colour to an open path, but no filling will take place unless you close it, changing it to a shape. If you insist that an open path should appear to hold a fill colour, you must use a separate shape with transparent lines to hold the colour and group this with the open path; this is how *ArtWorks* interprets an open path with a fill colour imported from *Draw*.

An existing lines or shape can be edited by selecting it and then clicking on a line or curve tool in the toolkit. A new set of icons appears in the status display; these are shown in Figure

5.15. They are equivalent to the Edit menu in most other drawing packages. Interestingly, the Break line at point tool will break a line into two daughter lines which of course are independent objects.

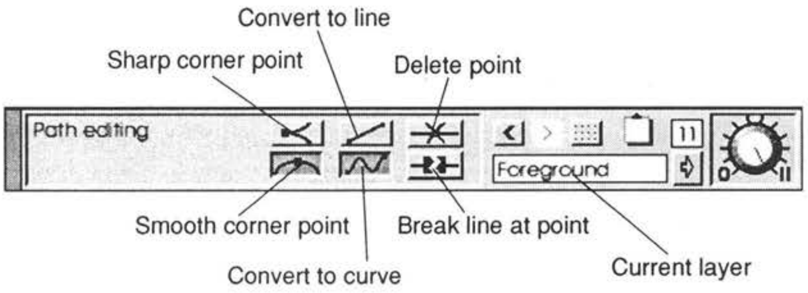


Figure 5.15—the line/shape editing icons which appear in the status display when editing

Because of the difficulties that thin lines cause in some processes, such as professional type-setting—these difficulties were described in Chapter 2—*ArtWorks* discourages their use. The default line width is 0.25 pt and this width is applied to thin lines in drawings imported from other packages. If you insist on having a thin line, you can set the line width in *ArtWorks* to 0. This is interpreted as a thin line when imported into *Draw*.

When editing curves, Bezier control points are only visible and adjustable if the endpoint to which they are attached is selected, another feature in common with *FontEd*. I find this system far preferable to the infuriating dense tangle of control points that appears when editing complex curved objects in other drawing packages and which has often compelled me to drag five control points out of the way to reach the one that needs attention. When dragging a control point, usefully its original position remains visible on the screen, making it easy to put the point back where it was if you cannot improve it (although a click on the Undo icon or press of F8 will also undo your last action).

Multiple point selection

Provided a segment drawing tool is selected, you can select more than one point in an object (by clicking Select on the first and Adjust on subsequent points following the normal RISC OS convention for multiple selections). If you drag one point with Select, any other selected segments also move an equal distance in the same direction. This is useful as it allows you to preserve the shape of part of an object while distorting another part of it. You cannot select points in more than one object.

Conspicuous by their absence are moves. As in *FontEd* moves are implicit. Moves between subpaths are possible by creating the subpaths as separate shapes, selecting them and applying a Join Shapes menu option which is similar to *Vector's* merge paths facility. When joined, the subpaths are all one object, the filling of overlapping areas being controlled by the familiar choice of winding rule.

A Make Shapes menu option will convert text to 'shapes', *i.e.* vector graphics, so that it can be combined with other shapes to create stencil-text and other special effects. Oddly, in *ArtWorks* there is rather less need for text-to-path conversion. *ArtWorks* incorporates its own font manager which allows all text objects to be given the same styles—separate line colours, fill colours, line widths and line patterns—as path objects, while remaining text objects with text editable and fonts changeable. Text can even be given graded fills!

Standard shapes such as ellipses and rectangles are supplemented by rounded rectangles. These are stored as tokenised objects rather than as editable vector graphics, but can be converted using the Make Shapes option. Ellipses and rectangles can be forced into squares and circles by a movement constraint facility, provided not by a tool but by the simple expedient of holding the Ctrl key down during the constrained operation. These shapes can be edited using the Make Shapes option.

Many style and transformation options that are controlled by menus in other packages are controlled by tools in *ArtWorks*. Thus there is a tool for line and fill colours and it applies equally to text and path objects. It leads via the status display (called the 'info bar') to the colour selectors described earlier. Graded fills—linear and radial—are provided by two further tools. Their operation is very similar to their counterparts in *Chameleon 2* described in Chapter 4. The rotation and scale tools control scaling or rotation through dialogue boxes—the usual rough rotation and scaling by dragging handles is also provided. Usefully the rotate facility allows you to place an axis of rotation within the object or outside it.

Two tools affect the nature of the display. One is the zoom facility which has a sophisticated range of options, including the ability to drag a rectangle over an area of interest and have the contained area enlarged to fill the screen. Zoom ranges from 1% (1:100) to 999% (9.99:1). Although the latter is not so high as that available in some other packages, the need for high magnifications is offset by the elimination of control points except on the selected point. Unlike *DrawPlus* and *Vector*, *ArtWorks* allows New Views into the same drawing, with different zoom settings on each.

The push tool, familiar to users of *Impression*, allows you simply to drag the drawing around until the required part of it is in the window—all very easy and intuitive.

The envelope tool provides the same kind of distortion facility as the moulds in *DrawBender* and *Typestudio*. The perspective tool is a variation on the same theme. It distorts the object so that it seems to disappear into the distance; with text this means that not only is the height reduced but also it is foreshortened (Figure 5.16); the effect is stunningly realistic. It even indicates a vanishing point, with the promise that later versions will allow more than one vanishing point.

The Edit menu in *ArtWorks* uses the clipboard principle to allow objects to be cut or copied and pasted elsewhere; these operations will be familiar to users of DTP software. It also offers a duplicate facility which is similar to the copy facility in *Draw* and a clone facility which I expected to produce replications, but the resultant copies are fully independent of the parent object.

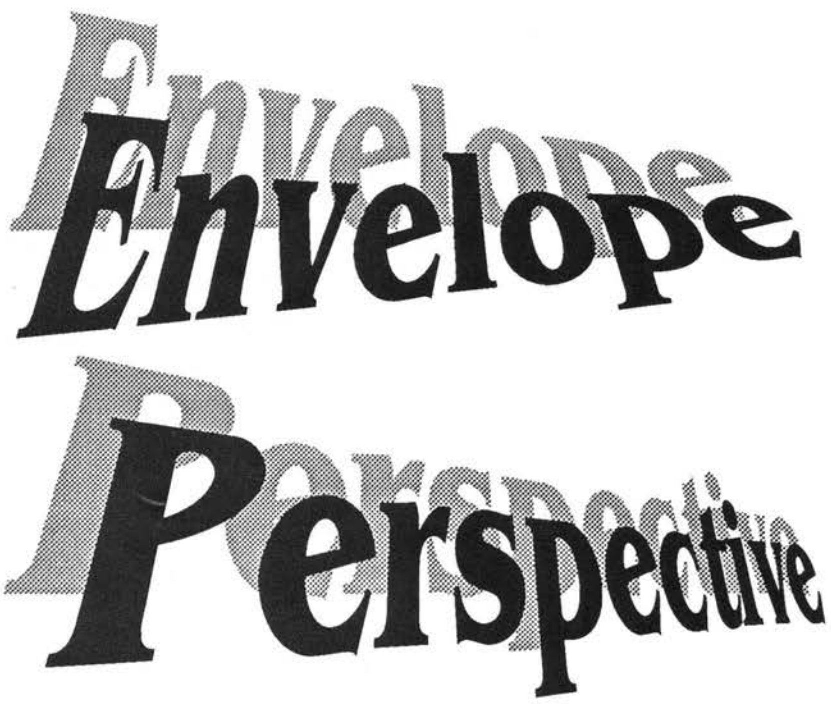


Figure 5.16—the Envelope and Perspective tools in *ArtWorks* offer similar distortion effects, but Perspective realistically condenses detail as it recedes into the distance

Order control, as in *DrawPlus*, is limited to back, front, backward and forward. But there is a difference: the backward and forward options are 'intelligent' in that they do not necessarily move the selection just one place up or down the stack, but as many places as are necessary to make a difference to the drawing. A forward operation, for instance, will move the object as many places as are required to bring the object in front of whatever object is hiding (part of) it.

A sophisticated interpolation facility is provided in the blend tool. There are several fundamental differences between this and the interpolate/grade facilities in some other packages.

Firstly, the parent objects need not have the same number of points (although better results are obtained if they do). Secondly the intermediate lines or shapes created are not discreet objects at all; the two parent objects and their 'offspring' form a 'blend object' in which only the definitions of the parents and the number of offspring are stored. The offspring are created 'on the fly' during redrawing. This not only results in worthwhile memory savings without imposing too great a burden on redrawing time, but also means that subsequent edits to either parent are immediately reflected in the offspring. Also, parent objects can be shared between blends. An expand facility will convert blend objects to discreet objects if the drawing must be compatible with *Draw* and DTP packages.

Layers are supported; the layers dialogue box is simpler than in *Vector* and *DrawPlus* although all the regular facilities are provided. Objects are only editable in the current layer although a multi-layer option allows you to access objects on all layers. To transfer objects from one layer to another you use the clipboard-based cut-and-paste operation. Objects can be locked independently of the layers system.

File interchangeability

ArtWorks will import files created by *Corel Draw*, Aldus *Freehand* and Adobe *Illustrator* and other packages provided they are saved in Adobe EPS (Encapsulated PostScript) format. It will export files in *Corel Draw* and two *Illustrator* formats (in fact *Illustrator* format is accepted by all three) as well as EPS. And of course it has its own file format and will also load and save in standard *Draw* format.

So, when running on a RISC OS 3 machine which can read and write MS DOS discs, *ArtWorks* is invaluable to those like myself who use Acorn computers in graphical work for clients who use PCs and who frequently need to swap graphics between several different formats.

And maybe this is where *ArtWorks* will find its niche. Acorn computers are widely used in schools and homes where the comparatively high price of *ArtWorks* may deter users. So far the Acorn 32-bit machines have not made significant impact in professional graphic design houses, but that is hardly surprising because, until recently, there was no professional-quality software. Since the purchase price of an A5000 and *ArtWorks* is less than half that of a comparable Apple Macintosh with drawing software and slightly less than that of a 486-based PC, Microsoft *Windows* and *Corel Draw* while providing a better performance and excellent file interchangeability, *ArtWorks* could open new and lucrative markets for Acorn computers.

6 Three Dimensions

As we have seen in the previous chapters, the images in vector graphics are stored as a sequence of co-ordinate data which the software handles in a stack. In the software considered so far, these co-ordinate data have always related to two dimensions, the x-axis (horizontal) and the y-axis (vertical). Two dimensions are sufficient for most purposes because ultimately the graphics will be displayed on a two-dimensional monitor screen or printed on a two-dimensional sheet of paper. It is true that, as the software redraws its way through the stack, objects at the bottom of the stack (back of the drawing) may become obscured by those higher up (further forward), giving an illusion of depth. But *illusion* is the operative word, for ultimately all the objects lie in one plane—that of the screen or printout.

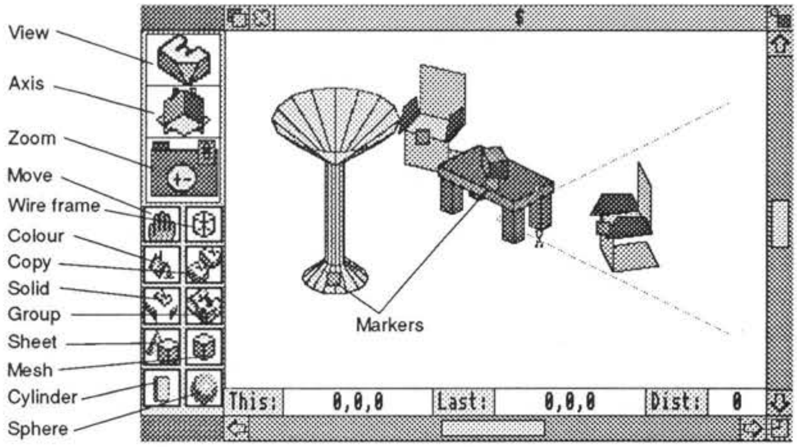
Since computers are by definition data processing machines, it is not a wildly extravagant extension of the principles of vector graphics to modify the format of the co-ordinate data so that they relate to not two but *three* dimensions. Obviously the new third axis will be perpendicular to the plane of the screen or printout.

Of course, some compromise is inevitable in any attempt to reproduce three-dimensional graphics on a two-dimensional monitor screen. But in one sense this does not really matter, for we accommodate ourselves to the same compromise whenever we watch television, video or cinema film. Of course, those three media are all *animated*; the pictures they present are, for most of the time, moving. Movement is a particularly powerful tool in the representation of three-dimensional graphics. If a man walks in front of a chair and temporarily conceals it, we know that he is in front of it, even if we could not have gleaned that from the perspective in the scene. And even if the objects depicted are static—as in a still life—by changing the viewpoint, *e.g.* by zooming the camera in on some point of interest, the change in perspective will reinforce our perception of the spatial relationships of the objects depicted, fostering the illusion that the monitor screen is a window into a three-dimensional world and not a two-dimensional one. For this reason, three-dimensional graphics and animation often go hand-in-hand.

Euclid

Several three-dimensional graphics packages have been written for the ARM machines, but probably the best known and certainly the oldest is *Euclid* from Ace Computing. *Euclid* claims to be a three-dimensional equivalent of *Draw*. In fact in some senses *Euclid* is much more than that, for it lies at the hub of a network of packages which offer a comprehensive range of linked facilities and is supported by an active user group called *Elements*. For instance, whenever you load *Euclid* you may find that *Mogul* has automatically been loaded as well. *Mogul* creates animations from *Euclid* and animation, as mentioned, is a potent force in the quest for three-dimensional realism.

A *Euclid* screen closely resembles any two-dimensional vector graphics screen except that the pointer co-ordinate display reveals three axes (Figure 6.1). Euclid is object-based; the objects



```
This:   0,0,0   Last:   0,0,0   Dist:  0
```

Figure 6.1—a typical editing screen in *Euclid*. The bottom eight tools in the toolbar vary depending on the operation in progress. Note the markers (rectangles)

themselves are usually three-dimensional and their relative positions are defined in three dimensions. It is perhaps easiest to think of the cluster of three-dimensional objects as being drawn at the centre of a sphere. The image in the editing window is the view through a mobile window which can be moved to any point on the sphere's surface using the 'move' tool identified by the hand icon. This allows you to view the objects from any side or from any angle above ground level or even beneath it (Figure 6.2), the ground being conveniently transparent! The zoom tool controls the size of the sphere; it can be contracted so that the window zooms in for close-up work or it can be expanded so that more of the scene can be viewed in the window. However, the view in the editing window is always isometric and therefore devoid of perspective. Views with perspective can be obtained using the camera facility described later.

All objects can be selected; in the editing window each object carries a marker, normally a blue rectangle. Clicking Select on a marker

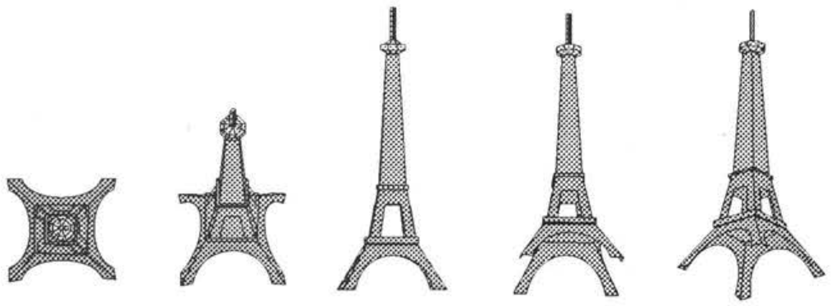


Figure 6.2—five views of Tony Cheal's *Euclid* drawing of the Eiffel Tower ably demonstrate the software's potential for generating different views of three-dimensional objects

changes it to red showing the object to be selected; multiple selections are possible by clicking Adjust on other objects' markers. Selected objects can be moved, rotated or scaled using drag operations which are reassuringly conventional. To eliminate the ambiguity that would otherwise result from attempting to define moves in three-dimensional space using a two-dimensional screen, only two of the three axes are available simultaneously; the axis tool allows you to choose the pair of axes with which you wish to work. The x- and y-axes are in the plane of the floor or ground of the scene; the z-axis is perpendicular to the floor plane and so relates to altitude above the floor, or depth beneath it. *Euclid* knows none of those conflicts of time and space which in the real world place restraints on our manipulation of solid objects; so you can make two objects share the same space—with bizarre or comic results (Figure 6.3).

Normally objects are viewed with their styles (mainly colour) in place. The wire-frame tool, however, eliminates colours so that objects are viewed as 'skeletons' of black lines. This can be useful as it allows you to see through solid objects to what lies behind.

Object types

Euclid recognises three types of objects: *solids*, which consist of a collection of flat, coloured surfaces; *meshes*, which consist of arrays of four-sided shapes; and *sheets* which are the product of two paths. It also recognises groups which may be made up of collections of any combination of object types including other groups. Interestingly, an object may belong to

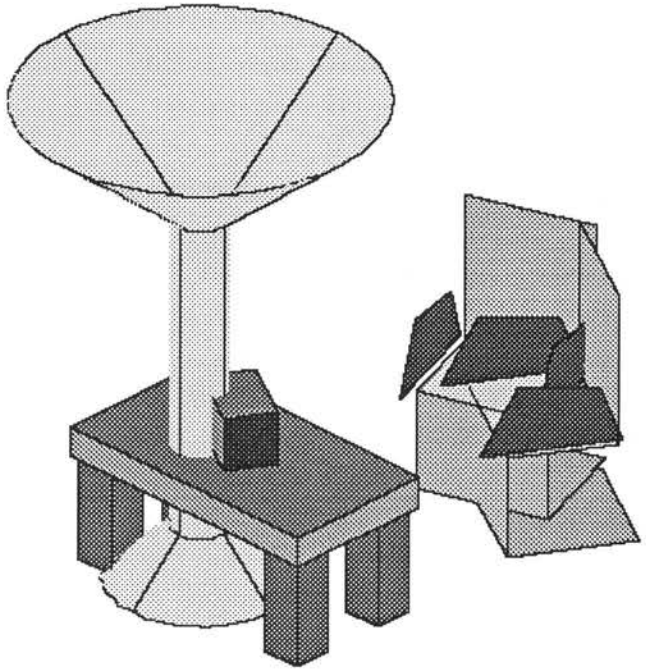


Figure 6.3—a few seconds play with the example scene provided with *Euclid* and the 'uplighter' (lampstand) has migrated into the table and the two chairs have become intertwined. Three-dimensional graphics are delightfully free from the spatial restrictions of the real world... Incidentally, by turning the Flatness up to 16 the uplighter appears genuinely round instead of polygonal

more than one group and consequently may appear in more than one place simultaneously and so is a clone or replication, edits to any instance being reflected in all instances.

Scenes in *Euclid* are built up using a hierarchical structure rather like that of the disc filing system. The whole scene is often named $ like a disc's root directory. This may contain a mixture of objects and groups, like a root directory containing both files and subdirectories. The groups may contain objects and other groups, and so on. Thus the table in the example scene is a group consisting of the tabletop (a mesh) and the four legs (four instances of another mesh) while the box is a single solid.

Tools are provided which allow the quick creation of some frequently used shapes such as cuboids, cylinders and spheres. The building up of solids, meshes and sheets demands the same drawing skills as *Draw*, although *Euclid* conveniently compensates for minor inaccuracies by assuming that a click within a certain radius of an existing point was intended to be coincident with that point. Bezier curves are provided, although their rendering and control are a little crude and, since the manipulation of curved solids is very time consuming, many normally curved surfaces are drawn as a succession of plane surfaces, the uplighter being an example. Colours are chosen using a standard 16-colour palette supplemented by a colour wheel which provides access to 256 colours in 256-colour modes; colours are stored internally by their RGB values and colours not in the current palette are simulated using dither patterns. Where an object is made up of separate components each may be given its own colour.

Euclid also provides 'system objects' which behave in special ways. A camera can be moved and rotated just like any other object, but clicking select on the camera tool changes the view in the picture window to the image 'seen' by the camera—this time with full perspective and with the markers invisible. A choice of lenses allows for special effects. You may have several cameras in the scene, offering a choice of views. Each camera is invisible to the others.

Lights allow the effects of shade to add interest to the scene. The 'front lit' option assumes the light is on the viewer's head (recalling a miner's helmet) so that as the move tool is used to change the view, the patterns of light and shade change in a realistic manner. User-defined lighting effects allow the experimenter almost limitless freedom to create special effects.

File exchange

Euclid allows you to save any scene as a sprite; it also allows the saving of *Draw* files. Note, however, that complex scenes may not leave the *Euclid* editor enough workspace to create a *Draw* file; also many *Euclid* scenes do not produce satisfactory *Draw* files. For example, it is hardly surprising that the scene in Figure 6.3 produced a horribly confused *Draw* file, because of ambiguity as to whether the uplighter was in front of the table or *vice versa*! An example of slightly less confusion in such a *Draw* file is evident in Figure 13.1.

Draw files can also be dragged on to the *Euclid* icon. Only path objects in them will be recognised and, subject to certain limitations, can be used as objects within *Euclid.* A dialogue box will ask for the depth of separation between successive objects. Subject to memory availability, a sprite file can be loaded as a background scene.

Incidentally *Euclid's* !RunImage file is written in BASIC and so, if you are competent in BASIC programming, you may adapt it to suit your own purposes. The possibilities are endless...

Mogul

Mogul is loaded automatically with *Euclid*, provided that its application directory has been 'seen' by the computer. It allows for animated sequences to be built up from scenes prepared in *Euclid* easily and without programming skills.

The technique is very simple. You arrange the objects (including cameras) as you wish them to be at certain critical stages in the film—these are called 'key frames'. A Show motion option in the Miscellaneous submenu opens a 'motion' window which consists essentially of a chart or program. Down the left-hand side are listed the objects which will change or move within the film. Across the top are the frame numbers, the limit being 250 (giving 10 seconds of action at normal speed). In the key frames the three-dimensional co-ordinates of the mobile objects are stored. There is no need to work out the intermediate positions of the objects: *Mogul* does the 'in-betweening' itself. A Preview option allows you to watch the action, but almost certainly in slow motion, since the positions of mobile objects must be calculated before each frame is set up.

There is a separate option to save the film. This sets up each frame as before but then saves that frame as a *sprite*. Each sprite is saved in a compressed format and an additional compression system (delta-compression) offers further memory savings in sprite-based animations by eliminating those parts of successive frames which remain unchanged. The need for such memory saving techniques is obvious when you consider that, without compression, if you wished to fit a sequence of 250 256-colour sprites on an 800-Kbyte floppy, you would be limited to a sprite size of just 3200 pixels, *e.g.* 75 × 43 pixels.

The resulting sprite-based film will be in Ace film format and can be played at its intended speed using a public-domain application called *Projector* which gives a useful range of facilities including freeze-frame, reverse and slow motion. We shall meet the Ace film format again later. Since the projector application is public domain and the film is your own creation, you may freely distribute copies of both without infringing copyrights.

3-D Construction Kit

'Build your own virtual reality' is how Domark have advertised their package, *3-D Construction Kit*. The package, however, uses the regular monitor display, not a personal stereoscopic display unit as usually associated with virtual reality. Like *Euclid* it is an object-based three-dimensional graphics package which includes an animation system. It also incorporates a BASIC-like programming language which allows you to write your own adventure games in which you interact with the three-dimensional world you have created. Not only can you explore the world, but you can collect objects, shoot at objects and score points. A thoroughly entertaining demonstration game is included. The games you create are saved in a stand-alone format which you may freely distribute to third parties.

That the software was originally developed for another machine and has been converted for the Acorn 32-bit machines is immediately apparent. It uses a pull-down menu system recalling much PC practice and fails to observe the niceties of the RISC OS desktop. It will not multi-task with other RISC OS applications and files are not interchangeable with them. In fact you cannot even use a screen grabbing module to save a screen since its presence causes the computer to crash—you are advised to switch off the computer before running the software to ensure that all other applications are cleared from the system first. For this reason it is sadly not possible to show any screens from the package in this book. Moreover, running the software reconfigures the system—the font cache, for instance, is reset to zero (to 32 Kbytes on an A5000) as you will quickly—or do I mean slowly?—discover when next you try to use a DTP package.

A comprehensive user manual and a tutorial video are very helpful—the video makes one appreciate the sophistication of the Acorn machines.

All the action takes place in the 'view window' which occupies the top half of the mode 15 screen. At first this wide and shallow window may seem somewhat restrictive, but you quickly get used to it. The lower half of the screen is used for messages, icons and menus.

Objects are created using the pull-down menu structure. A Create menu allows you to create two types of three-dimensional object—

pyramids and cubes—and five types of two-dimensional object—triangles, rectangles, quadrilaterals, pentagons and hexagons—besides (straight) lines. There are also special objects such as sensors which detect your presence. Groups are also supported. There are no true curves in the application, presumably because they would make the redrawing unacceptably slow. Remember that the program must redraw the scene interactively and interactive vector graphics are very demanding of computing power. Nearly all virtual reality systems are similarly devoid of curves.

The created object is an arbitrary size and colour and suspended rather menacingly above the ground. You can then colour it (each face of a cuboid can be assigned a different colour) and you can move it, turn it, stretch it and squash it. So what was created as a cuboid can become tall and thin, like the leg of a table, or short, deep and wide like a table top. With a little imagination most shapes can be constructed from the two three-dimensional primitives, pyramids and cubes. The two-dimensional shapes can be used to ornament the three-dimensional ones, *e.g.* to add windows to houses. Each new object is given a default name (such as 'cuboid002') which you can change to something more informative if you wish.

Besides objects, you can have different areas or zones with doorways or passageways between them. In fact, each area is itself a cuboid (cuboid001) which explains why the first cuboid object you create is always 002 and not 001. Facilities are provided to edit areas and to create and edit animated sequences within areas.

The Freescape Command Language allows you to program certain functions to occur under specific circumstances. For example, your score is incremented when you collect a certain object, a door opens when you shoot at it or the game ends when you collide with a certain object. The language will pose no difficulties to anyone familiar with BASIC. A built-in editor allows you to construct the necessary routines. Even sound effects using sampled sound are also supported. Six sounds are provided and up to 26 more samples, in a suitable format, can be added.

7 Vector Graphics and BASIC

Because a *Draw* file consists of blocks of data in a standard format, any application may use suitable source data as the basis of *Draw* graphics. The application itself may have no pretensions to being a graphics application. It might be, for example, a financial program which creates a graph of statistical trends in *Draw* format. Or a process control program may use vector graphics as the basis of a mimic diagram. The graphics not only appear on screen but can also be saved as a *Draw* file for printing or incorporation into DTP documents.

Applications which create *Draw* files in this way need not be highly sophisticated. ARM code or C are not necessary; BASIC is quite sufficient as

DrawAid from Carvic Manufacturing demonstrates. And BASIC applications can also display *Draw* files imported from other sources. This chapter includes a short listing of routines that will display the path objects in a *Draw* file.

DrawAid—Vector Graphics in BASIC

DrawAid from Carvic Manufacturing allows the user's own BASIC applications to create *Draw* files. This facility is especially useful when creating certain types of drawing which are extremely repetitive or tedious or require accurate calculation. Examples include drawing repeating patterns (Figure 7.1) or the teeth on a gear wheel, plotting the points on a graph and creating a pie chart with many segments. This description is based on version 1.05.

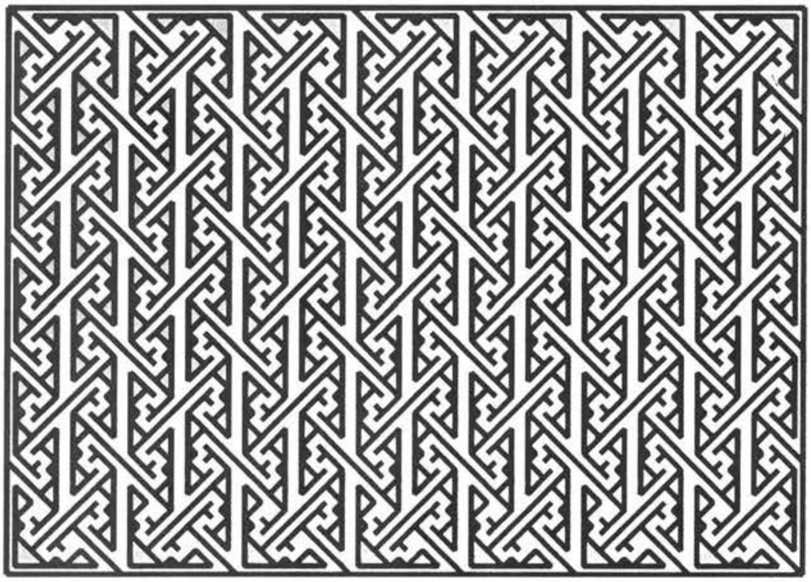


Figure 7.1—Creating this repeating pattern would be very tedious in *Draw* but the *Draw* file (32 Kbytes long) was generated by a BASIC program only 3957 bytes long (excluding *DrawAid's* comprehensive library of procedures)

Essentially *DrawAid* provides a library of BASIC procedures which not only draw the graphics on the screen but also create the necessary data structure which ultimately can be saved as a *Draw* file. A front end is provided which makes access to the RISC OS BASIC Editor more user friendly (RISC OS 3 users may find it easier to use the BASIC handling facilities in *Edit*) and a blank BASIC 'shell' into which the user's own code may be inserted. This program runs outside but from the desktop environment and so can be used with confidence by programmers familiar with BASIC even if they do not understand the niceties of WIMP programming. Provided on the disc are numerous examples which demonstrate the system in action.

Also provided is a 'vector font' (Figure 7.2). An

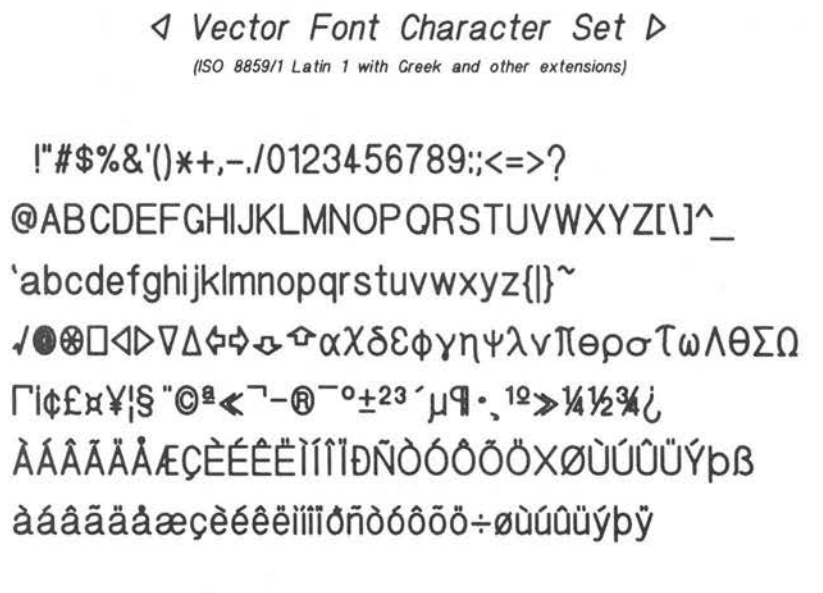

◁ Vector Font Character Set ▷
(ISO 8859/1 Latin 1 with Greek and other extensions)
!"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_
`abcdefghijklmnopqrstuvwxyz{|}~
αχδεφγηψλνπθρστωΛΘΣΩ
Γ¡¢£¤¥¦§¨©ª«¬-®¯°±²³´µ¶·¸¹º»¼½¾¿
ÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞß
àáâãäåæçèéêëìíîïðñòóôõö÷øùúûüýþÿ

Figure 7.2—the Vector Font supplied in *DrawAid*. Characters are path objects and so can be scaled and rotated. Unlike outline fonts strokes are single paths having finite but adjustable width

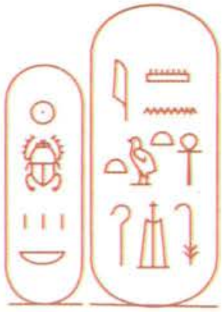
attractive alternative to the system font, this consists entirely of path objects, but unlike characters in outline fonts each stroke consists of a single path having a finite (but adjustable) width. Consequently text displayed in the vector font can be scaled and rotated like any other path object. Procedures are provided which display strings of text in the vector font in various sizes and orientations. It is ideal for use in labelling diagrams, especially when the labelling needs to be oriented out of the horizontal, *e.g.* the vertical axes of graphs. The vector font is also miserly on memory compared with text converted from outline fonts.

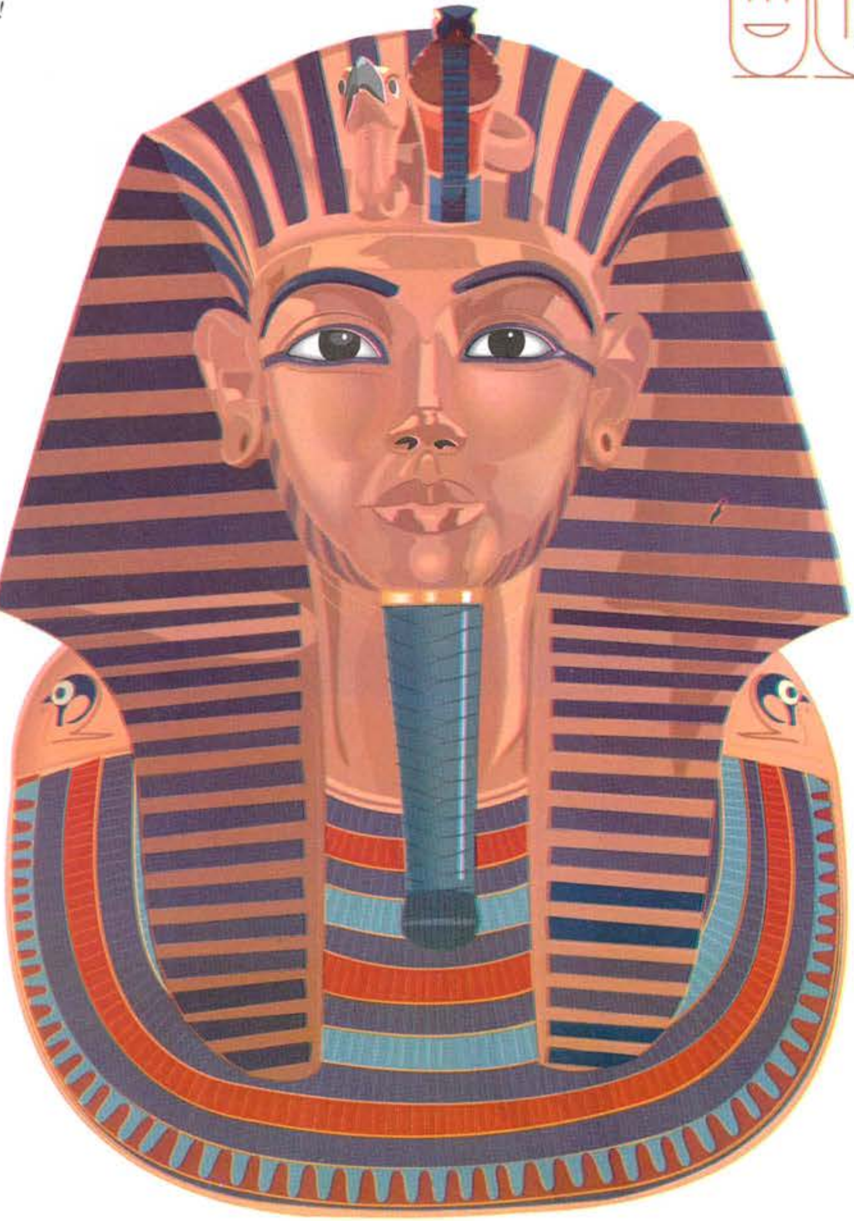
The procedures provided in the *DrawAid* library fall into four categories relating to standard objects, path objects, text objects and groups. By default all dimensions are in mm, but the user may select from a range of metric or Imperial units and the screen may be scaled to suit.

Standard objects are ready-made complete path objects. The choice includes line (single segment), Bezier curve (single segment), arrow (forwards), arrow (backwards), triangle, rectangle, polygon, arc, sector (*i.e.* a shape like a slice of a circular pie), segment (an arc and its chord), circle and ellipse. Each procedure takes a number of arguments which determine the size, location, line thickness and colours of the object. Usefully the desktop colours can be entered by names such as grey3 and light_green which saves the user from having to look them up.

The path object facilities allow any path object to be built up segment by segment just as in *Draw* itself—indeed in the same sequence that you

Plate 1 — The mask of Tutunkhamun as drawn by Hans Rijnen using **Vector**. *This drawing and Plate 3 demonstrate that vector graphics need not lack detail!*





Graphic Plates

*Plate 2a, b — You don't need expensive software or a university degree to produce interesting graphics. This is the work of 13 - year old Rosemary Harris using **Paint** and **Draw**. Henry VIII displays some subtle graded fills (in his hat). The wild flowers are a "montage" of three Draw files. Rosemary has sold many discs of clip art.*

1937 Jaguar SS100 - 2.5 Litre

Produced by AB Designs - February 1992

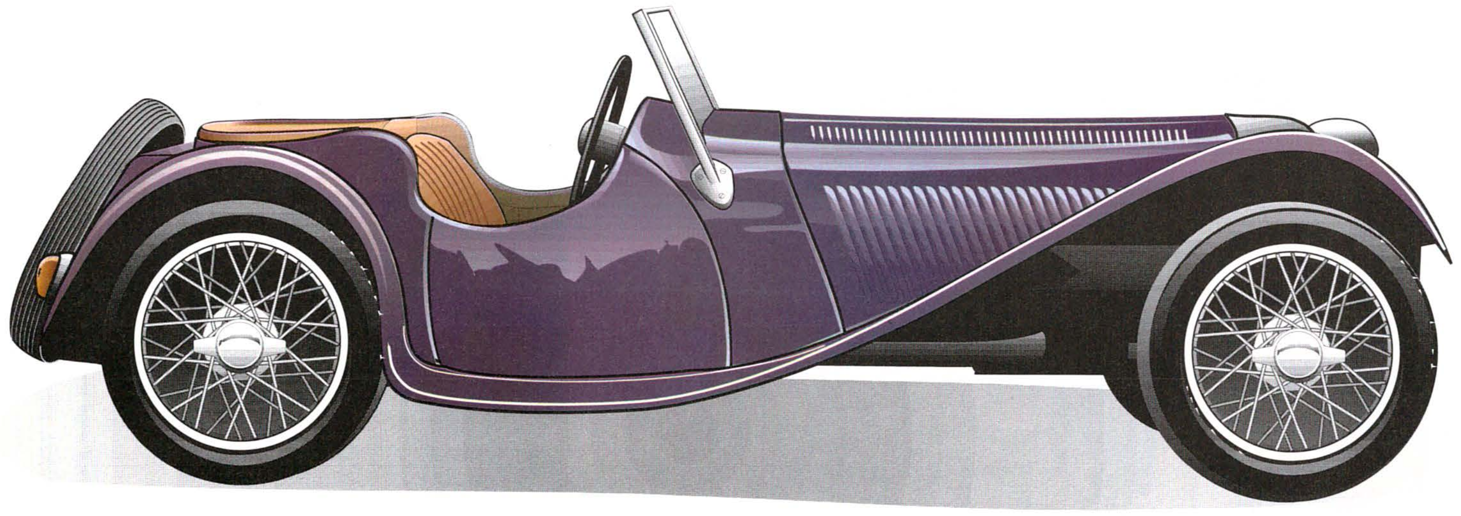


Plate 3 — *Alan Burns' drawing of a Jaguar SS100 demonstrates the capabilities of* ***ArtWorks.***

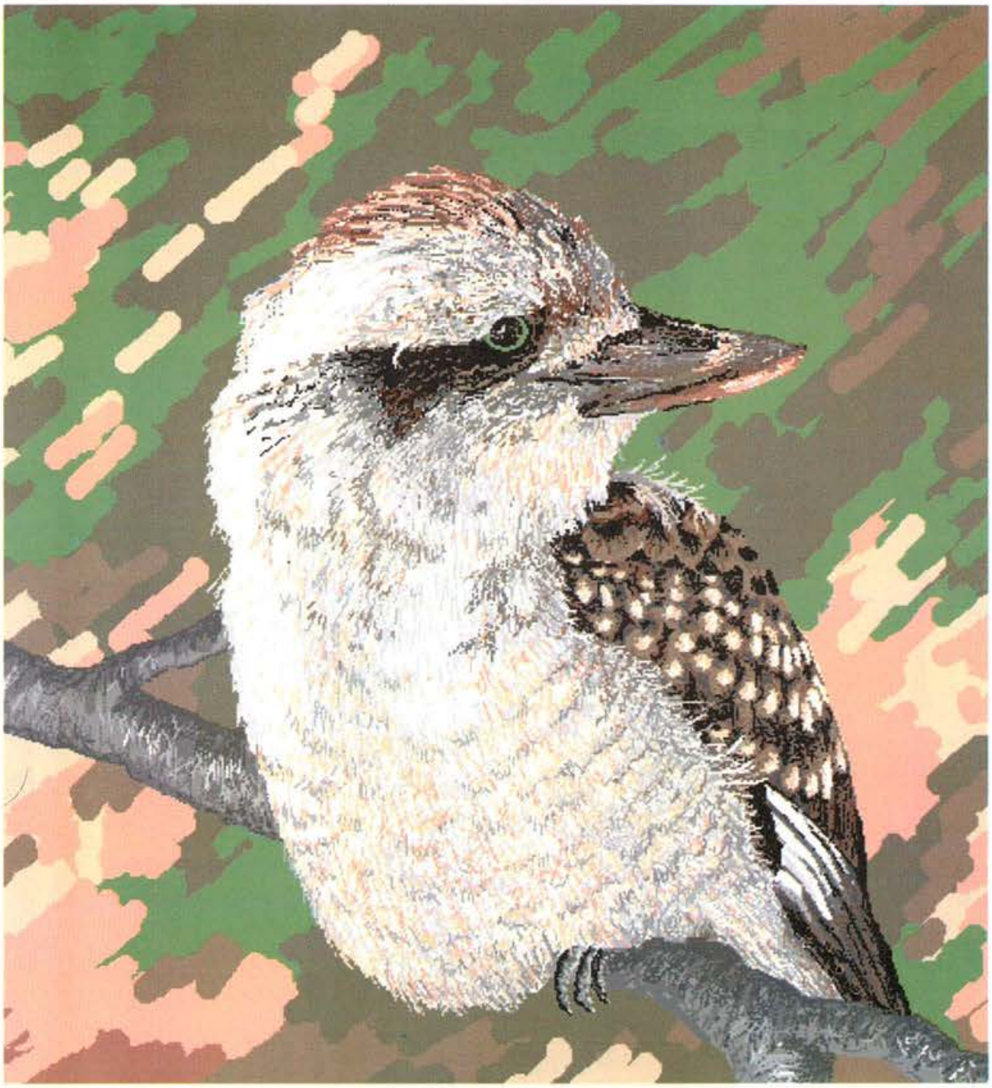


Plate 4 - A joint effort by husband and wife team - Raymond and Kim Keefe. Raymond wrote the software, ***Art20****. Kim, an artist, used it to produce this picture of a kookaburra, based on a photograph Raymond had taken. Some 15 hours of work were involved. It's hard to believe that this is mode 20, just 16 colours.*

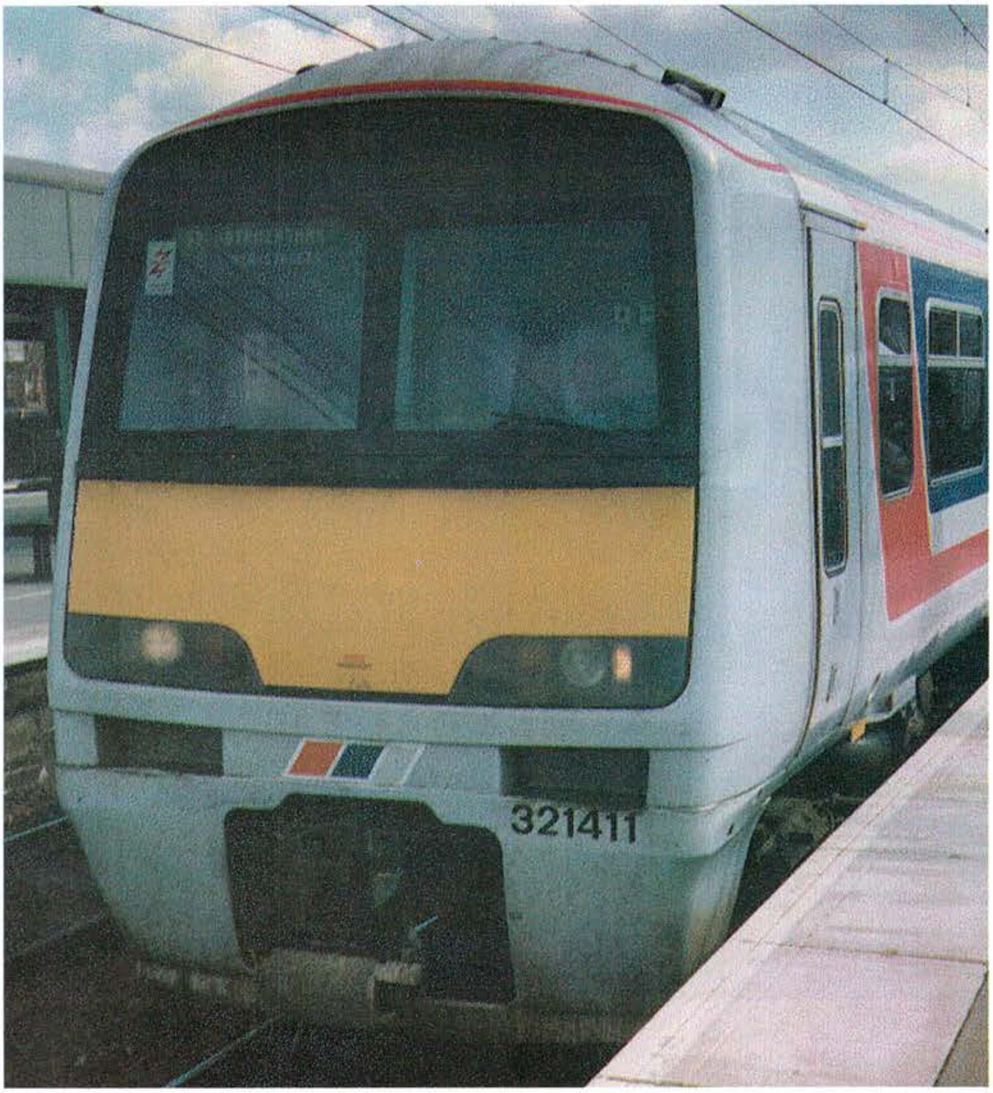


Plate 5 — an example of the output of the ***Epson GT6000 scanner*** *and software supplied in Irlam Instruments* ***Proimage*** *package. Floyd and Steinberg error diffusion allows the limited 256-colour palette to provide a remarkably lifelike rendering of the natural colours in a colour photograph. The print, scanned at 200dpi, is another example of the work of Max Amos (see Figure 10.2) and was taken at Watford Junction*

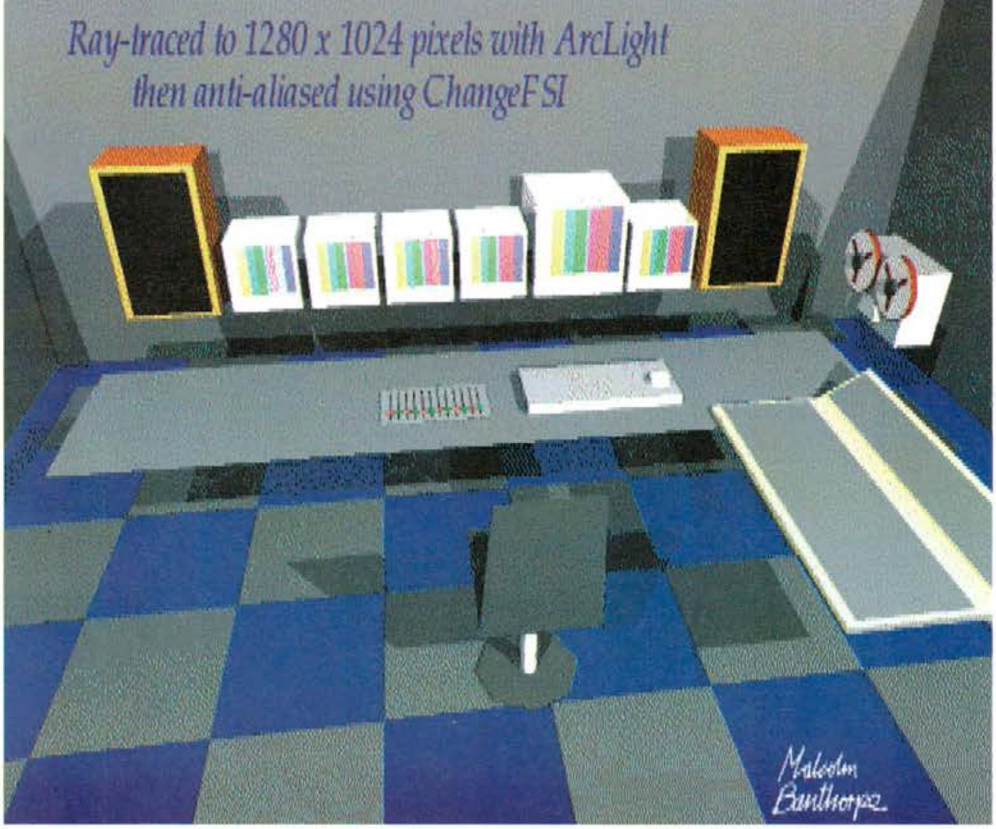

Ray-traced to 1280 x 1024 pixels with ArcLight
then anti-aliased using ChangeFSI
Malcolm
Banthorpe

Plate 6 — Malcolm Banthorpe's Studio is an exercise in antialiasing, and its needs to be since it is in mode 15 and nearly all the lines are oblique. The drawing originated in ***Euclid*** *and was ray traced using* ***ArcLight*** *with final processing in* ***ChangeFSI****. Malcolm is a video editor and video editing suites like this one provide the inspiration for much of his graphics work.*

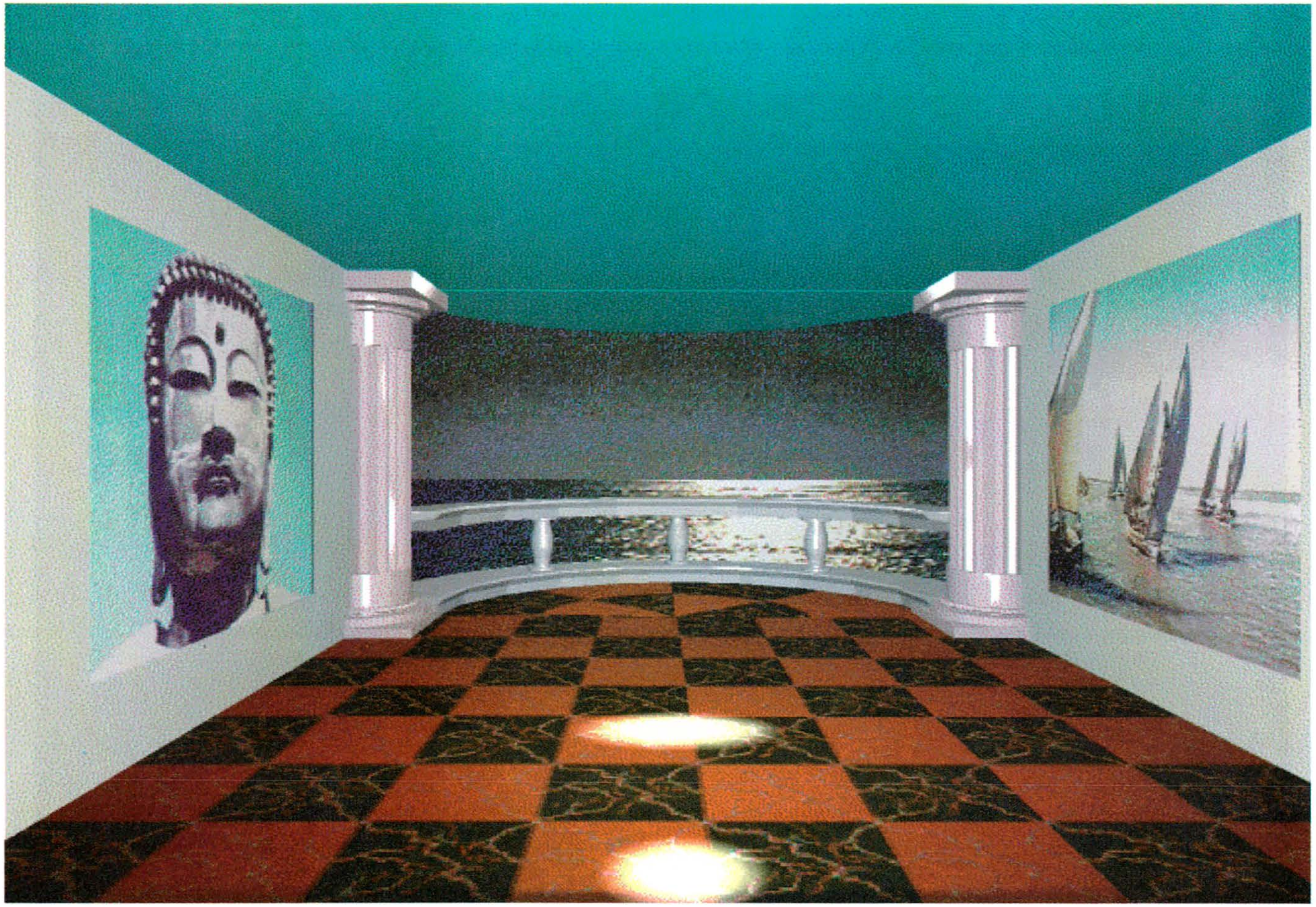


Plate 7 — One of the packages that drew attention to the graphics capabilities of the Archimedes in the early days was ***Clare's Render Bender****. Today an enhanced* ***Render Bender II*** *continues to attract the artistic talent with its facilities for ray tracing images and animation, as this scene, Balcony reveals.*

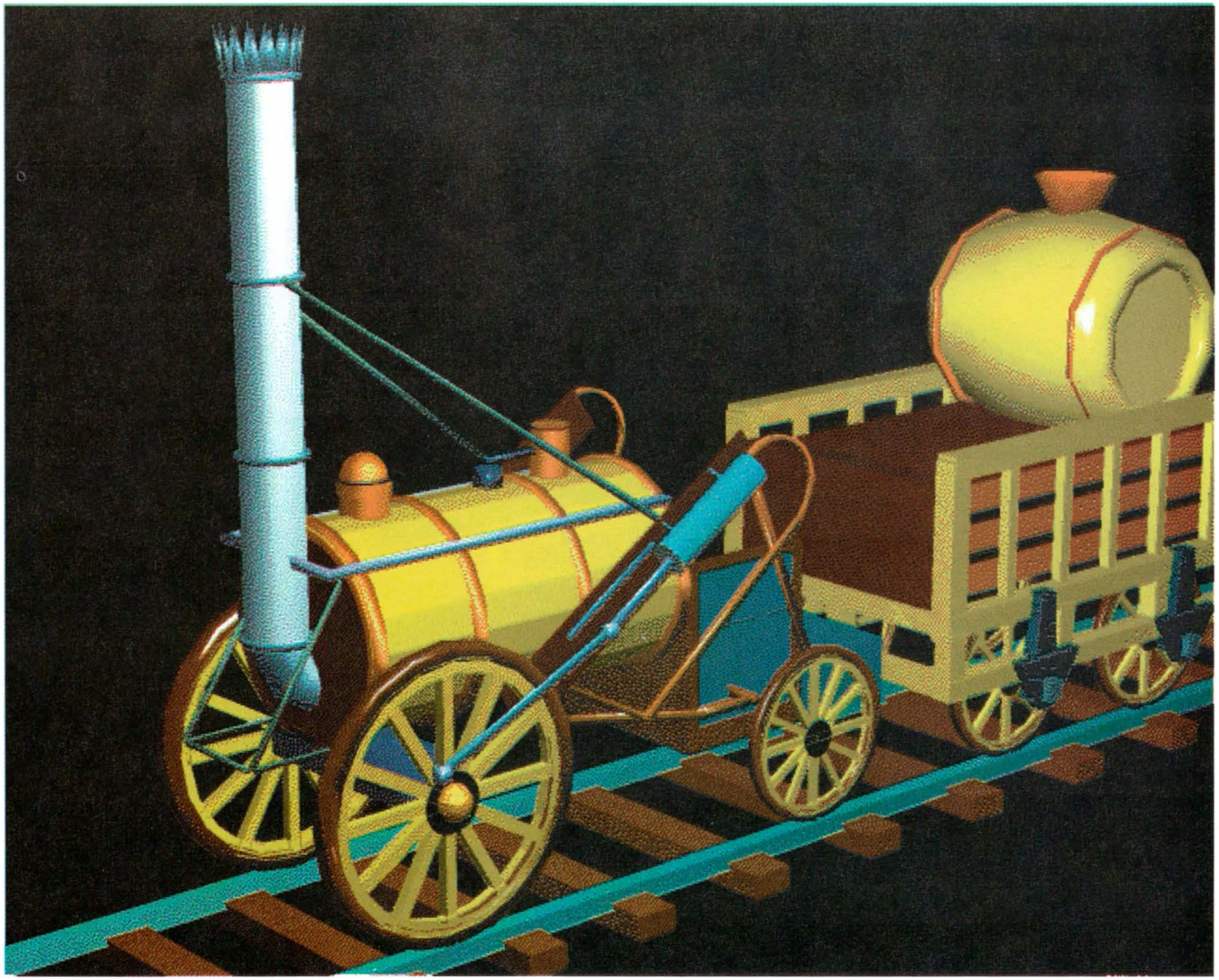


*Plate 8 — Render Bender has now been supplemented by **Illusionist** which is not a true ray tracing program, but nevertheless produces images of stunning realism and beauty. **Illusionist** is aptly named: it is easy to overlook the absence of true curves in this picture, but as a lifelong railway enthusiast what worries me is the absence of wheel flanges…*

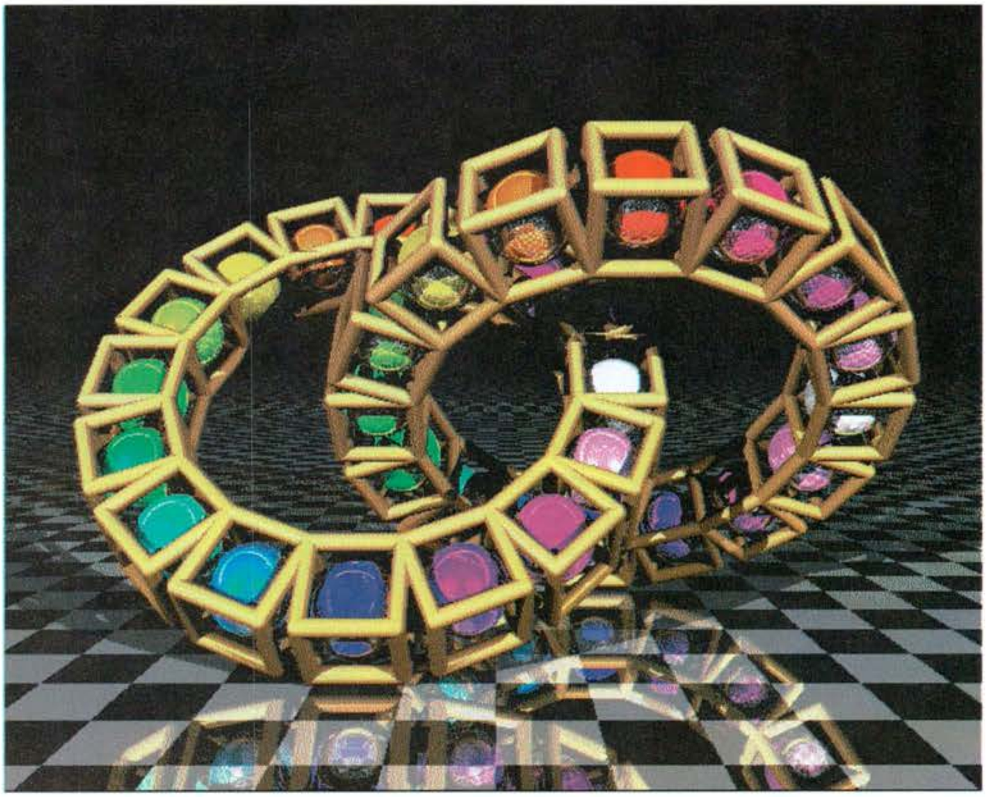


*Plate 9 — Ballring2 exemplifies the remarkable capabilities of Roger Attrill's **PowerShade** which creates ray-traced images from textual descriptions of the scene. This is a mode 21 sprite.*

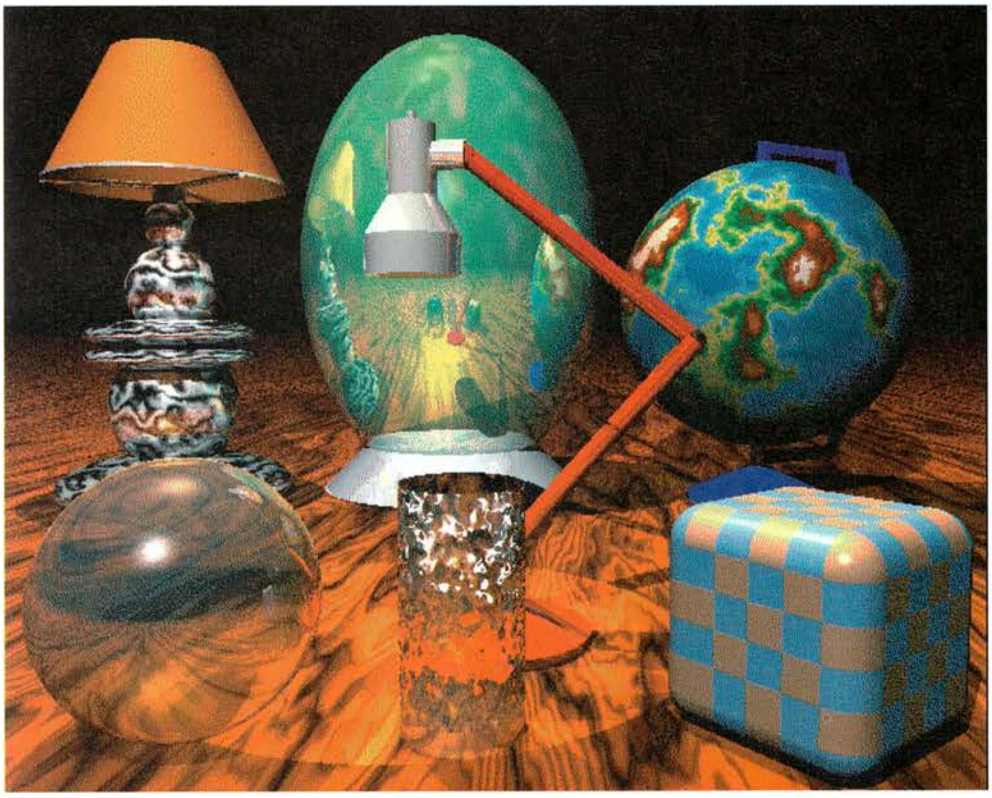


Plate 10 - Still Life, ray-tracing style. The artist is Roger Attrill and the program ***PowerShade****.*
Screen mode is 21. Some of the text file that produced this image is listed in Chapter 13.

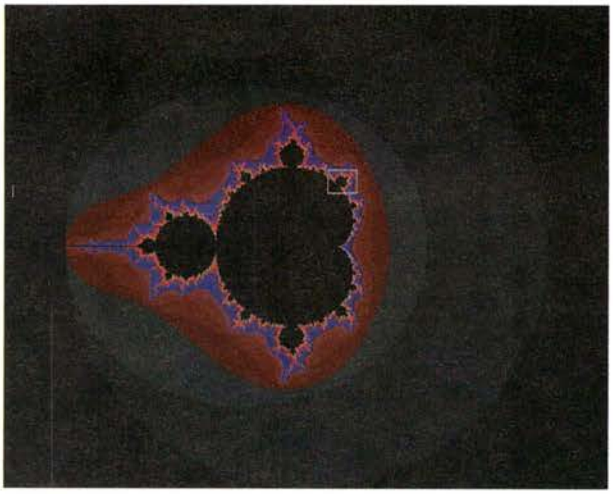
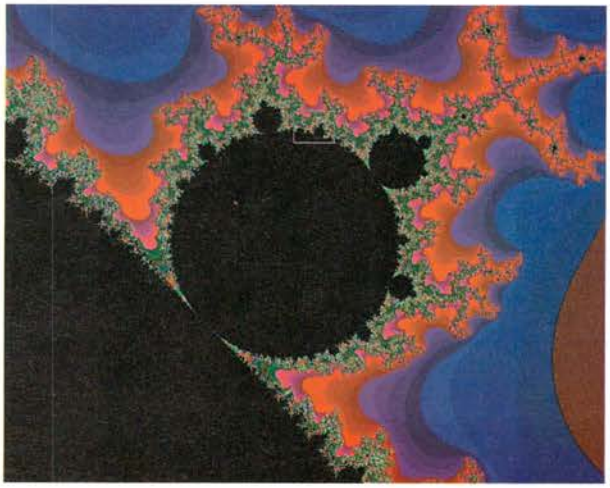
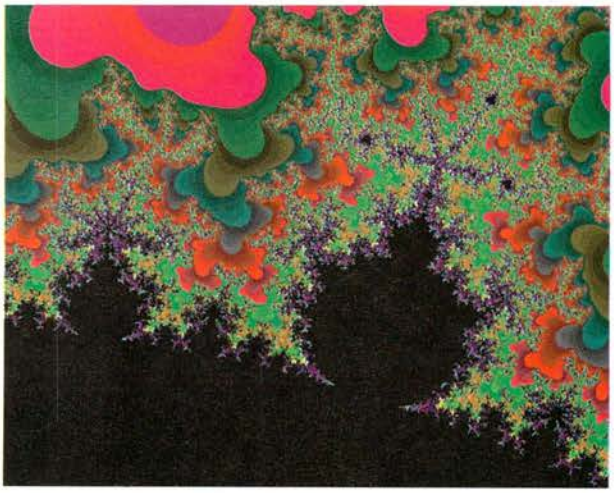




Plate 11 a, b, c — the unearthly beauty of the Mandelbrot set explored using Raymond Keefe's ***Mandlbrot*** *software set to 256 levels.*

Top: the whole set with an area highlighted for zooming.

Middle: the highlighted area zoomed to fill the screen with another area highlighted for zooming.

Bottom: the above highlighted area zoomed to fill the screen. This sequence well illustrates the Mandelbrot's recursive nature.

Plate 12 - even more unearthly than the Mandelbrot set, an example of the output from another of Raymond Keefe's creations, ***Lyapunov***

*Plate 13 — a **Julia set**, part of the range of software offered by the Third Millennium which specialises in abstract graphics with more than a hint of the psychedelic sixties...*

Plate 14 — Fractal by Jan Van Mourik is another of the Third Millennium's offerings

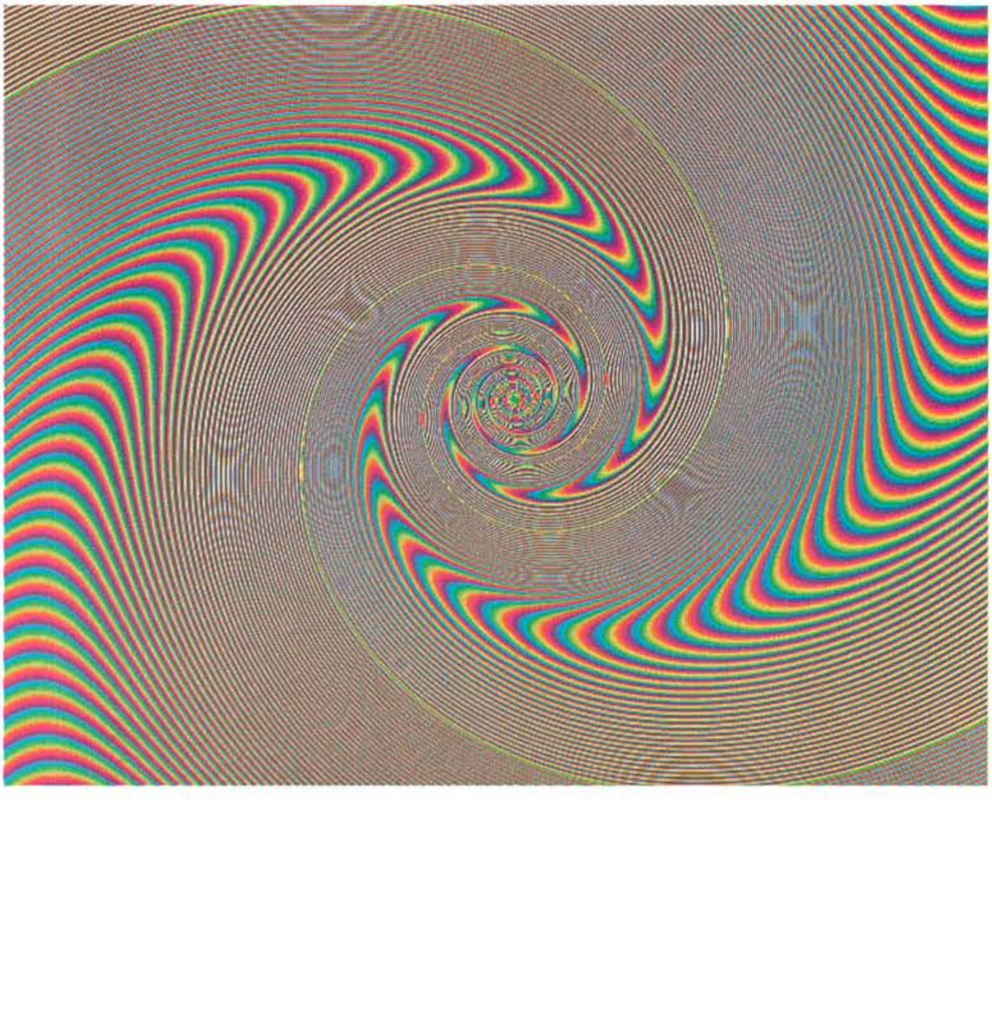


Plate 15 - a two-dimensional transformation created and animated by the ***BASIC program*** *listed in Chapter 16. Its ever changing patterns provide endless fascination.*

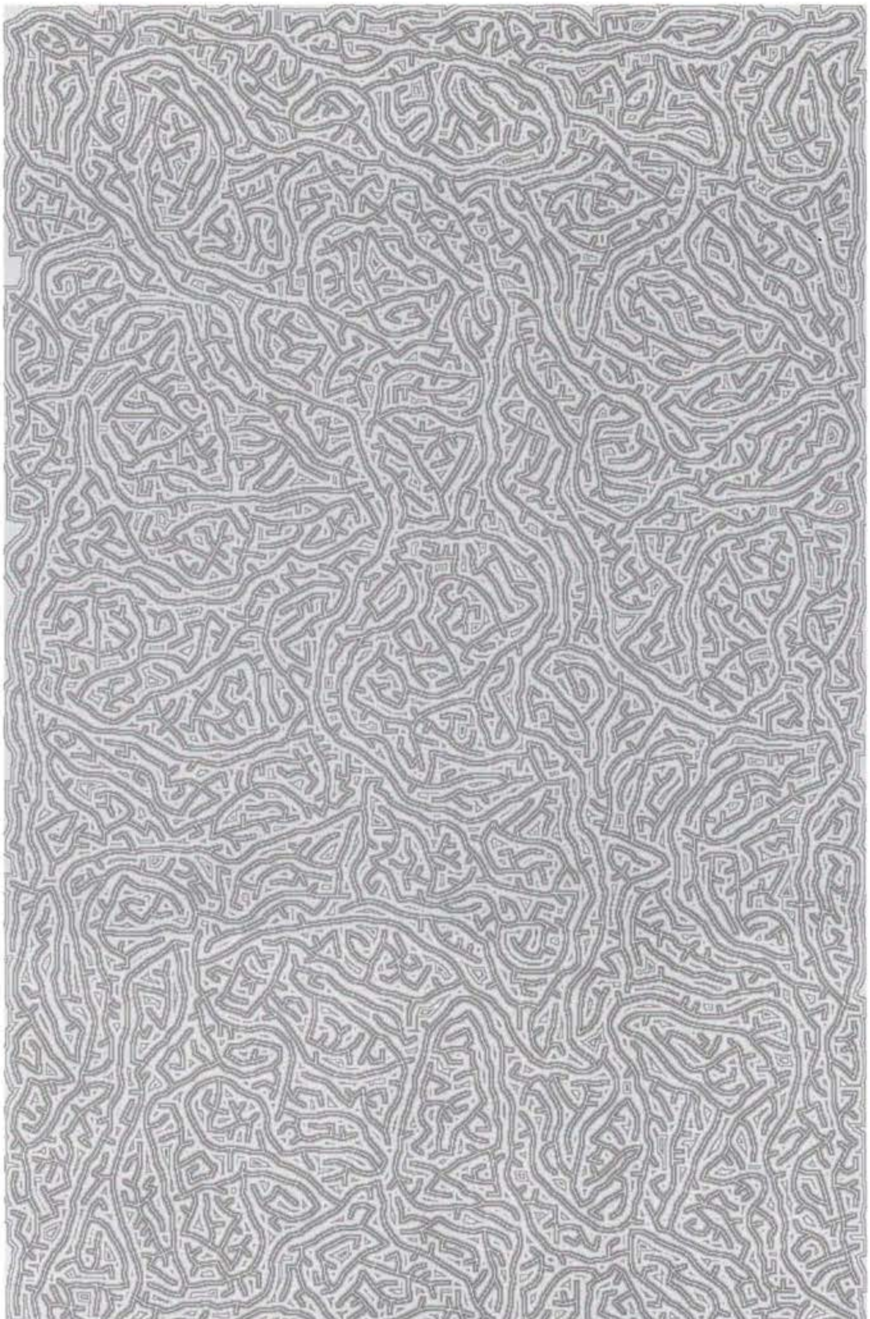


Plate 16 — Maze, another of Jan van Mourik's creations, available from the Third Millennium

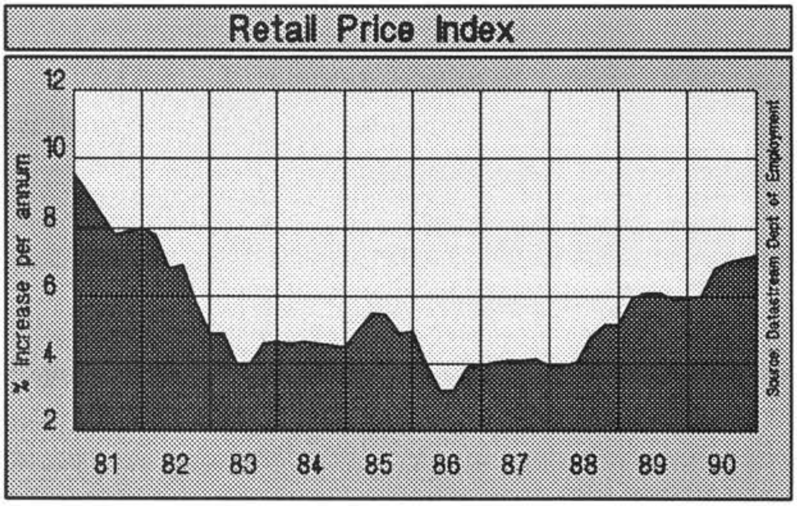


Figure 7.3—Graphs can be tedious to produce in *Draw*, but *DrawAid* allows BASIC programs to create graphs like this one from a block of raw data. All text is in the vector font

would do it in *Draw*. Facilities allow for the starting, ending and closing of paths, the creation of line, curve and move segments, the location, rotation, mirroring and scaling of path objects, including applying separate scale factors to the x- and y-axes. Facilities are provided for shearing the paths, that is slewing one axis only.

At present, text handling facilities are limited to the creation of text objects using the system font or the vector font. Those using the system font can be produced in only one size and horizontally. Those using the vector font, however, can be produced at any size (height is specified in mm rather than the customary points), any weight (*i.e.* line thickness), although four named weights are provided (thin, light, medium and bold), any style (*i.e.* inclination from the vertical), although two named styles, regular and oblique, are provided and any angle, this referring to the orientation of the theoretical line on which the text sits.

The grouping facilities are limited. Two procedures are provided, PROC_new_group and PROC_end_group. After PROC_new_group has been executed all new objects created will be grouped together until PROC_end_group is executed. In addition, all objects are collected into a single group in the final *Draw* file.

Version 2.00 of *DrawAid* is under development; it will greatly extend the facilities offered. It will handle sprites, including scaling and flipping, text in outline fonts, 16- and 256-level grey levels and HSV colour data, and the creation of polygons and splined curves.

Running the BASIC program

The BASIC program is run from the desktop. It makes three passes through the drawing definition (recalling assemblers which usually require more than one pass). The first pass checks the format and syntax and reports any fatal errors. The second pass creates the data block in *Draw* format and the third pass saves the data as a *Draw* file.

Displaying *Draw* files in BASIC programs

It is not complicated to display path objects from *Draw* files from within BASIC, although it is not quite so simple as displaying sprites (see Appendix 2). The drawing must be loaded into a reserved area of memory and a pointer set to the first object. Objects must be tested for type and sent to the appropriate routines in the modules that make up the operating system. At the end of each call the end of the file must be tested for.

Listing 7.1 is a simple BASIC program that prompts for the filename of a *Draw* file, loads it and displays the path objects in it. Note that in its present form it is limited to files no longer than 32 Kbytes (although this is easily changed). It displays only path objects, whether alone or grouped; the objects are displayed at whatever locations they occupied when the *Draw* file was saved. Other types of object (font lists, text objects, sprites, text areas and draw options) are simply ignored.

Listing 7.1—a simple BASIC program for displaying *Draw* path objects

```
REM This program displays path objects in Draw
files
REM It skips over other object types
DIM file% 32*1024:REM You can expand this as
necessary
DIM buffer% 255
INPUT "Enter filename"' filename$
SYS "OS_File",&FF,filename$,file%,0 TO
,,,,filelength%
CLS
PROCDraw
END

DEF PROCDraw
pointer%=40
REPEAT
objecttype%=file%!pointer%
IFobjecttype%=2 PROCPath
IFobjecttype%=6 pointer%+=36:ELSE
pointer%+=(file%!(pointer%+4))
UNTIL pointer%>=filelength%
ENDPROC
```

```
DEF PROCPath
fillcol%=(file%!(pointer%+24))
outlinecol%=(file%!(pointer%+28))
linewidth%=(file%!(pointer%+32))
!buffer%=0
buffer%!4=&20000
IF fillcol%<>-1 THEN
SYS"ColourTrans_SetGCOL",fillcol%,,,0,0
SYS"Draw_Fill",file%+pointer%+40,&32,0,0
ENDIF
IF outlinecol%<>-1 THEN
SYS"ColourTrans_SetGCOL",outlinecol%,,,0,0
SYS"Draw_Stroke",file%+pointer%+40,&3A,0,0,line
width%,buffer%,0
ENDIF
ENDPROC
```

8 Pixel Graphics and *Paint*

Although pixel graphics lack the versatility of vector graphics, almost certainly they are more widely used. Text and icons on the screen of your computer are pixel graphics. The output from scanners and video digitisers consists of pixel graphics. Saved screens, including screens imported from other computer systems, are pixel graphics.

Pixel graphics are simply two-dimensional arrays of pixels which often—but not always—correspond with the pixels on the monitor screen. Graphics built up pixel-by-pixel in this manner are commonly called *sprites*. Indeed it is often convenient to think of a sprite as a block of screen memory.

Sprites vary in size from just a few pixels representing tiny designs—such as individual characters on the screen or the icons used in the RISC OS desktop environment—to huge arrays larger than whole screens. Sprites obtained from the output of scanners are often larger than a screen.

One advantage of sprites over vector graphics is the speed with which they are plotted on the screen and also the ease with which user programs can make use of them. RISC OS contains many routines for sprite handling (see Appendix 2) which are readily accessible from BASIC, Assembler and other languages. If used with care, these even allow sprites created in one screen mode to be displayed in other screen modes, maintaining their original proportions even when the pixels in the current mode have a different shape.

Two warnings to newcomers. Firstly, bear in mind that pixel graphics can easily take up vast amounts of memory. For example, a mode 21 screen occupies 320 Kbytes, so you will only get two of them on an 800 Kbyte floppy disc (perhaps more if you use a file compression technique). On a 1-Mbyte machine if you start incorporating sprites into *Draw* documents or DTP documents, you may soon run out of memory. For serious work in pixel graphics 2 Mbytes is essential and 4 Mbytes is desirable. If your sprite only needs to use two colours (perhaps black and white) it will clearly save precious storage space if you create it in a two-colour mode such as mode 0 or mode 18 rather than the default 16-colour mode.

Secondly, never forget that you are dealing simply with a block of pixels. There is no data stack storing the contents of the graphic. And this has important consequences regarding the deletion of unwanted detail. In vector graphics it is easy to delete an object—you simply remove the object's definition from the stack and this forces the software to redraw the artwork with that object missing. But there is no corresponding operation in pixel graphics. All you have is a block of pixels. You cannot simply delete the pixels that make up the unwanted detail: every pixel must contain one or another of the palette colours. Some software allows you to delete whole columns or rows of pixels, but if you use these to delete unwanted detail you may also delete other material which you wished to preserve. The only safe way to remove unwanted detail in pixel graphics is to change the pixels in that area to whatever colour(s) they would otherwise have been. This may simply involve resetting them to a background colour, or it may involve reconstituting detail that the removed object had hidden. It is not surprising, then, that some pixel graphics software—including *Paint*—has no erase facility. Some pixel graphics software, however, does include an undo facility that reverses the effect of the last operation.

Sprite attributes and sprite files

Acorn has established a standard sprite format which is recognised by all RISC OS applications which make use of sprites. This makes the exchange of pixel graphics between applications very easy.

Each sprite consists of a block of data which contains, besides the details of the individual pixel colours, other vital data about the sprite called the sprite *attributes*.

Each sprite has its own *name* which may be up to 10 letters long. The sprite name is a convenient means of specifying an individual sprite in a file containing many sprites.

Each sprite also carries a record of its width and height (in pixels). These are essential since a block of, say, 4000 pixels may be 20 × 200, 40 × 100, 80 × 50, 160 × 25 or some other combination of width and height. Each sprite also has a definite physical size depending on both the number and size of its pixels. Pixels in modes 18 to 21 and 25 to 28 are assumed to be square having sides 1/90 inch long. Thus a mode 18 sprite 90 pixels wide and 90 pixels high is one inch square. Pixels in mode 13, however, are exactly twice as high and twice as wide so a 90 × 90 pixel mode 13 sprite will be size 2 × 2 inch.

The screen mode in which the sprite is intended for display is stored, since this affects the shape and size of the pixels and therefore the proportions of the sprite. It also determines the number of colours that it may use.

Optionally, a palette may be saved with the sprite. This allows the sprite to use specially created colours (except in a 256-colour mode) and ensures that it is always displayed in those colours if possible, otherwise in the closest colours available from the current palette.

Optionally a sprite may have a *mask*, more correctly called a *transparency mask*. This allows you to make any of the pixels in the sprite transparent. If a pixel is transparent, its own colour is not displayed—instead the colour of the background—or any other object behind the sprite—is displayed. By making pixels around the edge of the sprite transparent, the sprite can be effectively made any required shape. If no mask is present, every pixel must be a colour from the current palette and the sprite is of necessity rectangular in shape.

The mask is essentially a map of the sprite showing which pixels are transparent. Note that adding a mask to a sprite always doubles its memory requirement. So if you add a mask to a 256-colour sprite in which one byte represents each pixel, in the mask a full byte is used to identify transparent pixels. A screen-size mode 21 or mode 28 sprite with a mask will therefore occupy at least 600 Kbytes with clear storage implications on 1 Mbyte machines or on 800 Kbyte floppy discs.

Although any sprite can be stored as a separate file, a sprite file may contain any number of sprites. In RISC OS 2 double clicking on a sprite file icon will load *Paint* (if the computer has 'seen' the *Paint* application) and then load the sprite file into *Paint*. If the computer has not seen the *Paint* application, the sprite file will be loaded into memory and the first sprite in the file will be displayed in the bottom left-hand corner of the screen, the screen mode being changed (if necessary) to that of the sprite concerned. In RISC OS 3, double-clicking on a sprite file icon will always load *Paint* (since it is held in the permanently available ROM-based resources filing system) and the sprite file will be loaded into *Paint*.

Paint

The RISC OS application *Paint* allows you to create new sprites and to edit existing ones. Unlike *Draw* which, as we have seen, is now rivalled by a host of sophisticated drawing packages, there have been very few serious rivals to *Paint*. Sprite-based art packages appeared as soon as the Archimedes came on the market and today there is a profusion of them, but many have been limited to mode 15 and many to whole-screen-size sprites. (Mode 15 was originally regarded as the most useful screen mode for art because it offers 256 colours, reasonably high definition and is available on 1-Mbyte machines. It is also the mode in which many video digitisers prefer to operate.)

Paint, in contrast, is infinitely flexible. It operates in any desktop screen mode and, provided there is sufficient memory, will create or edit sprites of any size and in any mode, whether or not that is the current mode. (Nevertheless, creating a 256-colour sprite while running in a mode having 16 or fewer colours would hardly be practical.)

An exceptionally easy-to-use application, *Paint* sustained rather fewer changes than *Draw* when RISC OS 3 was introduced. Sadly the few quirks and bugs which afflicted the RISC OS 2 version also survived the change. These, however, are irritating rather than disastrous.

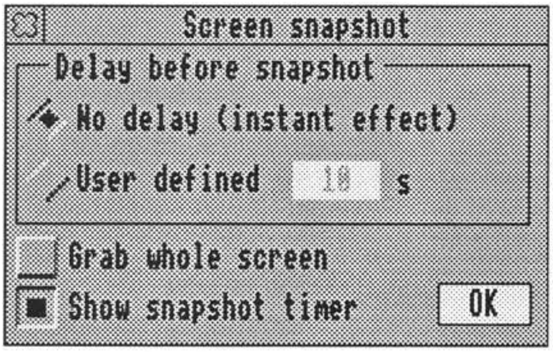
In general, the user guide explains *Paint* well, but the following may help you on points which are not clear. When using *Paint* it is even more important to save your work regularly than when using other creative software. This is not because of bugs in the software, but because of the very nature of pixel graphics. A wrong key press or misjudged drag operation can easily ruin your work and there is no undo facility. Consequently it is useful to be able to return quickly to an earlier but recent stage.

Get screen area/Snapshot

Paint is loaded in the normal manner, its icon appearing on the icon bar. Clicking Menu on the icon offers the usual information about the version in use and the option to leave *Paint*, saving any unsaved work. But it also leads to a very useful utility—Get screen area (RISC OS 2) or Snapshot (RISC OS 3). These allow you to save any rectangular area of the screen as a sprite. You should either ensure that the destination for the resulting sprite file is accessible on the screen, since the save dialogue box disappears if you alter anything else on the screen, or be prepared to enter the full pathname for the file. The destination may be a directory display or an application window or an application icon such as the *Paint* icon itself. Indeed it is good practice to drop the sprite straight into *Paint* so that you can check that you did indeed capture the required detail; you can also trim surplus matter from the edges of the sprite if necessary.

In RISC OS 2 the facility is basic but nevertheless very valuable. When you click Select on this option, the pointer changes to a camera with an arrow. Move this to one corner of the area you wish to save and press Select. Keeping Select pressed, drag the pointer to the opposite corner of the area to be saved—you will drag out a 'box' as you do so. When the box encloses the required area release Select, the box disappears and a standard save dialogue box now appears.

In RISC OS 3 the facility has been extended and made more useful. Clicking select on the Snapshot option opens the dialogue box shown in Figure 8.1. With the default settings (as shown in the illustration) a click on OK leads to the same camera routine that was offered in RISC OS 2. The alternatives are a Grab whole screen option and a timer whose period is adjustable from 1 to 999 seconds. The timer allows you to set up a screen and grab it; you need not have the *Paint* icon on the screen at the time. It allows you to save screens out of software that, perhaps, offers no opportunity to access the desktop or to save screens including menus or other details that disappear when the pointer is moved out of them.


Screen snapshot

Delay before snapshot

[✓ No delay (instant effect)]

User defined 10 s

Grab whole screen

[✓ Show snapshot timer]

Figure 8.1—the screen snapshot dialogue box in the RISC OS 3 version of *Paint*

Loading and creating sprite files

To load an existing sprite file into *Paint* drag its icon on to the *Paint* icon. Alternatively, if *Paint* is installed or if the computer knows where to find it, double-click Select on the sprite file icon. This opens a sprite file window which will display all or some of the sprite file's contents, depending on how many sprites the file contains. The window scrolls in the normal way and can be resized. It includes a small image of each sprite. This may be full sized if the sprite itself is small, but if the sprite is large, it will be a scaled-down version which may have a fuzzy appearance, especially if the original uses extended colour fills or other dither patterns.

Note that in version 1.41 of *Paint* (that supplied with early A5000 machines) the loading of sprite files into *Paint* from floppy disc is rather slow. To make matters worse, they are loaded twice. The loading of long (400 Kbytes or more) sprite files is made quicker by setting up a sufficiently large RAMdisc (if you have the memory to spare), loading the sprite file into *Edit* and then saving it to RAMdisc. Then double-click Select on the sprite file in the RAMdisc directory.

To create a new sprite file, click Select on the *Paint* icon and a blank sprite file window will be opened. In RISC OS 2, select the Create option in the sprite file menu to start your first sprite. In RISC OS 3 a Create new sprite dialogue box (Figure 8.2) will be opened automatically with the new sprite file window and it can be reopened whenever needed by moving right on the New sprite option in the sprite file menu (described below). This dialogue box sets the attributes for the new sprite.

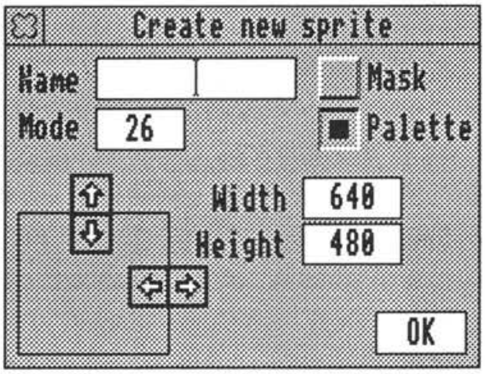


Figure 8.2—the create new sprite dialogue box in the RISC OS 3 version of *Paint*

Note that you can also extend sprite files by merging them. If you drag a sprite file icon into a sprite file window, the sprites in the new file will be added to those in the existing file. However, if any sprites in the newly introduced file have the same names as any in the original file, those in the original file will be deleted. The new sprites having the duplicated names will be deemed to have replaced the originals. Caution! No warning is given.

The Sprite File Menu

Clicking Menu with the pointer in the sprite file window leads to the following menu which concerns operations on the whole sprite file.

Misc (RISC OS 3)—this leads to a submenu of two items, Info and File. Info repeats the icon bar info about the version of *Paint* in use. File gives vital information about the sprite file such as its filename (and path), the number of sprites it contains, whether it has been modified since it was loaded and the length of the file in bytes.

Info (RISC OS 2)—this gives the same information about the sprite file as the File option from the Misc menu in RISC OS 3 (see above).

Display (RISC OS 3)—this leads to a submenu of options concerning the display of the contents of the sprite file window. A tick appears by each option currently in force. The default is Drawing and name which is self explanatory. The alternative, Full info, displays the entire sprite attributes, saving space by using a smaller image of the sprite itself. Use Desktop Colours, if ticked forces sprites that lack palettes to be displayed in the current desktop colours. Otherwise they are displayed in the logical colours used by BASIC and may look rather odd, especially since in 16-colour modes some colours may be flashing.

Save—This saves the sprite file using the normal RISC OS conventions.

Create (RISC OS 2)—use this if you wish to create a new sprite in the current file. A dialogue box will prompt you for the standard sprite attributes, *i.e.* a name for the sprite, its width and height in pixels, its screen mode (which need not be the mode currently in use) and whether or not it is to have a mask and a palette.

Sprite "name"—if the pointer was not over a sprite image or name in the sprite file window when you clicked Menu, there will be no sprite name between the quotes and the option will be greyed out showing that it cannot be selected. Otherwise it leads to a menu of operations that affect the sprite named. These will be considered below under the Sprite menu.

Display (RISC OS 2)—this leads to the submenu of options concerning the display of the contents of the sprite file window described earlier for RISC OS 3.

Use Desktop Colours (RISC OS 2)—this is the same as the sub-option of the RISC OS 3 Display menu described earlier.

New sprite (RISC OS 3)—this leads to a dialogue box (Figure 8.2) which prompts for the attributes of a new sprite to be added to the sprite file. By default the sprite is in the current screen mode and is full-screen size, with a palette but without a mask. Arrow icons provide for easy adjustment of the sprite size.

The Sprite Menu

The sprite menu is opened by clicking Menu on the entry in the sprite file window for the individual sprite of interest and then clicking Select on the Sprite 'name' entry, where 'name' represents the name of the sprite concerned. The order of the menu below is that in RISC OS 3; in RISC OS 2 the order is slightly different. The options themselves, however, are identical.

Copy—this prompts for a new sprite name and creates in the same sprite file a copy of the original sprite identical in all respects except for its name. This option is very useful for making a backup copy of a sprite if you wish to improve it, but are afraid your improvement may be unsuccessful. You can always delete whichever version is worse.

Rename—this prompts for a new sprite name for the sprite concerned.

Delete—this removes the sprite from the sprite file. Caution! It is lost irretrievably unless you have a copy elsewhere. No warning is given.

Save—this saves the sprite as an individual sprite file.

Info—this displays the attributes of the sprite.

Print—this leads to a dialogue box which allows a copy of the sprite to be printed, assuming of course that the printer driver has been installed and the printer is ready. Facilities allow for the sprite to be scaled and the printer origin to be offset. For the printing of very large sprites (such as those obtained from high-resolution scanners) scaling ratios must be chosen with care—see Chapter 10 and especially Table 10.1.

Creating a new sprite

Unlike vector graphics which can simply grow as you add new lines and shapes, each sprite has a definite size which is a function of its width and height in pixels and its screen mode (which determines pixel shape and size). To create a new sprite, you must first enter its intended size and screen mode in the create new sprite dialogue box, along with its name. You can always adjust its size later on, although changing its screen mode is less easy (see later). Having set up the sprite attributes (name, mode, width, height, palette and mask options) clicking OK creates the sprite in the current sprite file. In RISC OS 3 it also opens a sprite window and colours window—if only a limited range of colours is available, drag the current palette into the sprite window. Initially all the pixels in the new sprite will be set to one colour, normally black or white. This of course makes for a very uninteresting piece of graphics, but the sprite now officially exists and the creative work of giving it an interesting design is a matter of sprite editing.

The entry in the sprite file window for your newly created sprite may not appear until you force an update of the window by resizing it. Because of a bug in some RISC OS 3 versions of *Paint* if you have chosen the Full info display option, even resizing the window will not display new sprites. You will need to cancel this option and then reinstate it to see the Full info window display for new sprites.

Sprite editing

To edit an existing sprite you must double-click Select over the sprite image in the sprite file window. This opens a sprite window containing an image of the sprite concerned. All editing takes place in this window. In RISC OS 3 it also opens a Colours window and Tools window. Just as *Draw* allows you to have several windows open showing the same document, so in *Paint* you may open several windows on to one sprite.

Clicking Menu with the pointer in the sprite window leads to a menu of operations concerning this sprite. Some of these operations duplicate those available from the sprite window considered earlier.

Misc (RISC OS 3)—this leads to a submenu of three options: Info (the application information), Sprite (the sprite attributes) and Print (which prints the sprite if the printer driver and printer are ready).

Info (RISC OS 2)—displays the attributes of the sprite.

Save—this leads to a submenu allowing the sprite and its palette to be saved individually.

Paint—this leads to the Paint menu which is concerned primarily with colours. This is considered later.

Edit—this leads to the Edit menu which is concerned primarily with the sprite's size and orientation. This is considered later.

Zoom—this leads to a zoom dialogue box like the one in *Draw* except that it supports zoom factors up to an incredibly generous 999:1 (at which size a single pixel in nearly all screen modes would more than fill the screen) and down to 1:999. For editing work on sprites a zoom of at least 4:1 is recommended.

Grid—the grid is a set of fine lines marking the edge of the pixels in the sprite. It makes it far easier to count pixels; this may be useful if detail in one sprite needs to be matched against another. The grid can only be displayed at zoom settings of 4:1 or higher. This menu option allows you to turn the grid on and off and to choose any palette colour for the grid lines.

Print (RISC OS 2)—this prints the sprite if the printer driver and printer are ready.

The Paint Submenu

This submenu, entered from the sprite main menu, is concerned mainly with the colours used in the sprite.

Select ECF—this allows you to use up to four previously defined dither patterns (Extended

Colour Fills) as though they were colours from the palette. The dither pattern is defined as the bottom left-hand corner of a named sprite in the current sprite file, so you must create a sprite containing the required dither pattern first. Note that there are some limitations as to the use of dither patterns. If the two sprites are in screen modes using differently shaped pixels, you may find that dither pattern consists of stripes rather than the usual chequerboard.

Select colour—in *Paint* as in other pixel graphics applications painting operations require a currently selected colour. The normal way to select this colour is to click Select over it in the colours window; the selected colour is highlighted by a white boundary box and the appearance in the square of the colour's number. The Select colour option provides the useful alternative of selecting a colour from within the sprite itself. Move the pointer over a pixel of the required colour, click Menu, choose Paint and then Select colour. The colours window will be updated to show that the colour over which you clicked menu has been made the current colour. This facility is most useful when working with 256-colour sprites in which you wish to match an existing colour but are not sure which colour was used. It is also useful if you do not wish the colours window to get in the way of the sprite.

Show colours—this opens a colours window for the sprite concerned. Note that each sprite has its own colours window and if editing windows for several sprites are open simultaneously, each will have its own current colour. Consequently you must choose colours for each sprite only from its own colours window. Clicking select in another sprite's colours window will not affect the current colour in the sprite on which you are working.

Show tools—this opens a window showing the range of brushes, pencils, sprayguns and cut and paste facilities available for sprite editing and creation. All tools use the currently selected colour. A 'help' line usefully explains the nature of operations which are not always obvious from the icons. Further information is given below.

Small colours—this toggles the size of the colours window between standard and small. It is most useful when working in 256 colour modes since a standard-size 256-colour-window occupies most of the screen.

Edit palette—this option, only available if the sprite concerned has a palette, allows you to redefine the colours used in the sprite. The procedure is as follows. In the colours window (or by other means) select the colour you wish to edit. Click Menu, choose Paint and move right on to the Edit palette option. This leads to a fairly standard 16-palette colours and 24-bit colour slider selector. You can change the currently selected colour instantly to any of the palette colours by clicking Select on it or you can mix yourself a colour using the sliders. In RISC OS 3 dither patterns are used to represent the current colour in the box in the colour selector, although they are not used in the colours window or in the sprite window itself.

The Edit Submenu

This submenu is concerned mainly with the size and orientation of the sprite.

Flip vertically—this inverts the sprite, *i.e.* turns it upside down.

Flip horizontally—this produces a mirror image of the sprite.

Rotate—this leads to a writable submenu in which you enter the number of degrees anticlockwise through which you wish to rotate the sprite. The default is 90 degrees. For clockwise rotation either enter a negative number, *e.g.* –90, or subtract the angle from 360. In RISC OS 2 rotation can be a very time consuming operation. In RISC OS 3 it is very fast.

Scale x (RISC OS 3 only)—this rescales the sprite's x-axis (horizontal) only, inserting as many extra columns as required. Fractional increments (*e.g.* 1.5) are permitted.

Scale y (RISC OS 3 only)—this rescales the sprite's y-axis (vertical) only, inserting as many extra rows as required. Fractional increments (*e.g.* 1.5) are permitted.

Shear (RISC OS 3 only)—This progressively slews a sprite image (*e.g.* making any text into italic, or rather, oblique text). The factor to be entered is the aspect ratio. A factor of 1 slews the top of the sprite to the right by its own width, *i.e.* through 45 degrees. See Figure 8.3.

Adjust size—this changes the boundary box of the sprite. Be careful! Making the sprite smaller may cause detail to be lost from around the top and right-hand edges. Once it has been lost, it

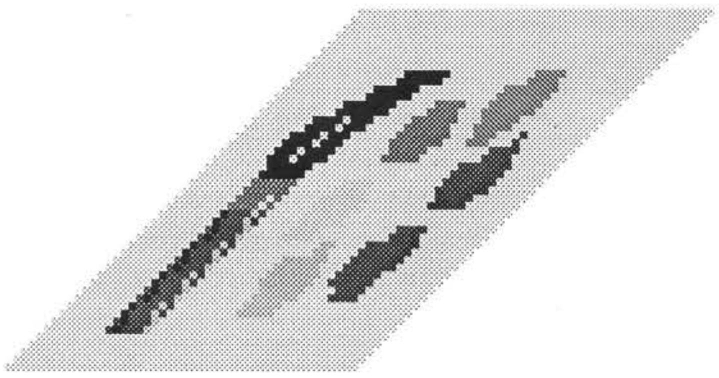


Figure 8.3—the effect of shearing a sprite (*Paint's* own icon) through a factor of 1

cannot be retrieved (unless you have it saved elsewhere). Two methods are provided—you can either click on directional arrows or write the new size in pixels. Note that despite the positioning of the arrows, all vertical adjustments are made to the top edge of the sprite and all horizontal adjustments to its right-hand edge. The bottom and left-hand edges are sacrosanct.

Insert columns, Insert rows—these allow additional columns (vertical) and rows (horizontal) of pixels to be inserted at any position in the sprite. Note that the new columns or rows will be inserted at the position of the pointer when you clicked Menu. A vertical or horizontal line will appear through the sprite at the position where the new material is to appear. A dialogue box will ask for the number of columns or rows to insert. You must press <Return> when you have entered the number. Clicking the mouse will cancel the operation.

Delete columns, Delete rows—these allow adjacent columns or rows to be deleted from the sprite. The operation is often used for trimming surplus matter from around the edges of large sprites such as those created by scanner-driver software. Note that these operations are somewhat temperamental. The range of columns or rows to be deleted is fixed at one end by the pointer position when you clicked Menu and at the other by moving the pointer to the required position. The pointer must not touch any other part of the menu structure since this cancels the operation. You may need to reposition the Edit menu to make this possible. Two coloured lines or one coloured band indicate the limits of the area to be deleted and a dialogue box displays the number of columns or rows affected. You can click Select or press <Return> to initiate the deletion. Pressing <Return> is more reliable; clicking select sometimes causes the menus to disappear without the requested deletion taking place. Trimming the top, bottom and right-hand edges of sprites is nearly always a two-shot operation, the first deletion leaving the outermost row or column unscathed; you get better results if you click Menu over the innermost row or column to be deleted. When deleting a single row or column, you must press <Return>; clicking Select has no effect.

Mask—this toggles the transparency mask on and off. Remember that adding a mask will double the sprite's memory requirements. It also adds an extra colour to the colours window. The extra colour representing transparency is shown as cross hatching. It can be used exactly like any other colour except that you cannot do a local flood fill (colour replacement) with transparent.

Palette—this toggles the palette on and off. If a sprite has a palette, it will always be reproduced in the currently available colours that give the closest match. In contrast, a sprite without a palette may be reproduced in a somewhat arbitrary fashion, using colours that are totally unsuitable. However, if the palette that was in force when the sprite was created has been saved elsewhere and is reloaded so that it is again in force when the sprite is displayed, the sprite will of course be shown in its original colours.

The *Paint* tools

The *Paint* tools (Figure 8.4) appear when you open a sprite window; they are also called up from the Paint menu. Unlike Colours, one set of tools is shared between all open sprite windows. The currently selected tool remains active if you cease working on one sprite and start working on another.

The range of tools provided is sufficiently comprehensive to enable *Paint* to create highly

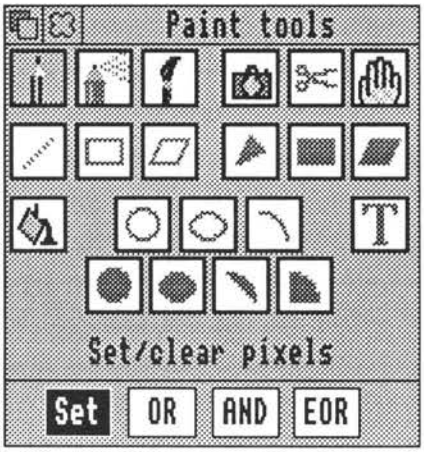


Figure 8.4—the *Paint* tools

original and artistic work. A help line provides a reminder of the function of each tool.

Four icons at the bottom of the toolbox give a choice of four plotting options. Most commonly used is Set which simply replaces the affected pixels by the currently selected colour. In the other three options the final colour of the affected pixels is influenced by both their present colour and the currently selected colour. On a white background OR generally has the effect of changes only being evident when they are made straight on to the background, *i.e.* they are ignored where pixels are any colour other than white. It is useful for creating drop shadows. AND has the opposite effect, changes against the white background being ignored but being visible where they overlie some other colour. EOR (exclusive-or) makes changes visible against any background and can result in some fascinating colour changes. If you repeat an exclusive-or plot on the same location, it reverses the effect of the previous one, causing the detail plotted to disappear.

The icons, from left to right and top to bottom are:

Most frequently used, the default tool is the pencil. When Select is pressed over the sprite with the pencil in use this simply changes the colour of the pixel at the pointer to the currently selected colour. It can be used in two ways: either to edit individual pixels (intermittent clicks on Select) or as a freehand drawing tool (Select held down continuously).

The aerosol can is a spraygun which sprays the area around the pointer with a mist of current-colour pixels. Useful for creating a variety of special effects (Figure 8.5), both the density of the spray and the radius are adjustable, although it is often easiest to obtain a denser spray by holding the (Select) button down a little longer.

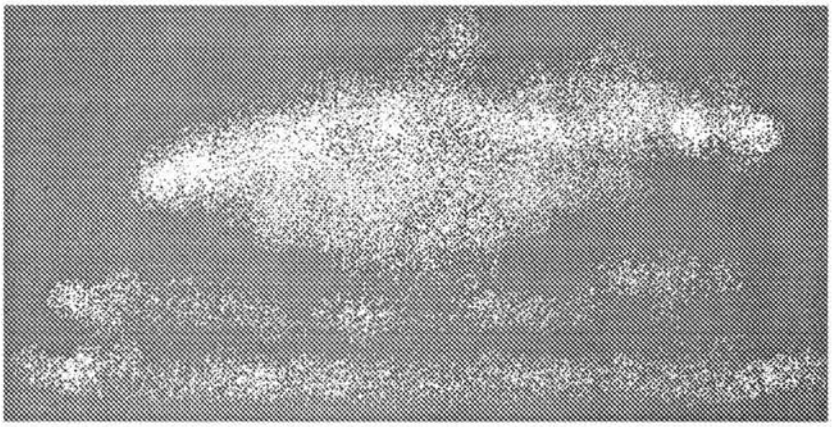


Figure 8.5—the spray can in *Paint* produces delightful cloud effects. (Imagine doing this in vector graphics!)

The paintbrush is a fascinating and extra-ordinarily versatile facility which allows you to paint using a named sprite as though it were a brush. Selecting this tool calls up a dialogue box in which you either enter the name of the sprite you wish to use (it must be in memory, but need not be in the same sprite file as the sprite on which you are working) or accept the default sprite 'brush' (RISC OS 2) or 'circle' (RISC OS 3).

The tool can be used in two ways depending on the setting of the shape switch. If selected, only the shape of the named sprite is used as a brush or template to deposit the currently selected colour in the sprite on which you are working. You can either drag the sprite around leaving a trail of colour or move it around clicking Select wherever you wish to leave a 'footprint'. It might seem that this facility is somewhat limited in that all sprites are inevitably rectangular. If the named sprite has a mask, however, only the non-transparent parts are regarded as the shape. So the sprite can have any shape you care to give it. The facility is therefore a brush of infinitely variable shape and size.

If the Shape switch is not selected, it is not only the selected sprite's shape but also its design that is copied into the destination sprite. The two sprites need not be in the same screen mode and this facility therefore provides a simple way of producing a copy of a sprite in another mode. You could copy a mode 20 (16-colour) source sprite into a mode 21 (256-colour) destination sprite or *vice versa*. If colours used in the source sprite are unavailable in the destination sprite's palette, the nearest available colour is substituted. You can also use this facility like a child's 'potato print' to repeatedly copy the detail of the source sprite into the destination sprite. For example, you can use it to reproduce a number of faces if you are drawing a picture of a crowd (Figure 8.6). Admittedly, all the faces are identical, but you could always edit each one a little later to add variety. Note that if the source sprite is larger than the destination sprite, you may find that you can only access a small portion of it in the destination sprite. But in these circumstances you are liable to lose all the original detail in the sprite you are editing. Instead of this routine you probably need the cut and paste facility described below.

The versatility of this facility is enhanced by the

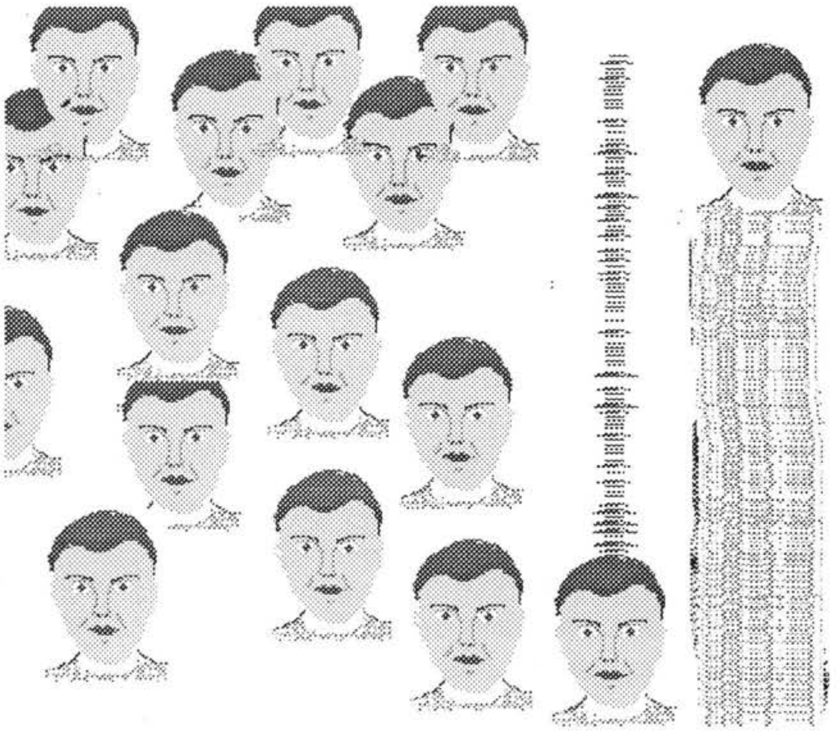


Figure 8.6—the crowd (left) was created with the Use sprite as brush facility in *Paint*. A single sprite for the face was 'stamped' repeatedly at intervals. Given time the faces could be edited to imtroduce variety. The streaks on the right were created by dragging the face sprite, first downwards and then upwards.

addition of scaling facilities. You can scale the source sprite up or down—this is useful if you are converting, say, a mode 13 sprite with its large pixels to mode 21 whose pixels are only one quarter the size.

Another multiple-copy effect is available from the camera tool. This drags a box around the rectangular area to be copied and then allows you to move the outline anywhere in the sprite; you click Select wherever you would like a copy. So you could build up a crowd by drawing a face somewhere in the sprite and duplicating this.

The scissors represent a cut-and-paste facility which is a variation on the same theme. The snapshot preserves the original; the cut-and-paste routine erases it, replacing it with colour 1 (often white) unless the sprite has a mask, when it is replaced by a transparent area. It is useful if you have drawn some detail in the wrong place and wish to move it.

Both these options offer an additional choice of 'local' (the default) and 'export'. Local means that the material can be copied only within the same sprite. Export means that the material copied or cut is converted to a sprite file which can be saved within the current sprite file or to disc or into another application. Incidentally, if you are exporting the material using the cut-and-paste facility, the original detail will be left intact; it is copied rather cut.

You can use this facility to transfer an area from one sprite to another: use the scissors to copy the area, save it as a separate sprite within the current sprite file and in the destination sprite name it as the brush for the Use sprite as brush facility. There is a bug in this facility. If you double click on the sprite you have exported, it may appear to have blank columns to one side and you may think, if you were cutting from near the edge of the original sprite, that you misjudged your cutting. Resist the temptation to delete the blank columns; if you attempt this *Paint* will crash. The problem appears not to arise if you drag your cutting box from right to left rather than from left to right.

The hand tool allows you to drag the entire contents of the sprite. It does not change the size of the sprite. Consequently, if you drag some detail over the edge, you will lose it unless you have a copy somewhere else. The matter lost is replaced at the trailing edge of the drag by background-colour material.

The line tool draws lines, one pixel wide. Place the pointer at the position for the start of the line, click Select, move the pointer where you wish the line to end and click Select again. The rectangle outline tool creates rectangle outlines, one pixel wide. You click Select at one corner and then at the opposite corner. The parallelogram outline tool creates parallelogram outlines, one pixel wide. You click Select at three corners.

The filled triangle tool used repeatedly provides one of the most effective methods of filling irregularly shaped areas. Click where you want the three angles. The filled rectangle and parallelogram tools are used in the same way as their outline counterparts, but produce filled, *i.e.* solid, shapes.

The spilled paintpot icon is useful but also dangerous and must be handled with care. It substitutes one colour for another (flood fill). It has both local and global options, the default, sensibly, being local. Select the colour that you want to change something to and then click Select over the area in the sprite that you want to change. If you have chosen the local option, the pixel over which you clicked and all others of the same colour that are contiguous with it will be changed; other areas of the same colour will be left unchanged. If you chose the global option, every pixel in the sprite having the same colour as that beneath the pointer will be changed to the currently selected colour. Caution! Save your work *before* using this option, since it is all too easy for the selected colour to spread to areas you did not intend. Also, when you have finished with this facility, select any other tool *immediately*. If you leave this tool selected, you may inadvertently click Select within a sprite and see hours of painstaking work disappearing under some hideous colour!

The circle outline tool draws circular outlines one pixel wide. Click Select where you want the centre of the circle and then click at any point on the intended circumference.

The ellipse outline tool is less straightforward. The first click determines the centre of the ellipse. The second places a point on the edge. The third click allows you to rotate the ellipse.

The circle section outline also takes some mastering. The first click determines the centre of the circle and the second the *starting* point of the circle section (arc) which is always formed *anticlockwise*. The third click determines the end of the arc. Obviously the centre of the circle must lie within the sprite. If it falls outside the sprite, you could always temporarily insert some columns or rows to make the sprite bigger.

The T icon writes text in the sprite. Only the system font is available, but it can be used in any size, the default being 8 pixels (wide) by 16 pixels (high), and in any palette colour. You enter your text in a dialogue box. Finally you place your text in the sprite. You may place it repeatedly if you need more than one copy. If you wish to use outline fonts in your sprites, load the sprite into *Draw*, superimpose your text over it and then grab the sprite off the screen using the Get screen area or Snapshot facility.

The filled circles and ellipses are formed in exactly the same way as their outline equivalents. The filled circle segments and sectors are both formed in exactly the same way as the circle section outlines.

Changing the screen mode of a sprite

Although *Paint* allows you to change the name and size of a sprite, it contains no explicit facility for changing a sprite's screen mode. Implicitly, however, it offers two routes for such a conversion, neither particularly elegant. One, involving the 'use sprite as brush' tool, was described earlier.

The other method is even less elegant, but simpler. Set your computer to the destination screen mode using the palette utility. Load any special palette necessary. Load the file containing the source sprite into *Paint* and open the sprite window. Select the appropriate zoom setting. (In changing a sprite from mode 13 to mode 21 or mode 28, for instance, you should set the zoom to 1:2 because of the difference in size of the pixels.) Now use the Get screen area or Snapshot facility to grab the sprite off the screen. Load it into *Paint* and trim the edges if necessary. There is a limitation, of course, in that the sprite must be small enough to fit on the screen.

A more elegant method is to use *ChangeFSI*, described in Chapter 11. There are also some public domain utilities which provide this function.

Sprite creation

Since you cannot simply rearrange the various parts of a sprite's design in the same way that you can in a vector graphic image, sprite creation repays careful thought. Begin at the back and choose a background colour. *Paint* usually provides a white background for new sprites, so if you want a different background colour you must select it and fill the sprite with it using either the filled rectangle facility or the replace-colour facility.

All design work should be undertaken at a zoom setting of at least 4:1. Use the grid if careful alignment is needed; even if alignment is not critical, the grid is often helpful.

You could do all your design work with the pencil and that might be acceptable for very small sprites such as icons. But it would be distinctly tedious if your sprite is a whole screen. For large design features use other tools such as rectangles, circles, ellipses and triangles. The filled triangle facility is very useful for filling in oddly shaped spaces.

Don't be afraid to import other sprites if their design is suitable for incorporation; it is futile to do the same work twice.

Final touching up can be undertaken with the pencil. Turn the zoom back to 1:1 regularly to see how the finished sprite looks at its natural size.

9 Art Packages

As mentioned in the last chapter, numerous art packages have been published for the ARM machines since the very earliest days. One of the first was Clare's *Artisan* which rather unusually was confined to mode 12 and which used dither patterns to provide more than that mode's 16 colours. Its successor, *ProArtisan*, which uses mode 15 (256 colours and fairly high resolution at 640 × 256 pixels) still enjoys considerable popularity. Many other art packages have been similarly restricted to mode 15, which needs a comparatively miserly 160 Kbytes of screen memory and therefore is suitable for 1-Mbyte machines.

This chapter examines the capabilities of three fairly recent contrasting art packages. The first is

Simon Hallam's *ARCtist* from the Fourth Dimension, well known for their high-quality games software, which is limited to mode 15. The other two packages are somewhat unusual in that they will run in any desktop mode. *Art20* from Tekoa Graphics, a package which, unusually, was originally designed to run in mode 20 (16 colours and 640 × 512 pixels) and *Revelation 2* from Longman Logotron was designed to run in all desktop modes. All three packages offer sophisticated facilities not found in *Paint.*

ARCtist

Like many art packages *ARCtist* is restricted to mode 15; it temporarily jumps out of the desktop environment and grabs the whole screen. This is not to evade the niceties of RISC OS, but to allow the full extent of the screen to be used for the artwork; if the artwork were contained in a conventional window (as in *Paint*) the top, bottom and right-hand edges would be hidden by the window frame. Although scrolling would make all of it accessible, you would never be able to see all of your picture at once. The package does partially multi-task in that it allows you to exit to RISC OS at any time, whereupon you will find any other installed RISC OS applications still as they were when you started *ARCtist*, and if you return to *ARCtist* you will find your artwork still intact.

Another point of interchangeability is that *ARCtist* has no file format of its own; all artwork is saved in standard RISC OS sprite format and, as such, can be loaded straight into *Paint* or any other applications which use sprites, including

DTP packages. There is also a facility to save any rectangular area of the screen as a sprite.

ARCtist is menu-driven, the menu of 19 items plus colour selector normally appearing down the left-hand edge of the screen. You can toggle the menu on and off using Adjust, allowing you access to the area of the screen hidden beneath it. Also you can edit the list of menu items, eliminating any which you never use. The menu philosophy is that you click Select once over an option to select it and double-click to call up a menu of options.

The colour selector seems at first sight impossibly small, each coloured square being just four pixels wide and two high (visible in Figure 9.1). In fact this does not really matter at all; the selected colour is displayed in a large rectangular pane just beneath the colour selector. Colours are selected by clicking Menu; whichever colour was beneath the pointer when menu was clicked becomes the selected colour whether that colour was in the colour selector or anywhere on the screen. The trick with the colour selector is to hold menu down and move the mouse around until the colour you want appears in the current-colour pane. But if you decide you *must* have something larger, the palette option on the menu puts a much larger and more logically arranged palette on the screen.

The zoom is unlike standard RISC OS zooms. It allows you to drag a small rectangle around the screen while a larger rectangle in one corner of the screen displays the area in the small rectangle at four times natural size. It does have

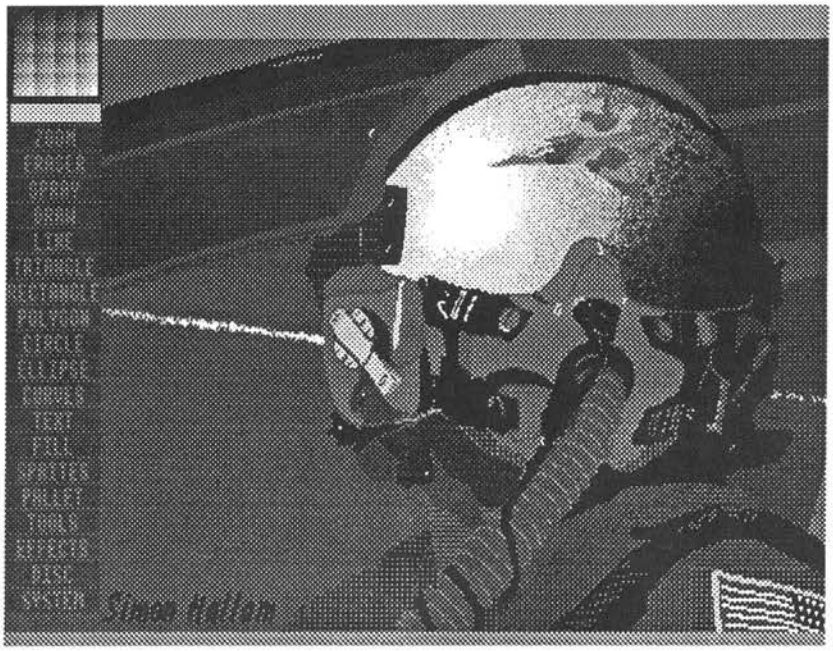


Figure 9.1—Simon Hallam's Highflyer with the complete menu and colour selector visible to the left of the screen. The menu can be toggled on and off to provide access to the whole screen

the advantage that you can simultaneously see the detail at its natural size and at a size that is big enough to allow fine editing. Moving the pointer into the enlarged rectangle allows you to carry on using tools at the higher magnification (Figure 9.2).

Unlike *Paint, ARCtist* offers two kinds of erase facilities. One is an undo function which reverses the effect of the last operation, although some complex operations cannot be undone. The other is a 'rubber' which simply paints the screen in whatever background colour you have chosen. (You can nominate any colour as the background colour; the screen will be filled to this colour.)

The spraygun offers a splendid range of facilities.

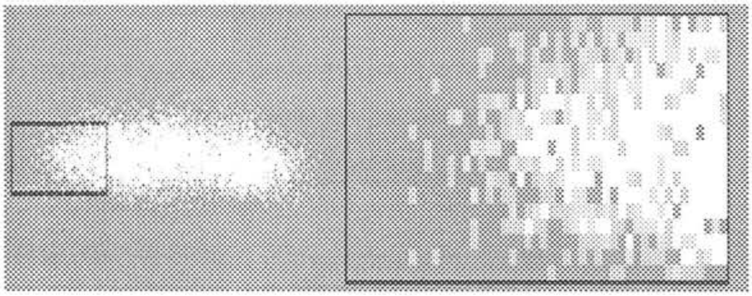


Figure 9.2—using *ARCtist's* zoom facility to examine the 'shade' option on its spraygun. The selected colour is white but some sprayed pixels are various shades of pale grey. Note the oblong shape of the mode 15 pixels

On its coarse setting it sprays the currently selected colour like the spraygun in *Paint*. On its default 'shade' setting it sprays shades of the current colour (Figure 9.2), reading the colour at present on the screen and choosing the most appropriate shade; this gives gentler and more controllable effects than the coarse setting; in fact it acts almost like anti-aliasing, giving the effect of a spray that is finer than mode 15 can manage. The size of the spray area can be adjusted and a thoughtful provision is a 'masking tape' option which allows you to protect precious details from the effects of the spray.

The Draw tool is like the brush tool in *Paint*. By default this tool sets not individual pixels, but quite a large circular area. There is a choice of shapes, including user-definable ones, and an infinitely variable range of sizes down to single-pixel level. So you can change individual pixels or flood huge areas of screen just as you please. There is also a choice between straightforward colours or patterns. A dozen predefined patterns are supplied—you have to load them from disc—and you can grab areas off the *ARCtist* screen to use as patterns or create them in *Paint*—a further benefit of using standard sprite file format.

The line tool and all the outline tools—triangle, rectangle, polygon, circle, ellipse and curve—all draw lines one pixel wide in the currently selected colour. All are straightforward to use and depend on clicking Select in two or three spots. The circle is created as in *Draw* by clicking in the centre and drawing out the circumference until correctly sized. The polygon begins exactly like the circle, but when correctly sized you convert it to a three-, four-, five-, six-, seven- or eight-sided figure by rotating a handle from the centre. The ellipse tool works like that in *Draw* but after setting its major and minor axes on the second click you can rotate it and fix its orientation with the third click. The curves begin as straight lines but can be pulled away from the straight into arcs.

The text facility in *ARCtist* is remarkably sophisticated. Its own outline font system, quite independent of the Acorn system, allows it to produce perfectly shaped text at any size and in any colour or having a pattern in place of a fill colour. Examples of the three fonts provided, Plantin, Flash and Broadway, are shown in Figure 9.3. The characters supplied are upper- and lower-case letters, numerals, full stop and comma.

The fill option determines whether solid operations like drawing and writing text are carried out in a colour or a pattern. Having selected fill, clicking anywhere in the screen will locally replace the colour where you clicked

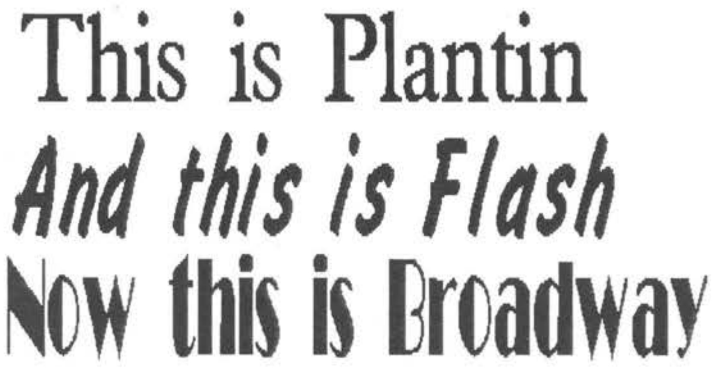

This is Plantin
And this is Flash
Now this is Broadway

Figure 9.3–non-Acorn outline fonts in action in *ARCtist*

with the current colour or pattern. This is useful as it allows you to draw polygons with one outline colour and fill them with a different colour. It also allows you to grab a portion of the screen and use it as a pattern for fill purposes.

The tools option provides a comprehensive collection of editing facilities including both coarse and fine copying and moving of parts of the screen—the area concerned need not even be rectangular—rotate (called 'spin') an area, and rescale an area, the x- and y-axes being separately scaled.

Some of *ARCtist's* most sophisticated facilities are found in the Effects menu. Quad Map, for example, allows you to define a rectangular area of the sprite and subjects it to the same kind of transformation that we have come to associate with mould facilities in vector graphics. You can drag the four corners to new positions and create perspective and other effects (Figure 9.4). Edge Detect works in a user defined area and searches for the borders between coloured areas, highlighting them in the currently selected colour; everything else is changed to the current

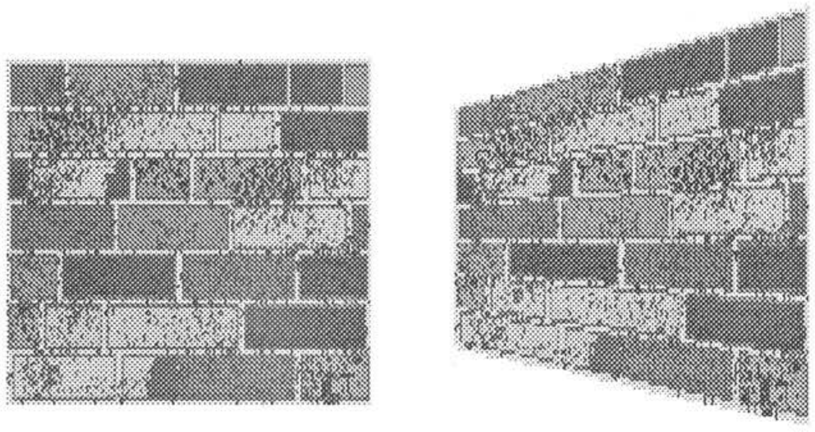


Figure 9.4—sophisticated distortion is provided by *ARCtist's* Quad Map effect

background colour. Trace Colour draws an outline in the currently selected colour around any coloured area. There are two Trace Sprite routines which also trace outlines, but they plot a named sprite rather than in a colour. A Blur effect plots darker pixels around shapes and makes them look blurred. Clean Up is almost the reverse of the spraygun and removes isolated pixels from a defined area. Anti-Alias is similar to blur but uses a wider range of colours; as in *ArtWorks* and the plotting of Acorn outline font characters it removes 'step' effects by introducing pixels of intermediate shades. There are three versions of a colour swap routine which handle tints in various ways. A HalfTone facility creates dither patterns; it is a colour swap in which only alternate pixels are swapped.

ARCtist has a whole submenu devoted to sprite handling. It can handle up to 30 named sprites which are stored in sprite files in standard RISC OS format. Facilities are provided to copy a rectangular or irregular shape as a named sprite (the irregular shape uses a mask) and to display a named sprite anywhere on the screen with scaling and rotation options. Utilities are provided to rename and delete sprites. There is even a simple sprite animator. Effectively this plots named sprites on the screen at preset locations and intervals, incorporates flight paths to gradually move sprites around and offers loop facilities for repeated action. It does not allow any background graphics.

The System option offers miscellaneous facilities not available elsewhere. One clears the screen to the currently selected colour, wiping out your current painting. Another sends the file to the printer, assuming of course that the appropriate printer driver is installed. The final option returns you to the desktop. Your artwork is preserved and you can return to it by clicking select on the *ARCtist* icon.

All in all *ARCtist* offers some very sophisticated facilities, some of which you might be excused for thinking not possible in pixel graphics. Its shortcomings lie chiefly in three areas. Frequently I found myself hankering for a grid or some means of measuring distances. Secondly, now that RAM expansion is readily available for all the ARM machines and many users can enjoy the more consistent high resolution of modes 21 and 28, it seems a pity to be restricted to mode 15. Moreover, on an A5000 the aspect ratio of mode 15 is not right, a circle being reproduced as an ellipse rather wider than high. Thirdly, its filing provisions are limited. Although you can run the main application from a hard disc, all loads and saves must use floppy drive 0 and named directories designated for pictures, sprites and patterns. So you cannot do a direct save to or load from a hard disc or a RAMdisc. But if you are willing to live with these limitations, *ARCtist* will help make the most of your talent.

Art20

Art20 by Australian programmer Raymond Keefe represents a different approach to computer art. Available from Tekoa Graphics, *Art20* was originally intended to provide sophisticated art facilities, but operating in mode 20. This mode offers very high resolution (640 × 512 pixels) and is available even on 1 Mbyte machines to owners of multi-sync monitors. It is, however, limited to 16 colours, although the limitations are to some extent overcome by ready access to dithering.

As the program developed, however, it was extended to use any desktop mode. If you choose a mode other than that currently in use, the mode will be changed to that mode when you enter the Paint screen and will be returned to the original mode when you leave it.

Like *ARCtist* its pictures are assumed to be full screen size, but portions of screens can be saved as sprites. It runs in two modes: a fully RISC OS compliant desktop mode used for previewing and filing operations and a Painting mode used for painting, editing and image processing which takes the whole screen. Subject to sufficient memory it will multi-task happily with other RISC OS software. The art files it generates are standard RISC OS sprite files, fully compatible with *Paint* and all other applications that use sprite files. It does have some file types of its own, however, used for storing shades, patterns and fills.

The desktop mode is entered by dropping a sprite file on to the application icon, by clicking Select on the application icon or by selecting New screen from the icon bar menu. The latter two options lead to a dialogue box which prompts for the required screen mode and opens a plain window. Dropping a full-screen size sprite file on to the icon leads to a dialogue box which asks whether you wish the sprite to be considered as a screen or a sprite. Dropping a smaller-than-screen sprite on to the icon causes the sprite to be added to the sprite library. *Art20* maintains a library of sprites which can be used to build up pictures. If a screen is loaded, it appears in the window. If the colours are arbitrary, drag the current palette into the window. Clicking menu within the window leads to a menu (Figure 9.5) consisting mainly of file save options. The other options display information about the current picture (its mode and size), Paint enters the painting mode and Kill screen is self explanatory; it kills the screen immediately, whether or not your work has been saved.

All painting and design work takes place in the painting mode entered by clicking on the Paint option in the desktop window. This opens a window which is initially empty (unless you have loaded a file) and which fills the whole screen. Clicking Menu brings up the main menu of 11 items, most of which lead to further menus. Incidentally, the Adjust button acts as an undo

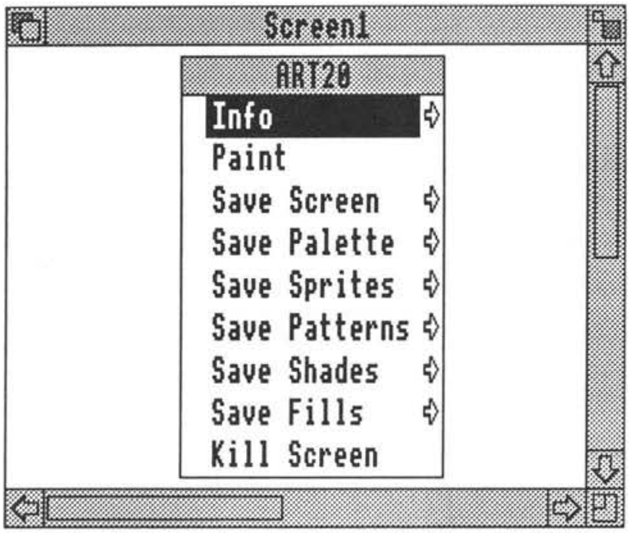


Figure 9.5—the Desktop window and its menu in *Art20*

button and usually reverses the effect of the last operation, a very valuable feature.

The Palette option concerns fills generally and not just palette colours. Besides access to a palette of the 2, 4, 16 or 256 colours available in the selected screen mode, a further palette offers 16 shades, *i.e.* dither patterns, which are user definable and can use up to 8 colours (Figure 9.6). Also available are patterns such as bricks, with a facility to design your own if none of those offered is suitable. There is a facility to set the current background colour. A wide choice of plot style includes NOT AND and NOT OR, useful for special effects. A colour pick option allows you to select the colour under the pointer from anywhere in the artwork. The 256-colour palette has the colours arranged in a very logical sequence of shades; a smaller palette is also available if the full-size one obscures too much of the painting.

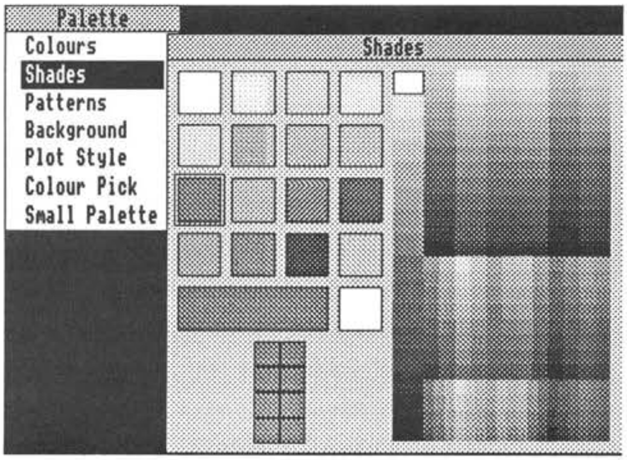


Figure 9.6—the shades window in *Art20* in a 256-colour mode. The current 16 dither patterns are shown to the left and the 256 palette colours to the right. The currently selected 'shade' has the structure shown bottom left; it can be edited as required, allowing the user access to thousands of different shades

The pen option offers a vast range of pens, brushes and other tools. The choice is point (*i.e.* single pixel), dot (*i.e.* circular brush), ellipse, triangle, diamond, square, rectangle, tall brush (*i.e.* tall rectangle), wide brush (*i.e.* wide rectangle) and fast and slow spray. All (except the point option) are offered in infinitely variable sizes.

The line option offers a choice of single or joined lines, continuous or dotted patterns and, a feature rarely found in pixel graphics, infinitely adjustable line width.

The shape option offers a choice of circle, arc, segment, sector, ellipse, triangle, square, rectangle, parallelogram and polygon (3 to 12 sides). All are produced with one-pixel outlines, but a fill option makes them solid. Two thoughtful provisions are a rotate option which allows the finished shape to be rotated and a move option which allows the finished shape to be moved around until correctly located.

The flood option provides a wide range of features. Simplest is a direct colour replacement whereby the area in which the pointer is located is changed to the currently selected colour or dither pattern or pattern. But there is also a wide choice of graduated fills adjustable in gradation direction, choice of colours and rate of change. There is even a radial fill. There are also options to force a named sprite into an area.

Text facilities, as in *Paint*, are limited to the system font. But it is scalable in both axes and may be in the full range of colours.

The area option leads to a diverse range of options concerning areas of the artwork. These allow an area defined by the user (the usual box drag routine) to be filled with a colour or pattern, copied, moved, scaled, reflected, rotated, distorted into a circle or semicircle and subjected to colour swaps. Three fascinating features are soften, sharpen and mosaic. Soften uses a Gaussian filter to reduce the amount of detail in the area by changing isolated pixels and adjacent pixels to their average colour, giving a blurring or smearing effect. It is ideal for enhancing the grainy images sometimes obtained from video digitisers. Sharpen uses the same type of filter to introduce subtle changes to the colours of pixels to highlight detail. Mosaic converts the affected area to a pattern of 'super-pixels' of user-defined size, making them the colour of the original pixel nearest the centre of the area covered. You have probably seen a similar effect used on television sometimes to conceal the identity of an interviewee. These effects are illustrated in Figure 9.

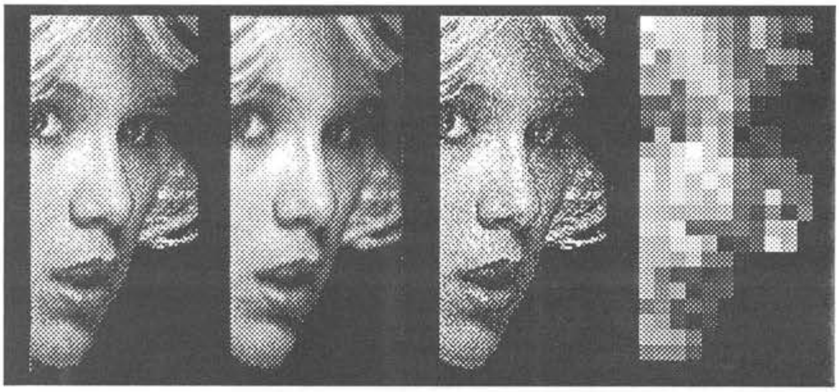


Figure 9.7—some image processing facilities in *Art20.* Left to right: the original image, the image after softening, the image after sharpening and the image subjected to a 12 × 12 pixel mosaic

The sprite option provides a host of operations on sprites. Like *ARCtist, Art20* supports a user sprite area. Sprites can be added to it at any time by dragging them on to the application icon. Sprites can be added to a picture even if there is a difference in screen mode. The original palette colours will be reproduced as closely as possible, but pixel shape differences are not taken into account. Thus a mode 15 sprite introduced to a mode 21 screen will appear to be squashed into half its original height, but you can always re-scale it. Similarly, rectangular areas of a picture can be grabbed and designated as sprites. While the sprite option is selected, moving the pointer around the screen drags the currently selected sprite and clicking Select plots it at the pointer position. The Sprite menu provides all the expected facilities for scaling, reflecting and deleting sprites. The Combine menu usefully allows interesting ways of plotting sprites on the screen such as plotting alternate columns or rows; this produces a 'ghost' effect, since the detail beneath will still be partially visible (Figure 9.8). An Add feature adds two screens to produce the best ghost effect.

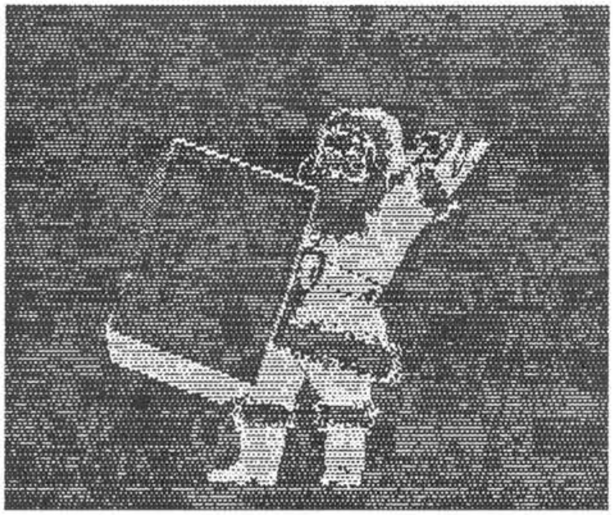


Figure 9.8—the spirit of Christmas past? A ghostly Santa Claus appears through a brick wall, courtesy of the Combine facilities in *Art20*. In this example alternate rows have been plotted

The Pixel Edit option provides the only zoom facility in *Art20*, all other options being performed at normal size. Zoom factors are up to 16:1.

The mouse option offers a choice of mouse speeds. The default in *Art20* is 1 which means that the pointer moves very slowly indeed.

One major facility was missing from early versions of *Art20*—printing. But since the resulting artwork is saved as a standard sprite file, it can be imported into *Paint* or *Draw* for printing, or indeed the file icon can be dropped straight on to a printer driver icon.

With its vast range of facilities, flexibility of screen mode and use of standard sprite files, *Art20* could be said to be a true alternative to *Paint* itself. Its principal limitations are that it cannot create or edit sprites larger than screens. Also I should have been grateful for a visible grid or other measuring system (although there is a grid lock). Anyone who works with pixel graphics will find this software very valuable.

Revelation 2

Longman Logotron's *Revelation 2* cleverly combines remarkable sophistication with commendable simplicity. Developed from the earlier *Revelation*, it is now supplemented by *Revelation Imagepro* which has greatly enhanced image processing facilities, although *Revelation 2* itself is by no means lacking in this area. Like *Arctist* and *Art20*, *Revelation 2* creates standard Acorn sprite files, compatible with *Paint* and DTP packages. Like *Art20* it will operate in any desktop screen mode, but some of its facilities are available only in 256-colour modes. Users are advised not to change screen mode while work is in progress, but mode changes are nevertheless permitted, *Revelation 2* issuing a warning that a degradation in picture quality and facilities available may result.

There is some new terminology to master. Clicking Select on the application icon calls up the Page size dialogue box shown in Figure 9.9; pictures are called pages. The default page size is

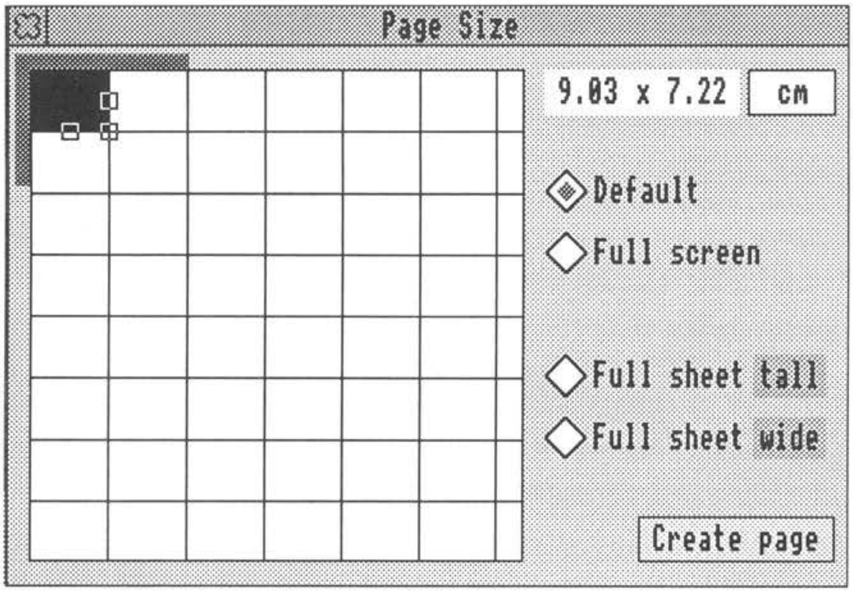


Figure 9.9—the Page size dialogue box in *Revelation 2*

9.03 × 7.22 cm which is approximately a quarter of a screen. Alternatively you can request a full-screen-size page or you can define your own page size by dragging the edges of the page area or clicking in the appropriate squares of the page definition grid. If you have a printer driver loaded or have another page window open, other ready-defined page sizes will be offered.

Revelation 2 works in a choice of two operating modes, special and RISC OS; both allow multi-tasking with other applications (if there is enough memory). In practice, the differences are subtle. In its special mode the main menu is given its own window which can be moved about or sent back, but not closed (shown in Figure 9.10). Submenus can similarly be 'torn off' and left permanently open. In the desktop mode the main menu is accessed when needed by clicking Menu with the pointer in a page window

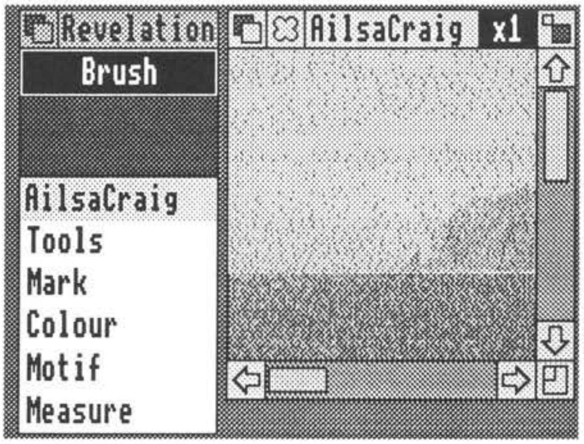


Figure 9.10—in its Special mode the main menu in *Revelation 2* is permanently available in its own separate window

and this and the submenus are closed automatically after use. The page window follows all the familiar RISC OS desktop conventions. A thoughtful touch is the provision for left-handed users which reverses the effects of the two outer buttons of the mouse, so that Select operations are easily accomplished with the index finger of the left hand.

Subject to the availability of memory, many pages may be open simultaneously and a New view facility allows you to open more than one window on to the same page.

Clicking Select on the Tools item in the main menu calls up a graphical display of tools not unlike that in *Paint*; it is shown in Figure 9.11. The currently selected tool is indicated by a boundary box.

Five of the tools put colour on the page. The distinction between the brush and the crayon is subtle; the crayon creates continuous lines and

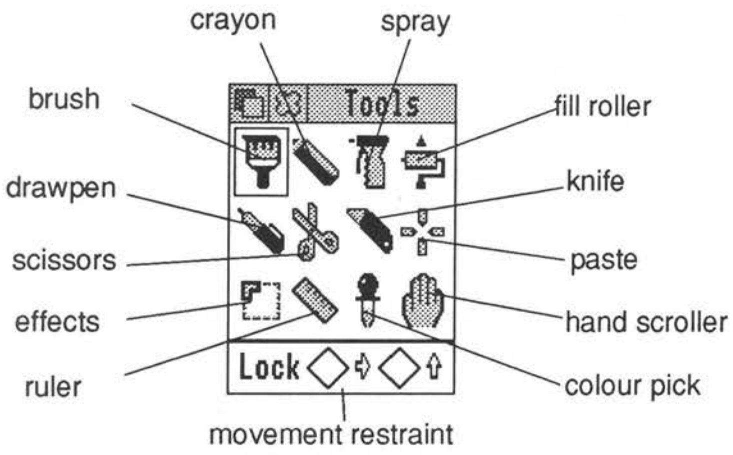


Figure 9.11—the tools in *Revelation 2*

the brush creates individual 'blobs' in the currently selected colour. If you drag the brush slowly, however, its blobs become a continuous line like that from the crayon. The spraygun creates a random cloud of pixels whose density can be controlled by varying the time you hold down Select or by adjusting a Density Slider. Incidentally all three tools are affected by the current selection in the Mark menu. This determines the size and shape of the blob deposited by the brush, the area sprayed by the spraycan and the width of the line made by the crayon. There is a choice of six square marks (the smallest being a single pixel), seven circular ones and six user-definable ones. Usefully the movement of these tools is affected by the direction constraint lock, which will limit movement to vertical or horizontal strokes.

The roller tool is particularly versatile. It is concerned with the filling of areas, that is shapes having a continuous boundary of differently coloured pixels. When this tool is selected, the direction lock is changed to a display which cycles through Normal fill, Fill to Boundary and Gradation as you click Select. Normal fill is a standard flood fill which fills the area containing the pointer with the currently selected colour. Fill to Boundary works in the same way, but only recognises boundaries whose pixels are the currently selected colour; other boundaries are ignored. Gradation is only available in 256-colour modes. It produces linear or circular graded fills and a Randomise facility allows the steps in the gradation to be blended into each other so that it appears infinitely smooth. The colour sequence in the gradation can also be edited. A Change all facility allows the roller tool to replace every pixel on the page of the colour beneath the pointer with the currently selected colour.

The drawpen provides a range of line drawing facilities including elastic straight lines and curves and a diversity of geometrical figures including squares, rectangles, triangles, parallelograms, circles, ellipses and regular polygons from 3 to 16 sides. Shapes are hollow by default, but by clicking on them again they become solid. The drawpen is unaffected by the Mark menu, but has its own line width slider allowing a choice of line width from thinnest possible to 64 OS units. There is also a choice of three line end and joint styles.

The scissors tool allows rectangular portions of the page to be cut and copied. The knife tool works in the same way but allows irregularly shaped areas to be cut. The area selected using scissors or knife is stored as the current 'motif' and can be viewed by clicking on the Motifs option of the main menu. You can copy the motif back into the page using the Paste tool. In this way you can build up multiple copies of part of the page.

The Paste facility itself offers a wide range of options. Blend allows underlying colours to show through the motif being pasted. Flip mirrors allow the motif to be flipped vertically and horizontally. Rotation allows it to be rotated and a lock facility constrains rotation to steps of 5 degrees. Motifs can also be scaled before pasting by dragging a box in the page window.

Motifs are conventional sprites which *Revelation 2* uses in a special way. They may be used to fill an area with a pattern rather than a simple colour. The size of a motif is immaterial. It may be a dither pattern consisting of only four pixels. Alternatively it may be a whole picture in itself. A selection of dither patterns is supplied in a Motifs directory with the package and you can easily supplement this selection with new ones of your own devising. As we have seen, motifs may be pasted—in this operation they are deposited individually at the pointer position. But a motif can also be selected as the current colour and will then be used instead of a single colour for most operations that use the current colour. Most usefully, the roller tool will fill an area with a motif, repeating it as many times as necessary, an operation called, aptly, 'motif tiling'.

The Effects tool offers five image processing operations: Trace, Edge, Blur, Sharpen and Dither. All except Dither offer three levels of sensitivity. All effects are confined to an area in which you drag an elastic box. Trace and Edge both detect the edges of coloured areas and draw them in the current colour, Trace replacing the present colours with a white background and Edge simply adding the boundaries to the existing image. Blur softens and smooths the image and Sharpen enhances contrast. Dither renders the area as a pattern of dots; if a motif is the current colour, Dither confines itself to the colours used in the motif. It can be used to reduce the numbers of colours used in an image in preparation for printing or use as a textile pattern.

The Ruler tool provides a valuable set of measurement facilities, making *Revelation 2* suitable for the kind of design work that would normally be the province of vector graphics packages. The Measure option in the main menu allows you a choice of inches, centimetres or Screen units (OS units). When the Ruler is selected, an x and y co-ordinate display is added to the main menu. You can set a point and then by moving the ruler to another point, you can read the distances and the angle from the first point to the second. You can also select an angle as your datum and measured angles will be relative to this. A measurement display is also available with drawing tools.

The Colour pick tool provides an alternative means of selecting a colour. It will pick colours out of any page or from the colour menu. The hand scroller allows you to move easily around a page that is too large to fit in the screen.

Support for colour in *Revelation 2* is second to none. Besides the palette colours appropriate to the current screen mode and the dither patterns available as motifs, a mixing palette is available in which you can mix your own dither patterns (in any screen mode) and your own blends (in 256-colour modes). The mixing palette is very simple. It is a miniature 'page'—a scribbling pad if you like—in which you can use any of the tools or effects to experiment with colours. You select from it using the Colour pick tool or you can cut a dither pattern for use as a motif.

In 256-colour modes several additional facilities are available. For instance, eight colour washes (white, yellow, cyan, green, magenta, red, blue and black) are available from the colour menu. These are effectively transparent colours. Applying a black wash over an area darkens it and a white wash lightens it, while a yellow wash gives a yellowish cast (as from a tungsten lamp). You may apply a wash over the same area several times to make the effect stronger. Black washes are ideal for simulating the effect of shadows.

A Colour shift dialogue box (shown in Figure 9.12) allows adjustments to the colours in the current page. These changes may be applied either globally (to the whole page) or to the current 'colour set' (that is the current colour selection if a motif or a wash) or to all colours except the current colour set. A colour can be adjusted in various ways including gradually shifting it towards another colour, or adjusting its saturation and darkness. A useful facility is 'posterisation' which gradually reduces the total number of colours on the page. A negative facility which creates a negative image of the page.

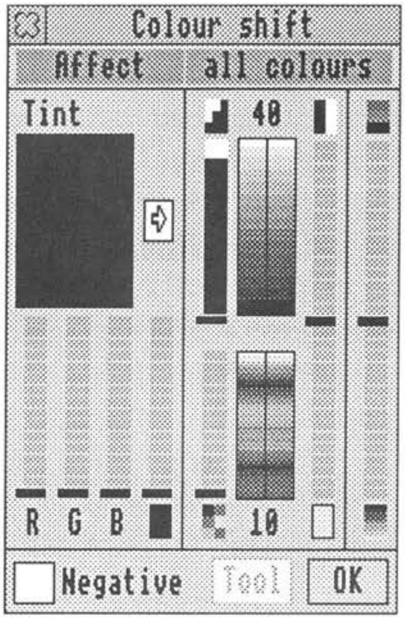


Figure 9.12—the Colour shift dialogue box in *Revelation 2*

When a page is active, its name appears as an item at the head of the main menu. This leads to a selection of operations affecting the current page. These include a valuable Undo facility, zoom (from 1:4 to 8:1) and New view which allows windows on to the same page at different zoom settings. It also accesses the Print and Save facilities.

Two options from the Motif menu warrant mention. The Pattern option leads to a Patterns dialogue box which offers various tiling patterns. This allows you to build up pages using motifs in various orientations and various offsets. This will be valuable for designers of repeating patterns, *e.g.* for textiles and wallpaper.

The other option is Text. Unlike many other art and pixel graphics packages, *Revelation 2* allows you to incorporate text using the outline font system as well as the System font. You should first ensure that the required colour is selected; if a motif or a wash is selected, the colour will be assumed to be black. You enter your wording in a writable field, select the required font name, height and width and the text in the chosen style becomes the current motif and can be used like any other motif or colour.

Certainly, in its versatility and its subtle simplicity *Revelation 2* is streets ahead of the competition. It sets new standards against which future art and pixel graphics packages will be judged.

10

Ready-to-use Sprites

Most of the graphics incorporated into DTP documents and computer-based audio-visual presentations are pixel graphics (sprites). Of these only a few represent the result of painstaking artistic endeavour using *Paint* or other creative software. Most were produced electronically in a matter of seconds from artwork that already existed but in some other format.

If the source material is a photograph, a drawing or some other conventional document, its contents can be converted into digital data using a *scanner*, software converting the data from the scanner into pixel graphics format. If the source material is video, *i.e.* derived from a video camera, videotape, videodisc or a television receiver with video output, it can be converted using a *video digitiser.*

There are two other commonly used sources of ready-made pixel graphics. If the artwork you are creating is for a software manual, catalogue or advertisement, you may wish to include screens generated in the software being described. These may even have originated on a 'foreign' computer type. These too are sprites. Finally, collections of ready-to-use graphics—'clip art'—are sold by some software suppliers for the express purpose of adding interest to computer-generated documentation.

Scanners

Scanners vary widely in size, shape, specification and cost. The simplest are hand-held devices that the operator rolls slowly over the document being scanned. On more sophisticated types, a motor winds the document through the machine, much as it does in a printer. On the most advanced types such as the Epson GT-6000 and GT-8000 the document is placed face down on a window and the machine does the rest, under software control.

All, however, work on the same fundamental principle: as the scanner head moves past the document (or *vice versa*), it is illuminated and the light being reflected from it is repeatedly measured by an array of tiny light-sensitive cells. The output from these cells is converted to a digital signal which is sent to the computer for processing.

All scanners need scanner-driver software and most need a special card, a *podule* or expansion card, which fits into the computer's backplane and into which the lead from the scanner is connected. Software converts the data from the scanner into standard sprite format, although some scanner-driver software offers alternative formats which use compression and so require less storage space. Image processing options may also be available.

Some scanners are strictly monochrome, discriminating only between black and white; colours and intermediate shades are regarded as either black or white, a threshold control being provided. Some scanner-driver software can accommodate intermediate shades by converting them to dither patterns. RISC Developments' *Scavenger*, for instance, has an option that uses clustered dot dithers. In these the dots are made up from a matrix of pixels in which the size of the dot is varied to simulate the shade being reproduced (see Figure 10.1). The dots are arranged diagonally since rectangular patterns are more conspicuous and distracting. This form of dithering is preferable to the dispersed dot dither used on the screen (in which shades are distinguished by varying the density of single-pixel dots) when the material is intended for printing. Indeed the process is very similar to the 'screening' used in the printing industry to convert half-tones such as photographs to a form in which they can be reproduced using black ink. An alternative option antialiases 2-colour sprites by arranging the pixels in groups of 4 × 4 and replacing each such group by a single pixel in a 16-colour mode with a special grey-scale

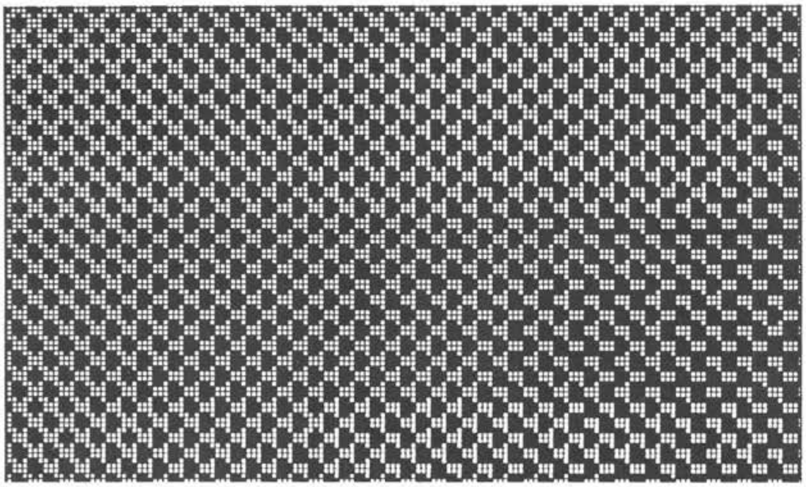


Figure 10.1—screen dump of part of a scanned photograph magnified in *Paint* with the grid switched on to make the individual pixels clearer. The scanner driver has used a clustered dot dither on a 6 × 6 matrix to render the half tones in the image. This particular area fades from about 50% (18 out of 36 pixels black) to about 67% (24 out of 36 pixels black) from left to right.

palette. The resulting sprite, of course, has only one sixteenth the number of pixels as the original scanned image, but since each pixel is represented by 4 bits instead of 1, it requires one quarter of the storage space.

Other scanner systems recognise tints and some recognise colours, offering 2-, 4-, 8-, and even 24-bit colour modes, almost like screen modes in reverse. Some colour scanners need to make three separate passes over the document, one for each primary colour; a filter is changed inside the scanner head before each pass. This of course lengthens the scanning time.

Proimage

Probably the most sophisticated scanning package for the Acorn 32-bit machines is

Proimage from Irlam Instruments Ltd. This includes an Epson GT6000 scanner, leads and software. Optionally a GT8000 can be supplied. Strictly speaking, no interface board is needed with the A5000 computer, the scanner connecting directly to the machine's printer port which is bi-directional. But you will be unable to connect a printer to this port at the same time as the scanner unless you use an additional Centronics switch box. For other computers you will need an interface card which provides its own port for the scanner, leaving the printer port available for other peripherals.

The scanner scans in monochrome or 24-bit colour with user-selectable resolutions from 50 to 600 dots per inch (800 dpi on the GT8000) and a maximum document size of 297 × 216 mm which accommodates both European A4 and US Letter sizes. The usual scanning procedure first makes a preview scan at low resolution and in monochrome. The user drags a box around the area of interest, selects colour or monochrome and required resolution and then starts the scan proper. This takes some time depending on document size and resolution. The image (in full 24-bit colour if colour was selected) is buffered internally to hard disc and subsequently processed to Acorn sprite standard using Floyd-Steinberg error diffusion (described in the next chapter). Colour rendering is excellent; plate 5 is an example of its output.

There is one vital point to remember. At a full 600 dpi and in 24-bit colour an A4 document generates an image of size approximately 100 Mbytes (uncompressed) which clearly cannot be loaded into an existing Acorn computer. For the time being this package is restricted to scanning small areas at higher resolutions or larger areas at lower resolutions. But even at 100 dpi a full-colour image is impressive.

Choice of scanner resolution

Most scanners, like most printers, offer a choice of resolutions. Even the simpler hand-held monochrome scanners frequently offer resolutions of 100, 200, 300 and 400 dpi. The choice of scanner resolution is, however, not so simple as it might appear and there are occasions when you should resist the temptation to scan at the highest available resolution. Ultimately it depends entirely upon what you want to do with the scanned image.

If you only wish to display the scanned image on the monitor screen at its original size, 100 dpi should be quite adequate; little would be gained from scanning at a higher resolution.

If you intend to import the scanned image into a DTP or drawing package destined for printing, the printout will be crisper if you use a higher scanner resolution, but you will almost certainly need to rescale the image in the DTP or drawing package and the scaling ratio is critical. Images from scanners are generally converted to sprites in modes 18 to 21; RISC OS regards the natural size of these sprites as 90 pixels per inch, so that their pixels match the nominal resolution of the monitor screen. Consequently, if you scan an image that is 4 inches wide at 300 dpi and import the resulting sprite, 1200 pixels wide of course, into a DTP or drawing document, you will find

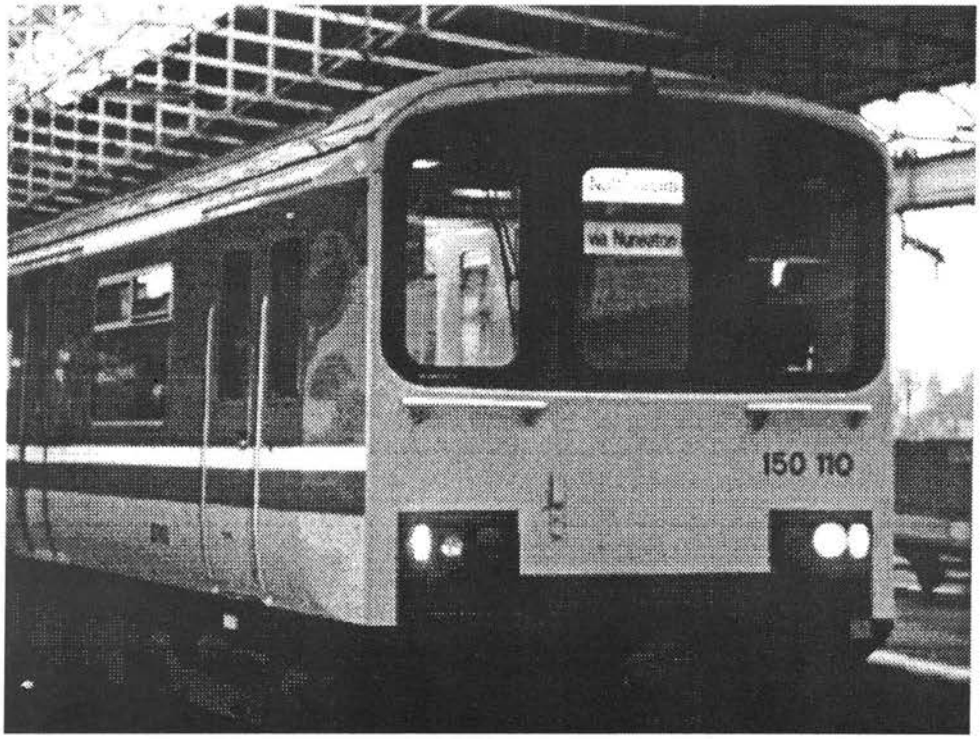


Figure 10.2—a colour photograph scanned at 400 dpi using RISC Developments' *Scavenger* hand scanner. The daily Rugby to Nottingham Sprinter was caught waiting to leave Rugby by the camera of the author's son, Max Amos

that its width is taken to be $13^1/_3$ inches. In other words, on a dimensions-only basis it has been scaled up by a factor of 300:90 and will be too wide to fit on an A4 page. So you will need to use the scaling facilities in your DTP or drawing software to reduce it to a more convenient size. *Note that these scaling facilities only affect the scale at which the graphic is reproduced on the screen or in printouts; the actual number of pixels in the sprite is not affected.*

So, what scaling factor should you use? Let's take the simplest possible option and scale the image back to its original width of 4 inches. The required ratio is of course 90:300, the inverse of that previously applied by the package. You may need to express this ratio in a different format such as 30% in *Ovation* or 0.3 in *Draw.* This rescaling will pose no problems and since it restores the resolution of the image to 300 dpi, it brings the added bonus of a dot density which is used by many printers. If you print the document on a 300 dpi printer, there will be an exact dot-for-dot correlation between the scanned image and the data sent to the printer, so you should obtain a clean, accurate reproduction of the original.

Now you may well have been tempted to scan at the highest resolution available on your scanner, thinking that this will give the best quality. Let's see if this is true. Let's return to the previous example and see what happens when we scan the same original at 400 dpi and then print it on a 300 dpi printer. The sprite obtained will now be 1600 pixels wide and the handling package will regard it initially as $17^{2}/_{3}$ inches wide. To restore it to its original size would need a reduction of 90:400 (or 22.5% or 0.225). With some packages, such as *Ovation*, we should now be in trouble. *Ovation's* scaling facilities only handle percentages in whole numbers, so we should not be able to restore the image to its exact original size. And the 0.5% error when printing out that image rescaled to 22% or 23% would lead to inaccuracies in the printout which may well be visible as Moiré fringes—periodically repeated stripes or streaks across and down the image.

Even if your DTP or drawing package does allow scaling as fractional percentages or by ratios such as 90:400 you still have a problem. You will have scaled the image back to its original dot density of *400* dpi but the printer prints at *300* dpi. This is not an exact correlation; the printer driver will need to interpret each 16 dots of the source image as 9 pixels for the printer, sufficient to introduce fringes, or if not fringes, some loss of definition.

Oddly enough, there is a simple solution. *You* know that your freshly imported sprite was scanned at 400 dpi, but your DTP or drawing software doesn't. It simply regards it as a large 90 dpi sprite. So you could pretend that it had been scanned at 300 dpi and scale it to 30% as before. It will now print out perfectly satisfactorily, but will be one third larger than life. You have effectively 'spaced out' the 400 dpi to 300 dpi.

In the previous examples, if you had intended to use a 360 dpi printer, this would have introduced a new set of problems. 300 dpi does not translate very well to 360 dpi, so a new set of scaling ratios is needed to scale the image to a suitable pitch for the printer. If the image was scanned at 300 dpi, scaling by 90:360 (25% or 0.25) will raise the dot density to 360 dpi and this will give a clean printout on a 360 dpi printer, which will be approximately one sixth smaller than life.

If you use a 600 dpi printer (such as *Laser Direct Hi-Res*) to print out an image scanned at 300 dpi at its original size, you gain no benefit. My experiments with photographs scanned at 300 dpi show that actual-size printouts at 300 and 600 dpi are identical. But if you scale the sprite to 15% forcing the dot resolution to 600 dpi and then print at 600 dpi you obtain a printout that is half the height and width of previous printouts but still contains all the original detail. Reproduction is needle-sharp and very attractive.

In all but the last of the examples considered so far the printout has been at the same or a similar size as the original scanned image. Supposing you want to print your scanned image at half or quarter or double its original size? You must first take account of the 'primary' scaling factor, *i.e.* the scaling factor needed to restore the image to full size or nearly full-size compensating for the printer resolution. With a 300 or 600 dpi printer this is 90% for 100 dpi scans, 45% for 200 dpi scans, 30% for 300 dpi scans, 22.5% for 400 dpi scans and 15% for 600 dpi scans. And you should only rescale your image to simple multiples of that primary scaling factor. Table 10.1 lists some scaling ratios which should give clean printouts with various scanner and printer resolutions. It does not claim to be exhaustive, so feel free to

Table 10.1—Scaling factors for clean printouts of scanned images

	Scanner pitch (dpi)						
Printer Resolution (dpi)	100	200	300	400	600	800	1200
300,600	30% = 1 : 3	15% = 1 : 3	10% = 1 : 3	5% = 2 : 9	5% = 1 : 3	5% = 4 : 9	5% = 2 : 3
	45% = 1 : 2	30% = 2 : 3	15% = 1 : 2	10% = 4 : 9	10% = 2 : 3	10% = 8 : 9	10%= 4 : 3
	60% = 2 : 3	45% = 1 : 1	30% = 1 : 1	15% = 2 : 3	15% = 1 : 1	15% = 4 : 3	15%= 2 : 1
	75% = 5 : 6	60% = 4 : 3	45% = 3 : 2	30% = 4 : 3	20% = 4 : 3	20% =16 : 9	20%= 8 : 3
	90% = 1 : 1	75% = 5 : 3	60% = 2 : 1	45% = 2 : 1	25% = 5 : 3	30% = 8 : 3	
	135% = 3 : 2	90% = 2 : 1	75% = 5 : 2	60% = 8 : 3	30% = 2 : 1	40% =32 : 9	
	180% = 2 : 1	120% = 8 : 3	90% = 3 : 1	90% = 4 : 1	40% = 8 : 3	45% = 4 : 1	
360	25% = 5: 18	25% = 5 : 9	25% = 5 : 6	5% = 2 : 9	5% = 1 : 3	5% = 4 : 9	5% = 2 : 3
	50% = 5 : 9	50% =10 : 9	50% = 5 : 3	25% =10: 9	25% = 5 : 3	25% =20 : 9	25%=10: 3
	75% = 5 : 6	75% = 5 : 3	75% = 5 : 2	50% =20: 9	50% =10: 3		
	100% =10: 9	100% =20: 9					

Note: *the scaling factor is given as a percentage; the accompanying ratio is that of the size of the printed image to the scanned image. For example, an image scanned at 200 dpi, rescaled to 60% and printed at 300 dpi will be 4/3 of its original size.*

These ratios should give clean fringe-free printouts in all circumstances, but they are not necessarily the only ratios that will do so.

experiment. But if you choose an awkward scaling factor such as 13% simply because that makes the image fit the space available, don't be surprised if you end up with a nasty touch of the Moiré fringes. I have sometimes seen manufacturers' literature prepared in-house using DTP equipment in which scanned images have been ruined by Moiré fringes; clearly the operator scaled them to fit without thinking of the possible consequences!

Oddly, for colour scanners you can choose lower resolutions without the deterioration in quality becoming evident. A full-colour scan at 100 dpi looks quite crisp, although a monochrome scan at the same pitch looks distinctly coarse. This is because the colours themselves provide automatic anti-aliasing. It is precisely the same optical illusion that is provided by the use of intermediate greys in on-screen text using the Acorn outline fonts.

And it is just as well that colour scans can use lower resolutions because, as we noted earlier, the combination of colour and high resolutions gobbles up the memory. Table 10.2 gives some guidelines. Clearly, if the sprite is larger than the RAM available in your computer, you will not be able to process it or to include it in documents. You may be able to store it on hard disc; this depends on the facilities offered by your scanner driver software. And even if it is too large for that, you may be able to pass the data coming from the scanner straight through to a colour printer, effectively giving you a colour photocopying facility. This may be useful (and profitable).

Table 10.2—Approximate storage requirements of A4* scanned images in various pitches and colours (in bytes)

pitch (dpi)	*2 colours 1 bit per pixel*	*4 colours 2 bits per pixel*	*16 colours 4 bits per pixel*	*256 colours 8 bits per pixel*	*24-bit*
75	60 K	120 K	240 K	480 K	1.4 M
100	107 K	214 K	430 K	860 K	2.6 M
150	240 K	480 K	960 K	1.9 M	5.8 M
200	430 K	860 K	1.7 M	3.4 M	10.2 M
300	960 K	1.9 M	3.8 M	7.7 M	23 M
400	1.7 M	3.4 M	6.8 M	13.6 M	40.8 M
600	3.8 M	7.7 M	15.7 M	31.5 M	95 M
1200	15.7 M	31.5 M	63 M	126 M	378 M

**A4 is taken as 11 × 8 inches. For A5 divide these figures by 2. For A3 multiply them by 2*

Don't be deterred by the appearance of the scanned image on your monitor screen. What looks hideous on the screen may nevertheless print out perfectly. Sadly the reverse is also true!

Video digitisers

A video digitiser is a card (podule) that plugs into the backplane of the computer. It includes a socket by which it can be connected to a source of video such as a camera, video cassette recorder or video disc system. As with the scanner, software is needed to drive the digitiser. Effectively it takes a 'snapshot' of the video image on its input and converts this to sprite format, most often in mode 15.

You may wonder how a mode offering just 256 rows of pixels (lines) can reproduce a video picture which in Europe is supposed to have 625 lines. The answer is that the 625-line TV picture is like a mythological beast—everyone has heard

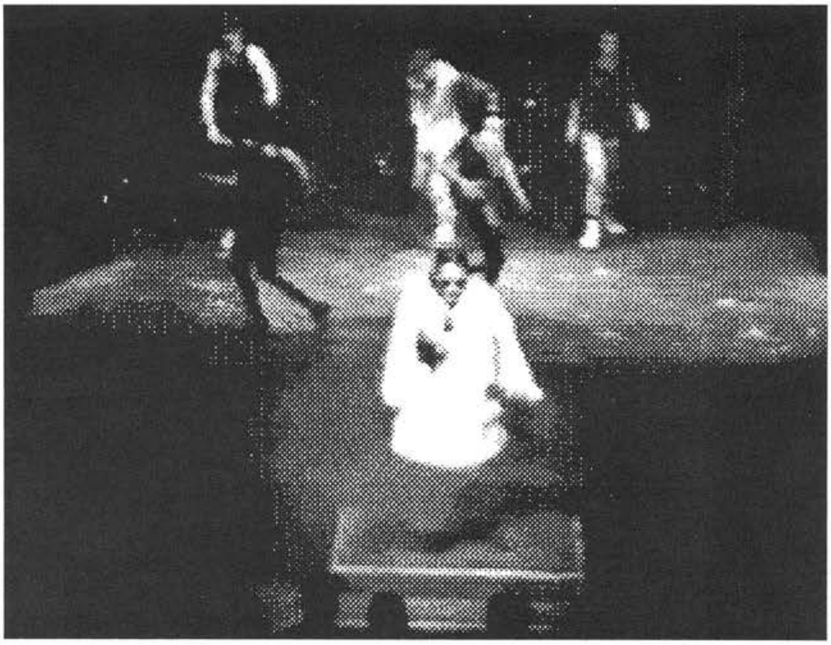


Figure 10.3—an example of a video image (an amateur recording) captured using the Pineapple video digitiser

of it, but it does not really exist. There are 625 lines belonging to each frame, but in order to reduce the flicker on a TV screen a system known as *interlacing* is used. This takes the odd numbered lines and displays them and then, in the next frame, displays the even numbered lines. So the maximum number of lines on a TV screen at any given moment is in fact only 312. But even that is a highly optimistic figure, since not all of these lines are visible. Many at the top of the picture are dedicated to special purposes such as the carrying of teletext and other data. Sometimes, on an incorrectly adjusted TV receiver, you can see the teletext data as a series of ever-changing dots on several lines at the top of the screen. In fact the number of visible lines on a TV picture is around 270, which is not far removed from the 256 rows of pixels in a mode 15 screen. The loss of the remaining few lines is not generally noticeable.

Video digitisers are available at a wide range of prices and with a matching range of facilities. If you wish to digitise standard-format TV or video pictures, you should check before purchasing any particular digitiser that it does capture the full *width* of the picture. Some digitisers produce mode 15 sprites of size 512 × 256 pixels which are of course square; these digitisers omit an appreciable area from one edge of the source picture.

The facilities offered vary widely. The Pineapple Video Digitiser offers some sophisticated hardware facilities including a real-time display of the source and a comprehensive range of support software which allows many image processing options. The digitiser has hardware controls for hue, saturation and value and, although a little practice is needed to set these for optimum results, they do provide a means of compensating for scenes that are dull or lacking in contrast. Each image is initially stored in 256 Kbytes of RAM on the digitiser board. This allows rather more than mode 15's 8 bits per pixel. Captured images can be stored on disc in this higher resolution format and then processed as necessary to give the most satisfactory sprite.

The sprites obtained from video digitisers are likely to be somewhat grainy since the 256 colours offered in mode 15 are a poor substitute for the millions of shades available in analogue video; some digitisers use dither patterns to render shades not available in the palette.

Consequently digitised video may be unsuitable for such purposes as conversion to vector graphics using tracing. But you may find that image processing makes them more suitable.

Video digitising also teaches some salutary lessons about the importance of the fourth dimension, time, in human perception. It sometimes happens that the scene that interested you is not really there; it was an illusion. What caught your attention was the juxtaposition of details in several successive frames, but which were never all present at once so that no one frame can 'tell the whole story'. On the other hand, you will sometimes find fascinating details on individual frames that pass unnoticed when the video is run at normal speed. If you have a video cassette recorder with a good freeze-frame facility, stepping frame by frame through sequences can be an enlightening—and sometimes a disillusioning!—experience.

Imported screens

Screens and parts of screens, as we have seen, are sprites. For the Acorn 32-bit machines there are a number of public domain/shareware utilities that allow screens to be 'grabbed' out of applications. Many of these take the form of a 'module'. When loaded into the computer's module area these interrupt the processor at regular intervals and test for the presence of a certain combination of keypresses. When the combination is detected, the current screen is saved as a sprite file in the currently selected directory. You can subsequently load this file into *Paint* or other software for editing or processing. For example, Figure 1.9 was culled from Fourth Dimension's flight simulator game *Chocks Away Special Missions* using just such a module.

Obviously you must ensure that the currently selected directory is on a disc that has sufficient room for the sprite file. One problem is that some applications reset the current directory. For example, you may set the current directory to be the root directory on your hard disc where you have megabytes to spare, but the games application out of which you wish to save that scintillating screen resets the current directory to floppy drive 0 from which the game is run on a protected floppy disc that is filled to the last byte! Also some applications—in order to run faster—disable interrupts so that this kind of screen grabbing module cannot be used.

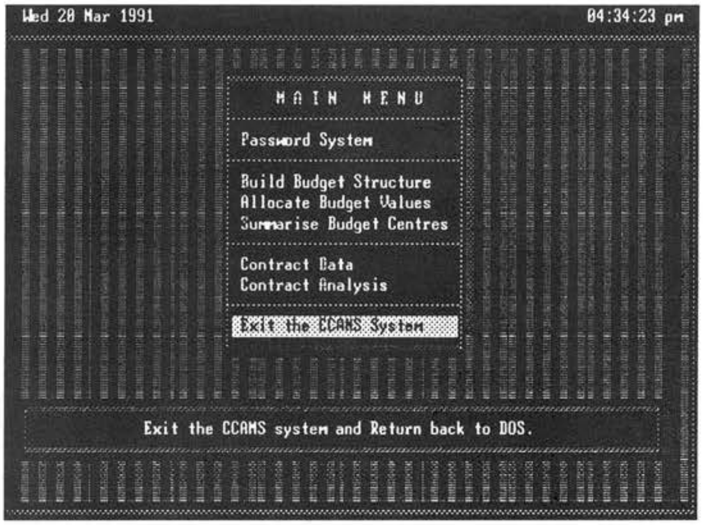
You may wish to import screens from some other type of computer. This is perfectly feasible, but you will need an application such as *ChangeFSI* (described in the next chapter) to convert the 'alien' screen format to Acorn sprite format. You will also need a screen grabbing utility to save the screens on the source computer. For example, I once used my Archimedes to prepare a user manual for a suite of IBM PC applications software. There is a utility called *Pzazz* for IBM PC-compatible machines that works in a similar way to the ARM modules described, but with rather greater sophistication. It multi-tasks with the application software and on detecting the necessary keypress it interrupts the application and displays its own menu offering a choice of image formats and destinations for the saved graphics. When you have made your selection, the application screen is restored and saved and the original application is resumed.

I used *Pzazz* to store screens in TIFF (Tag Image File Format) on 720K MS DOS discs. These were then loaded into the Archimedes A440 running *PCDir*, *ChangeFSI* and *Ovation* simultaneously. Each TIFF file in turn was dragged out of the *PCDir* directory viewer on to the *ChangeFSI* icon where in a few minutes the original PC screen appeared in an Acorn window with an option to save it as an Acorn sprite—and as such it was imported into *Ovation*. Although I had not used *Pzazz* or *ChangeFSI* before, everything worked perfectly at the first attempt. An example screen is shown in Figure 10.4.

Clip Art

Clip art consists of collections of ready-made graphics, often following a particular theme such as animals, plants, transport, music or sports, which are sold by software suppliers primarily for the purpose of decorating magazines, newsletters and other documents prepared on the user's computer system. Some clip art is in sprite format, but increasingly clip art for the ARM machines consists of vector graphics (*Draw* path objects) because this is easier to scale and in general more graphics can be fitted on to a disc.

Although intended for use in documents, clip art could be put to other use such as in animations or games programmes. But if these were distributed to third parties, you might be infringing copyright—see below.



```
Wed 28 Mar 1991                                04:34:23 pm

                    M A I N   M E N U
                    Password System
                    Build Budget Structure
                    Allocate Budget Values
                    Summarise Budget Centres
                    Contract Data
                    Contract Analysis
                    Exit the CCAMS System

          Exit the CCAMS system and Return back to DOS.
```

Figure 10.4—an example of a screen grabbed from a PC application and converted to a RISC OS sprite using *ChangeFSI*

Copyright—a warning

This chapter has described the copying of documentary material using scanners and of video material using video digitisers. Either form of copying could in some circumstances be illegal and, if you use either, you should first ascertain that you are not infringing copyright law.

The general principle of that law is as follows. You are allowed to copy up to 10 per cent of a document for your own personal use and presumably this implicit permission extends to electronic copies made using scanners and stored on floppy or hard disc for your own personal use. Technically it is illegal to record a TV broadcast even for your own personal use, although I have never heard of anyone being prosecuted for such personal use. Your copying of individual frames from TV broadcasts, either live or recorded, even for your own personal use, is therefore also illegal. If you distribute copies of copyright material on floppy disc or as hard-copy printouts, you are breaking copyright law unless you have obtained the consent of the copyright owner. The situation regarding the copying of screens from software is a little uncertain, but you should certainly not distribute such copies to third parties without the prior consent of the copyright owner.

Clip art falls into a slightly different category. It is art sold in the form of software for the express purpose of embellishing the purchaser's publications. Your purchase of clip art software necessarily includes permission to reproduce it *ad lib* on paper in documents that you create using your computer system. But you cannot copy the software itself without the permission of the copyright owner.

11 Image Processing

We met the term *image processing* in the two previous chapters. It is a generic term for several types of operation that are applicable to pixel graphics, but not to vector graphics. The operations are generally applied to the whole image, although some may occasionally be applied to only a part of an image.

The following types of operation are generally regarded as image processing:

(i) conversion of image format, *e.g.* changing PC screens to RISC OS sprites.

(ii) scaling of images, *e.g.* converting a mode 15 (640 × 256 pixel) image to a mode 21 (640 × 512 pixel) image or a half-size mode 21 (320 × 256 pixel) image. In these scaling operations *the number of pixels in the image is changed.* This stands in contrast with the scaling of sprites imported into DTP and drawing packages, which changes the size of the image on the screen or in the printout, but leaves the number of pixels unchanged.

(iii) changing the screen mode of a sprite.

(iv) changing all pixels of a certain colour to another colour, *e.g.* darkening or lightening the image or expanding its dynamic range (darkening the dark parts and lightening the light parts).

(v) more complex pixel colour changes which take into account or even affect the colour of adjacent pixels, *e.g.* in image sharpening and softening.

The simplest example of image processing is the substitution of one colour by another. *Paint* allows this, offering the choice of *global* substitution, that is, every pixel of the affected colour is changed throughout the sprite, or *local* substitution, which changes only pixels of the affected colour that are contiguous with the pixel beneath the pointer.

Some public domain applications offer a more sophisticated version of this process. Applicable usually only to 256-colour sprites, the application examines each pixel in turn and multiplies its red, blue and green content by conversion factors set up on a set of sliders not unlike those in the system palette except that they are usually scaled in percentages. Since you may wish to increase the content of a certain colour, the sliders generally allow settings over 100%—they may go as high as 1000%. Clearly this kind of application can be used to lighten dark scenes or darken pale scenes or to remove, say, a reddish cast that is spoiling an otherwise acceptable image.

In Chapter 9 we encountered edge detection in *ARCtist* and in *Art20* some sophisticated operations that could soften or sharpen an image or reduce its resolution giving a 'mosaic' effect (Figure 9.8). In these operations the changes that are made to pixels must take into account the status of adjacent pixels; this is true of many image processing operations.

Of course, there should normally be no reason to apply any such processing to a sprite that was composed pixel by pixel in *Paint* or an art package or, for that matter, to a sprite which was 'screen-grabbed' from a vector graphics package. If the artist was competent, he or she should have achieved the very effect that was intended. Image processing is sometimes applied to such sprites, however, to prepare them for printing.

But, as we saw in the last chapter, many sprites are created electronically. They are the output from scanners or video digitisers used to convert into digital format images that originated on paper or, ultimately, from a video camera. Now unlike an artist who can idealise, depicting the scene as it would appear under perfect conditions, a (video) camera records the real world with all its imperfections. It may be true that 'the camera never lies' (although it can be tricked), but the camera does not see as the human eye sees. It records, but unlike the human faculty of sight, it does not interpret what it records. For this reason its output sometimes fails to do justice to the scene portrayed—although sometimes it can also reveal detail that the living photographer failed to notice! When a photograph or video frame is converted into digital format, this adds an additional set of limitations to an image that was probably already less than perfect. Consequently, sprites from these sources often benefit from image processing.

Let me make one fact plain from the outset. Image processing is not a panacea for all evils. As in all branches of data processing, the rule 'rubbish in—rubbish out' applies to image processing. If the vital information is absent from the source material, no amount of image processing will restore it. Consequently, if you attempt to process a sprite obtained from a video digitiser connected to a worn out video cassette recorder playing back a recording made on a different machine of a television broadcast that was itself of less than broadcast quality (such as a vintage film or an outside broadcast in poor conditions), you must not be surprised if you fail to obtain any improvement.

On the other hand, in some sprites vital data are present but eclipsed by other detail. And sometimes, if the process parameters are correctly set, you will obtain a worthwhile improvement. As in so much of life, it takes practice. After a while, you learn to recognise the kind of condition that can be improved and you will know instinctively the steps to take. Until then, the golden rule is: don't be afraid to experiment. Don't be disappointed by your failures; learn from them.

There is one imperative: *never process your only copy of an image; always keep a copy of the original in a safe place.* Most image processing operations are irreversible. So if the chosen operation ruins the image—and sometimes this happens—there will probably be no way in which you can restore it to its former condition. But if you have a copy of the original, clearly you can discard the failure and try some other processing operation.

There are several image processing packages available, but the one that you are most likely to encounter is Roger Wilson's *ChangeFSI.*

ChangeFSI

Although *ChangeFSI* originated at Acorn Computers, it is widely available from public domain and shareware sources. It is also distributed with some scanners, such as those supplied by Irlam Instruments, as a means of converting their output to RISC OS sprite format. Its name is derived from Change (self explanatory), the names of R. W. Floyd and L. Steinberg, who in 1975 devised a process called 'error diffusion' which the application uses to improve the rendering of images, and Integer since the application performs all its arithmetic in 32-bit fixed-point integers.

Floyd and Steinberg error diffusion is used principally to improve the rendering of images having 24-bit colour information when they are converted to 256-colour sprites. Clearly only 256 of the 16,777,216 original colours can be reproduced exactly; the other 16,776,960 are simulated using dither patterns. When the colour is applied over an appreciable area, the dither pattern is carefully contrived so that the red, green and blue content of the original colour is conserved. Generally two colours from the current palette are deployed in a pseudo-random pattern and any error in the weight of the overall red, green and blue content is compensated for by adjusting the colours of individual neighbouring pixels. *ChangeFSI* works from right to left in one row and then left to right in the next to minimise the likelihood of error diffusion resulting in conspicuous repeating patterns.

ChangeFSI's icon suggests that its purpose is to convert any pixel graphic material to a RISC OS sprite. And for many that is its main use. Drop a file of 'alien' pixel graphics on to its icon and after a while (the length of time depending on the size and format of the source file) up pops a RISC OS window containing the image sought. Clicking Menu over the image produces a save icon which allows you to save the image as a RISC OS sprite. But quite apart from this very useful facility, *ChangeFSI* contains a valuable selection of image processing operations.

ChangeFSI is loaded in the normal way, its icon appearing on the icon bar. (Unusually for an ARM application it can also be run entirely from the operating system command line.) Clicking select on the icon does nothing—you start it off by dragging the source file on to the icon. But first you will almost certainly wish to set up some parameters. You do this by clicking menu over the icon; it brings up a menu of seven options.

Most important are Scaling, Processing and Output which lead to dialogue boxes described later. Fast gives faster operation. Save choices saves the options you have selected from the dialogue boxes, allowing you to reuse processing combinations that worked well on further material from the same source. Info and Quit have their usual functions.

Input material

ChangeFSI (version 0.82) accepts as input the following RISC OS file types:

(i) standard RISC OS sprite files

(ii) the ArVis format from Video Electronics Ltd (file type FF9)

(iii) the Pineapple colour digitiser format from Pineapple Software

(iv) the Watford Video Digitiser 'picture' format from Watford Electronics Ltd (file type DFA)

(v) the *ProArtisan* compressed picture file format from Clares (file type DE2)

(vi) two different TimeStep satellite image formats (file types 300 and 7A0)

(vii) CCIR601 (file type 601)

(viii) the AIM and Wild Vision V10 format (file type 004)

(ix) the Wild Vision V9 format

(x) the *Translator* Clear format (file type 690)

(xi) the Atari ST 'Degas' format (file type 691)

ChangeFSI recognises and attempts to convert the following image types which are most likely to have originated in other computer systems. They are recognised by their contents and structure rather than by directory information such as file type. Some of these formats such as TIFF and GIF are very widely used for image transfer between computers; others such as the RT format are used by certain pixel graphics applications.

(i) the Millipede Prisma format (also used by CadSoft)

(ii) the Aldus/MicroSoft TIFF format (also filetype FF0 is assigned to TIFF)

(iii) the CompuServe GIF format

(iv) the Electronic Arts IFF ILBM format

(v) the MicroSoft Windows 3 .BMP format

(vi) the Digital Research GEM .IMG format

(vii) the PC .PIC format

(viii) the MacPaint format

(ix) the ZSoft .PCX format

(x) the RIX Softworks ColoRIX format

(xi) the Sun pixrect format

(xii) the UNIX rle format

(xiii) the TrueVision Targa format

(xiv) the 'Flexible Image Transport System' (FITS) format

(xv) the PC .DSP format

(xvi) the QRT .raw format

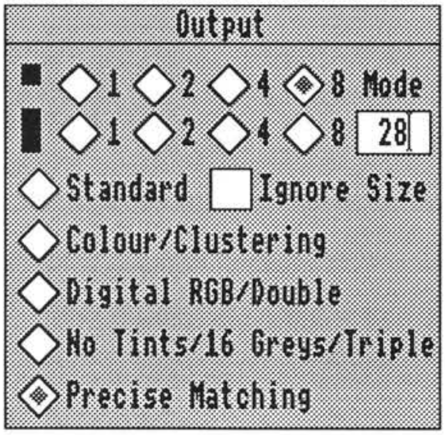
(xvii) the MTV pic. format

(xviii) the RT image format

More recent versions of *ChangeFSI* may recognise other image formats. As new formats are discovered, their data are added to the application which is highly accessible, being written in a mixture of BASIC and Assembler.

Output options

Output options are set up in the output dialogue box shown in Figure 11.1. Output is always a RISC OS sprite (although there is an option to output AIM/Wild Vision V10 format images) and *ChangeFSI* defaults to the current screen mode, although any desktop mode can be selected. (Clearly this provides an elegant means of changing sprites from one screen mode to another.) The radio buttons at the top of the dialogue box provide a quick way of selecting screen mode by pixel shape and number of bits per pixel, useful for folk who can never remember which mode is which.


Output

■ 1 2 4 [✓ 8] Mode

▮ 1 2 4 8 28

Standard ☐ Ignore Size

Colour/Clustering

Digital RGB/Double

No Tints/16 Greys/Triple

[✓ Precise Matching]

Figure 11.1—the output dialogue box in *ChangeFSI*

The other options in the box control various output parameters.

Ignore Size disables all pixel size information in the program, so that both source and output pixels are considered to be square.

Standard gives crude colour output. In a 16- or

256-colour mode you should use Precise Matching if you want full colour rendering.

Colour/Clustering gives coloured output in 4- and 16-colour modes. In 2-colour modes it converts output to a clustered dither using a 4-pixel matrix.

Digital RGB/Double in 16-colour modes gives a digital RGB output (i.e. red, green, blue, yellow, cyan, magenta, white, black). In 2-colour modes it converts output to a clustered dither using a 2-pixel matrix.

No Tints/16 Greys/Triple in 256-colour modes ignores the tints, using just the base 64 colours (see Appendix 3 for more details). In 16-colour modes it uses a 16 grey levels palette. In 2-colour modes it converts output to a clustered dither using a 3-pixel matrix.

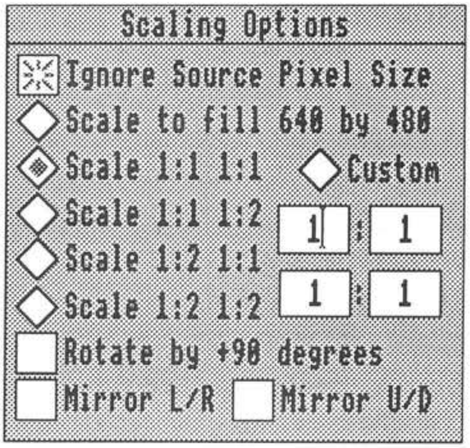
Precise Matching in 16 and 256-colour modes uses the default palettes and gives the closest possible colour matching.

The clustered dither options on 2-colour sprites are particularly useful in preparing material from scanners for printing.

Scaling options

ChangeFSI offers a comprehensive range of scaling operations, set up in a scaling dialogue box (Figure 11.2). Note that these scaling operations change the number of pixels present, not just the size of the image on the screen or in the printout.

Ignore Source Pixel Size is an option that can be set when the source material is from a format that specifies the pixel size. Some files get their


Scaling Options

[✓] Ignore Source Pixel Size

◇ Scale to fill 640 by 480

◆ Scale 1:1 1:1 ◇ Custom

◇ Scale 1:1 1:2 1 : 1

◇ Scale 1:2 1:1

◇ Scale 1:2 1:2 1 : 1

☐ Rotate by +90 degrees

☐ Mirror L/R ☐ Mirror U/D

Figure 11.2—the scaling options dialogue box in *ChangeFSI*

pixel size wrong! If this option is set, all pixels will be assumed to be the same size as a mode 18 pixel (*i.e.* 2 OS units or 1/90 inch per side).

Scale to fill x by y scales the source material to fill a screen, the current screen mode being assumed. x and y are replaced by the width and height of the current screen mode in pixels.

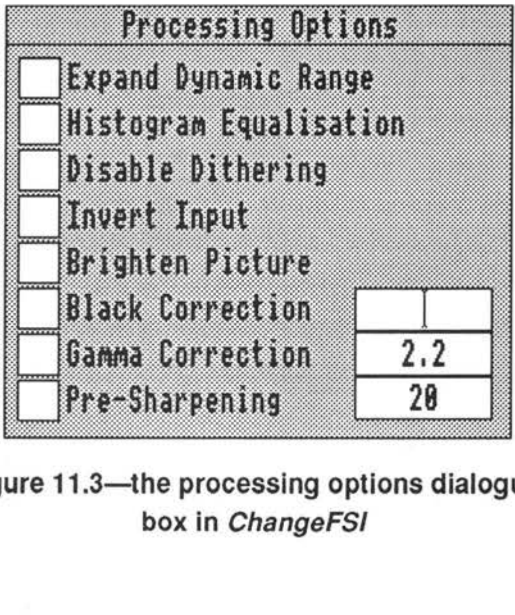
In the four standard scalings that are offered the first figure relates to the horizontal (x) axis and the second the vertical (y) axis. Thus the scalings offered are no change, either axis halved or both axes halved. Halving refers to an actual halving of the number of pixels, not simply the size of the image on the screen. You are quite likely to wish to scale an image downwards in this way. Halving the number of pixels forces *ChangeFSI* to merge pairs of adjacent pixels into single pixels and this is one way of eliminating dither patterns. You may wish to eliminate a dither pattern if, for instance, you wish to apply sharpening. If you use sharpening on a dithered image, you will sharpen the dither itself which will result in an appalling mess.

Besides the four pre-set scalings, you may enter any scaling factor you wish. You must click in the Custom box and set up your ratios in the four boxes beneath it. The upper pair relate to the horizontal (x) axis and the lower pair to the vertical (y) axis. Either box will accommodate figures up to three digits, so you can enter ratios such as 30:400 if you wish.

The last three options may be set individually or in any combination. They rotate the image anticlockwise through 90 degrees, mirror it left-to-right or reflect it vertically.

Processing options

The processing dialogue box in version 0.82 is shown in Figure 11.3. The eight processes are quite independent and theoretically all could be selected and applied, although you are unlikely to want both to expand the dynamic range *and*



Processing Options

- ☐ Expand Dynamic Range
- ☐ Histogram Equalisation
- ☐ Disable Dithering
- ☐ Invert Input
- ☐ Brighten Picture
- ☐ Black Correction
- ☐ Gamma Correction 2.2
- ☐ Pre-Sharpening 20

Figure 11.3—the processing options dialogue box in *ChangeFSI*

to brighten the picture.

Expand Dynamic Range causes *ChangeFSI* to survey the image and see how much of the dynamic range (from black to white) it uses. If it does not use the full range, the range is expanded to fill it—it is rather like the contrast control on a monitor or TV receiver. Thus a dark picture (containing no white) will be lightened and a light one (containing no black) will be darkened, while a picture that contains neither black nor white will be selectively darkened and lightened. Beware of using this option on images from animated sequences unless all the images in the sequence have the same dynamic range. Otherwise you may end up with frames whose brightness varies in a rather disconcerting manner.

Histogram Equalisation is a rather brutal process, which forces the widest use of the colours available. Usually it makes pictures inferior, but it can sometimes recover a picture that is otherwise useless or reveal detail locked in a small part of the input scale.

Disable Dithering prevents *ChangeFSI* from using dithering in the output sprite. Normally it uses dithering to represent colours not available in the palette. Moreover, as we have seen, it is able to use both dispersed and clustered dot dithers. This option will force it to use only the colours in the palette.

Invert Input as its name suggests inverts the colours. Although rarely useful in colour images, monochrome images frequently need inverting.

Brighten Picture lightens all colours by adding 1 to each RGB level. Obviously, it is unwise to use it in a picture that is already pale.

Black Correction is a facility for improving the appearance of images printed on 'write black' printers (such as dot matrix and ink jet types). The parameter tells *ChangeFSI* how much larger an inked dot is than a pixel in the sprite; the default setting is 32 which represents a printed dot having an area 50 per cent greater than a pixel. Use of this facility generally makes screen images look pale. Of course, to print pictures which have received black correction you must scale the sprite so that its pixels match the printer pitch, *i.e.* by 90:300 for a 300 dpi printer.

Gamma Correction is a useful option for output intended for the screen. Monitors vary widely in the brightness with which they display colours; many make dim colours appear unnaturally dark. Gamma correction can be used to compensate for such discrepancies. The default value of 2.2, the standard in the TV industry, lifts dim colours appreciably and will often improve pictures from video digitisers or from other computers having more bits per pixel.

Pre-Sharpening is a powerful tool. It enhances the edges of objects in the picture, usually resulting in a more acceptable output. A value must be entered: the lower the number the greater the level of sharpening. A value of 8 performs edge detection. Values in the range 10 to 20 are normally used to sharpen images, while values in the range 20 to 30 can be used to compensate for the fuzziness introduced by dithering. It is worthwhile experimenting to find the best setting for each individual source image.

The image menu

When *ChangeFSI* has finished processing the image, the finished sprite is displayed in a standard RISC OS window. Clicking Menu in this window calls up a menu of five options.

Sprite info gives useful information about the sprite: its name (*ChangeFSI* allocates a name based on the processing options used), screen mode, size in pixels and file size.

Source info gives similar information about the source file that was processed.

Range info gives information about the dynamic range if the Expand Dynamic Range operation was used.

Zoom leads to a standard zoom facility exactly like that in *Paint*. The maximum zoom settings are 999:1 and 1:999 which should be adequate for most purposes.

Save sprite leads to a standard save dialogue box allowing the sprite to be saved to disc or into another application.

12 Tracing—Pixel to Vector Graphics

One of the disadvantages of pixel graphics is that they cannot be scaled easily. If you increase a sprite's size (by, say, doubling the number of pixels on each axis), each pixel in the original is represented by a block of four pixels in the enlarged version. So the pixel structure becomes more obvious in the jagged edges of oblique lines and curves. And if you reduce its size, detail is lost as pixels are merged (Figure 12.1).

Vector graphics, in contrast, can be magnified or reduced indefinitely without incurring any such penalties. Consequently, a means of converting sprites to vector graphics offers distinct attractions. Applications that do this are called *tracing packages*.

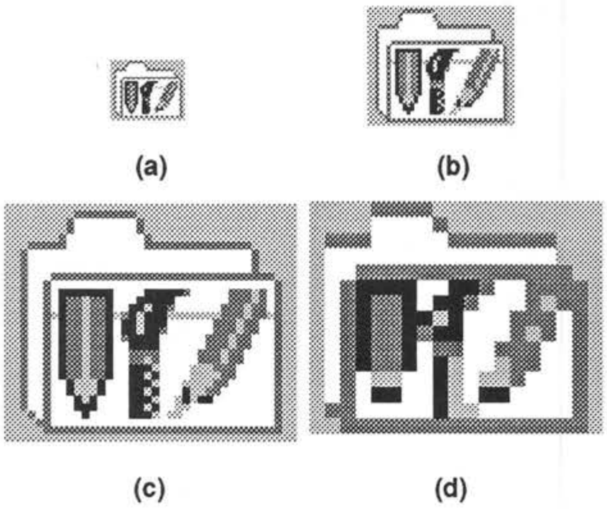


Figure 12.1—the high-resolution version of the RISC OS 3 Apps icon at (a) natural size, (b) magnified by 2, (c) magnified by 4, to show how the pixel structure becomes increasingly obvious, and (d) reduced to half size and then remagnified to show the loss of detail

Tracing programs use one of the image processing operations considered earlier, *edge detection*, to find the edge of each coloured area in the sprite in turn. Having identified the edge, they then create a *Draw*-style closed path to fit the outline.

There are, of course, limitations to the process. Tracing programs always work in terms of filled shapes bounded by thin lines. So, a line of uniform thickness in the sprite will nevertheless be translated into a shape, even though it would have been more logical and economical to represent it by a single line.

Some tracing programs attempt to reproduce the colours of the original sprite; others do not, leaving you with the tedious task of selecting objects and changing their path styles one by one if colour is important to you. Although you can easily scale the resulting *Draw* image, you may find that other editing tasks are even more complex than they would have been on the original sprite. A sprite is two-dimensional; what you see is all there is. A *Draw* file, in contrast, can contain one interesting detail on top of another, so that some objects are hidden behind others, the data stack itself providing a kind of rudimentary third dimension. Although a sprite contains no depth information whatever, tracing programs generally assume that any coloured area that is completely surrounded by another coloured area is in front of it. So if there is one coloured area in the sprite that surrounds everything else—the background colour in many sprites being the most obvious example—that will become the first (rearmost) object in the *Draw* file. In general this supposition does sort the objects into quite a sensible order.

Not all sprites are suitable for conversion. Those that contain a multiplicity of isolated pixels, for example in dither patterns, will not produce satisfactory *Draw* files. This, sadly, rules out the creation of *Draw* files from most images obtained from video digitisers—unless you have the patience to subject the image to a great deal of softening and loss of detail first. Indeed, the sprites that give the most satisfactory conversions are those that most resemble vector graphics in the first place, in that they have large expanses of uniform colour.

Before you purchase a tracing package, it is worthwhile to stop and consider carefully the task you wish it to perform. If the sprite you wish to trace is fairly simple, it could be that your best bet is to import it into a drawing package and draw around it by hand. (The stickback chair in Figure 1.4 was produced by hand tracing around a scanned image.)

There are several tracing packages for the ARM machines. These include Midnight Graphics' *Tracer* and Iota's *Outliner*. Probably the best known is David Pilling's *Trace* which is modestly priced and, unlike some of its competitors, produces full-colour drawings.

Trace

David Pilling's *Trace* is supplied on the same disc as *D2Font* which converts *Draw* files to Acorn-format outline fonts (described in Chapter 3). This immediately suggests one interesting application for a tracing program: to convert scanned images of a typeface to *Draw* files which *D2Font* can then process into an outline font. In practice the resulting fonts usually require some further attention to make them usable.

Trace is absolute simplicity to use. Clicking on the icon opens two standard RISC OS windows labelled Sprite and Draw (Figure 12.2). If you have not used the application since you installed it, the two windows will of course be empty. You drag the sprite you wish to convert into the Sprite window, click Menu, select Choices, adjust the error setting or accept the value offered (more on this later) and click on Trace to start the operation. The time taken depends on the complexity of the sprite. On an A5000 three or four scanned characters take just a few seconds. It is fascinating to watch the Draw file appearing in the Draw window.

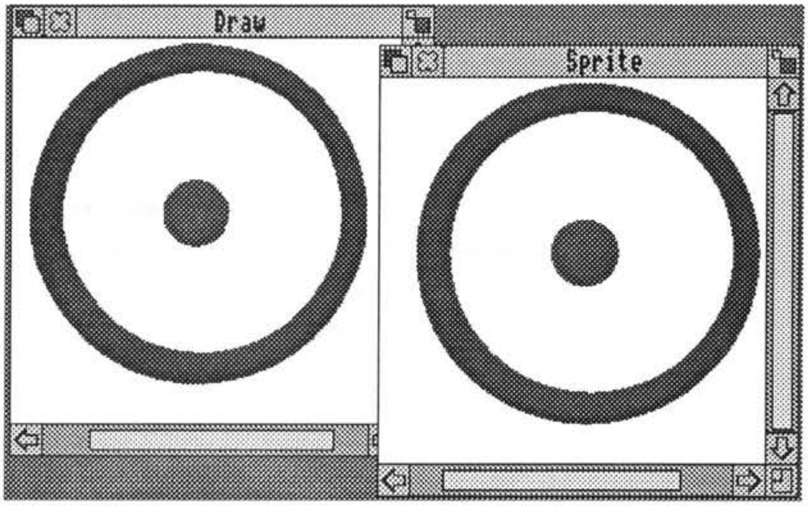


Figure 12.2—the two windows in *Trace* when it has finished a tracing operation. That the two graphics look almost identical is a tribute to its abilities. Now for an abstruse philosophical question: Does a roundal consist of a hollow blue ring surrounding a small red dot or is it a small red dot over a much larger white dot over an even larger blue dot? For the answer see the next Figure.

Each window has an identical menu. Options allow you to save either the original sprite or the *Draw* file produced from it, to initiate tracing, to zoom in and out (zoom ranges from 999:1 to 1:999 and is applied simultaneously to both windows), to choose the error setting and save your choice and to show vital information about both sprite and Draw file.

The error setting in *Trace* determines the accuracy with which the resultant drawing

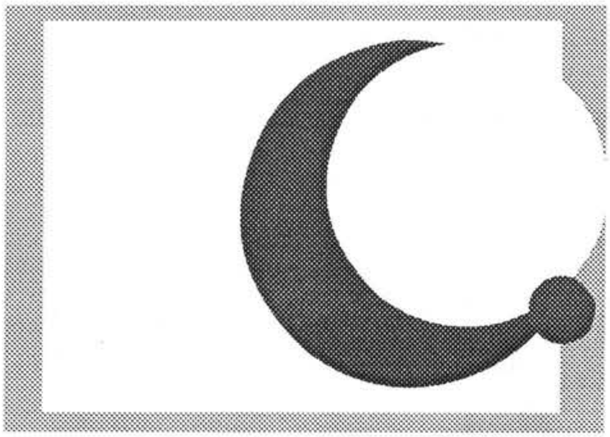


Figure 12.3—the *Draw* file from Figure 12.2 disassembled to show its structure; a pale grey background has been added. It clearly reveals the answer to the abstruse philosophical question posed by the previous Figure: neither, it's a small red dot on a larger white dot on a larger blue dot on an even larger white rectangle!

reproduces the edges of complex coloured areas in the sprite being traced. If you argue that it should reproduce them *exactly,* you are in trouble. For if it did that, it would then reproduce the outlines of the individual pixels of which the sprite's design was composed. And if you subsequently scaled the drawing upwards, the jagged edges would become every bit as obvious as they would if you had magnified the sprite itself. It was largely to eliminate this very effect that tracing packages were developed! So, ironically, to gain any real benefit from the tracing process, a degree of error must be built into it—the paths created must follow an *average* edge rather than the *exact* edge. It is worth experimenting with different error settings. The default setting of 0.80 pixel gives excellent results with many sprites, but you may find with certain types of sprite a different setting is preferable (see Figure 12.4). Also, if you are converting characters that you intend to incorporate into an outline font using *D2Font* or an equivalent application, it helps to make the sprites as big as possible. The larger they are, the more accurate will be the renderings from *Trace* and the less work you will have to put into subsequent tidying by hand.

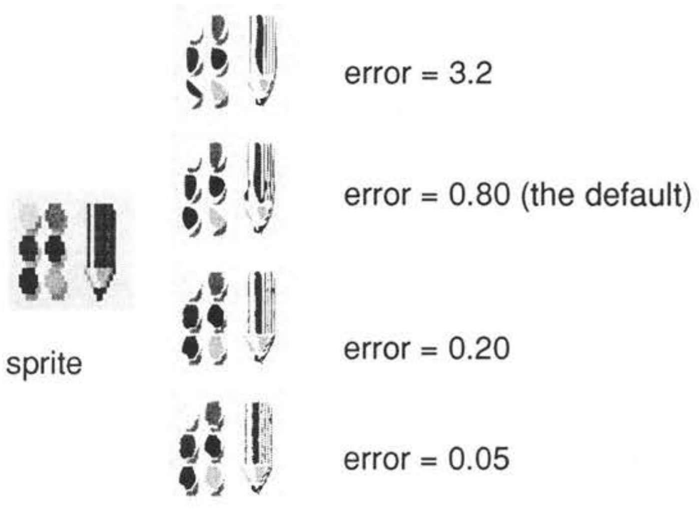


Figure 12.4—effect of various error settings in *Trace*. The higher the setting, the more freedom the package has to interpret the course of the edge of the coloured area being copied. Which setting you prefer is to some extent a matter of taste

13 Ray Tracing

Ray tracing is almost as old as computer graphics itself. The philosophy of the technique is as follows: if you have a mathematical model of an object or a group of objects and you know precisely how it is illuminated, then given sufficient computing power you can reconstruct the image of it that would be seen by an observer or captured by a camera trained on it at any point. Working through the image pixel by pixel, the computer retraces the history of the light falling on each part of the image and traces it back to all possible sources, taking into account reflection off surfaces and refraction within transparent objects. For this reason the system needs details of not only the colour but also the reflective and refractive properties of the objects in the scene. The resulting images—in pixel graphics format of course—are usually startlingly realistic. 'Photographic' is not an apt description; they are more than that. They represent idealised photography—photography that is not subject to the limitations of lenses, film and chemical processes. You will find some examples in the colour plate section of this book.

There are two essential prerequisites for ray tracing, apart of course from sufficient computing power (and time!). You need a ray tracing algorithm, *i.e.* a ray tracing program, and a mathematical model of the scene, including the full reflective properties of all surfaces.

Given these demanding prerequisites, you might be excused for thinking that not many would undertake the daunting prospect of writing ray tracing software for the ARM machines. In fact many such packages are available and one of them, Clare's original *Render Bender*, dates back to the earliest days of RISC OS. This chapter examines four contrasting packages, one of which is not true ray tracing at all, but offers similar facilities. While most packages could theoretically run in any desktop screen mode, the very nature of ray tracing presupposes a 256-colour mode (13, 15, 21 or 28).

ArcLight

ArcLight is one of that family of linked applications from Ace Computing whose members *Euclid* and *Mogul* we met in Chapter 6. It was natural that this family should include a ray tracing member because the other members of the family provide some of the prerequisites.

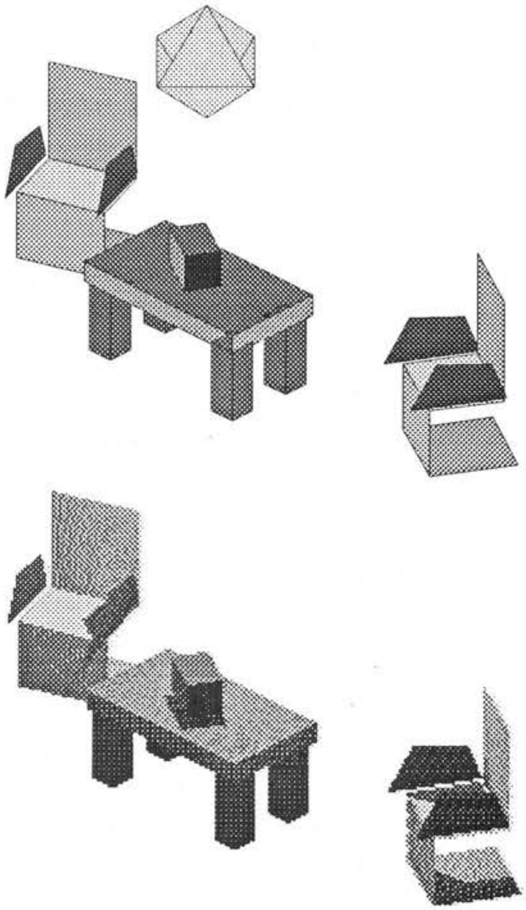
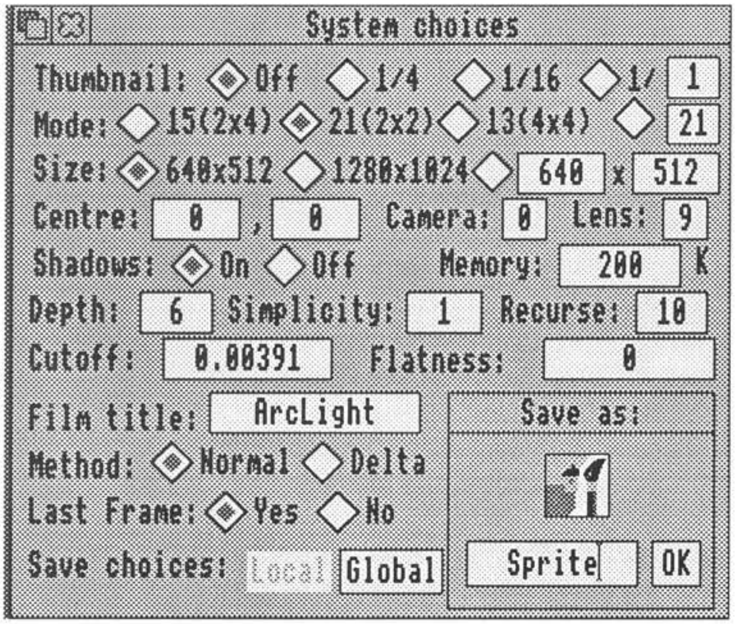


Figure 13.1—the use of *ArcLight* to ray trace a *Euclid* drawing. Top: the *Euclid* file as rendered in *Draw* (distortion in the table is a consequence of the 3D-to-2D conversion). Bottom: the same scene rendered by ArcLight, analysing over 45,000 rays. Note the shadows and texture effects

One prerequisite for ray tracing is a mathematical model of the three-dimensional scene to be ray-traced. Now a *Euclid* file contains just such a three-dimensional model. Indeed *Euclid* itself contains the elements of a ray tracing system in that it allows you to place one or more cameras trained on the scene and, with some limitations, to define how it is lit. Each camera can then show its particular 'view' of the scene with proper perspective and with lighting and shading effects. *ArcLight* uses the *Euclid* file as its mathematical model and takes the process a stage further.

To use *ArcLight* is very simple. You drag the *Euclid* file on to the *ArcLight* icon. This calls up the dialogue box shown in Figure 13.2 in which you set up various options. Finally, drag the sprite file icon into a directory display to start the process. The sprite image builds up in a window on screen and is finally saved in the destination directory. Another dialogue box shown in Figure 13.3 provides interesting statistical information


System choices

Thumbnail: [✓ Off] 1/4 1/16 1/ 1

Mode: 15(2x4) [✓ 21(2x2)] 13(4x4) 21

Size: [✓ 640x512] 1280x1024 640 x 512

Centre: 0 , 0 Camera: 0 Lens: 9

Shadows: [✓ On] Off Memory: 200 K

Depth: 6 Simplicity: 1 Recurse: 10

Cutoff: 0.00391 Flatness: 0

Film title: ArcLight

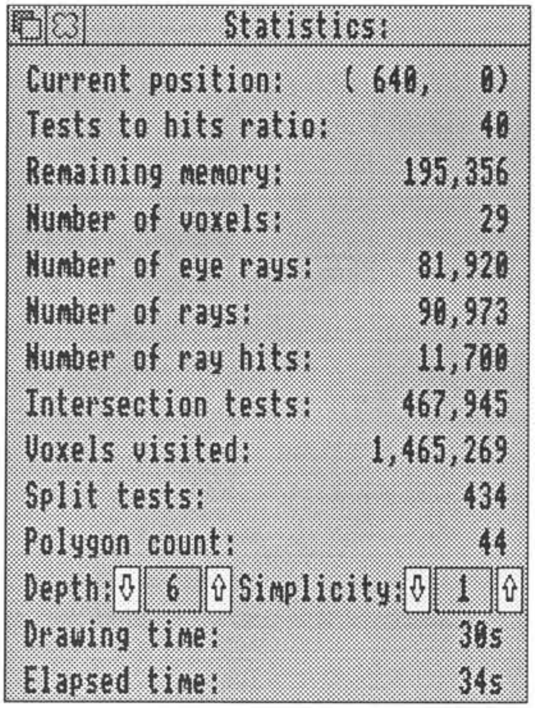
Method: [✓ Normal] Delta

Last Frame: [✓ Yes] No

Save choices: Local Global

Save as: Sprite OK

Figure 13.2—the choices dialogue box in *ArcLight*. Note that the picture size is in OS units, not pixels. The option of saving local or global choices is valuable



```
Current position:     ( 640,   0)
Tests to hits ratio:          40
Remaining memory:        195,356
Number of voxels:             29
Number of eye rays:       81,920
Number of rays:           90,973
Number of ray hits:       11,700
Intersection tests:      467,945
Voxels visited:        1,465,269
Split tests:                 434
Polygon count:                44
Depth: 6  Simplicity: 1
Drawing time:                30s
Elapsed time:                34s
```

Figure 13.3—The statistics dialogue box in *ArcLight* after a typical ray tracing operation

on the process, including the time taken. *ArcLight* is much faster than most ray tracing software; with an ARM3 many pictures take under a minute.

ArcLight will also ray-trace whole animations made using *Mogul*. You create a directory for the new film and place in it a *Euclid* file named *Picture* and a *Mogul* sequence file named *Sequence*. Then you drag the entire directory on to the *ArcLight* icon. The process can take some time depending on the number of frames in the film and the screen mode in use. But the process operates in the background so, subject to memory availability, you can use your computer for other tasks at the same time.

To speed up what would otherwise be a tediously slow operation *ArcLight* examines the scene and if it contains more than a certain number of polygons, it divides the scene into 8 areas called *voxels*. Each of these is examined and, if it is too complex, it is subdivided again and so on, the maximum number of such successive subdivisions being set as the *depth*. Voxels are too complex for handling if the number of polygons inside them exceeds an arbitrary value known as the *simplicity*. Both depth and simplicity are user-definable. In general, increasing the depth and reducing the simplicity speed up rendering, but also increase the likelihood of the machine running out of memory.

Another facility for speeding the ray tracing is the Fast option. This suspends normal RISC OS multitasking and allows *ArcLight* to take over the whole of the computer's processing power. Normally multitasking operation is resumed if any mouse button is clicked.

Render Bender II

The original *Render Bender* from Clares probably did more than any other package to stimulate interest in the Archimedes' potential for graphics and, especially, animation in the early days. Who wasn't captivated by the line of toy soldiers marching endlessly out of the castle, the swinging balls in the cradle or the flying saucer buzzing the suburban street?

Today *Render Bender II* is a suite of four main and several subsidiary applications. *Illuminator* is a 3D vector graphics application similar to

Euclid. You use this to create the mathematical model that will be ray traced. *Render Bender* itself is the ray tracing application which requires an *Illuminator* file as its input. It converts this to a ray traced image. *Animator* takes a sequence of pixel-graphics images and converts them to an animated sequence stored in a compressed format. Although the source images are usually *Render Bender II* images, it will work with sprites from a variety of sources. *Converter* converts compressed screens to sprites and also allows you to extract individual screens from animations for editing.

Illuminator provides a graphical front end for the *Render Bender* suite; prior to its introduction, the mathematical model of the scene had to be constructed as a text file which was less user friendly. *Illuminator's* main screen (Figure 13.4) recalls that of *Draw* or, even more, *Euclid.* The top six icons in the tool panel produce primitives: sphere, cube, pyramid, disc, cone and

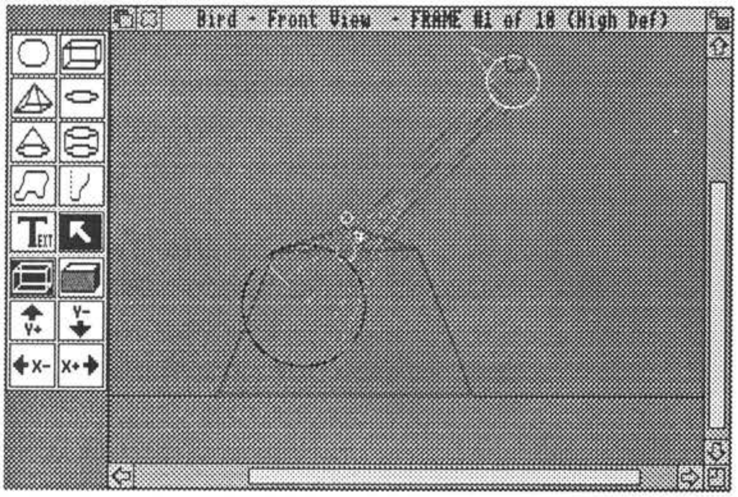


Figure 13.4—A typical (wire-frame mode) screen in *Illuminator*

tube. The next two icons, outline and sweep, allow for the design of more complex objects. A text facility allows the entry of text objects using three special fonts provided. The select facility works exactly as in *Draw* or *Euclid*, selected objects being distinguished by their grey outlines. The next pair of icons provide a toggle switch, toggling between wire-frame and solid rendering within the drawing. The bottom four tools are indicators only, showing which pair of axes are represented in the current screen view.

The main menu allows files to be saved and duplicates the tools in providing entry to the Select menu and for the creation of objects. From the Select submenu you can delete, copy, move, group and ungroup objects—and change their colours. From the Enter submenu three types of light can be added to the scene; these are not available from the tools; each scene must contain between one and four lights. The Display submenu is concerned with various aspects of the display including an optional background, grid and zoom facilities. Views determines which view of the scene is displayed; the default being the front. Clicking left, for instance, opens an additional window showing the scene from the left. Clicking top opens yet another window showing the scene from above. Examples are shown in Figure 13.5. The Function menu allows you to move, rescale and rotate objects. A Show option provides very fine control of object placing. The Animate option allows you to choose how many frames the animation will include and which frame is being edited. Most powerful is an Enter Formula option which

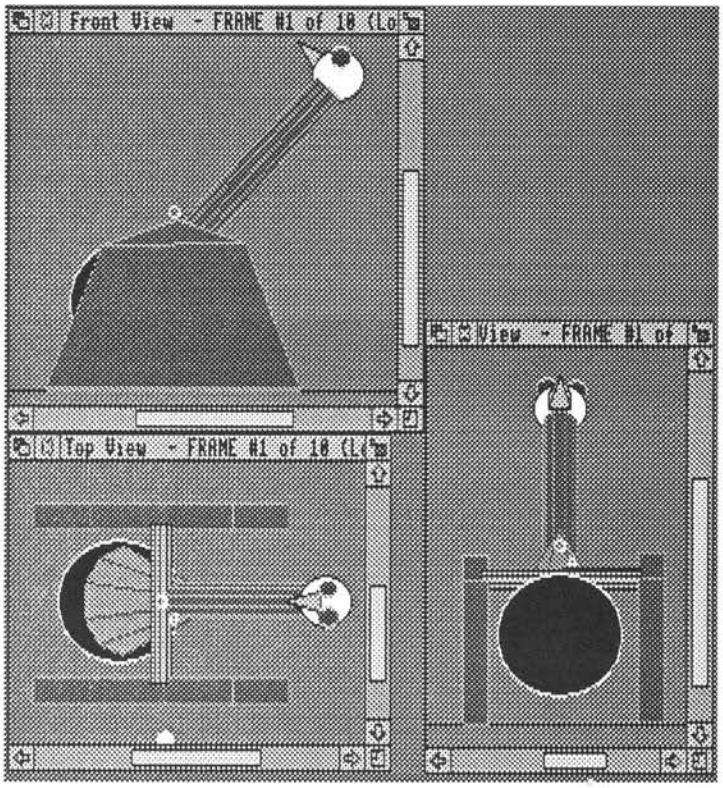
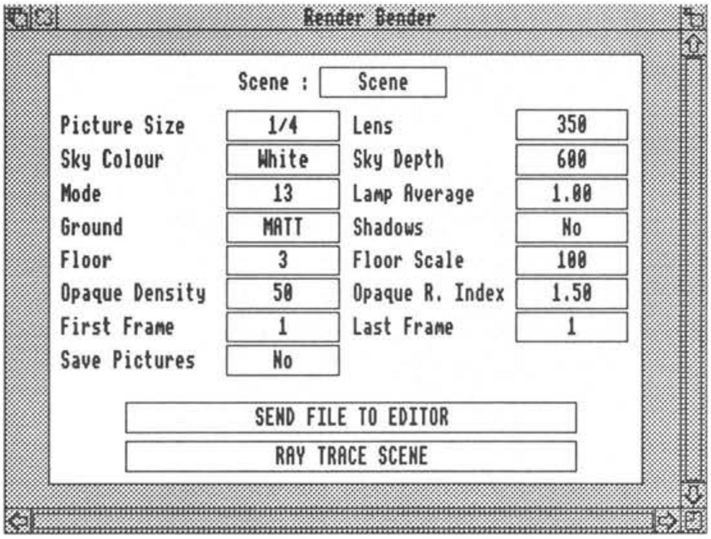

Front View - FRAME #1 of 10 (Lo
Top View - FRAME #1 of 10 (L(
View - FRAME #1 of

Figure 13.5—Front, top and left views (solid option) in *Illuminator*

determines how an object moves throughout the animation. Inbetweening, *i.e.* the generation of intermediate frames, is performed automatically.

Render Bender II is the application that converts scene files from *Illuminator* into ray-traced images. Its operation is quite simple, the hard work having been undertaken in *Illuminator*. A click on the icon opens a window (Figure 13.6) in which you enter various parameters for the ray-traced image(s). Within this window a menu is available offering further choices. A total of 14 floor designs is provided with *Render Bender* and these can be seen from the Floor item in the



Scene : Scene

Picture Size	1/4	Lens	350
Sky Colour	White	Sky Depth	600
Mode	13	Lamp Average	1.00
Ground	MATT	Shadows	No
Floor	3	Floor Scale	100
Opaque Density	50	Opaque R. Index	1.50
First Frame	1	Last Frame	1
Save Pictures	No		

Figure 13.6—the main processor options dialogue box in *Render Bender*

window. If none suits your purposes, you can design your own as a sprite and import it; a special palette is provided for this purpose.

The ray-traced images may be in any 256-colour screen mode. They may also be in any of several sizes from full size down to 1/1024 size. The reason for the provision of such very small sizes is that the time taken to ray trace the image increases dramatically with size and it is disappointing after waiting on tenterhooks perhaps an hour or so for the computer to process your image to find that, owing to some silly mistake, the very effect sought has been ruined. Instead you can quickly produce a very small rendering of the image which will show if anything major has gone wrong and allow you to correct it. Then, when you are sure that all is well, you can leave the computer to produce the final version.

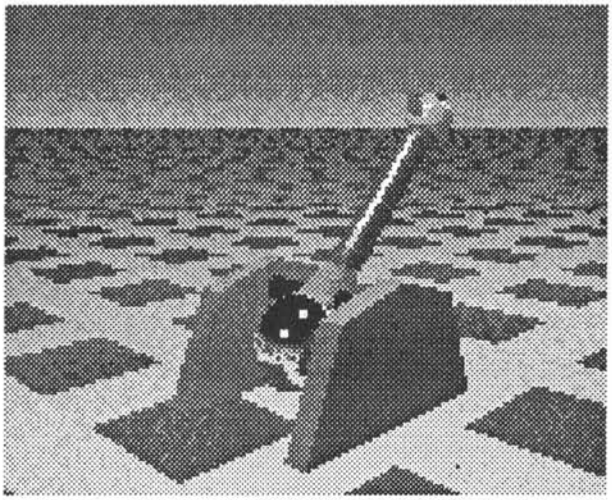


Figure 13.7—a ray-traced image produced by *Render Bender*

Like *ArcLight,* both background and fast ray tracing options are available. The fast option takes over the whole computer and displays the number of the row of pixels on which it is working; this gives you an idea of its progress. Ray-traced images are stored in a compressed format shared by several of Clare's applications. They can be converted to standard RISC OS sprites using the *Converter* application.

Animator creates animations from numbered sequences of *Render Bender II* images, but it also accepts RISC OS sprite files, *ProArtisan* files and *Illusionist* (see later) files. An animated sequence may use images from several different sources, but all must be in the same screen mode. Click on the *Animator* icon to open a window and drag one of the files into the window. All files in the same directory are examined and all scenes having the same name are noted and indicated by number in the window. You can then edit the sequence in which screens are displayed by editing the sequence of numbers. The same number may be used as often as you wish, allowing repeated or cyclical motion with minimal memory requirements. When you are satisfied with the sequence, you can compile the animation; this introduces additional compression (delta compression) which essentially saves only the differences between successive frames. You can convert an animation back to individual frames using *Converter.*

The quality of the rendering in *Render Bender II* is almost legendary. But the process can take a long time. To produce an animation of the quality of the demonstrations is a commitment to many hours of painstaking work and, depending on your choice of screen mode and size, is likely to tie up your computer for several hours of processing.

Illusionist

Strictly speaking *Illusionist* should not be in this chapter at all; it is not a ray tracing application. It describes itself simply as a 3-D art and design package. But this application from Clares, the same stable that produced *Render Bender* (above), uses similar techniques and also produces high-quality images. The difference is that instead of building up the final image pixel by pixel, analysing the rays of simulated light that fall on each, it divides the image into polygons and works through them one by one, calculating the exact colour and shade needed to paint each. It does not allow shadows or reflections as *Render Bender* does, but it is faster and somewhat easier to use.

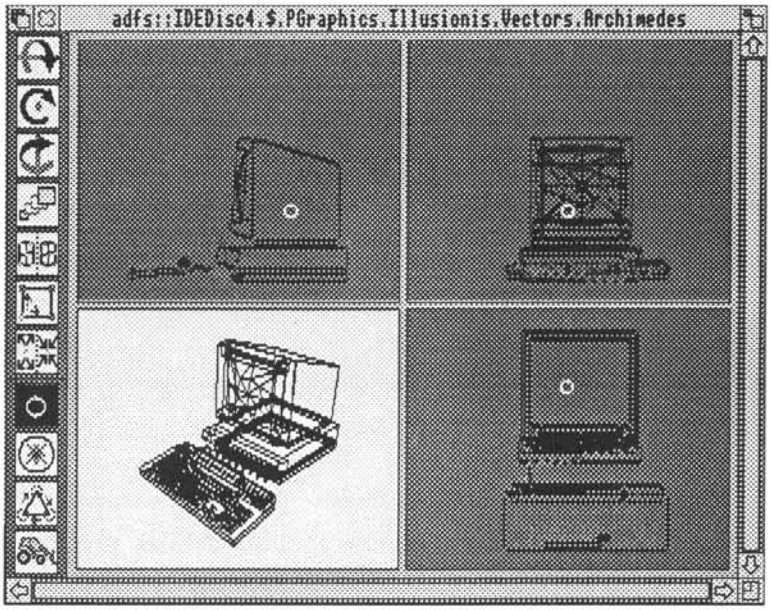


Figure 13.8—the 3D editing screen in *Illusionist*. The bottom left-hand quarter shows a perspective drawing of the subject

One valuable feature is a facility to 'map' a sprite or picture file on to any surface. Supplied for this purpose are reproductions of woodgrain, marble and a stunningly realistic ancient brick wall. Any sprite or picture file can be used provided it is in mode 12 or 13. So you could even wrap a map of the world around a sphere if you wished to reproduce a geographer's globe.

Illusionist is all contained in just one application. As in *Render Bender* the production of an image is a two-stage process involving drawing in a vector-graphics environment, followed by rendering. The vector-graphics drawings are referred to, appropriately enough, as vectors. There is a facility to export these as *Draw* files. The resulting pixel-graphics images can be saved in either RISC OS sprite format or Clare's own compressed format used in *Render Bender* and

ProArtisan. *Illusionist* itself does not produce animations, but the *Animator* application supplied with *Render Bender* will accept *Illusionist* images.

The editing facilities provided are comprehensive. The main screen is split in four, three divisions providing simultaneous views: side elevation, front elevation and plan. These are thoughtfully labelled with the six directions for identification: N, S, E, W, U (up) and D (down). All changes are reflected immediately in all three views. The fourth view is a wire-frame full perspective drawing and it is this that can be exported to *Draw*; it appears as a single path object (over 170 Kbytes for the drawing of the Archimedes).

The tools are mainly concerned with movement and editing. Selection is normally applied to points rather than objects and the first three tools rotate selected points about three axes; the other tools are concerned with zooming, selection, scaling, polygon creation and moving. Apart from the polygon creator (which handles polygons having up to 32 sides) objects are always created by the Add object option in the Edit menu. This provides for eight primitives: cube, pyramid, sphere, torus, hemisphere, tube, cone and disc. Also available are sweep objects (user-defined complex objects) and surfaces which are flat planes consisting of intersecting lines. All objects, even spheres, are made up from polygons; there are no curves in *Illusionist*. However, both smoothing and anti-aliasing facilities are offered which can give the illusion (this is the most aptly named software!) of smooth curves.

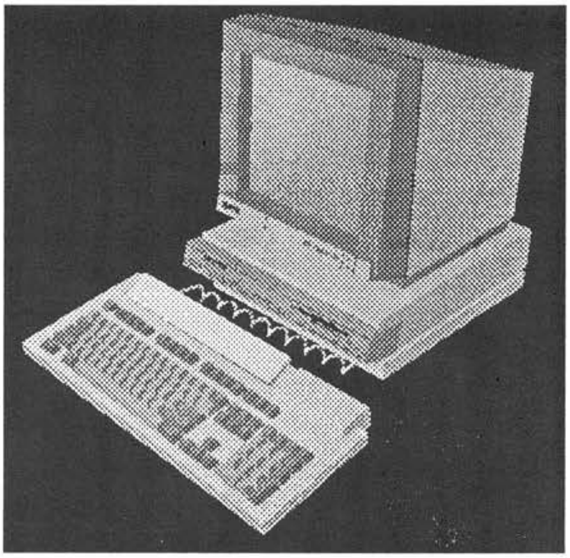


Figure 13.9—the fully rendered version of the vectors in Figure 13.8

Whole objects can be deleted or just selected polygons (surfaces) can be deleted. Objects can be given a variety of materials whose texture will affect their appearance. Polygons can be single or double faced—if double faced the appearance of their inner faces is also taken into account; this may be important if part of the object is transparent or missing.

Up to 14 lamps can be used in a scene; the colour, position, target, distance and type (point or spot) can be predefined. The observer or camera position can also be moved to any position within the scene. There are settings for exposure and ambient light (it is fairly easy to underexpose and end up with a rather dark image especially if the light is oblique to the main surfaces).

When you are finally ready to render your scene, there is a useful 'mini-test' facility which quickly produces a small version of the picture. This allows you to make any necessary adjustments before starting the full-size rendering. Although faster than *Render Bender*, *Illusionist* images still take appreciable time. The picture of the Archimedes in mode 21 (Figure 13.9) took about 5 minutes using an ARM3.

PowerShade

Roger Attrill's *Powershade* has been described as the most powerful and sophisticated ray tracer available for the ARM machines. Several examples of its output are featured in the colour plate section of this book. At the time of writing (mid-summer 1992) its publication by Arxe Systems is imminent.

It differs from the other applications described in that it normally takes its input in the form of a text file (but it can also use *Euclid* files). A typical extract from a text file is given below. This is written in a simple but comprehensive object definition language which allows the following primitive object types: sphere, plane, box, cylinder (open or closed), cone, triangle, phong triangle, superquadric and *n*-sided polygon. Each object's surface is defined with the following properties: ambient colour, diffuse colour, specular colour, specular coefficient, reflection coefficient, transmission coefficient, refractive index, translucency coefficient and phong transmission coefficient. In addition, each object may have any of the following textures: chequer, marble, wood, tile, cork, gloss and sky. And if those don't provide sufficient choice, you may alternatively apply bump mapping, fractal or picture mapping to get just the effect you want.

Each object can be rotated by any angle about any vector and can be scaled in each axis. Objects may be grouped and groups may include other groups.

Three types of light source may be used: point source, directional source at infinity and area light sources (for penumbra effects). The image may be traced by taking a number of samples in each pixel or by anti-aliasing with adaptive sub-dividing of a pixel colour until there is even contrast at each corner.

Output is available as an 8-bit RISC OS sprite with Floyd and Steinberg dithering (see Chapter 11) or as a 24-bit RGB file. Support is provided for 24-bit video enhancers. Effects which may be available include stereo views for left and right eyes, mist or smoke effects and depth of focus.

Powershade was written entirely in Assembler for speed and compactness. Nevertheless its comprehensiveness is such that tracing takes appreciable time; its progress is reported. Even with an ARM3, several hours should be allowed for a picture to be completed.

Listing 13.1—Opening of 'Still Life' Scene File

```
Password Fred
jittered samples 1
eyep 1.1 -9 3.2
lookp 0 0 2.5
up 0 0 1
fov 58 47.88
screen 640 512
mode 21
background 0 0 0
surface wood 0.15 0.1 0.045 1.
```

```
0.75 0.13 1 0.75 0.13 9. 0 0. 0.
surface shade1 0.1 0.1 0.1 .9 .9
.9 .9 .9 .9 40 0.5 0 0
surface shade2 0.1 0.05 0.01 .9 .6
.1 .9 .6 .1 24 0 0 0
surface stand 0 0 0 .9 0 0 .9 0 0
30 .3 0 0
```

14 Animation

You may be excused for thinking that there is no need for this chapter since several animation facilities have been described in previous chapters. These were offered by *Euclid/Mogul* and *3D Construction Kit* described in Chapter 6, *ARCtist* described in Chapter 9 and *ArcLight* and *Render Bender* described in Chapter 13. But in those chapters the animation facilities were incidental; it was some other feature of the software that merited its description there.

Animation techniques

Every animation system that man has devised (apart from 'live' puppets and automata) relies on the same optical illusion. If you show a sequence of static pictures in quick succession and each picture shows a successive stage in the same process, perhaps a man walking, the observer will be tricked into thinking that he is seeing not a sequence of still pictures but one moving picture. From the schoolboy flicking the corners of his notebook to achieve a crude animation of his doodlings, through the Victorian 'what the butler saw' machines on seaside piers, and on through cinematography to video, this same fundamental principle has always applied. And computer animation is no exception.

It is generally accepted that a minimum of 12.5 frames (*i.e.* pictures) per second is needed to achieve the illusion of motion. Increasing the number of frames per unit time gives smoother animation and reduces the flicker effect. VHS video recorders in Europe use 25 frames per second. Broadcast television in Europe actually displays 50 pictures per second, but only 25 frames; each frame is shown twice but using alternate lines. On the A5000 computer the monitor displays 70 frames per second, so that in theory it can display computer animations that are smoother than is possible on a standard video system!

Computer animation may use either vector graphics or pixel graphics. For interactive animations such as adventure games, flight simulators and virtual reality, vector graphics is the only choice, since it would be impossible for the computer to store all the sprites needed to display every possible view of every possible scene. Instead the computer stores a mathematical model of the whole scene and repeatedly redraws the user's view of it. Even for an ARM3 this imposes a massive burden on the processor and a special form of vector graphics is essential to speed up the processing. For example, medium-resolution mode 13 is often used because it combines 256 colours with a quick refresh time. No curves are permitted since their plotting involves onerous calculation. Even in the most sophisticated virtual reality systems wheels and other round objects when closely examined turn out to be polygonal: curves are a luxury that present virtual worlds cannot afford!

Most computer animation, however, is sprite based. Sprites have the advantage that they can be thrown on the screen very rapidly indeed and RISC OS' sprite plotting routines are readily accessible from BASIC or even the command line interpreter (see Appendix 2). The principal drawback of sprite-based animation systems is their massive memory requirement. If your animation consists of full-size mode 15 screens you will only have room for four of them on an 800 Kbyte floppy unless you employ a compression system. And unless the decompression system works in real time, another limitation will be the number of frames you can simultaneously hold in RAM.

Consequently, as in so many aspects of computer graphics, animation is an exercise in the art of compromise. Practical animation on the ARM machines tends to use the following techniques to keep its films to manageable sizes:

(i) frames considerably smaller than full-size screens

(ii) 16-colour screen modes (sometimes less than that)

(iii) repeated action (so that the same sprites are used over and over again)

(iv) compression of individual frames

(v) delta compression, *i.e.* compression applied to successive frames so that only the differences between them are stored.

These techniques used individually or in various combinations allow interesting animations of several seconds duration to be created and stored in files of conveniently manageable size. A typical example is Tony Cheal's animation *SmallFly* created using *Euclid* and *Mogul*. This fascinating little film, lasting about 5 seconds, gives you a 'fly's eye view' as it buzzes through the rooms of a modern house. The animation consists of 70 mode 15 frames, each 320 × 256 OS units, *i.e.* 1/16 of a mode 15 screen. But compression techniques have reduced it from the expected 70 × 160 × 1/16 = 700 Kbytes to just 44 Kbytes!

This kind of animation is not interactive: the viewer must accept the film as it is shown, just as in a TV broadcast or cinema performance; he or she cannot influence the development of the action.

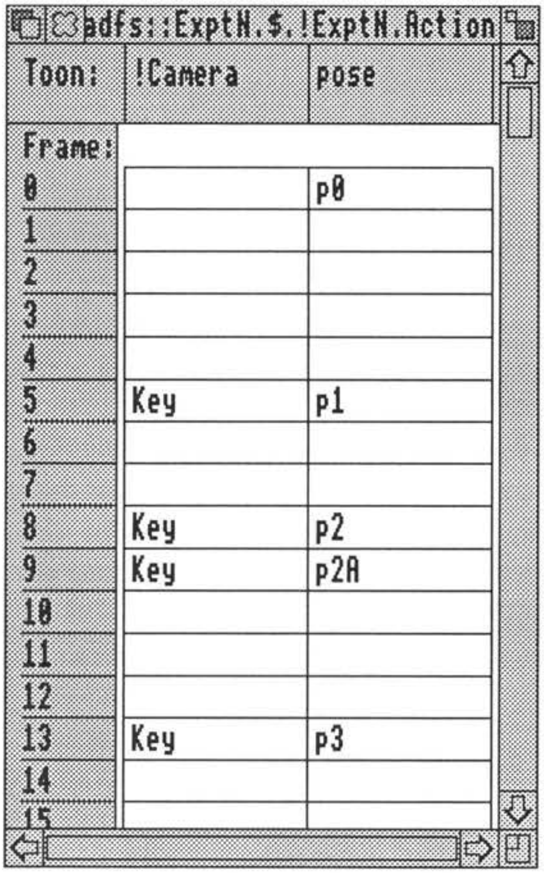
This chapter describes three software applications, all from Ace Computing, which are designed to take the anguish out of the creation of computer animations: *Tween* allows you to create animations from *Draw* files; *Splice* allows you to edit animations in Ace film format and *Projector* is a utility for playing animations in Ace film format. Finally, the chapter describes a few animation techniques for those who wish to go it alone and produce animations with or without the help of proprietary software.

Tween

Tween is to *Draw* what *Mogul* is to *Euclid.* If you want that in plainer English, here goes. *Tween* takes a pair of *Draw* files and produces from them as many intermediate frames as you specify; it does the 'inbetweening'—hence its name. And it will convert the resulting frames to a sprite-based film in Ace film format if you require.

The difference between *Tween* and the interpolate/grade/blend facilities offered in *Draw3*, *Vector* and *ArtWorks* is that *Tween* interpolates whole drawings consisting of many objects including groups, and produces intermediate independent drawing files, whereas the other packages only interpolate between single path objects within the same drawing. In *Tween* the parent drawings must of course contain the same numbers and types of object in the same sequence and each object must contain the same numbers and types of segments. Size, position and colour are all interpolated. But in fact it's more sophisticated than that since it will simultaneously interpolate between several pairs of drawings in different directories and will superimpose the results in a user-defined order. In those different directories you might have drawings of two or three different characters performing different actions but whom you wish to include in the same scene; and in one you might have a background scene which remains unchanged throughout the action. Each such directory is called a 'toon'.

As in *Mogul*, an Action chart (Figure 14.1) appears containing across its top the names of the relevant directories, one for the camera and one for each toon. Beneath these you enter the filenames of the *Draw* files relating to the key frames. If you click Menu in an empty space the main menu will offer you the chance to see that 'missing' frame. When this is drawn—it takes a little while, depending on complexity—it


adfs::ExptN.$.!ExptN.Action

Toon:	!Camera	pose
Frame:		
0		p0
1		
2		
3		
4		
5	Key	p1
6		
7		
8	Key	p2
9	Key	p2A
10		
11		
12		
13	Key	p3
14		
15		

Figure 14.1—the Action window in *Tween*. p0, p1, p2, p2A and p3 are all *Draw* files in the directory *pose*. Intermediate frames between them can be seen (and saved) by clicking in intermediate spaces

appears in a window and you can save it as a new *Draw* file.

Although the files are described as *Draw* files, the originating software need not be *Draw* itself. *Vector* offers a special advantage for users of *Tween*. When the drawings being 'inbetweened' are fairly complex (say 20 objects, three of which are groups of six further objects each) it often happens that in editing the drawings for the key frames, the component objects get out of order, usually as a consequence of grouping operations. Sorting so many objects into the right order is a hit and miss affair in any drawing package except *Vector* whose order dialogue box tells you the precise position in the stack of the currently selected object and allows you to move the selected object to any position by number. The trick is to make a master list of the objects in the drawing (and in the groups) and after finishing each key version, to select each object in turn and move it to its proper place if necessary.

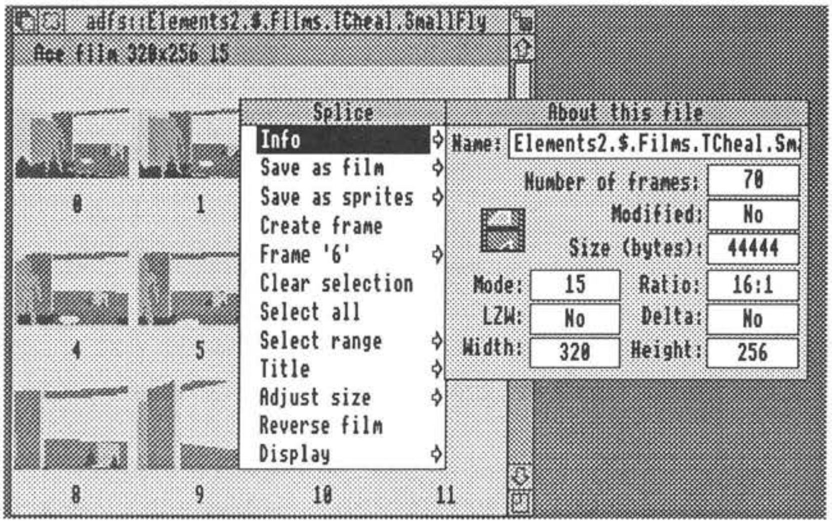
If you intend to save the film in Ace film format, you may also need to attend to the camera. This is a special rectangular object whose size determines the size and aspect of the drawing contents as they will appear in the finished film. You can even change the size of the camera during the film's progress—it can be 'inbetweened' like the drawings—so that the camera can, for instance, zoom in on some point of interest in the action. Often, however, no camera motion is required and the camera column can be left blank.

Another of the options from the main menu is to save the complete film in Ace film format. This may be a long process, depending on the number of toons and frames involved, as each drawing in turn must be interpolated and then compressed. But even if you use some other film format, if you are an animation enthusiast who appreciates the exciting potential of vector graphics, you will find *Tween* an invaluable tool.

Splice

Splice is a utility for editing Ace films. It cannot edit the contents of individual frames—for that you must use *Paint* or some other pixel graphics editor—but it can be used to step through films frame by frame, to examine or export frames, to delete or insert frames or even to join films together, *i.e.* to splice them, hence the application's name.

Predictably, you start up *Splice* by dragging an Ace film icon on to its icon. (Double-clicking on an Ace film loads it into *Projector* instead.) This opens a viewer window similar to that of *Paint* in that it contains a miniature version of each frame in the film (Figure 14.2). Clicking Select in a miniature selects that frame. Double-clicking Select opens a window showing that frame full size. Clicking Select in the full-size frame changes the view to that of the next frame and clicking Adjust to the previous frame. Holding down Select or Adjust causes the film to run forwards or backwards at more or less normal speed within the window. Clicking Menu in that window opens a dialogue box with a save option allowing you to save the frame as a sprite. To edit an individual frame you would use this option, drag the frame into *Paint* and use that



adfs::Elements2.$.Films.TCheal.SmallFly

Ace film 320x256 15

About this file

Name: Elements2.$.Films.TCheal.Sm

Number of frames: 70

Modified: No

Size (bytes): 44444

Mode: 15 Ratio: 16:1

LZW: No Delta: No

Width: 320 Height: 256

Figure 14.2—the frame display and main menu and film information box of *Splice*. The film being examined is Tony Cheal's *SmallFly* which well illustrates the interaction between 3D drawing (*Euclid*) and animation (*Mogul*)

package to edit the picture. When your editing is finished, you save the sprite, dragging it back into the *Splice* window where it will replace the existing frame.

Individual frames or groups can be selected and subjected to such manipulations as vertical and horizontal flipping, although I confess to being uncertain as to why one should ever wish to flip individual frames.

A useful facility allows the size of the frames to be changed. All frames in an Ace film are the same size and sizes are always expressed in OS units rather than pixels. Interestingly, resizing Tony Cheal's *SmallFly* to full screen size of 1280 × 1024 (multiplying the area of each frame by 16) only multiplied the file size by about 5 to a little over 200 Kbytes, slightly more than a single mode 15 screen.

A scale-to-fit toggle, if selected, ensures that the design is scaled to fit the new size. If unselected, scaling up introduces lots of free space into each frame, so that additional action could be added into a film by those with sufficient dedication.

You can join two films together very easily by dragging one film's icon into the other's window. The newcomer is appended to the end of the original film. If the films have a different frame size, the newcomer will be scaled to the same size as the original, an operation that may take some time. Since *Splice* allows you to save a selection of marked frames as a separate film and to delete selections of marked frames, by combinations of joining, deleting and saving you can insert one film into another and introduce repeat action. A facility for reversing the sequence of a film makes it easy to produce reciprocating action, such as a swinging pendulum.

You can convert a whole film to a sprite file, but beware! It might be too big to load into *Paint*. Similarly, if you drag a sprite file into *Splice* it becomes an Ace film. So, *Splice* could be used in conjunction with *Paint* alone to create animations. But my experience suggests that the preparation of animations is simpler using *Vector* and *Tween* or *Euclid* and *Mogul*, with *Splice* being used for edits to nearly finished films. The range of Ace products certainly allows limitless opportunities to the computer animation enthusiast.

Projector

The last of the trio from Ace computing, *Projector* is an application for playing Ace films.

Ace have placed it in the public domain so that you can freely distribute copies of it together with Ace films that you have created (and to which you therefore presumably own the copyright).

You can play an Ace film by dragging it on to the *Projector* icon or by double clicking on the film icon provided that *Projector* is installed or the computer knows where to find it. The film starts up automatically in a window in the current screen mode (whether or not that is the film's 'native' mode). When the film finishes it will automatically restart; this cycling continues until you intervene.

Clicking Menu within the window reveals just a few of the application's facilities such as freeze frame and reverse, which are self-explanatory (Figure 14.3). Yo-yo plays the film alternately

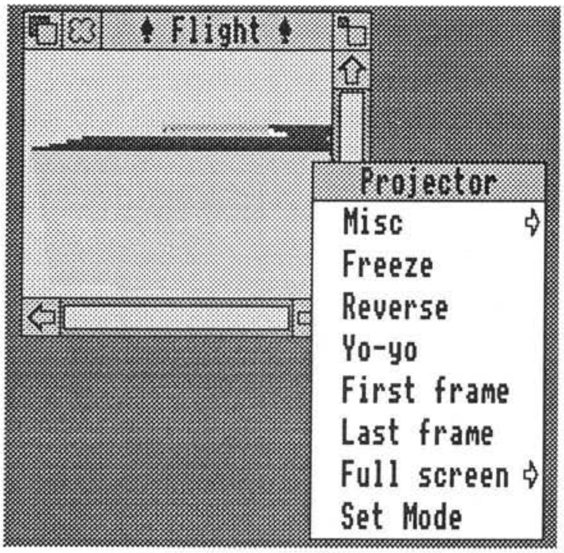
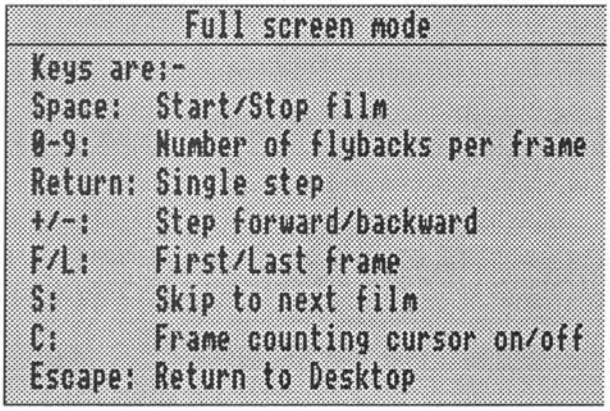


Figure 14.3—a typical *Projector* window with the application's main menu. The film is Tony Cheal's *SmallFly*

forwards and backwards. First frame restarts the film (forwards) at the beginning and last frame restarts the film backwards at the last frame. Set mode changes the screen mode (if necessary) to the film's native mode.

Full screen reveals the other facilities in *Projector*. As its name suggests, it temporarily exits the desktop and commands the whole screen. The benefit of this is that animations run faster in full-screen mode than in a window and some larger films may only be playable at their proper speed in this mode. You can return to the desktop at any time by pressing Escape. All applications will be intact and the film will again be playing in its original window.

If you slide right on the Full screen option without clicking on it you call up a list of controls available in Full screen mode; this is shown in Figure 14.4. The space bar starts and stops the film. The speed of the film can be controlled by entering a number between 0 and 9: the higher the number the slower the film will



```
          Full screen mode
Keys are:-
Space:  Start/Stop film
0-9:    Number of flybacks per frame
Return: Single step
+/-:    Step forward/backward
F/L:    First/Last frame
S:      Skip to next film
C:      Frame counting cursor on/off
Escape: Return to Desktop
```

Figure 14.4—options in *Projector's* full-screen mode

play. Return allows you to single-step through the film, while + and - single-step forwards and backwards. F is equivalent to the First frame option and L takes you to the last frame, but does not reverse the film; it simply starts again. Skip to next film only works if you load several films and choose the Roll option from the icon bar menu or if you use the projector to control a whole rolling demo using an obey file—this is explained in the file *!Help* in the *!Projector* directory. The frame counting cursor is an optional numerical indication of the current frame number displayed at the pointer position.

Clearly a projector is an essential prerequisite if you wish to enjoy films! It was indeed thoughtful of Ace to place this application in the public domain—and in their interest as well as the public's since it can only lead to a greater awareness of the enjoyment available from animations on the ARM machines.

Getting into animation

You may well feel that you want to make your own way in computer animation without the constraints (although they are minimal) of proprietary film formats. In fact it is not that complicated. Computer animation is not some startling innovation that has only been made possible by the advent of the ARM chip. Can you remember those halcyon days when we huddled around the BBC Model B (then seemingly the ultimate in computing power) enthralled by such games as *Frogger*, *Monsters* and *Snapper*? These involved simple, but very effective, sprite-based animation. And they achieved it, hard to believe though it now seems, in just 32 Kbytes of RAM which also had to include the screen memory, operating system workspace, the application program and such variable data as the scores and players' names! Bearing in mind what was possible with 32 Kbytes, just think what ought to be possible with 1 Mbyte—or 4 Mbytes!

Let's examine the practicalities of sprite-based animation. In the BBC games mentioned above, animation consisted almost exclusively in moving small brightly coloured sprites around the screen against a generally black background. This is indeed the simplest *scenario*. Moving a sprite from A to B involves plotting the sprite at A, erasing it at A and plotting it at B. Depending on the distance between A and B and the speed with which you wish the sprite to cover that distance you may also plot the sprite at one or more intermediate locations (and erase it again of course before the next plot). There may be instants when the sprite is not on the screen at all, *i.e.* when it has just been erased at one location and has not yet been plotted at its next location. But provided the time between plotting and erasure exceeds that between erasure and the next plot, this is unlikely to be noticed. Moreover, if the background is black and the sprite is brightly coloured, the human eye will fail to notice such temporary disappearances. Indeed with the ARM and a small sprite, if the plotting instruction follows immediately after the erasure instruction, both may occur in the same interval between frames so that the sprite is never absent from the screen at all.

Plotting a sprite is one matter; erasing it is another. There are several ways this may be

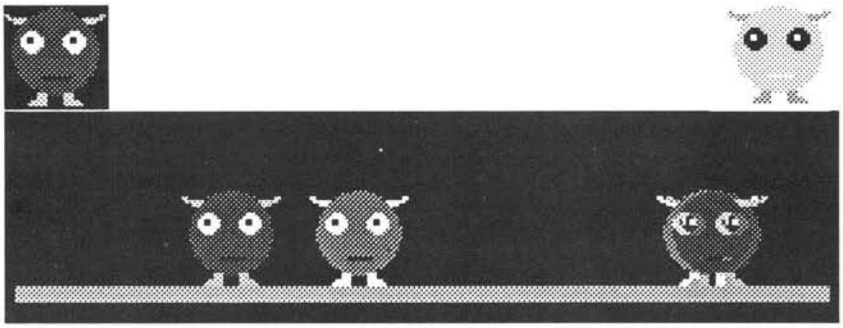


Figure 14.5—The value of the EOR plotting action in games and other animations. Top left: a sprite consisting of a red monster on a black background. But when EORed on to the intended black background it becomes a cream monster on a white background (top right). So we EOR it with black before introducing it into the animation or game when it looks as intended (below left). Below right: what happens when a red and green monster pass each other? We get a temporary dark grey monster! (With apologies to the former Acornsoft company!)

done. Simplest is to first plot the sprite on the screen using the exclusive-OR plotting mode (plotting action 3 in the ARM SWIs). Then to erase the sprite you simply plot it again at the same point. This has the advantage that it puts the background behind the sprite back exactly as it was before you first plotted it there. But it has the disadvantage that the background colour will affect the appearance of the plotted sprite. Not that this really matters, for you could always design your sprites in their intended colours and then exclusive-OR them with the background colour (by dragging a square of background colour over the design in *Paint* having first selected EOR) before using it in the animation. Seen in the raw in *Paint* its colours will now look extraordinary. When it is first EORed against the intended background colour the sprite will be restored to its design colours. And when you EOR it again in the same location, it will conveniently disappear.

An even simpler way to achieve the same effect is to design the sprite with a margin of background colour around its edges. Having plotted the sprite you can now move it in any direction up to one margin's width and simply replot it. Ordinary 'overwrite' plotting action is sufficient. The previous instance will be overwritten by the margin of the new instance and so will disappear. All commendably simple, but the margins make the sprite rather larger than it need otherwise be. Of course, you may not need all the margins. If the sprite is, say, a car that is only required to travel repeatedly from right to left along a level road across the screen, it only needs a margin at its rear end.

But few games or other animations are satisfying if all they do is move a single sprite around a screen of solid background colour. To add interest we need a few other sprites to move around as well. What happens when two sprites meet?

The answer is with the EOR system, all still works as intended. It's true that you get funny colour effects when one sprite is plotted on top of another having different colours (of course, if you plot a sprite right on top of its identical twin, both disappear). But as each sprite is replotted to make it disappear so it restores the other sprite and the background to its original condition.

So too with the margin system: all will sort itself out in the end, but you may find that the sprite that was first on the scene at the point of collision becomes invisible for a while, hidden beneath the other sprite's margin.

But what happens if, in the interests of realism, you want a detailed background? Suppose you want a car to drive past houses and shops or a man to walk in front of a detailed brick wall or a train to pass a signal box and trees and telegraph poles? If you want all this, you can have it, but it will be costly in terms of computing power.

If the sprite has an odd shape (*i.e.* if it is any shape other than rectangular) or if its design has windows or gaps through which the background can be seen, you will need to give it a transparency mask. That immediately doubles its memory requirement. Also, if you are using your own software to plot the sprite and if you are calling a SWI (operating system routine) you must remember to set bit 3 of the plotting action number or the mask will be ignored (and transparent pixels will be reproduced as the highest value palette colour).

As the sprite travels along it will overwrite parts of the background. These will have to be put back. The easiest way to do this is to calculate the exact area where the sprite will next be plotted (you know the sprite's size and intended location so this is no problem) and to save this area of screen as a temporary sprite using OS_SpriteOp 14. (This will of course take more memory; I said that realism was demanding on computer power.) When you have saved the background as a temporary sprite, you then plot your sprite over that location. Then when you wish to move your sprite on, you plot the temporary background sprite in its original location; this puts the background back as it was, erasing the mobile sprite. Now repeat the process at the mobile sprite's next location. You could use the same sprite name over and over again to save the background, but if, as is likely, you have several such mobile sprites in your animation, each will need its own background sprite.

If two of these mobile sprites, each having its own background sprite, should overlap (for example, if they depict two cars or two trains passing each other) you have further complications. If it is important that one particular sprite should appear to be in front, then that sprite must be plotted last, even if this is out of the normal plotting sequence. Another problem is that when you save the area of background where the second sprite is about to be plotted, it will already include the image of the first sprite and we probably will not wish to include that in the background which we replace when the second sprite moves on, because by then the first sprite itself should presumably have moved on. If your animation allows the possibility of two mobile sprites occupying the same area of the screen, you must before each plot check to see if any other sprites are already there and, if so, then you must erase them immediately before saving the newcomer's background. Then replot the other sprites and lastly the newcomer itself.

All this applies well to cars and trains which don't change their shape as they move, although perspective may well change the aspect ratio of, for instance, a train coming towards you. That can be accommodated by careful separate scaling of the sprite's x- and y-axes as you plot it.

But what about people and animals walking—or even people staying in one place but striking different poses (such as waving their arms)? People (and animals) have such awkward shapes for animation! Forget what I have said about masks and backgrounds. The easiest way to animate people and animals is to use *Draw* or another vector graphics package. Introduce your background and draw your person or animal in front of it. Then grab the portion of screen with *Paint*'s Grab screen area/Snapshot facility. And then change the pose a little for the next frame. This is an ideal task for vector graphics. In fact the layers facility in *Vector*, *DrawPlus* and *ArtWorks* is ideally suited for this. You can lock the background in an unselectable layer, put the first pose in the next layer and copy it into another layer for editing into the next pose and so on. Each time you copy the drawing you copy a piece of background so you need not worry about the background, except that you will need to use the equivalent of the 'margin' system to replace background behind a walking person or animal. It's costly in memory and processing power, but I did warn you it would be.

It's also costly in artistic ability and powers of observation. You may be a brilliant portrait artist, but to create an animation of a person walking is something else. You may well discover how little you know about human locomotion! A suitable passage of videotape seen repeatedly in slow motion or frame by frame may help a lot in this respect. So may standing on a corner and watching all the folk go by..

15 Mathematical and Abstract Graphics

There remains an area of computer graphics that does not belong properly in either vector graphics or pixel graphics. This is the highly specialised but fascinating field of graphics generated using a mathematical process or algorithm. Probably the best known of these is the Mandelbrot set which has received much attention in computer media. Similar in some respects are the Julia set and Lyapunov Space. All create images of engaging but somewhat unearthly beauty; you will find examples of these and others in the colour plate section at the centre of the book.

The basis of most of these images is the *fractal.* The essential idea of fractals is a very simple one.

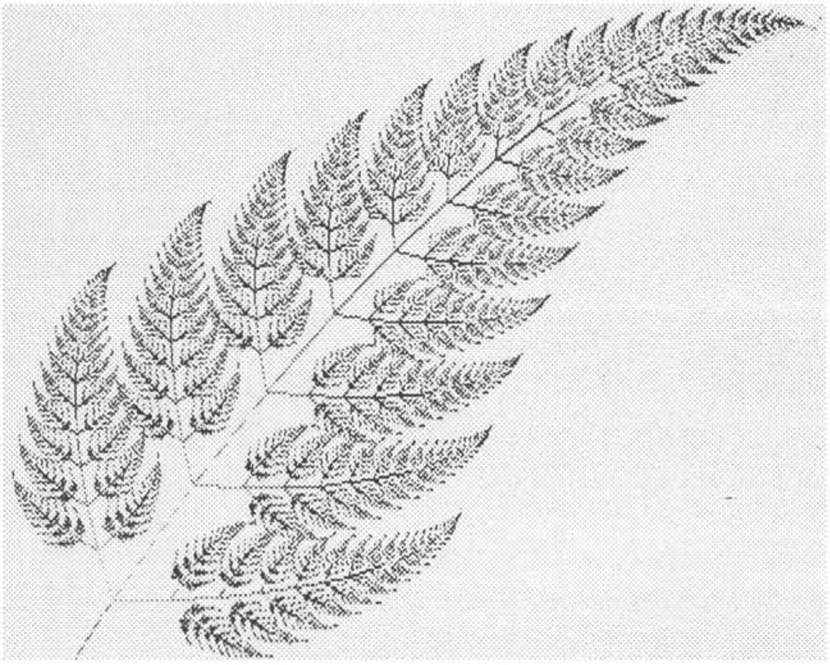


Figure 15.1—the structure of a fern leaf provides an everyday illustration of the concept of fractals. This image was created by software from the Third Millennium

Perhaps a fern leaf (Figure 15.1) provides the most familiar example. Many ferns have leaves whose edges, instead of following a gentle curve around the periphery, are deeply divided into lobes which botanists call *pinnae.* A leaf thus divided is said to be *pinnate.* Often, however, the edges of the pinnae are themselves similarly divided, so that there are two distinct levels of division and the leaf is then said to be *bipinnate.* And in some examples even the edges of these second divisions are further subdivided, making the leaf *tripinnate.* If you examine a pinna from a tripinnate leaf under magnification, it may look remarkably like the whole leaf seen without magnification. And that is significant since the characteristic property of a fractal pattern is that it tends to look the same irrespective of the magnification factor with which it is examined. There are always further levels of detail too small to see. Indeed there is in theory no end to the detail contained in the pattern.

The Mandelbrot set

Without introducing complex mathematics which you have no real need to understand, suffice it to say that the Mandelbrot set is similar to an infinitely pinnate fern leaf that has been turned inside out. It is a geometrical figure having a number of levels or contours of ever increasing complexity enclosing a central void. Ideally the number of levels is infinity, but since our computers are finite we are obliged as ever to make a compromise, so let's make the number of levels 256. This is a convenient number, since in a 256-colour mode we can assign each level its own colour and thereby ensure that (theoretically) the resulting graphic uses every colour in the palette.

The following description is of necessity a gross simplification. The first level in the set is a filled circle, *i.e.* it is the simplest possible curved shape, having a single radius. Inside this we draw the second level, a slightly more complex shape, an oval, which has two radii. Inside this we draw the third level, slightly more complex still—a pear shape—having 4 radii. Each successive level has twice as many twists in its contour as the previous and each hugs the contour of the previous a little more closely. This continues until, in our example, the 256th level.

Although the first few levels hardly inspire great interest, by the fifth and sixth levels interesting shapes begin to emerge in the overall graphic. Some 'wobbles' in earlier levels become exaggerated in successive levels leading eventually to near-circular bays or sharp promontories. Indeed the whole figure resembles a contour map of fabulous mountains surrounding an inland sea whose coastline is a maze of inlets and natural harbours. As you examine successively deeper levels, the coastline becomes more and more detailed and its fractal nature becomes ever more apparent: zooming in on one of the near-circular bays will reveal that its edges are not as smooth as they appeared at lower magnification, they themselves have inlets and smaller near-circular harbours. The sequence in Plate 11 shows this very clearly.

Mandlbrot

There have been many Mandelbrot generators for the ARM Machines, especially in the public domain and shareware market. But one of the finest applications—available from Tekoa Graphics and not in the public domain—is simply called *Mandlbrot* and it comes from Raymond Keefe, originator of *Art20* described in Chapter 9. Like *Art20, Mandlbrot* will run in any desktop screen mode, but obviously the most attractive graphics are obtained in the 256-colour modes. You really need a machine having at least 2 Mbytes of RAM.

Although not as fast as some Mandelbrot generators, it is a very flexible program. Any number of levels can be set with a wide choice of ways in which colours can be assigned to levels—the colours can even be changed after the set has been plotted. But one word of warning: the time taken to plot the Mandelbrot increases dramatically as you add more levels and probe deeper into the set. Even with an ARM3 the third picture in plate 11, which magnifies an area of a 256-level Mandelbrot by about 100, took about an hour.

Clicking Select on the application icon or on Create New Mandelbrot from the icon bar menu leads to a dialogue box in which you enter the required screen mode. The Mandelbrots produced occupy full screens in the selected mode.

When you have entered your choice of screen mode, you are taken into a sprite window (Figure 15.2) which is of course initially empty. Its menu allows you to save the sprite (you would of course only choose this option later on

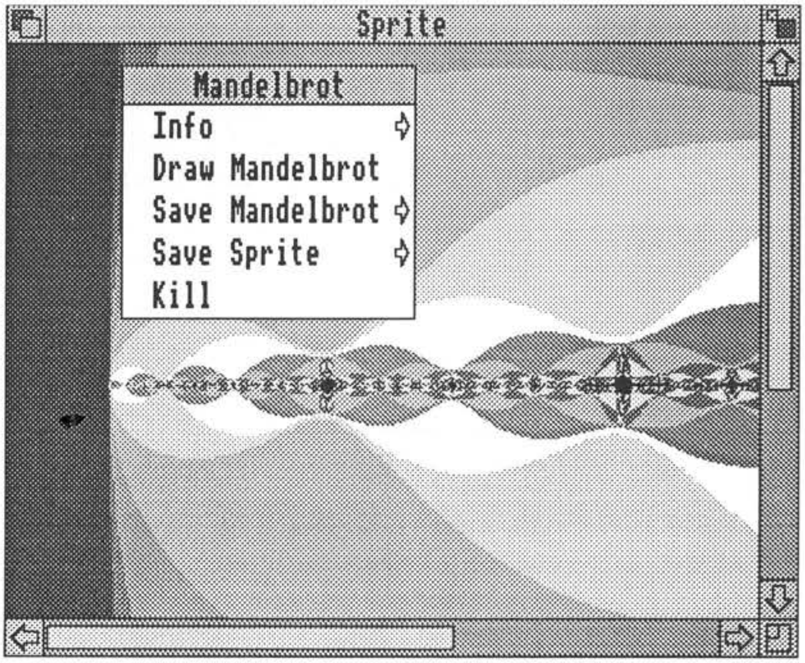


Figure 15.2—the Sprite window and its menu in *Mandlbrot*

when there is something worth saving) or the Mandelbrot (*i.e.* all the settings and calculations so you can resume an exploration that was previously interrupted). To start a new Mandelbrot you must select the Draw Mandelbrot option and this takes you into the main window which, in order to provide the largest possible graphics area, occupies the whole screen (Figure 15.3). You can return to the desktop environment by selecting the last option, Desktop, from this screen's menu. Any other applications that were running will be intact.

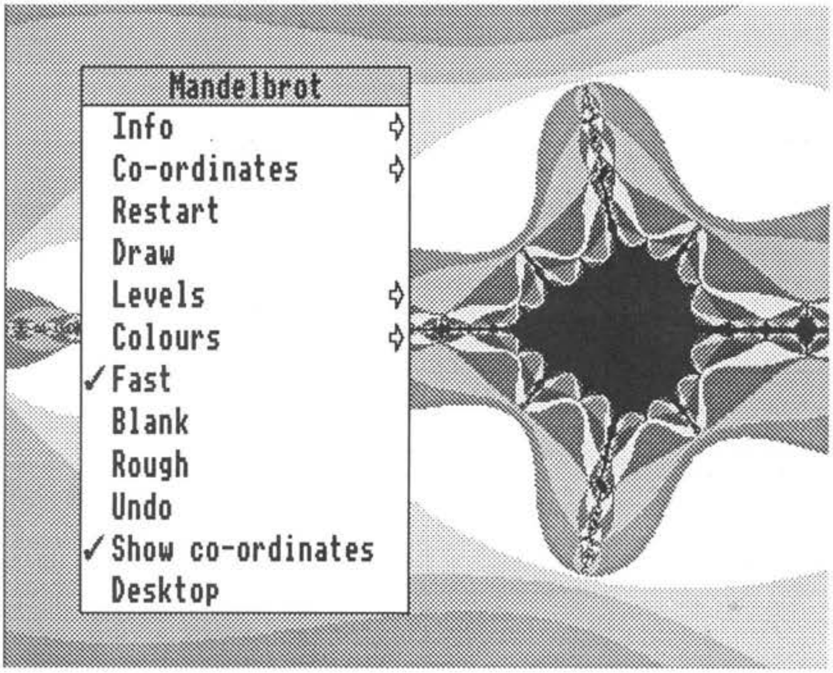


Figure 15.3—the main screen and menu from *Mandlbrot.* The set shown in this and the previous Figure uses only 16 levels and is in mode 27 (for speed)

In fact Mandelbrot drawing does not begin when you enter the main window. It only begins when you select Draw from the main menu. But first you will probably wish to select a few options such as the number of levels and the colour system. The number of levels need not be the same as the number of colours. If you have more levels than colours (*e.g.* 32 levels in a 16-colour mode) each colour will be used twice. And if you have more colours than levels, not all colours will be used. You can even change your colour *régime* after the Mandelbrot has been drawn and the screen will be updated to reflect your decision. Usually each palette colour is used in turn. You can reverse the colour sequence or introduce an offset so that the sequence starts on a different colour.

Two provisions are made for magnification. The simplest is to drag a box anywhere in the drawing. It will be constrained to screen proportions. Thereafter clicking Select will redraw the area enclosed by the box at full screen size. This involves more calculation than the original screen and will therefore take longer to draw. Alternatively you can enter the coordinate of the bottom left-hand corner of the area of interest together with the width of the area. There is an option to display pointer co-ordinates permanently, so you can quickly find the values for any areas of interest. You can return to the previous view using the Undo option. The Rough option provides a rapid drawing of the requested area at 1/64 of full screen size and much faster than a full draw; this is useful for checking that you have selected the right area before waiting for it to be drawn in detail. The Faster option (selected by default) provides more rapid drawing, but the calculations are not quite so accurate.

Any screen of interest can be saved as a standard RISC OS sprite and this of course can be loaded into *Paint*, *Art20* or any other application that accepts sprite files. You can also save the whole Mandelbrot complete with settings and calculations in order to resume a session that must be interrupted. If you are interested in this kind of graphic, then *Mandlbrot* will occupy you and your machine for many happy hours.

Lyapunov

Also from Raymond Keefe and available from Tekoa Graphics is an application entitled *Lyapunov* which creates images in Lyapunov Space. Indeed they look just like something that you might expect to see in outer space! For further information on the work of Aleksandr Lyapunov see the article 'Leaping into Lyapunov Space' in *Scientific American*, September 1991.

The program itself works in almost exactly the same way as *Mandlbrot*. The formula that creates the images, however, is different and one of the differences is that it requires a seed value. Whereas there is only one Mandelbrot set whose only possible variations are in number of levels and colours assigned to levels, there are limitless possible Lyapunov images. You can make four settings: the Seed value allows you to enter a value that will be used in calculations; Settling allows you to enter how many iterations (repeated calculations) of the formula are undertaken before any averaging is done—its purpose is to allow the formula to 'settle down'

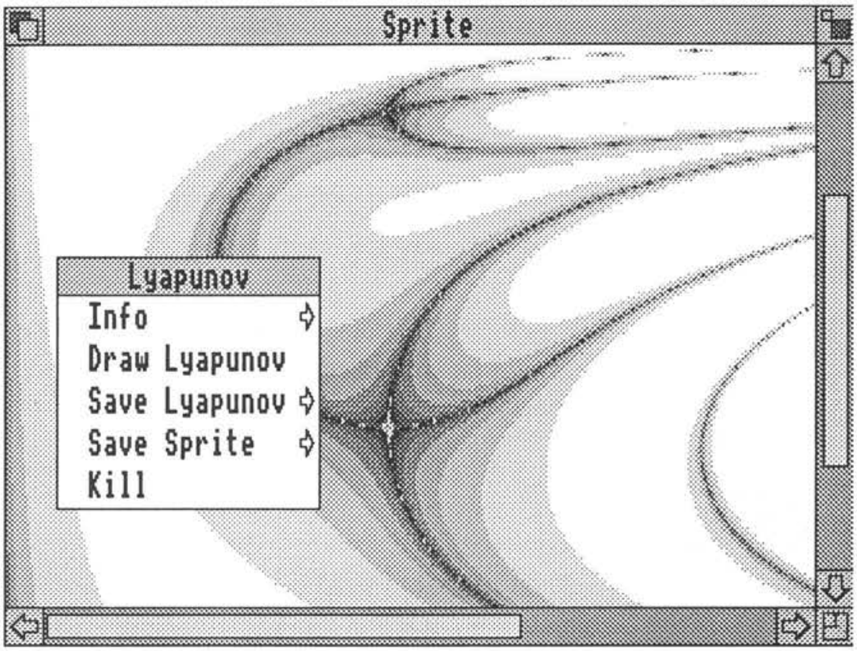


Figure 15.4—the Sprite window and its menu in *Lyapunov*

before its results are used; Iterations determines how many iterations of the formula are averaged; Sequence determines the sequence in which x and y values are entered into the formula. These four variable inputs, then, allow considerable variation of the images produced.

Colour control is much as for *Mandlbrot* and the Autoscale setting is recommended as it works out a spread of colours to cover the values obtained. Chaotic regions, however, are always shown dark. It is another application offering endless fascination to those with a taste for the abstract.

Third Millennium

Third Millennium is a public domain library maintained by Jean van Mourik which specialises in the kind of abstract and mathematical graphics described in this chapter. Some examples of the images available are included in the colour plate section; another is reproduced in Figure 15.1. Also available are various dithering techniques to extend the colours available from regular modes.

Besides software, Jean sells postcards and tee-shirts depicting his graphics creations, a video of fractal-based graphics backed with ambient music and he is working on a book containing a collection of his graphics.

If you enjoy abstract graphics of this kind you can do no better than purchase the Third Millennium's latest *Rainbow Demo* disc which is packed with graphics, some animated, guaranteeing you a colourful time. Many of the graphics are created and animated by simple BASIC programs which you can list and adapt to suit your own purposes.

Among the most fascinating of his creations are *cellular automata*. These are patterns based on cells which undergo transformations and multiplications which in some ways recall the growth of populations of micro-organisms. Each cell is evaluated in turn on the basis of its proximity to other cells and the screen replotted with new values for each cell. The *Game of Life* (not itself on the Rainbow Demo disc) is an example of a cellular automaton that has enjoyed occasional bursts of popularity.

Other examples are the two-dimensional transformations such as *Whirl* (plate 15) which uses the short BASIC program in Listing 15.1. When the screen has been plotted (which takes some time) it is animated by the simple

Listing 15.1 A two-dimensional transformation

```
   10REM >WHIRL1
   20REM JEAN VAN MOURIK, THIRD MILLENNIUM, 4
PANTLLYN, LLANDYBIE, AMMANFORD, DYFED, U.K.
SA18 3JT
   30F=9999:MODE20:ORIGIN640,512:OFF:REM
INPUT"AMOUNT OF SKEW (SUGGEST 9999)";F
   40DIMR%(15):DIMG%(15):DIMB%(15):DIMD%(15):
DIMC%(15):FORX=0TO15:READ C,D,E:COLOURX,C,D,E:
R%(X)=C:G%(X)=D:B%(X)=E:NEXT
   50FORX%=-511TO512STEP2:D%=X%*X%:A=1:FORY%=
-639TO640STEP2:S=F/SQR(D%+Y%*Y%):A=SQRS:P%=
X%*COSA+Y%*SINA+1234:IFP%>1234:GCOL(P%MOD15)+1:
POINTY%,X%:POINT-Y%,-X%:NEXT:NEXT:GOTO90
   60NEXT:NEXT
   70TIME=0:REPEAT:R%=R%(1):G%=G%(1):B%=B%(1):
FORX%=2TO15:R%(X%-1)=R%(X%):G%(X%-1)=G%(X%):
B%(X%-1)=B%(X%):NEXT:R%(15)=R%:G%(15)=G%:B%(15)
=B%:WAIT:FORX%=1TO15:COLOURX%,R%(X%),G%(X%),
B%(X%):NEXT:UNTIL0
  80DATA0,0,0,192,0,240,144,48,240,96,96,240,48
,144,240,0,192,240,0,240,192,48,240,144,96,240,
96,144,240,48,192,240,0,240,192,0,240,144,48,24
0,96,96,240,48,144,240,0,192   REM R,G,B VALUE
OF THE PALETTE
   90REM TO SAVE THE IMAGE DELETE "REM " FROM
THE FOLLOWING LINE
  100REM *SCREENSAVE IMAGE1
```

expedient of changing the colours on a cyclical basis.

Fortran Friends

Fortran is a programming language ideally suited to mathematical and scientific work. On the ARM machines C, Assembler and BASIC reign supreme and little is heard of Fortran, but there is

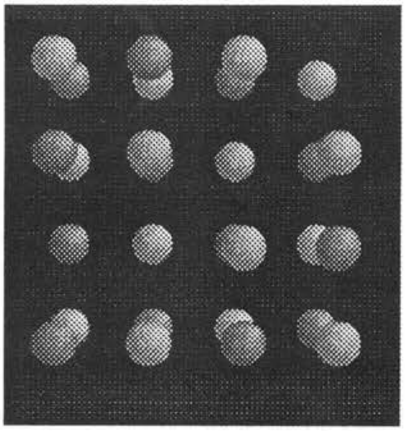


Figure 15.5—an example sprite from the *SphereRot* application from Fortran friends. The radial fills add to the 3D effect

an active user group called Fortran Friends (address in Appendix 5). They have compiled a disc of graphics software which is now available as Shareware Disc 44 from Norwich Computer Services (address in Appendix 5).

SphereRot is an example of sprite manipulation; it creates a sprite showing spherical objects of various sizes arranged in different ways—again the three-dimensional effect is most attractive. An example is shown in Figure 15.5. *WimpPoly* draws a large selection of three-dimensional polyhedra which can be saved as *Draw* files. Examples are given in Figure 15.6.

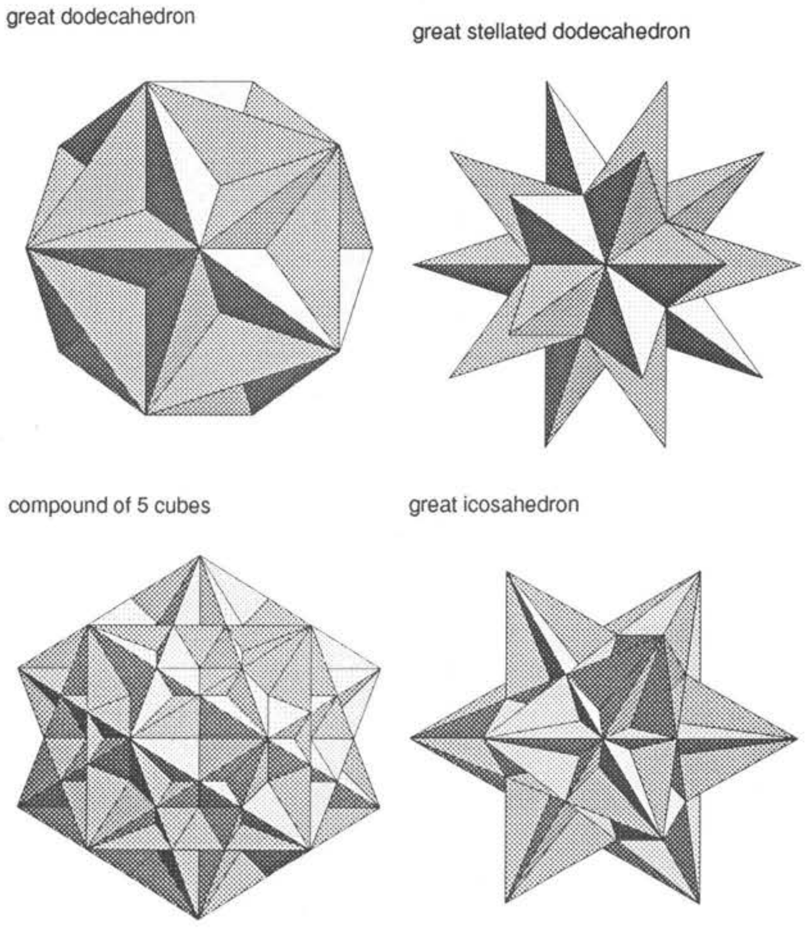


Figure 15.6—example *Draw* files created by a public domain Fortran application from Fortran Friends

16 Epilogue–the Future

One of our philosophers has said that if you want to know where you are going, sometimes you can get a clue by seeing where you have come from. So let's begin our look into the future with a nostalgic look at the past.

Ten years ago the BBC Microcomputer was launched by Acorn in connection with the BBC Computer Literacy Project. Even though the Sinclair Spectrum from the same period sold in greater numbers, the influence of the BBC Micro was unequalled because of its widespread use in schools; this was the machine on which millions of schoolchildren gained their first hands-on experience of computing.

The original BBC Model B had just 32 Kbytes of

RAM; the first disc drives for it offered a mere 100 Kbytes of storage. Its highest resolution screen mode was mode 0, 640 × 256 pixels and 2 colours only. I bought a BBC Micro in 1984 and used it for both work and leisure activities until 1989 when I bought an Archimedes A310; I still own that BBC Micro and it remains in daily use, although by my sons rather than myself.

In all the time that I used that BBC Micro only twice did I run out of memory. When I changed over to the Archimedes with its seemingly massive 1 Mbyte of RAM, 32 times as much as in the Beeb, I reflected that it was unthinkable that I or anyone else could ever run out of memory on so vast a machine. I was wrong. So completely wrong that within a year I was compelled to trade it for a 4-Mbyte Archimedes. And even on that and a 4-Mbyte A5000, running out of memory is an all-too-common occurrence.

Why the difference? If 32 Kbytes were adequate once, why are 4 Mbytes barely adequate now? Because computing suffers from its own version of Parkinson's Law. Applications and data expand to fill the memory available. In common with other ARM users, I am undertaking tasks today that were beyond my wildest dreams when I first became acquainted with the BBC Micro. Requirements and expectations have grown even faster than the hardware.

So, in the past ten years we have progressed from 32 Kbytes of RAM and 100 Kbytes of floppy disc storage to 4 Mbytes of RAM, 1.6 Mbyte floppy discs and, commonly, 100 Mbytes or more of hard disc storage. What can we expect of the Acorn machines that will be available in the early years of the new millennium? Simple extrapolation suggests machines having 512 Mbytes of RAM, 100 Mbytes of demountable or floppy storage and 10 Gbytes (10,000 Mbytes) of fixed storage. And as to their graphics capability, a screen mode of 1280 × 1024 pixels supporting full 24-bit colour does not sound far-fetched; the screen memory requirement of 3840 Kbytes is paltry in such a machine. And all must of course be supported by dramatic increases in processor speed.

And I am prepared to bet that even with such machines as these, running out of memory will remain a problem. Why? How could one possibly fill a machine having 512 Mbytes? It's easy. Table 10.2 suggests one way. There are scanners available now that scan A4 documents at 1200 dpi and in 24-bit colour. The resulting sprite (without compression) occupies 378 Mbytes which goes a long way towards filling 512 Mbytes of RAM. And that, I repeat, is available *now*. And there is a direct-drive laser printer available *now* that will handle A3 sheets at 1200 dpi. The printer image without compression occupies 32 Mbytes and is presumably buffered on hard disc in present machines. And scanner and printer technology will have developed further in 10 years' time. We might reasonably expect to see 24-bit colour laser printers commonplace then. And they would certainly be more demanding in RAM than today's machines.

All this will need to be supported by storage devices of ever increasing capacity. Perhaps the 'floptical discs' currently widely advertised which offer 21 Mbytes of demountable storage will replace the 3.5 inch floppies that they so closely resemble. Improvements in technology may increase their storage capacity to my extrapolated 100 Mbytes or more over the next decade. Already CD-ROMs and WORM drives are offering greater storage than is available on hard disc.

But I confess to uncertainty as to the long-term future of electromechanical storage systems such as floppy and hard drives and even CD-ROM. All suffer ultimately from reliability problems—even the best-made bearings wear out in time. The future of storage media surely lies in purely solid-state devices which of course have no moving parts. Already one company serving the PC world is offering a product called 'Silicon Drive' for use primarily in harsh industrial environments where dust, damp, vibration or electromagnetic interference preclude conventional floppy and hard discs. In Silicon Drive the storage medium itself is a PCMCIA card, a credit-card-sized demountable RAM board which is inserted into a slot on the drive, like a miniature floppy disc. On this tiny card is a staggering 2 Mbytes of RAM backed up by an ultra-thin battery. Flash EPROM-based versions are also available and need no battery. The present system provides for future cards having storage capacities up to 64 Mbytes. No doubt tomorrow's technology will extend their capacity still further. Here is a more likely contender for the first 100-Mbyte 'floppy'.

Silicon Drive connects to the computer's IDE bus and, since IDE is now an Acorn standard, presumably, with suitable software, it could be used now on A5000s and other ARM machines equipped with an IDE interface. At present, expense limits this medium to critical industrial applications, but we all know how prices tumble as technology catches on. And this particular technology has the potential to catch on in fields far broader than just pure computing. For example, within ten years it could revolutionise the audio/video world, superseding today's mechanical CDs and DATs as the ideal medium for digital audio and video. Imagine jogging listening to your favourite music on a solar-powered walkman reading its digital audio from a credit-card sized ROM board!

So much for the hardware—what of the graphics software? Clearly that will develop to make full use of the hardware facilities, removing many of the restrictions that frustrate creativity today. So animations will get longer and will use the full screen, perhaps even in new ultra-high-resolution screen modes. Combining tomorrow's vast RAM banks with today's less demanding screen modes might make sprite-based interactive animation feasible. Animations may acquire sound effects—that is quite feasible now, of course, but beyond the scope of this book. The faster processors may make Bezier curves feasible in vector graphics-based animations, allowing more realistic graphics in flight simulators and other interactive animations. Stereoscopic graphics could use virtual reality-style viewers—or, more cheaply, a split screen monitor display with a special attachment to ensure that each eye sees only its half of the screen.

But otherwise, who knows? The very challenge of the future resides in the certainty that none of us knows what is waiting round the corner. Ten years ago I could not have dreamt I should be writing this today. I have conjectured about what lies 10 years in the future. But the reality may in fact be beyond our wildest dreams.

A1

Appendix 1: The Structure of *Draw* files

A *Draw* file consists of a 40-byte header followed by the objects that make up the drawing. If the file contains any text objects (and in some applications, whether it contains any or not) the first object will be a list of the fonts used (object type 0). If the file originated in a RISC OS 3 version of *Draw* and contains an options object (object type 11) this will follow the font list or, if none is present, the options object will be the first object in the file. The visible objects in the document follow sequentially, from the bottom of the stack (back of the drawing) to the top (front of the drawing). Groups consist of a

36-byte group header (object type 6) followed immediately by the objects in the group. Path objects in *Draw* are composed entirely of 4-byte words and all object lengths are constrained to multiples of 4 bytes.

The *Draw* file header

This begins with the four bytes that make up the ASCII for *Draw*. It is these that identify the file as a *Draw* file. If omitted, most software will issue an appropriate error message such as 'This is not a Draw file'. Optionally, beginning at byte 12, the name of the application in which the drawing originated may be inserted. This is 12 bytes long, any excess bytes being made up be spaces (ASCII &20).

Object types used in *Draw*

All objects begin with a four byte word representing the object type number. Although this appears to allow for a maximum of 4,294,967,296 different object types, which might appear rather optimistic, in practice some applications use the object type number for other purposes, as we shall see.

The second word represents the length of the object in bytes and therefore provides a pointer to the next object.

Since this is an established convention, it is easy to arrange for software that handles *Draw* files simply to skip over any object whose type it does not need to recognise. Indeed, some applications deliberately manipulate object type numbers. In *DrawPlus*, for instance, when an object is moved to a layer, bits 8 to 12 of its object type number are ORed with the layer number. So a sprite object (normally object type 5) on layer 2 will appear to be object type 5 + 512 = 517. When the picture is redrawn, the layer number is extracted from the object type number and, if the layer is visible, the object's original type number is presented to the redraw routine. So only objects on visible layers are redrawn.

Font list (object type 0)—The list begins at the ninth byte. Each entry consists of a single byte representing the number by which the font will be known; the first entry is 1, the second 2 and so on. This is followed by the path name between *!Fonts* and the Outlines and IntMetrics files, directory names being separated by full stops. Each entry is terminated by 0. At end of the list up to three extra 0s may be inserted to round the length of the object to a multiple of four bytes.

Text object (object type 1)—This includes the location and style data followed by the text content in plain ASCII. The font used is represented by a single byte identifying its position in the font list (see above).

Path object (object type 2)—The location and style data are followed by the coordinates of the points through which the path passes.

Sprite object (object type 5)—The location, orientation and scaling data are followed by the sprite data in standard Acorn sprite format. The sprite may include a palette.

Group (object type 6)—A group consists of a header which is 36 bytes long, followed immediately by the objects in the group. The object length word refers to the overall length of the group. The rest of the data include such information as the overall bounding box of the group.

Text area (object type 9)—This consists of a header of variable length followed by the text file in the form in which it was imported. The header includes a number of text column (type 10) objects, *q.v.*, one for each column requested in the text file.

Text column (object type 10=&0A)—These text column objects have a semi-independent existence in that they can be separately selected, sized and moved, but they cannot be copied or sent to the front or back. Since they reside in the header of their parent text area (type 9) object, they behave as though permanently grouped with it. Each text column object is just 24 bytes long, containing only the standard object header and the coordinates of the column it defines. It contains no text data, since this is contained in the initial text area object.

Options object (object type 11=&0B)—Used only in RISC OS 3 versions of *Draw*, this object stores the user's preferences in use when the drawing was saved.

Other packages

Other vector graphics packages use the familiar *Draw* objects outlined above and some use other object types, peculiar to themselves. *Vector*, for instance, uses the following:

Static replicate (object type 102=&66)

Dynamic replicate (object type 103=&67)

Masked object (object type 105=&69)

Radiated object (object type 106=&6A)

Skeleton for replications (object type 107=&6B)

If you load a *Draw*-format file containing objects of these types into any other package than *Vector*, the objects concerned will usually be ignored. Replicated and radiated objects can be converted in *Vector* to separate objects which will be reproduced correctly by *Draw* and other packages accepting *Draw*-format graphics.

In *DrawPlus* the visible object types are the same as in *Draw* but an object type 65637 (=&10065) is also used for storing internal data such as prefered settings.

ArtWorks certainly uses some object types peculiar to itself, but if the drawing is saved as a *Draw* file, these objects are converted to a *Draw*-compatible form first. For instance, a blend object that has *not* been expanded to separate objects will be found to consist simply of the two parent objects which are conventional path objects (object type 2).

A2

Appendix 2: Pixel Graphics SWIs

Following is a selection of operating system routines (SWIs) that will be useful to those wishing to handle sprites from BASIC, Assembler or other languages. All use the call OS_SpriteOp (which must be spelt in exactly that way). The exact operation depends on the contents of the ARM's registers. From BASIC the registers are set up by specifying their contents as a sequence of arguments.

General considerations

Before you can use sprites you must set aside an area of memory in which the sprite definitions will be stored. RISC OS provides its own system sprites area, but its use is now discouraged, current practice being for applications to set up their own user sprites area. If the size of the sprites together with that of the application software exceeds 640 Kbytes, you will need to expand the size of the Next slot either manually from the Task Display before loading the application or by using the appropriate WimpSlot command in the application's *!Run* file.

The operation to be applied is determined by the least significant 8 bits of Register 0, giving each operation a number between 0 and 255. Bit 8 (which adds 256 to the register value) and bit 9 (which adds 512 to the register value) are used as flags to indicate respectively the choice of sprite area and the means of identifying the individual sprite concerned. If both flags are 0, the system sprites area is used and the sprite must be identified by its name. If either flag is set, the user sprite area is used. If bit 8 is set, the sprite will be identified by its name. If bit 9 is set, the sprite will be identified by its position in the area, *i.e.* its position in the sprite file. This is a faster means of locating sprites than by name, since it eliminates the need to search the sprite area for the occurrence of a string. It is illegal to set both bit 8 and bit 9.

Register 1 contains the address of the user sprite area and so is only used if the operation relates to such an area. In BASIC its use is simple. The BASIC application probably created the user sprites area by an instruction such as DIM sprites% 512000. It is sufficient then to specify sprites% as the content of this register. BASIC will look up the address contained in the variable and will copy it into the register. You need never know the exact address.

Register 2 identifies the individual sprite on which the operation is to be performed. If you are identifying the sprite by number, that number is inserted in the register. If you are identifying the sprite by name, the sprite name must be 'indirected' since it may well be too long to be accommodated in a 32-bit register. The register therefore must contain a memory address where the sprite name (terminated by a null) can be found. Again, however, BASIC simplifies this dramatically, it being sufficient to include the sprite name between double quotes as an argument.

Plot action

Some of the calls involve plotting the specified sprite on the monitor screen. The plot action itself offers a wide range of options determined by the contents of one of the registers. These are as follows:

value	action
0	overwrite colour on the screen
1	OR with colour on the screen
2	AND with colour on the screen
3	exclusive-OR with colour on the screen
4	invert colour on the screen
5	leave colour on the screen unchanged
6	AND colour on screen with inverted sprite pixel colour

7	OR colour on screen with inverted sprite pixel colour
8	If set, use mask. Otherwise ignore mask (if present).
&10	Extended colour fill pattern 1
&20	Extended colour fill pattern 2
&30	Extended colour fill pattern 3
&40	Extended colour fill pattern 4
&50	Giant extended colour fill (patterns 1-4 placed side by side)

Note that options 0 to 7 are mutually exclusive. Any of those eight options can be combined with option 8 (by adding 8 to it) and also with any one of options &10 to &50 by adding its number on to it.

OS_SpriteOp 9 Initialise sprite area

R0=9 or 265 (&109) or 521 (&209)

R1=address of user sprite area (if appropriate)

This initialises a sprite area. If the same area has previously been used for sprites, these are all deleted, leaving the sprite area empty. If using the system sprite area, it is equivalent to the OS command *SNEW.

If you are initialising a user sprite area, you must also initialise two words in the area header. The first word (32 bits) must contain the total size of the area and the third word the offset to the first sprite, usually 16. Thus a BASIC program which sets up a user sprite area might include the following line:

```
DIM sprites% 512000:!sprites%=
512000:sprites%!8=16
```

OS_SpriteOp 10 Load sprite file

R0=10 (&0A) or 266 (&10A) or 522 (&20A)

R1=address of user sprite area (if appropriate)

R2=pointer to filename (in a BASIC program the filename alone is sufficient)

This loads the named file into the specified sprite area, overwriting any sprites already present. The sprite area must have been previously initialised. If using the system sprite area it is equivalent to *SLOAD.

For example, in a BASIC program the following line would load a file called 'mysprites' (assumed to be in the currently selected directory) into the user sprite area defined in the previous example:

```
SYS"OS_SpriteOp",&10A,sprites%,
"mysprites"
```

OS_SpriteOp 11 Merge sprite file

R0=11 (&0B) or 267 (&10B) or 523 (&20B)

R1=address of user sprite area (if appropriate)

R2=pointer to filename (in a BASIC program the filename alone is sufficient)

This loads the named sprite file into the specified sprite area without overwriting sprite definitions already present. There must, of course, be sufficient space for the new as well as the existing sprite definitions. After loading, any of the original sprites having names which occur also in the newly loaded sprite file are deleted. If using the system sprites area, the call is equivalent to *SMERGE.

OS_SpriteOp 12 Save sprite file

R0=12 (&0C) or 268 (&10C) or 524 (&20C)

R1=address of user sprite area (if appropriate)

R2=pointer to filename (in a BASIC program the filename alone is sufficient)

This saves all the sprites in the current sprite area as a sprite file. If using the system sprites area, the call is equivalent to *SSAVE.

OS_SpriteOp 14 Get screen area

R0=14 (&0E) or 268 (&10C)

R1=address of user sprite area (if appropriate)

R2=pointer to sprite name

R3=palette flag (0=exclude palette, 1=include palette)

Use this call to copy a portion of the current screen as a sprite. The operating system stores the last two positions of the graphics cursor in a buffer. You should therefore arrange to take the graphics cursor to two diametrically opposite corners of the area to be saved and then issue the call. You must designate a name for the new sprite; it cannot be identified by position in the user sprite area. The call creates the sprite in the designated sprite area; a separate save operation will be needed to save it to a disc or other filing system.

OS_SpriteOp 28 Plot sprite at graphics cursor

R0=28 (&1C) or 284 (&11C) or 540 (&21C)

R1=address of user sprite area (if appropriate)

R2=pointer to sprite

R3=plot action

This call is useful in games where the sprite is moved around the screen in response to the mouse or cursor keys. If the plot action is 3 (exclusive-OR), replotting the sprite in the same position will erase it, restoring whatever detail was present previously.

OS_SpriteOp 33 Flip sprite about its x-axis

R0=33 (&21) or 289 (&121) or 545 (&221)

R1=address of user sprite area (if appropriate)

R2=pointer to sprite

This flips the sprite about its horizontal axis, effectively inverting it. The sprite is only inverted in memory—to see the inverted sprite you must subsequently plot it using an appropriate call.

OS_SpriteOp 34 Plot sprite at user co-ordinates

R0=34 (&22) or 290 (&122) or 546 (&222)

R1=address of user sprite area (if appropriate)

R2=pointer to sprite

R3=x co-ordinate

R4=y co-ordinate

R5=plot action

A fast and easy-to-use routine for plotting a sprite at any required position on the screen without affecting the graphics cursor.

OS_SpriteOp 47 Flip sprite about its y-axis

R0=47 (&2F) or 303 (&12F) or 559 (&22F)

R1=address of user sprite area (if appropriate)

R2=pointer to sprite

This flips the sprite about its vertical axis, effectively mirror-imaging it. The sprite is only reversed in memory—to see the reversed sprite you must subsequently plot it using a plotting call.

OS_SpriteOp 52 Plot sprite at user co-ordinates with scaling

R0=52 (&34) or 308 (&134) or 564 (&234)

R1=address of user sprite area (if appropriate)

R2=pointer to sprite

R3=x co-ordinate

R4=y co-ordinate

R5=plot action

R6=scale pointer (see below)

R7=colour translation table pointer (see below)

This call, because of its versatility, is widely used in graphics software for plotting sprites which must be scaled or for plotting sprites in screen modes other than those in which they originated. When you import a sprite into a *Draw* document, for instance, it is this call that plots it on the screen.

In order to use the scaling facilities you must set apart a block of 16 bytes as a scale table. In BASIC this might take the form DIM scale% 15. This is divided into four four-byte words, respectively the x multiplier, the y multiplier, the x divisor and the y divisor. The multipliers and divisors may be thought of as the numerators and denominators of fractions which determine the size of the sprite being plotted. In this way the call will enlarge or reduce a sprite and can scale the two axes independently if required. The following lines in a BASIC program will cause a sprite called 'mysprite' to apparently 'grow' while remaining centred at a certain position on the screen. It is assumed that the sprite has the same number of colours as the current screen mode, so that no colour translation table is needed.

```
DIM scale% 15:scale%!8=&100:
scale%!12=&100
FOR x%=1TO256
!scale%=x%:scale%!4=x%
WAIT:SYS"OS_SpriteOp",&134,
sprites%,"mysprite",682-(106*
x%/256),64-(62*x%/256),0,scale%
NEXT x%
```

Beware of scaling a sprite whose design includes dither patterns. Since these will not be reproduced consistently at non-native sizes, the sprite is likely to flash in a thoroughly disconcerting manner. (Of course, if you are looking for a psychedelic effect, you might like to try it.)

The colour translation table may be omitted if the sprite was created in a mode having the same number of colours as the mode in which it is being plotted. If, however, you wish to plot in a 256-colour mode a sprite that was created in, say, a 16-colour mode, you must include a colour translation table. This is simply a block of memory, one byte for each colour used in the sprite's *original* mode, which represents colours

0, 1, 2 *etc*. respectively. The contents of each byte represent the number in the current palette of the colour in which this colour should be rendered. In *Draw* and other applications which use this call, the colour translation table is set up automatically by the application, the original colours being matched as closely as possible to the present palette for their RGB content. In some other applications, such as *FontEd*, there is no such automatic creation of colour translation tables and so importing a two-colour sprite into a 16-colour mode screen results in the error message 'Bad colour translation table'.

A3

Appendix 3: Colour in RISC OS

Despite the very attractive graphics that can be produced on the ARM machines, examples of which appear in the colour plate section, RISC OS is in fact quite limited in its colour capabilities. The VIDC which handles output to the screen can only handle 12-bit colour. That is to say, it recognises 16 levels (defined by 4 bits) of each of the primary colours red, green and blue, giving a total of 16 × 16 × 16 = 4096 possible different colours. But no conventional screen mode allows as many as 4096 colours on screen simultaneously. So colour, so far as the monitor display is concerned, is yet another matter of compromise.

The screen modes provided in RISC OS offer a maximum of 2, 4, 16 or 256 simultaneous on-screen colours. And there is a fundamental difference between the 256-colour modes and those offering fewer colours. In modes having less than 256 colours you can redefine the palette to include any selection from the 4096 available. In 256-colour modes, however, you have no such choice; you are restricted to a fixed set of 256 colours.

Note that output to a colour printer is not subject to the same restrictions. If the application stores its colour data in 24-bit format, as most vector graphics and DTP packages do, the full 24-bit colour data will be sent to the printer driver. If the printer is capable of handling 24-bit colour, your graphics will be printed in the full colour stored by the application; this is entirely independent of the monitor display. If your printer is more restricted, then, like the screen display, it will use the closest available approximation to the specified colour.

2-, 4- and 16-colour modes

In screen modes that offer 2, 4 or 16 different colours you may use the sliders in the system palette to adjust any of the current palette colours to any of the 4096 colours of which the system is capable.

However, the colours in the palette are not completely independent of each other. Colour 0 (white by default) is assumed to be the background colour and colour 7 (black by default) is assumed to be the current foreground colour. For the purposes of text anti-aliasing any adjustment to either colour 0 or colour 7 will automatically cause colours 1 to 6 to be set up as six intermediate shades between the new colours 0 and 7. So, if you wish to redefine colours 0 to 7 for your own purposes, adjust colours 0 and 7 first and then colours 1 to 6. Subsequent adjustment of colour 0 or colour 7 may upset your carefully formulated colours 1 to 6.

A 16-colour mode may sound restrictive, but with careful choice of colours and judicious use of dithering, it can produce very attractive graphics—indeed graphics which the casual observer would regard as using a much broader palette. *ArtWorks* includes a carefully contrived 'primary palette' which is supposed to reproduce in 16-colour modes the colours available from full colour ink-jet printers; it gives excellent results although a little deficient at the purple end of the spectrum.

Remember that 16-colour sprites take only half as much memory as 256-colour sprites and this can make a big difference to the size of sprite collections, such as sprite-based animations.

To save precious memory further I have used 4-colour and even 2-colour sprites as part of the background in 16-colour animations. For example, if you need a brick wall as the background, you may well find that with care you can create a sufficiently realistic one using just four colours. Similarly pavements and roadways can often be represented using only two colours, probably two contrasting shades of grey, and skies can be created from cyan and white alone. When shown in conjunction with 16-colour sprites, the fact that part of the scene uses only two or four colours is not at all obvious. Note, however, that colour translation tables will be needed if you wish to display 2- or 4-colour sprites from user applications in 16- or 256-colour modes—see Appendix 2.

256-colour modes

In 256-colour modes you cannot redefine the colours in the palette; you are restricted to the 256 colours which RISC OS offers. If while in a 256-colour mode you drag the sliders on the 16-colour system palette, you will change the currently selected colour, usually at three points on each slider, but only to another of the 'legal' 256.

Although the 256 colours offered form a selection that may seem strange at first, there is a sound technical reason for its content. The selection is made up as follows. The colours are numbered 0 to 255, in binary 00000000 to 11111111. Two bits are allocated to each of the primary colours red, green and blue, giving four levels of each and therefore $4 \times 4 \times 4 = 64$ basic colours. Remember, however, that the operating system recognises 16 levels of each primary—the interval between these primary-colour levels is in fact equal to four operating system levels—the four steps are equivalent to 0, 26, 53 and 80% of full colour. They are quite coarse steps. These 64 basic colours are indeed the same 64 colours that are obtained using BASIC's COLOUR statement in 256-colour modes.

This leaves us with 2 spare bits and these are used to add 0%, 6%, 13% or 20% of white, *i.e.* of red plus green plus blue, to the base colour. These fine shades are steps at the operating system level of 16 steps per colour. And they correspond to BASIC's TINT keyword. So white in a 256 colour mode consists of 2 bits each of red, blue and green giving 80%, plus 2 bits of white which add 20% to each of the primaries giving 100%, or pure white.

This selection does have the advantage that it encompasses the full range of values of each of the primary colours. So, by dithering it ought to be possible to simulate almost any colour. It also has the advantage that an equal number of bits are available for each primary; this helps to maintain hue consistency.

Dithering

Dither patterns are patterns in which two or more colours alternate in order to simulate a third colour that is not available in the palette. There are two sorts of dither pattern: dispersed dot dither and clustered dot dither.

In dispersed dot dithers the dots are pixel-sized and colour density is controlled by the concentration of the dots. These are used primarily on screen, *e.g.* by RISC OS 3 in *Draw* and many other applications. RISC OS 3's dispersed dot dithers are based on a four-pixel matrix. For a dither involving two colours—let's call them dark and light—this allows three intermediate shades: 25% dark + 75% light, 50% dark + 50% light and 75% dark + 25% light. RISC OS 3 also supports a dither pattern using three colours within the matrix of four pixels. These are illustrated in Figure A3.1. Dither patterns of this sort work best when the colours used are similar. Using the small square pixels of modes

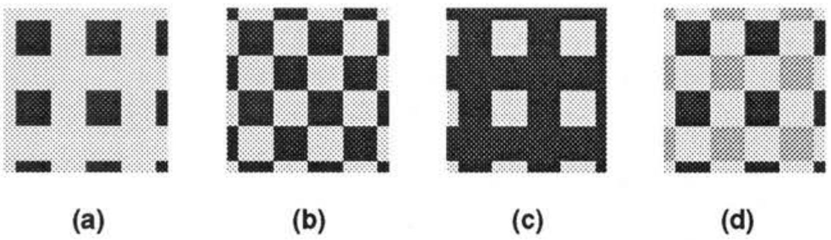


Figure A3.1—four-pixel dispersed dot dithers used by RISC OS 3 to simulate colours not available from the palette. (a), (b) and (c) represent two-colour dithers consisting of 25%, 50% and 75% of the dark colour respectively. (d) represents a dither using three colours in a 50%, 25% and 25% ratio. For best results there should be minimal contrast between the colours used

18-21 or 25-28, dither patterns are often unobtrusive and could be mistaken for solid colours.

Clustered dot dithers in contrast use much larger dots and at a fixed pitch; the strength of the colour is varied by changing the size of the dots each of which is made up of several pixels, *e.g.* 36 (6 × 6). They are generally unsuitable for use on screen as the dots would be too obtrusive, but are ideal for use with printers working at 300 dpi or higher resolutions. They are often used by scanner driver software to render intermediate tones encountered in photographs. For an example of the structure of a clustered dot dither see Figure 10.1 and for an example of an illustration that uses them see Figure 10.2.

24-bit colour

Professional image processing equipment and software uses 24-bit colour, *i.e.* the colours are represented by 8 bits of red, 8 bits of green and 8 bits of blue giving a total of 16,777,216 colours. In fact *Draw* and most other vector graphics/

object-based software for the ARM machines stores its colour information in 24-bit format and so is compatible with 24-bit hardware. It is only the hardware in current Acorn computers that is not capable of displaying 24-bit colour on the screen and therefore replaces shades that are unavailable by the nearest shade (or dither pattern) that is available. It is widely believed that future Acorn computers will handle 24-bit colour on screen. A 640 × 480 pixel screen mode in 24-bit colour would require 900 Kbytes of screen memory.

A4 Appendix 4: Printing Graphics

Any comparison of the merits of various printer types will inevitably be to some extent subjective. 'Quality' is hard to define since what looks good to one observer looks poor to another. So only general guidelines can be given here.

Dot matrix printers

Impact dot matrix printers are primarily text printers. Most have graphics modes and so *can* be used to print graphics. The highest definition available on the Epson FX-80 and compatible 9-pin machines, for instance, is 240 × 216 dpi which requires three passes of the print head for each 1/9 inch of paper, making printing a slow process at the best of times. Some 27-pin dot matrix machines offer higher dot densities of up to 360 dpi.

Dot matrix machines are not recommended for graphics printing for two reasons. Firstly, their ribbons are subject to uneven wear. Consequently within one page and even within the same area of a page the paper is subjected to uneven inking and this looks untidy. Secondly, the mechanism for moving the print head is rarely accurate. Slight inaccuracies which may pass unnoticed on text printouts have a tendency to become glaringly obvious when they occur, for example, in fine vertical lines in graphics.

Ink-jet printers

Ink-jet printers operate by squirting tiny blobs of ink at the paper. In general their printouts are superior to those of dot matrix printers since they offer more consistent inking and more accurate print head placement. Two popular machines are the Hewlett Packard *DeskJet* and the Canon *BJ10EX*. Offering 300 dpi and 360 dpi resolutions respectively, their printouts have been described as 'near laser quality'.

There are four disadvantages to ink-jet printers. Firstly, some are expensive to operate, gobbling costly ink cartridges at an alarming rate if heavily used. Secondly, the ink takes appreciable time to dry, especially when applied in large solid areas as sometimes happens in printing graphics. Even if the paper dries without crinkling or smudging, an accident with a cup of coffee may wash all trace of text and graphics off the paper. Thirdly, the ink is fired through microscopic nozzles which sometimes become clogged. When a nozzle stops firing, fine uninked streaks appear throughout the printout and if one of these corresponds to a thin line, that line will of course not be printed. Fourthly, the paper quality is critical. Some paper absorbs the ink like blotting paper causing it to spread out in unsightly 'whiskers' that ruin graphics.

Despite these disadvantages, ink-jet printers offer an excellent compromise between cost and quality. An important advantage over dot matrix printers is their quiet operation.

Laser printers

Laser printers generally offer consistent blackness and highly accurate positioning of text and graphics. Until recently their only real disadvantage was their price, but this has fallen steadily, while specification has generally improved. Most laser printers offer 300 dpi and some 600 dpi; some newer machines are offering 400, 800 and even 1200 dpi resolutions.

So far as ARM users are concerned, laser printers fall into three categories: PostScript printers which use the Adobe PostScript page description language, direct-drive laser printers and others such as the Hewlett Packard LaserJet and compatible machines. PostScript printers tend to be expensive, but RISC OS does provide an excellent PostScript printer driver. So far as the LaserJet and other machines are concerned, do ensure if you purchase one of these that it contains at least 1 Mbyte of on-board RAM. You will not be able to print a full A4 page of graphics on it if it contains less. And remember that even text which uses the Acorn outline font system is printed as graphics.

Direct-drive laser printing is a field in which the ARM machines are way ahead of some other computer types. The idea is a very simple one. You take a basic laser printer, disable most of its electronics and drive its mechanism (its 'engine') directly from the computer. In practice this means that a special board, a 'podule' or expansion card, must be installed in the computer. It also imposes another limitation on the computer: since the image that the printer puts on the paper must be built up in computer RAM rather than in the printer's RAM, the computer must have sufficient RAM for that massive image as well as applications. In practice this means at least 2 Mbytes of RAM are needed for 300 dpi printing or 4 Mbytes for 600 dpi (A4 paper size assumed). An interesting benefit of direct drive is that, because the ARM is a much faster processor than that normally fitted in laser printers, printing is far quicker than on conventional laser printers. It has been said that ARM3-driven direct-drive laser printers offer the fastest laser printing available anywhere.

The two best-known systems are Computer Concepts' *Laser Direct* which currently offers a 600 dpi option for less than £1000 and Calligraph's *ArcLaser* which now includes an A3 machine having 600 and 1200 dpi facilities.

Colour printing

The systems described above offer monochrome printing only. Colour printing is a little more problematical. Colour laser printers do exist but are very expensive and none is so far offered for the Acorn market. Colour dot matrix printers are comparatively cheap, but they suffer from the same disadvantages as monochrome machines and the colour produced is in general somewhat crude.

Which leaves colour ink-jet printers. For some years, the Integrex printer had the field to itself, but it uses special paper on a roll which is not always convenient. There are two alternatives, both of which use cut paper, although on both, special high-quality paper is recommended. Hewlett Packard's *DeskJet C* offers 300 dpi using cyan, magenta and yellow inks in one combined cartridge. This has the disadvantage that there is no true black; black areas are printed by overlaying all three process colours and the result is sometimes brownish and sometimes bluish. Computer Concepts is about to launch a Canon *BJC800* bubblejet machine which uses separate cyan, magenta, yellow and key (black) cartridges together with a Turbo Driver that is optimised for speed. This machine has a resolution of 360 dpi and can handle A3 paper.

Structure of the printed image

There is a fundamental difference between the structure of the image which appears on the screen and that which any of the printers described above puts on paper. The smallest dot which appears on the screen, a pixel, may be any of the colours in the current palette. A printer, however, can only make marks which are the colour of its ink (or toner). In other words a monochrome printer has a two-colour palette and a colour printer has a four- or five-colour palette, the paper being the additional colour. Variations of colour or intensity therefore can only be provided by dithering.

Both dispersed and clustered dithers can be used, but because at high resolutions (over 300 dpi) even the tiniest inaccuracy can cause unsightly streaks in dispersed dithers, clustered dithers are preferred. When a colour image is printed using clustered dithers, the original is sampled at intervals and the colour at that point is used as the basis of the current cluster. For this reason if you attempt to print an image that already uses dithering, such as Floyd and Steinberg error diffusion, you are likely to obtain disappointing results. Some of the clusters will appear to be in entirely inappropriate colours. Also see chapter 10 on the scaling of sprites for printing.

Other printing systems

Video imagers are more commonly used in the PC world rather than the ARM world, although I have seen a Mitsubishi imager connected to an Archimedes. These units are connected to the computer's *monitor* socket; they print a copy of the monitor image on a roughly postcard-sized Polaroid photographic paper. Both the machine and the paper are expensive (over £1.00 per sheet) and you are, of course, restricted to what you can get on a screen. The prints are very attractive, however, the gloss finish of the paper giving them an expensive look. For some specialist purposes these may be worth considering.

If your graphics need to look even better than a laser printout, you can get a printing company to typeset them for you using professional equipment. This often has a resolution of 1200, 1800 or 2400 dpi. Expect to pay about £5.00 per side of A4.

Most of this equipment uses the PostScript page description language and reads MS DOS discs. So you will need to load the RISC OS PostScript printer driver and select its 'file' option. If you have RISC OS 3 or a utility such as *PCDir* you will be able to save the output direct to an MS DOS disc. Otherwise, save the output to an ADFS disc or RAMdisc and transfer it to a DOS disc later from the PC Emulator using the file exchange utilities provided.

Printing—the practicalities

Files in Acorn *Draw* or sprite format can be printed at any time by dragging them on to the printer driver icon, assuming of course that the printer is on line.

Most graphics software incorporates a print option. If your printer is on line and you have installed the appropriate printer driver, you can leave your printing to the software to handle. You may need to specify the orientation —vertical (portrait) or horizontal (landscape) of the paper. If your graphic is so large that it will not fit on one sheet, you may need to make special arrangements to print over several sheets that will be joined together later. Some application software (such as *Vector*) handles this automatically; over-size *Draw* files can be handled by *Placard.*

If you are using a 1-Mbyte machine you may find that printing is unacceptably slow and you may even get a message stating that there is insufficient memory available to print the document. This is because the printer driver needs memory in the computer to build up an image of the document at the graphics resolution used by the printer. You can release memory for the printer driver in the following ways: (i) quit any unwanted applications; (ii) change the screen mode to mode 0 (don't be deterred by the appearance of the screen, the printout quality will not be affected); (iii) having ensured that the printer driver is correctly set up, delete it from the icon bar. This may sound counter-productive, but it is only the front end that appears on the icon bar and this takes 64 Kbytes. Deleting it will release that precious 64 Kbytes while still leaving the module in memory that is needed for the printing process.

A5 Appendix 5: Suppliers

I am indebted to the individuals and organisations listed below who either supplied software in connection with this book or gave their time and assistance in various other ways. Technical inquiries about the software should be addressed to them. You may find, however, that commercial software and hardware are available from other suppliers.

4Mation

14 Castle Park Road, Barnstaple, Devon EX32 8PA. Software: *Chameleon2, smArt, Vector*

Ace Computing

27 Victoria Road, Cambridge CB4 3BW. Software: *Euclid, Mogul, Projector, Splice, Tween*

Arxe Systems

P.O. Box 898, Forest Gate, London E7 9RG. Software: *PowerShade*

CARVIC Manufacturing

Moray Park, Findhorn Road, Forres, Moray, Scotland IV36 0TP. Software: *DrawAid*

Clare's Micro Supplies

98 Middlewich Road, Rudheath, Northwich, Cheshire CW9 7DA. Software: *Illusionist, Render Bender II*

Computer Concepts

Gaddesden Place, Hemel Hempstead, Herts HP2 6EX. Software: *ArtWorks, Turbo Drivers*

The Data Store

6 Chatterton Road, Bromley, Kent BR2 9QN. Software: *FontFX*

David Pilling

P.O.Box 22, Thornton Cleveleys, Blackpool FY5 1LR. Software: *D2Font, Trace*

Domark Ltd

Ferry House, 51-57 Lacy Road, London SW15 1PR. Software: *3D Construction Kit*

Fortran Friends

c/o Ms Kate Crennell, Greytops, The Lane, Chilton, Didcot, Oxon OX11 0SE. Software: graphics demonstrations of the capabilities of the Fortran programming language. This software is available as 'Shareware 44' from Norwich Computer Services, 96a Vauxhall Street, Norwich NR2 2SD.

The Fourth Dimension

1 Percy Street, Sheffield S3 8AU. Software: ARCtist

Miss Rosemary Harris

Beck Bottom, 9 Moorland Close, Embsay, Skipton, N. Yorkshire BD23 6SG. Software: clip art

ICS

1 Kington Road, West Kirby, Wirral, Merseyside L48 5ET. Software: *DrawBender, Fontasy, Placard*

Irlam Instruments Ltd

Brunel Institute for Bioengineering, Brunel University, Uxbridge, Middlesex UB8 3PH. Hardware/software package: *Proimage*

Longman Logotron

124 Cambridge Science Park, Milton Road, Cambridge CB4 4ZS. Software: *Revelation 2*

Orion Computers Ltd

250 Leyland Lane, Leyland, Preston PR5 3HL. Software: clip art

RISC Developments Ltd

117 Hatfield Road, St Albans, Hertfordshire AL1 4JS. Software: *Typestudio*; Hardware/software package: *Scavenger*

Tekoa Graphics

16 Murray Road, Rugby, Warwickshire CV21 3JN. *Art20, Lyapunov, Mandlbrot*

Third Millenium

4 Pantllyn, Llandybie, Ammanford, Dyfed SA18 3JT. Public domain software specialising in abstract graphics, pattern generators, Julia sets *etc.*

Wiseword Software

16 Murray Road, Rugby, Warwickshire CV21 3JN. Fonts, *Cir-Kit* electronic circuit symbols.

Index

A DABHAND GUIDE

Four years after the launch of the Acorn Archimedes in 1988, there is a surge of interest in graphics using the Acorn 32-bit machines which now include the A3000, A3010, A3020, A4000, A5000 and the A4 portable.

Recent major graphics software releases include Computer Concepts' *Artworks*, Longman Logotron's *Revelation 2*, 4Mation's *Vector* and Arxe Systems' *PowerShade*. It is as though the fraternity of Acorn users has suddenly become aware of the immense power of their machines as graphics workstations.

Recent hardware add-ons include video digitisers for under £100 and scanners that will convert whole A4 pages to graphics in full colour.

Nevertheless the subject of graphics is in general not well understood. This book surveys the graphics hardware and software for the Acorn 32-bit and, in so doing explains the underlying principles. Its aim is to give you a greater understanding of just what your machine is capable of doing.

If you have ever been baffled by;
- moves, thin lines or winding rules in *Draw*
- scaffolds in *FontEd*
- layers in *DrawPlus*, *Vector* or *ArtWorks*

or if you have ever wanted to know how to;
- change the screen mode of a sprite
- plot sprites or Drawfiles in BASIC
- import images from other computer systems
- prevent fringes from appearing in printouts of scanned images

you will find help in these pages.

Roger Amos is a freelance writer, editor and publicity consultant specialising in the high-technology industries. He has written user manuals and designed fonts for Beebug (now Risc Developments) and wrote Budget DTP on the Acorn Archimedes for Dabs Press.

COLOUR PLATES SECTION

£14.95

ISBN 1-870336-13-5

9 781870 336130