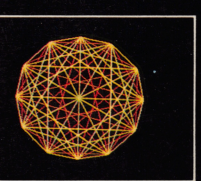
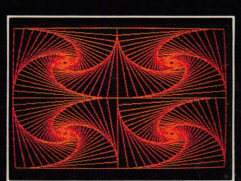
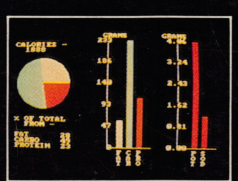
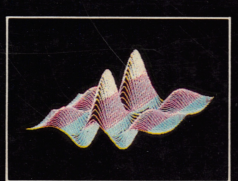
COMPUTER GRAPHICS FOR THE IBM PERSONAL COMPUTER

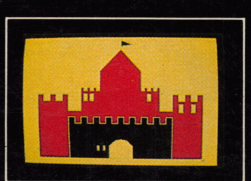
DONALD HEARN AND M. PAULINE BAKER











COMPUTER GRAPHICS for the IBM Personal Computer

COMPUTER GRAPHICS for the IBM Personal Computer

Donald Hearn

M. Pauline Baker

Computer Science Department
Western Illinois University

Prentice-Hall, Inc.
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging in Publication Data

Hearn, Donald.
Computer graphics for the IBM Personal Computer.

1. Computer graphics. 2. IBM Personal Computer—Programming. 3. Basic (Computer program language) I. Baker, M. Pauline. II. Title. III. Title: Computer graphics for the I.B.M. personal computer.
T385.H39 1983 001.64'43 83-4463
ISBN 0-13-164335-5
ISBN 0-13-164327-4 (pbk.)

Editorial/production supervision
and interior design by Kathryn Gollin Marshak
Cover design by Jeannette Jacobs
Manufacturing buyer: Gordon Osbourne

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-164335-5
ISBN 0-13-164327-4 {PBK}

Prentice-Hall International, Inc., *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Whitehall Books Limited, *Wellington, New Zealand*

Contents

List of Programming Examples

List of Color Photographs

Figure A Pictures can be created using the PC character set and the COLOR command.

Figure B Color spectrum displayed by Program 2-8, using character patterns.

Figure C A variety of color shades can be displayed by alternating the color of points plotted in adjacent positions.

Figure D Color pictures can be painted on the screen using the special graphics commands available on the PC.

Figure E Color bar graph produced with the PC character set.

Figure F Color bar graph produced with PC graphics commands.

Figure G A color menu can be used with a light pen to select colors and areas of a picture to be painted.

Figure H Three-dimensional scenes can be created by painting color areas onto the screen. Color areas farther away are painted first, with nearer areas painted over them.

Figure I Color pattern formed with a series of rotated hexagons. Each subsequent hexagon is formed by moving all the vertices of the previously drawn hexagon a small distance along the lines joining the vertices. The hexagon sides are alternately colored cyan and magenta.

Figure J A color surface pattern formed with curved lines using three-dimensional plotting techniques. The lines shown are generated from the equation Y=A*ABS(SIN(B*Z))*EXP(−K*X)*SIN(X). The constants A, B, K and the range of values for X and Z are chosen so as to position the curves on the screen, as discussed in Section 10-6. Variations in color are obtained by plotting different colors in different Y regions.

Figure K Color pattern created with a series of triangles. The design is started with four triangles, positioned to form a rectangular boundary with the magenta side of each triangle. The other two sides of each triangle (cyan) lie along the diagonals of the rectangle. A succession of triangles is then drawn inside each of the original four triangles, with each subsequent triangle rotated slightly and diminished in size.

Figure L A color pattern formed with rotated rectangles. A series of rectangles are drawn inside each of the four adjoining rectangular areas. Each subsequent rectangle is drawn so that its vertices are moved a small distance along the sides of the previous rectangle.

Figure M Three-dimensional bar graph displayed by painting color areas on the screen from the "back" of the graph to the "front".

Figure N Exploded pie chart, produced by the methods discussed in Section 12-1.

Figure O Band chart, using colors to emphasize the areas between data curves.

Figure P Biorhythm graph, plotting the theoretical highs and lows for physical, emotional, and intellectual energy levels.

Preface

In this book we discuss the basic concepts and techniques of computer graphics, and we explore the capabilities of the IBM Personal Computer (PC) for graphics applications. Methods for creating two- and three-dimensional pictures and graphs are considered, together with ways to manipulate and animate our displays. We look into the makeup of the PC, and we examine the graphics features of the PC's BASIC in detail.

Our discussion is arranged in five parts. Part I is about the IBM Personal Computer. We see what makes the system tick, how the different hardware components function, and what options are available for expansion boards, video monitors, and other input/output devices. We also take an introductory look at the software capabilities of the PC in this first part.

In Part II, we introduce fundamental methods for constructing pictures and graphs in two dimensions. We see how to create displays using the alphabet and special graphics characters or using graphics commands and pixels. Color, shading, and the use of light pens, joysticks, and tablets in graphics programs are investigated.

Techniques for manipulating displays are taken up in Part III. We present procedures for translating, scaling, and rotating displays, and then use these ideas as the basis for a detailed development of animation. We also consider how spotlighting and clipping of pictures and graphs can be incorporated into our graphics programs. Applications are discussed for both character and pixel displays.

Three-dimensional graphics is introduced in Part IV. Here we look at methods for erasing hidden lines, for developing perspective views of objects, and for performing three-dimensional transformations on both pictures and graphs.

Applications of computer graphics in business, education, and the home are

surveyed in Part V. Topics discussed in this final part include additional graph-drawing techniques, simulations, computer-assisted instruction, household budget charts, nutrition charts, and game playing.

The graphics methods and applications discussed in this book are illustrated with programs written in BASIC. We developed and tested all programs on an IBM Personal Computer running under DOS 1.1. Suggestions for extensions and revisions to the programming examples are given in the main discussion and in the list of programming projects at the end of the chapters.

Donald Hearn
M. Pauline Baker

COMPUTER GRAPHICS for the IBM Personal Computer

Part I

THE IBM PC

To create pictures and graphs with our Personal Computer, we need to understand the operational features of the system. So we begin with a survey of hardware and software components and how our graphics creations are to be input, processed, and displayed.

Chapter 1

System Overview

IBM's Personal Computer, the "PC," provides us with a variety of graphics capabilities. Pictures, graphs, charts, and animation are all possible in both text and pixel graphics modes. But before we get into a discussion of these graphics capabilities, we should take a tour of the PC system and get acquainted.

The basic hardware components of the PC are the system unit, keyboard, and a video display monitor. To this beginning setup, we can add a number of input/output (I/O) devices, several different types of monitors, and numerous software options. Figure 1–1 shows one possible system configuration.

1–1 SYSTEM UNIT

Here we have the activity center for all of the PC's operations. This cabinet contains the system board (with memory, central processor, and expansion slots), the power supply, and a speaker. As options, we can add one or two floppy disk (diskette) drives or hard disk drives. The system unit layout is diagrammed in Fig. 1–2.

SYSTEM BOARD

An Intel 8088 is the central processing unit (CPU) for the PC. This microprocessor chip sits on the system board, processing our instructions and serving as the main control for all system operations. The 8088 is a 16-bit CPU chip that gives us fast processing speed and the ability to address over 1 million bytes of main memory locations.

Main memory, also called internal memory, comes in two denominations: random-access memory (RAM) and read-only memory (ROM). RAM is the kind

Figure 1–1 The IBM Personal Computer in a system configuration composed of the monochrome display, keyboard, 80-character matrix printer, and system unit, with two diskette drives installed.

used to temporarily store a program or a data file, while we are working with it. Sometimes RAM is called read/write (R/W) storage because the system can both read from and write into this type of memory. For the PC, we can add RAM chips to the system board in units of 64K (65,536 bytes on each chip). There is room for four such chips, so that the "smallest" PC system would have 64K of RAM and three empty spots. Filling in these three spots gives us a total of 256K of memory on the system board. (We can add more RAM by putting memory chips on the option boards that plug into the expansion slots.) ROM is the type of memory used for permanent storage of the system programs and data. The system board holds several ROM chips that are used to store the cassette operating system, BASIC interpreter, I/O drivers, disk loader, and patterns for the graphics characters.

On the system board we also find a number of switches. These are called dual in-line package (DIP) switches. We use them to tell the PC about our system configuration. DIP switch settings indicate the amount of RAM memory we have installed, the type of monitor we have, the number of disk drives attached, and the type of display adapter board we are using.

OPTION BOARDS

The back end of the system board contains five expansion slots, numbered 1 through 5 from left to right (Fig. 1–2). By inserting option boards into these slots,

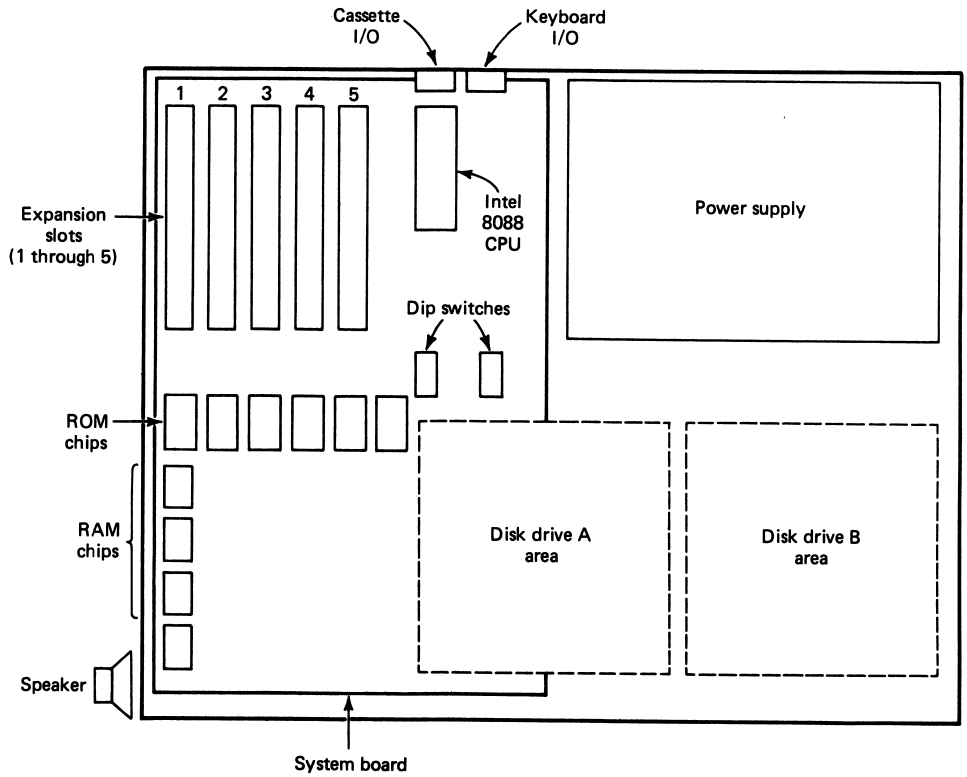


Figure 1–2 System unit layout, showing locations of the system board, power supply, disk drives, and I/O connections.

we can increase the RAM available or we can select one or more of the system options. At least one slot must be used for the board that, among other things, contains the system option to run the display monitor.

Two primary video display option boards are available. They are the Monochrome Display and Printer Adapter board and the Color/Graphics Monitor Adapter board. Both use a Motorola 6845 chip to control operation of the video monitor. With the Monochrome board we can attach the IBM monochrome monitor, giving us characters of exceptionally high quality on a green phosphor screen. We can make pictures and graphs with the Monochrome board from the symbols provided in the ROM character set. This board also includes a connector for the IBM printer. Alternatively, we can use the Color/Graphics board and connect a color monitor or TV. This board lets us make use of special graphics commands and up to 16 colors. There is no connector for a printer on the Color/Graphics board, so we need to use a second expansion slot for the Parallel Printer Adapter board. We could use both the Monochrome and Color/Graphics boards if we wanted to support both the IBM monochrome display and a color monitor. Additional boards are available that allow graphics commands to be used with the monochrome monitor, but color, of course, would still not be possible.

Several other options are available to us by using additional boards in the expansion slots. The Diskette Drive Adapter board is needed if we want to attach disk drives to our PC. The Asynchronous Communications Adapter board provides an RS-232 port for attaching serial devices, such as modems, graphics tablets, and some printers. The Game Control Adapter board allows us to use joysticks or game paddles. We can also use expansion slots to increase the RAM available. An add-on memory board usually starts with 64K and is expandable up to (and in some cases beyond) 256K, in increments of 64K.

With all the options that we might want to add to our PC, it is easy to quickly use up the five expansion slots. Fortunately, various options can be put together on a single board. Combination boards are available with almost any selection of options. Typically, such boards have memory chips, one or two RS-232 interfaces, and a parallel printer port. Some combination boards might also include a clock/calendar, which automatically sets the date and time on system startup, and a game adapter for joystick or paddle attachment.

MAIN MEMORY

We have noted that our PC has the capability to access over 1 megabyte of internal storage. Some of the address areas in this memory space are available to us as RAM for temporarily storing our programs and data files. The other areas are reserved for system use, either as system RAM or system ROM. Figure 1–3 shows the way that storage addresses are partitioned for the various uses. Memory locations are given in hexadecimal.

All RAM addresses are assigned to the first 768K. The ROM areas are addressed as the last 256K of the memory space. Within the RAM addresses, the user space (our available area for temporarily storing programs and data) is

Figure 1–3 Table of internal storage partitions, stating the beginning and ending byte address for each area of main memory and the purpose of each area.

Start address	End address	Memory area description
00000	3FFFF	From 64 to 256K RAM on the system board.
40000	9FFFF	Up to 384K RAM on option expansion boards.
A0000	A3FFF	Reserved 16K RAM area for system use.
A4000	AFFFF	Reserved 48K RAM area for use by the system in setting up video displays.
B0000	B0FFF	Reserved 4K RAM display buffer area on the Monochrome Display and Printer Adapter board.
B1000	B7FFF	Reserved 28K RAM area for video use by the system.
B8000	BBFFF	Reserved 16K RAM display buffer on the Color/Graphics Monitor Adapter board.
BC000	BFFFF	Reserved 16K RAM area for color graphics.
C0000	EFFFF	Reserved 192K ROM expansion area.
F0000	F3FFF	Reserved 16K ROM area.
F4000	FFFFF	Reserved 48K ROM area on the system board.

designated as locations 0 through 9FFFF (640K). Beyond this area, the system makes use of RAM addresses for several purposes. A 4K display buffer, starting at location B0000, is set up for use with the IBM monochrome display. The 4K memory chip on the Monochrome board is accessed with these addresses. A 16K display buffer, starting at location B8000, is used with graphics monitors. The memory chip for this buffer area is on the Color/Graphics board.

1–2 KEYBOARD

Attached to the system unit through 6 feet of coiled cable is the keyboard, which is our main input device to the PC. Eighty-three keys are positioned on the keyboard to give us a standard typewriter layout, a 10-key numeric keypad, and 10 function keys. We can use the numeric keypad portion of the keyboard for fast data entry, as we would with an adding machine, or we can use these keys for cursor control. We change from one mode to the other (cursor control or numeric data entry) by pressing the NUM LOCK key. This key works like a toggle switch, changing from one mode to the other each time we press it. The 10 function keys, on the left of the keyboard, are set for commonly used commands, such as LIST or RUN. Hitting the appropriate function key is equivalent to typing that command. We can change these function keys to whatever commands we want.

We also get a touch of luxury with our keyboard. The keyboard slant can be set with a tilt adjustment to either a 5-degree or a 15-degree angle. With the coiled cable, we can drag the keyboard around to suit our working environment.

The internal workings of the keyboard are under the control of an Intel 8048 microprocessor chip. This chip is inside the keyboard and continually scans the keys to determine when one has been pressed. It also does a self-test on the keyboard during system startup, performs key-debounce checks, and handles the buffering of up to 20 keys. Buffering allows us to type in a command even while the CPU might still be busy executing our previous command. The second command is then stored in the keyboard's buffer until the CPU is ready for it.

1–3 VIDEO MONITORS

A wide choice of display units can be used with the IBM PC. These video displays attach to connectors on the back of the system unit and include the IBM monochrome display, other types of monochrome or color monitors, and television sets. The operation of a video monitor is based on the standard **cathode-ray tube (CRT)** design.

BASIC CRT OPERATION

Figure 1–4 illustrates the basic operation of a CRT. A beam of electrons (cathode rays), emitted from an electron gun, passes through a focusing and deflection

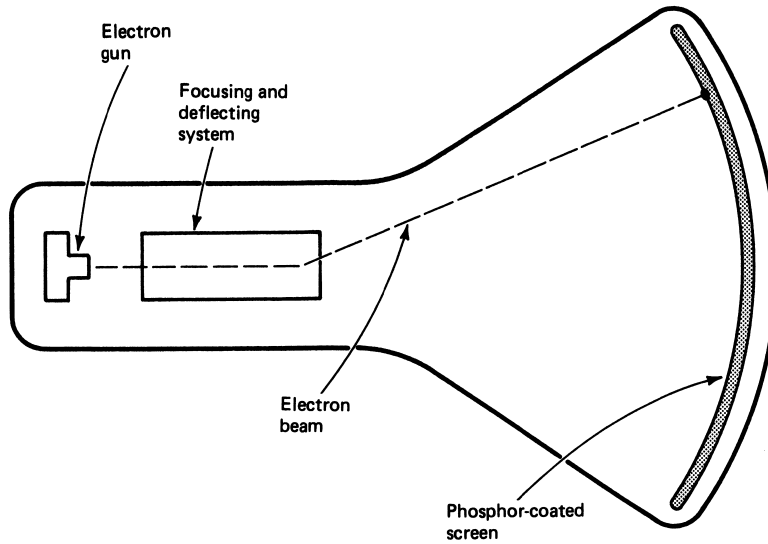


Figure 1–4 Basic operation of a CRT.

system and strikes a phosphor-coated screen. Voltages applied to the electron gun determine the number of electrons emitted. The focusing and deflection system, also controlled by voltages, produces electric and magnetic fields to focus the beam onto a particular spot on the screen. When the electron beam strikes the phosphor coating, the screen lights up at that spot. The intensity of a light spot depends on the number of electrons in the beam. By directing the beam to various points on the screen, we are able to display text or a graphics pattern.

The light emitted by the phosphor coating on a display screen lasts only a small fraction of a second. Therefore, we need some method for maintaining the screen picture so that we can see it. One way to keep the phosphor glowing is to repeatedly pass the electron beam over the same screen points. This type of display is called a **refresh CRT**. It turns out that we need to refresh a screen picture at least 25 to 30 times each second; otherwise, it flickers.

Refresh CRTs used with the PC operate as **raster-scan** displays. These monitors pass the electron beam over all parts of the screen, turning the beam intensity on and off to coincide with the information to be displayed. The electron beam is made to sweep across horizontal lines of the picture tube from top to bottom. This refresh cycle is set up so as to sweep the beam across every other horizontal line on one pass, then return to the top of the screen to sweep across the remaining lines on the next pass. Interlacing of the scan lines in this way helps to reduce flicker, since we essentially see the entire screen display in one-half the time it would have taken to sweep across all the lines from top to bottom in one pass.

How does the CRT know when to turn the beam on and off? This is governed

by reading the pattern of bits stored in the display buffers. Both the Monochrome board and the Color/Graphics board contain memory chips, called display buffers, that are used to store the definition for the screen display. A 4K buffer is on the Monochrome board and a 16K buffer is on the Color/Graphics board. The address areas for these buffers are given in Fig. 1–3.

IBM MONOCHROME DISPLAY

A green phosphor screen is used on the IBM monochrome display. This monitor is driven by a Motorola 6845 chip as a raster-scan CRT with a refresh rate of 50 times a second. Each screen frame is presented in 350 horizontal lines (from top to bottom) with 720 individual dots across each line. Characters displayed on this screen are formed with rectangular dot patterns that are 9 dots across and 14 dots high. This gives us a screen area that contains 25 print lines with 80 character positions in each line.

Two cables connect from the monitor to the system unit. One cable provides the drive interface to the 6845 chip on the Monochrome board through a nine-pin connector. The other cable attaches to the AC power ON/OFF switch on the back of the system unit. In this way, the monochrome monitor is automatically switched on and off with the system unit.

The monochrome monitor weighs about 17 pounds. We can safely place it atop the cabinet of the system unit—a convenient location. If we use a different monitor with our PC, we may not want to do this. A heavy video display could press down and bend the option boards in the expansion slots.

COLOR MONITORS

Using color on a monitor requires some changes to the basic CRT operation. Color is produced by using more than one type of phosphor coating on the screen. Different phosphors emit different-colored light, and combinations of light from two or more phosphors can produce a range of colors.

Displays typically use a **shadow-mask** CRT to produce color. This is the type of CRT used in color TV sets. A shadow-mask CRT has the screen coated with tiny triangular patterns, each containing three different closely spaced phosphor dots. One phosphor dot of the triangle emits a red light, another emits a green light, and the third emits a blue light. This CRT has three electron guns, one for each color, and a shadow-mask grid just behind the phosphor-coated screen (Fig. 1–5). The purpose of the shadow mask is to focus the electrons from each gun so as to strike only the correct color dot in any triangle. Setting intensity levels for the three electron guns sets the color combination for each triangle of phosphor dots so that the triangle appears as one small color point on the screen. Thus, a blue spot on the screen would result from activating only the blue phosphor dot. A white (or gray) area is the result of activating all three dots with equal intensity. Yellow is produced with the green and red dots; magenta is produced with the

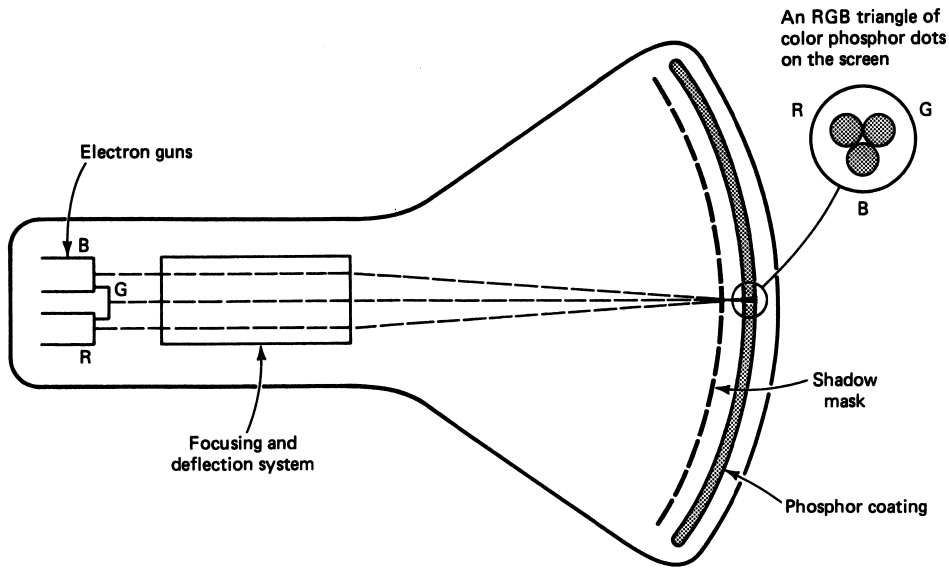


Figure 1–5 Shadow-mask CRT. Three electron guns, arranged in a pattern to coincide with the arrangement of each triangle of color dots on the screen, are directed onto the dot triangles by a shadow mask. The mask allows only the electrons from the B gun to strike the B dot, the R gun to strike the R dot, and the G gun to strike the G dot.

blue and red dots; and cyan shows up when blue and green are activated equally. When no dots are activated, we have a black spot. Eight additional colors (for a total of 16) on the PC are obtained by changing the intensity levels of one or more of the three electron beams.

The raster-scan circuitry on the Color/Graphics board generates 200 horizontal screen lines with either 320 or 640 dots across each line. Three types of color monitors can be attached to the Color/Graphics board: a television set with an RF modulator, a composite monitor, and an RGB (red-green-blue) monitor. These monitors differ in the way the signal is transmitted from the system, and they produce images of varying quality. We can also connect a black-and-white monitor to the Color/Graphics board and do graphics without color.

TELEVISION SETS

To use a color (or black-and-white) TV as a video monitor with the PC, an RF modulator must be hooked up between the TV and the four-pin connector on the Color/Graphics board. The purpose of the RF modulator is to simulate the signal from a broadcast TV station. This means that the color and intensity information of the picture must be combined and superimposed on the broadcast-frequency carrier signal that the TV needs to have as input. Then the circuitry in the TV takes this signal from the RF modulator, extracts the picture information, and

paints it on the screen. We can expect this additional handling of the picture information by the RF modulator and TV circuitry to result in lower-quality images.

COMPOSITE MONITORS

We can get higher-quality pictures by using monitors that eliminate the need for the TV broadcast frequency input. Monitors that are adaptions of TV sets, allowing bypass of the broadcast circuitry, are called **composite** monitors. These display devices still require that the picture information be combined, but no carrier signal is needed. Connection to a composite monitor is made from the "composite signal phono jack" at the rear of the system unit. Since picture information is combined into a composite output and then separated by the monitor, the resulting picture quality is still not the best attainable.

RGB MONITORS

The third type of color monitor, the **RGB** monitor, produces the highest-quality picture image. This monitor takes the intensity levels for each electron gun (red, green, and blue) directly from the system without any intermediate processing. In this way, fewer signal distortions are generated. A nine-pin connector on the rear panel of the system unit outputs color signals directly to the RGB monitor.

1–4 INPUT/OUTPUT OPTIONS

A number of I/O devices can be connected to our PC. Tape cassette players, floppy disk drives, and hard disk drives provide external (or auxiliary) storage. These are the devices that we use to store our programs and data permanently. A five-pin connector at the rear of the system unit is used for the tape cassette, while disk drives are connected to the Diskette Drive Adapter board. Up to four floppy disk drives (either single-sided, double-sided, or a combination) can be attached. Two of these drives can be installed inside the system unit, and two more can be attached through a connector on the rear panel of the system unit.

Several interactive graphics I/O devices are also available for use with the PC. One or two joysticks or up to four game paddles can be operated from the Game board. A light pen, attached to a six-pin connector on the Color/Graphics board, and a graphics tablet, attached to an RS-232 port, provide other forms of interactive input. These devices can be used in graphics programs to construct pictures, to select program options, or to create animated displays.

Both serial and parallel printers can be operated from the system unit. Serial printers are attached through an RS-232 port. Parallel printers are attached through a parallel port, such as the 25-pin connector on the Printer board or on the Monochrome board.

1–5 GRAPHICS PROGRAMMING

The patterns we see on our screen are determined by the contents of the system display buffers. We set the contents of the display buffers with graphics programs written in BASIC, assembly language, or using the UCSD p-System.

Four versions of BASIC are available from IBM. We can write graphics programs using a BASIC interpreter, which comes in three versions, or with the BASIC compiler. The three interpreter versions are called cassette, disk, and advanced BASIC. Varying levels of graphics commands are available on the PC for these versions of the interpreter. Figure 1–6 lists BASIC graphics commands and their availability in the three versions of the interpreter. The BASIC compiler operates under DOS and includes the same commands as the advanced interpreter.

Cassette BASIC is permanently stored in ROM and is available with all system configurations. Disk and advanced BASIC are optional. They come with DOS and provide for operations with the disk drives. Certain enhanced graphics techniques, such as the CIRCLE command, are available only in advanced BASIC. The BASIC compiler is also an optional package. It supplies the capability for increased speed, which is an important consideration in some graphics applications.

Assembly language graphics routines can be written by setting the graphics buffer locations (Fig. 1–3) to the values that produce the desired screen picture. In this case, the "graphics commands" are instructions to store certain numeric

Figure 1–6 List of PC graphics commands and their availability in each of the BASIC language interpreters (cassette, disk, advanced).

Graphics command	Purpose	Cassette	Disk	Advanced
CIRCLE	Draw arcs, circles, ellipses.	no	no	yes
COLOR	Set screen pixel or character colors.	yes	yes	yes
DRAW	Draw outline of specified objects.	no	no	yes
GET, PUT	Animation and clipping.	no	no	yes
LINE	Draw lines or rectangles.	yes	yes	yes
ON PEN	Branching command used with the light pen.	no	no	yes
ON STRIG	Branching command used with joysticks or paddles.	no	no	yes
PAINT	Paint colors into specified screen areas.	no	no	yes
PEN	Provide coordinate and status input from the light pen.	yes	yes	yes
POINT	Check the color of a specified screen point.	yes	yes	yes
PSET, PRESET	Plot or erase specified points.	yes	yes	yes
SCREEN command	Set resolution mode, turn color on/off, and set pages for display.	yes	yes	yes
SCREEN function	Check ASCII code or color attributes of a specified character position.	yes	yes	yes
STICK	Provide coordinate input from joysticks or paddles.	yes	yes	yes
STRIG	Provide button input from joysticks or paddles.	yes	yes	yes

values into certain buffer locations and to display the buffer contents. We could also do this with the BASIC language PEEK and POKE statements. The UCSD p-System, which supports both the FORTRAN-77 and Pascal languages, features "turtlegraphics" methods.

Part II

BASIC GRAPHICS

Now let us see how to create graphics displays with BASIC on our PC. We start with simple picture-drawing concepts, then consider techniques for generating graphs, curved lines, and interactive displays.

Chapter 2

Making Pictures: Character Style

The PRINT statement and the PC character set give us a simple and often effective means for doing graphics. Character graphics is a technique that is available whether we have the Monochrome or the Color/Graphics option. So, to get started, we first see what we can do using this method.

2–1 CHARACTER GRAPHICS CONCEPTS

We can use any character on the keyboard (letters, digits, or other symbols) in PRINT statements to construct graphics patterns. A box outline is displayed with the statements

```
10  PRINT "************************"
20  PRINT "*                      *"
30  PRINT "*                      *"
40  PRINT "************************"
```

We make oversize letters with a program segment such as

```
10  PRINT "IIIIIIII  BBBBBBBBB    MMMMMM          MMMMMM"
20  PRINT "IIIIIIII  BBBBBBBBBB   MMMMMMM        MMMMMMM"
30  PRINT "  IIII      BBB   BBB    MMMMMM    MMMMMM"
40  PRINT "  IIII      BBBBBBBBB    MMMMMMM  MMMMMMM"
50  PRINT "  IIII      BBBBBBBBB    MMMM  MMMM  MMMM"
60  PRINT "  IIII      BBB   BBB    MMMM   MMM   MMMM"
70  PRINT "IIIIIIII  BBBBBBBBBB   MMMMMM    M    MMMMMM"
80  PRINT "IIIIIIII  BBBBBBBBB    MMMMMM         MMMMMM"
```

Program 2–1 Snowflake pattern using keyboard characters.

```
10 'PROGRAM 2-1. SNOWFLAKE USING CHARACTERS & PRINT STATEMENT
20  CLS
30  PRINT  "                    **                      "
40  PRINT  "                    **                      "
50  PRINT  "                  **  **                    "
60  PRINT  "              ##    **    ##                "
70  PRINT  "  **            ##<<**>>##            **    "
80  PRINT  " *%%****       *[<<oo>>]*       ****%%*   "
90  PRINT  "   $****  o     <<oooo>>     o  ****$      "
100 PRINT  "  *   &&<<mm     <<oo>>     mm>>&&    *    "
110 PRINT  "     oo<<>>     ^^ oo ^^     <<>>oo         "
120 PRINT  "          *pp     & ** &     qq*            "
130 PRINT  "        pp  >>& ****** &<<  qq              "
140 PRINT  "                  **  **                    "
150 PRINT  "        bb  >>& ****** &<<  dd              "
160 PRINT  "         *bb     & ** &     dd*            "
170 PRINT  "      oo<<>>     vv oo vv     <<>>oo         "
180 PRINT  "  *   &&<<ww     <<oo>>     ww>>&&    *    "
190 PRINT  "   $****  o     <<oooo>>     o  ****$      "
200 PRINT  " *%%****       *[<<oo>>]*       ****%%*   "
210 PRINT  "  **            ##<<**>>##            **    "
220 PRINT  "              ##    **    ##                "
230 PRINT  "                  **  **                    "
240 PRINT  "                    **                      "
250 PRINT  "                    **                      "
260 END
```

In Prog. 2–1 we produce a pattern using several keyboard characters. The CLS statement on line 20 of this program clears the screen and places the cursor in the upper left corner (the "home" position). Our figure outline is then displayed from this starting position.

We can put graphics patterns anywhere along the available print lines of the screen. The IBM PC provides 25 horizontal print lines with either 40 or 80 characters per line. Lines are numbered from 1 at the top of the screen to 25 at the bottom of the screen. Character positions along each print line are numbered from 1 to 40 or 1 to 80, left to right. We choose which width we want for the print lines using the WIDTH statement. WIDTH 80 gives us the maximum number of characters across a line, and WIDTH 40 provides half as many "double-wide" characters. If we wanted to center the word GRAPHICS using the 40-character width, we would position the word to start at location 17 on line 12. We can do this with the LOCATE statement, which selects line and character starting positions for the next PRINT statement:

```
10  WIDTH 40
20  LOCATE 12, 17
30  PRINT "GRAPHICS"
```

The LOCATE statement sets up the screen position for only one PRINT statement. After executing line 30, the screen cursor will be positioned at the beginning of the next print line: column 1 of row 18. Another PRINT statement, following line 30, would then print in this position. If a character string is too long

to fit along one row, it is either continued on the next row or entirely moved down to start on the next row. A character string will be started at the beginning of the next row whenever we specify it to start after column 1 and it will not fit in the remaining positions of the row that we specified. Row and column positions in a LOCATE statement that are outside the screen limits cause an error message to be displayed.

There are several options with the LOCATE statement that can turn the screen cursor on and off and set the cursor size. For many of our graphics applications, we will only need to use LOCATE to set the line number (row) and character position (column). We should note that the WIDTH statement clears the screen if we are changing screen modes from 80 characters to 40 characters per line. When we are already in the 40-character mode, a CLS statement is needed to clear the screen.

2–2 CONSTRUCTING CHARACTER PICTURES

Pictures can be set up for display by first sketching the picture outline on graph paper. We then determine the character print position for the picture outline or silhouette from the sketch. Each horizontal line of characters on the graph paper becomes one print line. We have one slight problem with this method. Graph paper is usually divided into squares, but the area occupied by a character on the screen is not square. Although character area in WIDTH 40 is almost square, a character displayed in WIDTH 80 is quite a bit taller than it is wide. If we use square graph paper to determine print positions, the displayed picture will be distorted. Graph paper that more closely matches the dimensions of our characters is provided in Appendix A for both WIDTH 40 and WIDTH 80. Although this graph paper will not duplicate exactly the dimensions found on all monitors, it will give better results than square graph paper. We can also make customized graph paper by printing a page filled with plus signs (+), as shown in Fig. 2–1. Program 2–2 outputs a silhouette of the figure outlined in this picture.

Lines 380 through 430 in Prog. 2–2 are identical. We could reduce these six lines to a single PRINT statement inside a FOR-NEXT loop. Any picture containing repeated or symmetric patterns can be set up with loops to construct the patterns. The pyramid of Fig. 2–2, for example, can be displayed with Prog. 2–3. By not including the variable name STARS in statement 80 and the name COUNT in statement 110 (an option available to us on the PC), we speed up the loop processing. Execution speed of our programs is an important consideration when we are producing complex pictures or when we are animating scenes.

We can also cut down on the number of PRINTs we have to type if we encode the character print positions in DATA statements. This encoded data can then be read from the DATA statements, decoded, and printed. There are a number of ways we can encode the print lines. One possibility is to specify each print line of a picture as a pair of numbers. The first number gives the starting print

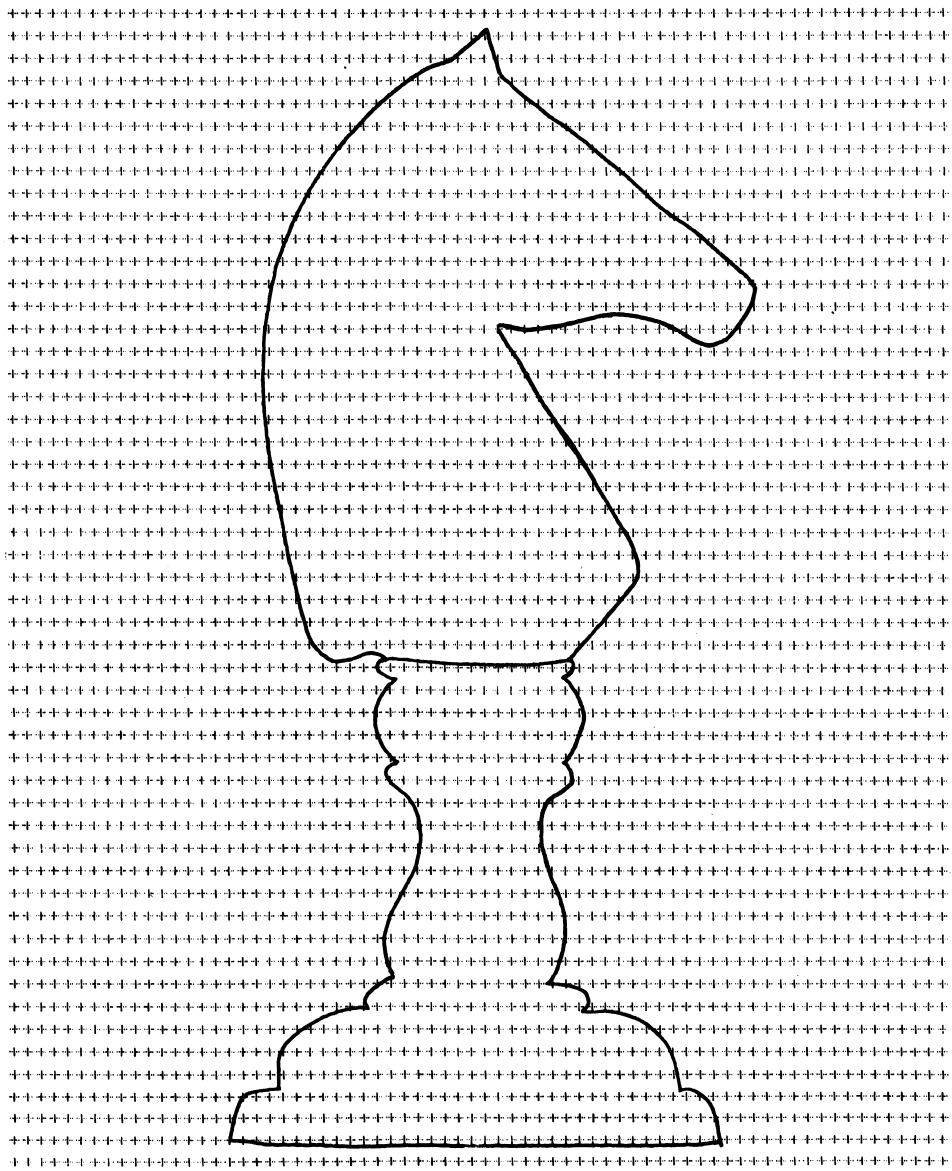


Figure 2–1 A figure outlined on customized graph paper (formed by printing a page of plus signs) is used to determine the character print positions for Prog. 2–2.

position and the second number states how many characters are to be printed on that line. This scheme is used in Prog. 2–4 to produce the chess piece.

2–3 GRAPHICS CHARACTERS

In addition to the characters that appear on the keyboard (letters, numbers, punctuation marks, and other symbols), the PC provides a variety of special

Program 2–2 Figure silhouette (chess piece) using character graphics.

```
10 'PROGRAM 2-2.  CHESSPIECE SILHOUETTE USING CHARACTERS
20  CLS: WIDTH 80
30  PRINT TAB(10);"                    &"
40  PRINT TAB(10);"                  &&&&&"
50  PRINT TAB(10);"                &&&&&&&&&&"
60  PRINT TAB(10);"              &&&&&&&&&&&&&"
70  PRINT TAB(10);"            &&&&&&&&&&&&&&&&&"
80  PRINT TAB(10);"          &&&&&&&&&&&&&&&&&&&&&"
90  PRINT TAB(10);"        &&&&&&&&&&&&&&&&&&&&&&&&&"
100 PRINT TAB(10);"      &&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
110 PRINT TAB(10);"    &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
120 PRINT TAB(10);"    &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
130 PRINT TAB(10);"  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
140 PRINT TAB(10);"  &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
150 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
160 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
170 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&                    &&&"
180 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&"
190 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&"
200 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&&"
210 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&&&"
220 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&&&&"
230 PRINT TAB(10);" &&&&&&&&&&&&&&&&&&&&&&&&"
240 PRINT TAB(10);"  &&&&&&&&&&&&&&&&&&&&&&&&"
250 PRINT TAB(10);"  &&&&&&&&&&&&&&&&&&&&&&&&"
260 PRINT TAB(10);"  &&&&&&&&&&&&&&&&&&&&&&&&&"
270 PRINT TAB(10);"  &&&&&&&&&&&&&&&&&&&&&&&&&"
280 PRINT TAB(10);"   &&&&&&&&&&&&&&&&&&&&&&&&&"
290 PRINT TAB(10);"   &&&&&&&&&&&&&&&&&&&&&&&&"
300 PRINT TAB(10);"    &&&&&&&&&&&&&&&&&&&&&&&"
310 PRINT TAB(10);"    &&&&&&&&&&&&&&&&&&&&&&"
320 PRINT TAB(10);"      &&&&&&&&&&&&&&&&&&&"
330 PRINT TAB(10);"           &&&&&&&&&&&&&"
340 PRINT TAB(10);"         &&&&&&&&&&&&&&&&"
350 PRINT TAB(10);"         &&&&&&&&&&&&&&&&"
360 PRINT TAB(10);"          &&&&&&&&&&&&&&"
370 PRINT TAB(10);"           &&&&&&&&&&&&&"
380 PRINT TAB(10);"            &&&&&&&&&&&"
390 PRINT TAB(10);"            &&&&&&&&&&&"
400 PRINT TAB(10);"            &&&&&&&&&&&"
410 PRINT TAB(10);"            &&&&&&&&&&&"
420 PRINT TAB(10);"            &&&&&&&&&&&"
430 PRINT TAB(10);"            &&&&&&&&&&&"
440 PRINT TAB(10);"          &&&&&&&&&&&&&&&"
450 PRINT TAB(10);"          &&&&&&&&&&&&&&&"
460 PRINT TAB(10);"          &&&&&&&&&&&&&&&"
470 PRINT TAB(10);"          &&&&&&&&&&&&&&&"
480 PRINT TAB(10);"        &&&&&&&&&&&&&&&&&&&&&&&&"
490 PRINT TAB(10);"    &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
500 PRINT TAB(10);"    &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
510 PRINT TAB(10);"    &&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
520 PRINT TAB(10);"&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
530 PRINT TAB(10);"&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
540 END
```

characters. We can get these symbols using the CHR$(AC) function, where AC is the ASCII code of the character. ASCII codes have values from 0 through 255 (Appendix B). Some of the values are used as control codes and do not produce a character. The statement PRINT CHR$(7), for instance, beeps the speaker, and PRINT CHR$(12) clears the screen. The other ASCII values are codes for characters that we can display on the screen. They include the characters that are on the keyboard, foreign language characters, special business and mathematical

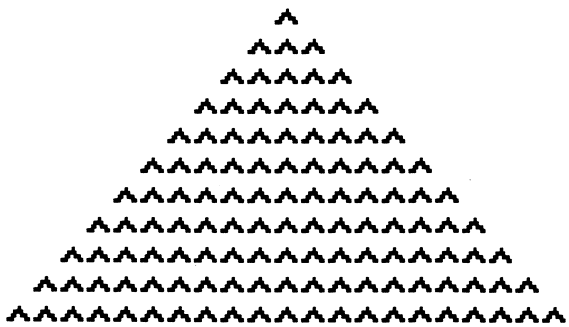


Figure 2–2 Symmetrical patterns, such as this pyramid output of Prog. 2–3, can be produced with loops that minimize the number of PRINT statements.

symbols, and characters designed expressly for producing pictures, graphs, and charts.

We can use any of the special character codes to create screen patterns with the Monochrome option or when in the text mode with the Color/Graphics option. The box outline that we previously made with asterisks can be formed with continuous lines using some of the PC's special characters. Program 2–5 shows how we can use the ASCII character codes in PRINT statements to produce the box of Fig. 2–3. We could also make a single-line box by using codes 179, 191, 192, 196, 217, and 218. Characters formed with combinations of single and double lines are also available.

Although we do not have keys for the special symbols, there is a way to access them from the keyboard. While holding down the ALT key, we type the ASCII code of the desired character on the numeric keypad. This character then appears on the screen. In this way, we can repeatedly type in codes and put together a string of symbols to form a picture. Thus the box pattern of Fig. 2–3 is also produced by Prog. 2–6, using this method.

Pictures displayed at positions specified by input, as in Prog. 2–5 or Prog. 2–6, could run off the screen or scroll down. To prevent such possibilities, we can include checks of the input positions. If a specified position is so close to a screen edge that the picture would go off screen, we can reject that input and ask for another.

Program 2–3 Symmetrical pattern (pyramid) using character graphics and program loops.

```
10 'PROGRAM 2-3. DRAW PYRAMID
20  CLS: WIDTH 40
30  COLUMN = 20
40  FOR COUNT = 1 TO 21 STEP 2
50      PRINT TAB(COLUMN);
60      FOR STARS = 1 TO COUNT
70          PRINT "^";
80      NEXT
90      PRINT                          'GO TO NEXT PRINT LINE
100     COLUMN = COLUMN - 1
110 NEXT
120 END
```

Program 2–4 Chess piece silhouette using character graphics and encoded data.

```
10 'PROGRAM 2-4. CHESSPIECE SILHOUETTE WITH ENCODED DATA
20      'EACH PRINT LINE IS STORED AS A PAIR OF NUMBERS.
30      'FIRST NUMBER IS POSITION TO START PRINTING,
40      'SECOND NUMBER IS HOW MANY CHARACTERS TO PRINT.
50      'INPUT IS TERMINATED BY READING 0,0.
60      'CHARACTER TO PRINT CAN BE CHANGED IN LINE 110.
70  CLS: WIDTH 80
80  READ POSITION, NUMBER
90  WHILE POSITION <> 0 AND NUMBER <> 0
100     PRINT TAB(POSITION);
110     FOR COUNT = 1 TO NUMBER
120         PRINT "#";
130     NEXT
140     READ POSITION, NUMBER
150 WEND
160 DATA 28,1,25,5,22,9,20,13,19,16,17,20,16,23,15,26,14,29,14
170 DATA 32,13,35,13,36,12,37,12,36,12,19,44,3,12,20,12,21,12,22
180 DATA 12,23,12,24,12,25,13,25,13,26,13,27,14,27,14,26,15,24,15
190 DATA 23,17,19,22,13,20,17,20,17,21,15,22,13,23,11,23,11,23,11
200 DATA 23,11,23,11,23,11,21,15,21,15,21,15,21,15,17,23,13,31,13
210 DATA 31,13,31,10,37,10,37,10,37,0,0
220 END
```

The many different characters available on the PC give us a wide range of picture possibilities. Figure 2–4 illustrates some shapes that we can form with characters. We can even introduce shading into our pictures by a careful choice of the characters we use. We can go from solid areas (code 219) to lightly shaded areas (codes 249 and 250). Codes 219 through 223, as well as the regular keyboard characters, give us lots of in-between textures. Three of the special character codes (176, 177, and 178) are designed specifically for shading. A few examples of character shading and texture patterns are found in the scene of Fig. 2–5.

Program 2–5 Box pattern using ASCII character codes in PRINT statements.

```
10 'PROGRAM 2-5.  BOX MADE FROM SPECIAL SYMBOLS
20  CLS: WIDTH 80
30  INPUT "ROW AND COLUMN PLACEMENT FOR UPPER LEFT CORNER"; ROW, COLUMN
40  INPUT "HOW LONG AND HOW TALL TO MAKE BOX"; HOWLONG, HOWTALL
50  CLS: WIDTH 40
60  LOCATE ROW,COLUMN
70  PRINT CHR$(201);                          'UPPER LEFT CORNER
80  FOR COUNT = 1 TO HOWLONG - 2
90      PRINT CHR$(205);
100 NEXT
110 PRINT CHR$(187)                           'UPPER RIGHT CORNER
120 FOR COUNT = 1 TO HOWTALL
130     PRINT TAB(COLUMN);CHR$(186);TAB(COLUMN + HOWLONG - 1);CHR$(186)
140 NEXT
150 PRINT TAB(COLUMN);CHR$(200);              'LOWER LEFT CORNER
160 FOR COUNT = 1 TO HOWLONG - 2
170     PRINT CHR$(205);
180 NEXT
190 PRINT CHR$(188)                           'LOWER RIGHT CORNER
200 END
```

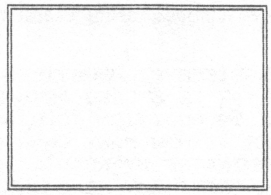


Figure 2–3 Box pattern formed with characters by Prog. 2–5.

Program 2–6 Box pattern using the ALT key and the numeric keypad to print special characters.

```
10 'PROGRAM 2-6.  BOX MADE FROM SPECIAL SYMBOLS
20  CLS: WIDTH 80
30  INPUT "ROW AND COLUMN PLACEMENT FOR UPPER LEFT CORNER"; ROW, COLUMN
40  INPUT "HOW LONG AND HOW TALL TO MAKE BOX"; HOWLONG, HOWTALL
50  CLS: WIDTH 40
60  LOCATE ROW,COLUMN
70  PRINT "╔";                          'UPPER LEFT CORNER
80  FOR COUNT = 1 TO HOWLONG - 2
90      PRINT "═";
100 NEXT
110 PRINT "╗"                           'UPPER RIGHT CORNER
120 FOR COUNT = 1 TO HOWTALL
130     PRINT TAB(COLUMN);"║";TAB(COLUMN + HOWLONG - 1);"║"
140 NEXT
150 PRINT TAB(COLUMN);"╚";              'LOWER LEFT CORNER
160 FOR COUNT = 1 TO HOWLONG - 2
170     PRINT "═";
180 NEXT
190 PRINT "╝"                           'LOWER RIGHT CORNER
200 END
```

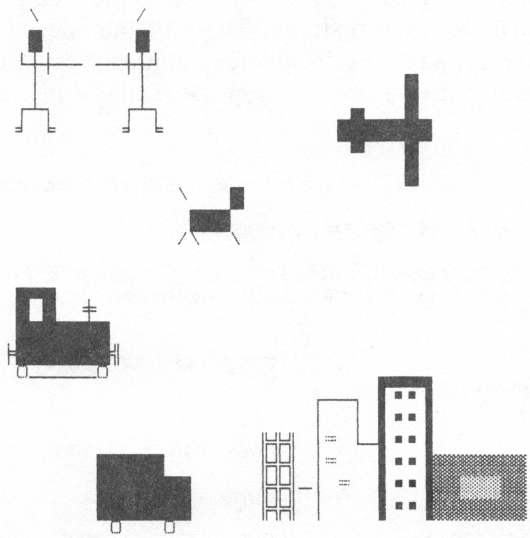
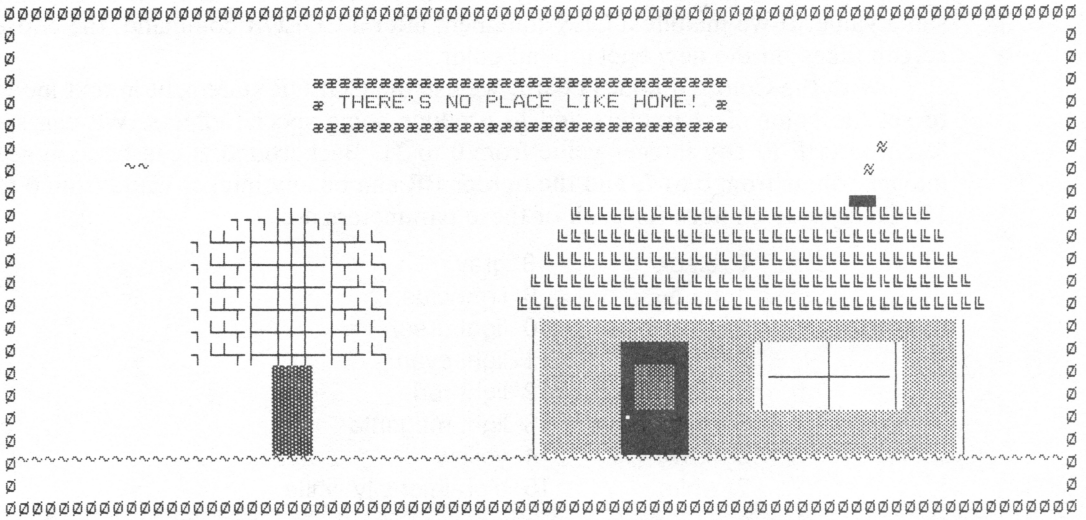


Figure 2–4 Shapes formed with character patterns.

2–4 SPECIAL EFFECTS AND COLOR

There are several options available to us for producing special effects with displayed characters. These options include blinking, highlighting, and underlin-

Figure 2–5 A picture formed with characters.



ing. If we have the Color/Graphics board, characters can also be displayed in color. We choose all of these different options by setting parameter values in the COLOR statement. With the Monochrome board or in the character (text) mode with the Color/Graphics board, the COLOR statement is used in the form

COLOR F,B,BR — Sets the special effect or color of characters displayed with subsequent commands to F (foreground) with a background B and a screen border BR.

Values for parameters F and B in the COLOR statement are set according to the special effect or color we want to select and according to whether we have the Monochrome or Color/Graphics option. In the Monochrome option, we have the following choices for the foreground parameter F:

0	Black
1	White character with an underline
7	White
15	High-intensity white

Adding 16 to any of these values makes the character blink. So if we set F to the value 17, we get a blinking, underlined, white character. Two values are used for the background parameter B and border parameter BR:

0	Black
7	White

Several COLOR statements may be used in a single program. Each time we use a COLOR statement, the next characters output on the screen are displayed in

the specified foreground color F with the rectangular background area of the character displayed in color B. We get a solid color block if F and B are set to the same value. If we include a CLS statement after a COLOR command, the entire screen takes on the new background color.

With the Color/Graphics board, we use the COLOR statement in text mode to set the color of characters and to produce some special effects. We can set foreground F to any integer value from 0 to 31. Background B can be assigned integer values from 0 to 7, and the border BR can be any integer value from 0 to 15. Color code choices for each of these parameters are

0 black	8 gray
1 blue	9 light blue
2 green	10 light green
3 cyan	11 light cyan
4 red	12 light red
5 magenta	13 light magenta
6 brown	14 yellow
7 white	15 high-intensity white

As with the Monochrome board, adding 16 to the value chosen for F makes characters blink in that color. We can display different-colored characters with multiple COLOR statements, and we erase characters using the same techniques discussed for the Monochrome option. Some monitors may not be able to produce all 16 colors.

Program 2–7 illustrates use of the COLOR command in text mode with the Color/Graphics option by displaying randomly selected colors and characters on the top 23 lines within a blue border. A typical screen pattern output from this program is shown in Fig. 2–6. To enter the character mode, we use SCREEN 0 in line 20 of Prog. 2–7. We will take a closer look at the SCREEN command in Chapter 3. Character codes that are used for line feeds, cursor control, and other operations that produce blank spaces were omitted by the test in lines 40 and 50.

Color shadings can be produced using different characters with different

Program 2–7 Random color pattern using characters.

```
10 'PROGRAM 2-7.  RANDOM CHARACTERS RANDOMLY PLACED IN RANDOM COLORS
20  SCREEN 0: COLOR ,,1: WIDTH 40: CLS
30  AC = INT(254 * RND) + 1                'GET AN ASCII CODE
40  IF AC > 7 AND AC < 14 THEN 30          'DON'T PRINT SOME CONTROL CODES
50  IF AC > 27 AND AC < 33 THEN 30
60  ROW = INT(23 * RND) + 1
70  COLUMN = INT(40 * RND) + 1
80  FOREGROUND = INT(32 * RND)
90  BACKGROUND = INT(8 * RND)
100 COLOR FOREGROUND, BACKGROUND
110 LOCATE ROW,COLUMN
120 PRINT CHR$(AC);
130 GOTO 30
140 END
```

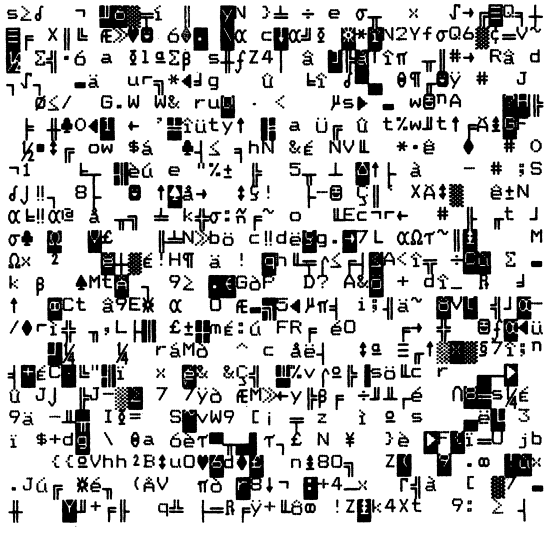


Figure 2–6 Random pattern formed with characters by Prog. 2–7.

foreground and background combinations. An example is the rainbow pattern shown in Fig. B of the color insert. The graduated shading patterns shown in this figure were displayed by Prog. 2–8.

The color options available to us can help our displays in a number of ways. We can use color to make a display easier to understand. Complex displays can sometimes be clarified by using colors to distinguish different objects or areas. Color can be used to accent or highlight important parts of a display, to add realism to a scene, or to aid design applications. We can also use color for fun, to produce a more interesting or attractive display.

Choice of color combinations for a particular display, whether in character or graphics mode, should be carefully considered. A random selection can produce a glaring, unpleasant effect. Using fewer colors in a display is usually best. A background chosen as the "complement" of one of the foreground colors can be attractive. Complement color combinations include red and blue-green, blue and orange, yellow and purple-blue, and green and magenta. A light background (say, cyan) can be used to good effect with darker colors. If several colors are to be used in a display, a gray background is best. Dark borders around the different color areas can help to reduce clashes when many colors are used.

2–5 PRINTING AND SAVING CHARACTER PICTURES

Screen patterns can be printed on paper by using the LPRINT statement in our graphics programs or with the PrtSc (Print Screen) key on the keyboard. When we press PrtSc and the shift keys together, we get a "hardcopy" of our character picture on the printer.

Program 2–8 Color shading patterns using characters.

```
10 'PROGRAM 2-8.  RAINBOW USING PRINT AND COLOR STATEMENTS
20  SCREEN 0: COLOR 0,0,0: WIDTH 80: CLS
30  FOR COLUMN = 1 TO 80
40      READ FORE,BACK,CODE
50      COLOR FORE,BACK
60      FOR ROW = 2 TO 23
70          LOCATE ROW,COLUMN
80         PRINT CHR$(CODE);
90     NEXT ROW
100 NEXT COLUMN
110 COLOR 7,0,0
120 DATA 4,4,219,4,4,219,4,4,219,4,4,219,4,4,219
130 DATA 4,6,178,6,4,176,4,6,178,6,4,176,6,4,176,4,6,177
140 DATA 6,4,177,4,6,177,6,4,177,4,6,177,6,4,177,6,4,178
150 DATA 4,6,176,6,4,178,4,6,176,6,4,178
160 DATA 6,6,219,6,6,219,6,6,219,6,6,219,6,6,219
170 DATA 6,2,178,2,6,176,6,2,178,2,6,176,6,2,178,2,6,176
180 DATA 6,2,177,2,6,177,6,2,177,2,6,177,6,2,177,2,6,178
190 DATA 6,2,176,2,6,178,6,2,176,2,6,178
200 DATA 2,2,219,2,2,219,2,2,219,2,2,219,2,2,219
210 DATA 3,2,176
220 DATA 2,1,178,1,2,176,2,1,178,1,2,176,2,1,178,2,1,177
230 DATA 1,2,177,2,1,177,1,2,177,2,1,177
240 DATA 1,3,178,1,3,178,1,3,178,1,3,178,1,3,178
250 DATA 1,5,178,5,1,176,1,5,178,5,1,176,1,5,178,1,5,177
260 DATA 5,1,177,1,5,177,5,1,177,1,5,177,5,1,178,1,5,176
270 DATA 5,1,178,1,5,176,5,1,178
280 DATA 5,5,219,5,5,219,5,5,219,5,5,219,5,5,219,5,4,176
290 DATA 0,7,219
300 IF INKEY$ = "" THEN 300
310 END
```

The hardcopy output produced by PrtSc prints characters in the "normal" print style (which is the way characters look on the screen in WIDTH 80). This is the case whether we have displayed our pictures in WIDTH 80 or in WIDTH 40. If we want to get characters printed as we see them in WIDTH 40 on the screen, we must use LPRINT and instruct the printer to go into the double-wide mode. In this way we create our picture directly on the printer, rather than on the screen. We get double-wide character printing by including the function CHR$(14) at the beginning of each LPRINT statement. This tells the printer to print that line with double-wide characters. Since the double-wide mode is automatically turned off at the end of each print line, we have to include CHR$(14) in each LPRINT, as in

```
510  LPRINT CHR$(14); "THIS LINE PRINTS DOUBLE WIDE"
520  LPRINT CHR$(14); "THIS ONE DOES TOO"
```

Whether we use PrtSc or LPRINT, some of the characters that we want in our picture may not get printed. This occurs because ASCII codes for all the different characters are sent from the system unit to the printer, and these two pieces of equipment may not agree on the meaning of these codes. Not all ASCII codes are standardized. Different manufacturers assign different meanings to some of the codes, so a particular printer may print blank spaces or characters

different from those we see in the screen picture. We can correct this situation with "screen-dump" programs. These programs use the information stored in the display screen buffer to activate the printer so as to produce the dot patterns for the PC characters we specified.

We can save pictures on disk or tape using the BSAVE command. They can then be displayed at any later time by loading them into the screen buffer with BLOAD. To use these commands, we need to state the address of the screen buffer area (for either the Color/Graphics or Monochrome option) in the DEF SEG command.

In disk or advanced BASIC we can reference our pictures with any descriptive file name. In cassette BASIC, we must use the file name CAS1. The following illustrates the use of BSAVE and DEF SEG with the Color/Graphics option.

```
1000  DEF SEG = &HB800
1010  BSAVE "RAINBOW",0,&H1000
```

Statement 1000 specifies which area in memory we want to save. Since we want to save a displayed picture, we state the "segment number" of the memory area that is used as the screen buffer. This number has one less zero than the starting address (B8000) of the display buffer (see the memory map table in Fig. 1–3). Thus, we use the segment number &HB800 (hexadecimal) in line 1000. We would change this to &HB000 for the Monochrome option. Any subsequent BSAVEing or BLOADing we do in our programs will use this memory area. Statement 1010 specifies the picture name as RAINBOW, an offset of 0, and a length of 4K (1000 in hexadecimal). This tells the system that we want to save the first 4K bytes of this screen buffer, which has a total size of 16K. Four thousand is the maximum number of bytes that we need to store a character picture (in WIDTH 80). We need 2000 bytes (25 lines times 80 characters per line) for character codes and 2000 more bytes for color information (one byte per character), for a total of 4000 bytes. In WIDTH 40, our total picture size would be 2K. The additional space in this buffer is needed when we use pixel graphics. On the Monochrome board, the total screen buffer size is 4K, since it is set up for character graphics only.

To load this saved display back into the screen buffer for viewing, we can use the program segment

```
2000  DEF SEG = &HB800
2010  BLOAD "RAINBOW", 0
```

Line 2000 again specifies the memory area as the screen buffer on the Color/Graphics board. This address would be changed to &HB000 for the Monochrome option. In line 2010, we state the file name as "RAINBOW" and the offset as 0, so that we load the picture information into the beginning of the screen buffer. No length specification is used with BLOAD.

PROGRAMMING PROJECTS

2–1. Display a star pattern using a number of different characters to produce texture, as in Prog. 2–1.

2–2. Using the LOCATE statement, write a program that will clear the screen and display the word "HELLO" in large letters (each formed with several characters) at the center of the screen.

2–3. Display the following patterns, using loops as in Prog. 2–3.

(a)
```
*******
*****
***
*
***
*****
*******
```

(b)
```
@
@@@
@@@@@
@@@@@@@
@@@@@
@@@
@
```

2–4. Sketch the outline of some figure or scene on graph paper. Fill in the outline with some character and display the silhouette on the screen, as in Prog. 2–2.

2–5. Produce the silhouette for Project 2–4 by encoding the data, as in Prog. 2–4.

2–6. Using the method of Prog. 2–5 or Prog. 2–6, display a word in large letters at the center of the screen that has a box border. Set up the program so that several different box borders (single line, double line, or block) can be selected with input statements.

2–7. Write a program segment that can be used with Prog. 2–5 or Prog. 2–6 to check the input data to be sure that the box pattern will fit on the screen.

2–8. Write a program to produce each of the shapes in Fig. 2–4. Include a LOCATE statement that will position the shape at any screen location specified by input. As an option, allow the color of each shape to be determined by input.

2–9. Write a program to produce the scene in Fig. 2–5.

2–10. Modify the program of Project 2–9 to color each part of the scene.

2–11. Display any picture, using encoded data in a DATA statement that will allow several different characters on each line (as in Fig. 2–5). Data for each print line is to include the following information: (1) the ASCII code for each character on the line, (2) the starting position of the character on each print line, (3) the number of consecutive positions occupied by each character, and (4) the color of each character. The ASCII code will be used in the CHR$ function to print out that character.

Chapter 3

Making Pixel Pictures

Special graphics commands, available to us on the PC with the Color/Graphics option, provide another means for making pictures. Instead of building up displays with characters, these commands enable us to "draw" pictures on the screen using points and lines.

3–1 PIXEL GRAPHICS CONCEPTS

Each character that we display on a video screen occupies a small rectangle. This area is subdivided into a grid of even smaller rectangles that we can use with graphics commands. These smaller rectangles are called **picture elements** or **pixels**, or simply **points**. The number of pixels contained in a single character generated by the Color/Graphics board is 8 points vertically and 8 points horizontally. Figure 3–1 illustrates this 8 by 8 **pixel grid** corresponding to the area occupied by a character. A video screen will then contain eight times as many points as characters across the screen and eight times as many points vertically. Since the PC can display a maximum of 80 characters across 25 print lines, we can plot a point in any one of 640 positions horizontally across the screen and in any one of 200 vertical positions. With the 40-character width, we get double-wide characters and double-wide points, so that 320 horizontal positions are available for plotting pixels.

Plotting a point on a video screen means that we instruct the computer to "turn on" the small rectangle of light at the specified pixel position. Individual pixel positions are referenced by **coordinates**. That is, we must specify the location of a pixel as a pair of integers (X,Y). The first integer, X, gives the horizontal distance across the screen, and the second integer, Y, gives the vertical distance.

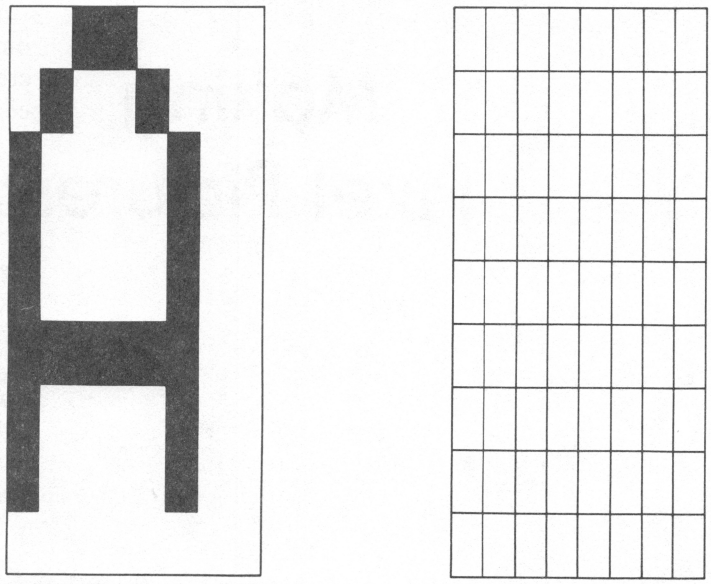
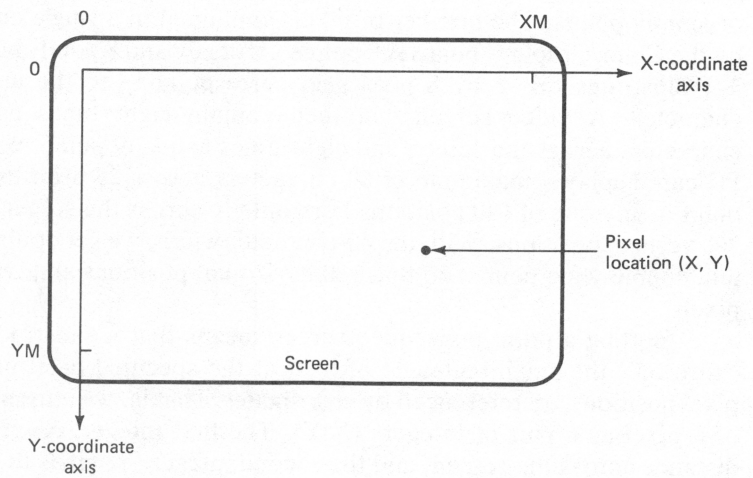


Figure 3–1 The area occupied by a character is divided into an 8 by 8 grid of smaller rectangular pixels.

Our IBM PC requires that these distances be measured from left to right and top to bottom. This means that the screen is referenced with the **origin** of the coordinate system at the upper left corner, as shown in Fig. 3–2. The X, or horizontal coordinate, can range from left to right through integer values 0, 1, 2, ..., up to

Figure 3–2 The coordinate system used by the PC places the origin at the upper left corner of the screen.



either 319 or 639. The Y, or vertical coordinate, can range from top to bottom through the integer values 0, 1, 2, ..., 199. For this coordinate reference, the position of the pixel at the upper left corner of the screen is (0,0). Pixel coordinates for the point at the lower right corner are specified as (XM,YM), where XM is either 319 or 639 and YM is 199.

The number of pixels that we can plot along a line is referred to as the **resolution** of the system. More precisely, resolution is the number of pixels that we can plot per centimeter (cm), in either direction. A larger screen plotting the same number of pixels across its width as a smaller screen will have lower resolution (fewer points per centimeter) in that direction. We can use monitors with different physical dimensions, but changing the screen size does not change the number of pixels we can plot. This number is fixed by the **resolution mode** we choose. The PC provides two resolution modes: **medium resolution** (with a pixel screen size of 320 by 200 for a total of 64,000 points), and **high resolution** (with a pixel screen size of 640 by 200, for a total of 128,000 points).

Resolution modes are selected with the SCREEN statement. This command is used to get us into one of the graphics modes and has the form

SCREEN M, BST, AP, VP — Selects a graphics mode or a text mode.

Parameters M, BST, AP, and VP take numeric values, and not all the parameters need to be specified each time we use this statement. We choose a mode by setting M to a value of 0 (for character, or text, mode), 1 (for medium- resolution graphics mode), or 2 (for high-resolution graphics mode). With BST (the "burst" parameter), we can block color off the screen, allowing black and white only. A value of 0 for BST turns color off in text mode, and a nonzero value turns color off in medium resolution. Reversing these values turns color on. No color is available in high resolution. In text mode, AP and VP are used to select a "page" of text to be written to (AP, the active page) or to be viewed on the screen (VP, the visual page). We can set up eight pages of text (numbered 0 through 7) with WIDTH 40, or we can set up four pages of text (numbered 0 through 3) with WIDTH 80. In Chapter 8, we will explore graphics uses of text pages.

3–2 PLOTTING POINTS

Pixels are directly accessible when we are in one of the graphics modes (SCREEN 1 or SCREEN 2). Once we get into a graphics mode, we can plot a point by stating the command

PSET (X,Y), — Places a pixel on the screen at coordinates (X,Y) with color C. Parameters X, Y, and C may be numeric constants or expressions. If noninteger, they will be rounded.

```
10 'PROGRAM 3-1.  PLOTS A SINGLE PIXEL
20  SCREEN 1: CLS
30  INPUT "ENTER X AND Y COORDINATES"; X, Y
40  CLS
50  PSET (X,Y)
60  END
```

Program 3–1 Plotting a point.

Color parameter C should be kept in the range 0 to 3. A value of zero is the same as background color. Three color options are set with the values 1 through 3. We will take up color in graphics mode a bit later and, for now, just omit this parameter. An example of the use of PSET is given in Prog. 3–1. This program plots a point at a location specified in the input statement after clearing the screen with the CLS statement.

We should not try to plot a point beyond the screen limits. This could produce an error in program execution, an invisible point, or a distortion in the point position due to **wraparound** effects. Wraparound occurs when a point, plotted beyond the limits on one side of the screen, "wraps around" and appears on the other side of the screen. To avoid these problems, neither the horizontal nor the vertical coordinate should ever become negative or exceed the maximum screen locations. We can ensure that we do not attempt plotting beyond the screen boundaries by including the tests of (3–1) in our programs.

$$\begin{aligned} 0 <= X <= XM \\ 0 <= Y <= 199 \end{aligned} \qquad (3\text{–}1)$$

In these tests, XM is the maximum horizontal pixel location allowed (either 319 or 639), and 199 is the maximum vertical pixel location. Program 3–2 illustrates the use of these tests.

For many applications it is convenient to be able to "erase" a point that we previously plotted. We have the following command to accomplish this:

PRESET (X,Y),C — Plots a pixel at position (X,Y) in color C. Parameters X, Y, and C may be numeric expressions or constants. If noninteger, they will be rounded.

Omitting the parameter C in the PRESET statement erases a point previously plotted by replotting that point in the background color. Otherwise, PRESET is

Program 3–2 Point plotting and off-screen tests.

```
10 'PROGRAM 3-2.  CHECK FOR OFF-SCREEN
20  SCREEN 1: CLS
30  INPUT "ENTER X AND Y COORDINATES"; X, Y
40  IF X < 0 OR X > 319 OR Y < 0 OR Y > 199 THEN 80
50  CLS
60  PSET (X,Y)
70  GOTO 100
80  PRINT "COORDINATES OFF-SCREEN. TRY AGAIN.
90  GOTO 30
100 END
```

```
10 'PROGRAM 3-3.  TURN A PIXEL ON AND OFF
20  SCREEN 1: CLS
30  PSET (160,100)
40  FOR DELAY = 1 TO 1000: NEXT
50  PRESET (160,100)
60  END
```

Program 3–3 Point plotting and erasing.

the same as PSET. We can also erase points with the statement PSET (X,Y),0. If the pixel at location X,Y is not turned on, PRESET (X,Y) will have no visible effect. Program 3–3 shows how this command can be used to turn off a pixel after a delay time of about 2 seconds.

Delay loops, as in statement 40 of Prog. 3–3, can be inserted into graphics programs to make points blink. They can also be used to hold a picture for viewing before the next picture is created. These loops give us a delay time of about 1 second for every 500 iterations of the loop.

A random pattern of points that blink on and off is produced by Prog. 3–4. The RND function used in this program will repeat the same pixel pattern each time we run the program. We can get different patterns each time by including the RANDOMIZE statement, but then we will have to respond to RANDOMIZE's request for a "seed" value. Alternatively, we could include a statement using RND(X) before the loop, where X is a negative number. We will need to use a different value for X each time the program is run if we want to generate new patterns. This number can be derived from the TIME$ variable, eliminating the need for any input. TIME$ returns an eight-character string indicating the current time. By taking the two rightmost characters from this string and converting them to a negative number, we get random values for X. The statements

```
25  SEED = VAL(RIGHT$(TIME$,2))
26  STARTRND = RND(-SEED)
```

could be included in Prog. 3–4 to generate a new pattern of random numbers on each run.

Coordinate values in the special graphics commands can be stated as **absolute coordinates** or as **relative coordinates**. We have been using the absolute form, which states coordinates directly in screen coordinates. Relative form specifies the coordinates as displacements, or offsets, from the last point referenced in the program. To indicate relative coordinates, we replace (X,Y) by STEP (DX,DY), where DX and DY give the X and Y displacements. Program 3–5 illustrates the use of relative coordinates by plotting a series of 31 points

```
10 'PROGRAM 3-4.  BLINKING PIXELS
20  SCREEN 1: CLS
30  X = INT(320 * RND)
40  Y = INT(199 * RND)
50  PSET (X,Y)
60  FOR DELAY = 1 TO 500: NEXT
70  PRESET (X,Y)
80  GOTO 30
90  END
```

Program 3–4 Plotting a pattern of random points.

```
10 'PROGRAM 3-5.  RELATIVE COORDINATES IN PSET
20  SCREEN 2: CLS
30  PSET (0,199)
40  FOR COUNT = 1 TO 30
50      PSET STEP (20,-5)
60  NEXT COUNT
70  END
```

Program 3–5 Plotting points using relative coordinates.

diagonally up the screen from the lower left corner. Each successive point is plotted 20 units farther to the right and 5 units up the screen from the previous point plotted.

3–3 DRAWING LINES

We have two basic methods for drawing lines. We can plot individual pixel positions along the line using PSET, or we can use the LINE statement.

LINE STATEMENT

A line-drawing graphics command is available to us in the form

LINE (X1,Y1)—(X2,Y2),C,BX — Draws a straight line or box from position (X1,Y1) to position (X2,Y2) with color C. Parameters X1, Y1, X2, Y2, and C may be numeric constants or expressions. If noninteger, they will be rounded. BX can take either of the following character values: B or BF.

Coordinates for the line endpoints, (X1,Y1) and (X2,Y2), may be specified in absolute or relative form. If parameter BX is omitted, this command draws a straight line between these endpoints. With BX set to the value B, we get a box with (X1,Y1) and (X2,Y2) as coordinates of any diagonally opposite corners. Setting BX to the value BF fills the interior of the box with color C. Parameter C should be set to values that stay within the interval 0 to 3. Zero gives the background color, and for other values we have a choice of three colors. Again, we will omit the color parameter for now.

In Prog. 3–6, we demonstrate use of the LINE command to draw a line using both absolute and relative coordinate specifications. The starting coordinates for

Program 3–6 Line drawing using the LINE command with both absolute and relative coordinate specification.

```
10 'PROGRAM 3-6.  DRAWS HORIZONTAL LINE
20  SCREEN 1: CLS
30  INPUT "ENTER STARTING POSITION FOR LINE"; X, Y
40  INPUT "ENTER LENGTH OF LINE"; L
50  CLS
60  LINE (X,Y) - STEP (L,0)
70  END
```

Program 3–7 Star pattern produced by specifying line endpoints relative to the last referenced point.

```
10 'PROGRAM 3-7.  CONNECTING RELATIVE STAR POINTS
20  SCREEN 2: CLS
30  READ X,Y
40  PSET (X,Y)
50  FOR P = 1 TO 5
60      READ X,Y
70      LINE - (X,Y)
80  NEXT
90  DATA 274,84,90,141,159,50,231,141,46,84,274,84
100 END
```

the line are given in absolute form and the ending coordinates are in relative form. This example lets us draw horizontal lines of length L from any input starting position.

At times, we may want to draw a line from the last point referenced in a program to another specified point. We can do this by leaving out the first coordinate pair. Program 3–7 gives an example of this method of line drawing to produce the star pattern shown in Fig. 3–3.

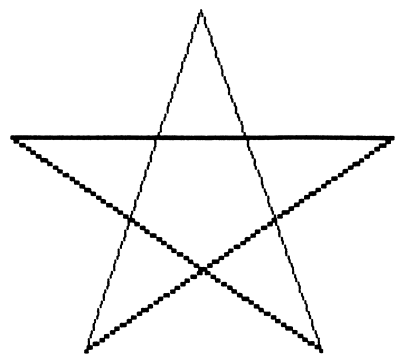


Figure 3–3 Pattern formed with straight lines, using the LINE statement, as output by Prog. 3–7.

PIXEL METHODS

We can also use PSET (or PRESET) to draw lines. Line-drawing methods using PSET will be useful to us later when we consider animation. To create a straight line in this way, we need to specify the position of each point along the line. Since pixels are 1 unit apart, PSET will draw a vertical line, for example, when the Y coordinate is repeatedly increased by 1 and the X coordinate is held constant. Programs 3–8 and 3–9 provide examples of drawing vertical and horizontal lines in this way. The lines drawn by these programs are identical to what we would get using the LINE command, since each pixel plotted is a small rectangle adjoined to the previously plotted pixel. In Prog. 3–8 we have used an absolute coordinate specification to draw the line, and in Prog. 3–9 we have used relative coordinates.

Drawing diagonal lines with PSET is less straightforward. To draw a diagonal line using this command, we must calculate coordinate values along the

Program 3–8 Drawing a vertical line by plotting points, specified as absolute coordinates.

```
10 'PROGRAM 3-8.  PLOTTING VERTICAL LINES USING PSET
20  SCREEN 1: CLS
30  INPUT "STARTING COORDINATES FOR LINE"; X, Y
40  INPUT "LENGTH OF LINE"; LENGTH
50  CLS
60  IF Y + LENGTH > 199 THEN LENGTH = 199 - Y    'SET LENGTH BACK TO MAXIMUM
70  FOR EACHPOINT = Y TO Y + LENGTH              'THAT WILL FIT ON SCREEN
80      PSET (X,EACHPOINT)
90  NEXT
100 END
```

path of the line. To do this we use equation (3–2), which relates X and Y values for a straight line.

$$Y = M * X + B \qquad (3–2)$$

In this equation, M is the **slope** of the line, which may be positive, negative, or zero. When M is zero, we have a horizontal line. For very large magnitudes of M, we have nearly vertical lines. The **Y-intercept**, B, is the value that Y has at the left edge of the screen where X is zero (the Y axis).

We can program a general line-drawing algorithm based upon equation (3–2) and the PSET command. Program 3–10 illustrates this method, with values for M and B entered as input. This program first determines whether the specified line can be drawn on the screen. If no part of the line can be plotted within the coordinate boundaries of the screen, the program simply prints that message. If some part of the line can be drawn, Prog. 3–10 plots the visible part of the line from one screen boundary to another. Figure 3–4 shows the output of Prog. 3–10 for the case M = 1. The line shown appears opposite from what we would expect when graphed in a conventional coordinate system, where a line with positive slope slants up from left to right. This happens because our point (0,0) is at the upper left corner of the screen. We can change the program to produce conventionally oriented lines by multiplying each input value of the slope by -1 and by changing each input B value to 199 – B.

Program 3–10 will not produce continuous lines when the magnitude of the slope is greater than 1. We will have gaps between the plotted pixels. This can be corrected by incrementing the Y coordinate by 1 unit instead of the X coordinate

Program 3–9 Drawing a horizontal line by plotting points, specified as relative coordinates.

```
10 'PROGRAM 3-9.  PLOTTING HORIZONTAL LINES WITH PSET AND STEP
20  SCREEN 1: CLS
30  INPUT "STARTING COORDINATES FOR LINE"; X, Y
40  INPUT "LENGTH OF LINE"; LENGTH
50  CLS
60  PSET (X,Y)
70  IF Y + LENGTH > 319 THEN LENGTH = 319 - Y    'SET LENGTH BACK TO MAXIMUM
80  FOR EACHPOINT = 1 TO LENGTH                  'THAT WILL FIT ON SCREEN
90      PSET STEP (1,0)
100 NEXT
110 END
```

Program 3–10 General line drawing using the line equation and point plotting.

```
10 'PROGRAM 3-10. GENERAL LINE DRAWING USING PSET
20  SCREEN 1: CLS
30  INPUT "SLOPE"; M
40  INPUT "Y-INTERCEPT"; B
50            'IF SLOPE IS NEGATIVE AND LINE INTERCEPTS Y AT A
60            'VALUE LESS THAN 0, LINE IS BEYOND SCREEN COORDINATES.
70  IF M < 0 AND B < 0 THEN PRINT "LINE OFF SCREEN": GOTO 390
80            'IF SLOPE IS POSITIVE BUT LINE INTERCEPTS Y AT A VALUE
90            'GREATER THAN 199, LINE IS BEYOND SCREEN COORDINATES.
100 IF M > 0 AND B > 199 THEN PRINT "LINE OFF SCREEN": GOTO 390
110           'OTHERWISE, FIND LEFTMOST POINT OF LINE.
120           'IF Y-INTERCEPT IS BETWEEN 0 AND 199 THEN LEFTMOST
130           'POINT IS AT LEFT EDGE OF SCREEN.
140 X1 = 0
150           'IF SLOPE IS NEGATIVE AND LINE INTERCEPTS Y AT A VALUE
160           'GREATER THAN 199, LEFTMOST POINT IS ALONG BOTTOM EDGE OF
170           'SCREEN (WHERE Y = 199) SO X1 = (199 - B) / M.
180 IF M < 0 AND B > 199 THEN X1 = (199 - B) / M
190           'IF THIS LEFTMOST POINT IS BEYOND 319, LINE IS OFF SCREEN.
200 IF X1 > 319 THEN PRINT "LINE OFF SCREEN": GOTO 390
210           'IF SLOPE IS POSITIVE AND LINE INTERCEPTS Y AT A POINT LESS
220           'THAN 0, LEFTMOST POINT IS ALONG TOP EDGE OF SCREEN
230           '(WHERE Y = 0) SO X1 = (0 - B) / M OR X1 = -B / M.
240 IF M > 0 AND B < 0 THEN X1 = -B / M
250           'IF THIS POINT IS BEYOND 319, LINE IS OFF SCREEN.
260 IF X1 > 319 THEN PRINT "LINE OFF SCREEN": GOTO 390
270           'OTHERWISE, LINE IS AT LEAST PARTIALLY ON SCREEN.  START AT
280           'LEFTMOST POINT OF LINE (X1).  USING INCREASING VALUES OF
290           'X, CALCULATE NEW Y VALUES AND PLOT X,Y.  CONTINUE UNTIL
300           '1.  X > 319 (LINE GOES TO RIGHT EDGE OF SCREEN)
310           '2.  Y < 0  (LINE GOES OFF TOP EDGE OF SCREEN) OR
320           '3.  Y > 199 (LINE GOES OFF BOTTOM EDGE OF SCREEN)
330 CLS
340 FOR X = X1 TO 319
350     Y = M * X + B
360     IF Y < 0 OR Y > 199 THEN 390
370     PSET (X,Y)
380 NEXT
390 END
```

when ABS(M) > 1. We could also modify the program to position the line so that it is drawn in the center of the screen. Methods for positioning lines on the screen are discussed in Chapter 4.

In certain graphics applications, we need to know the slope and Y-intercept of lines plotted between given endpoints. Specifying the line endpoints as (X1,Y1) and (X2,Y2), we can calculate M and B from the relations

$$M = (Y2 - Y1)/(X2 - X1)$$
$$B = Y1 - M * X1 \qquad (3\text{–}3)$$

We will use the LINE command whenever we want to draw a line between specified endpoints. If we want to draw a line with a certain slope M, we can either use PSET or figure out where the endpoints are and use the LINE statement. The PSET method is also useful for animating objects along straight-line paths.

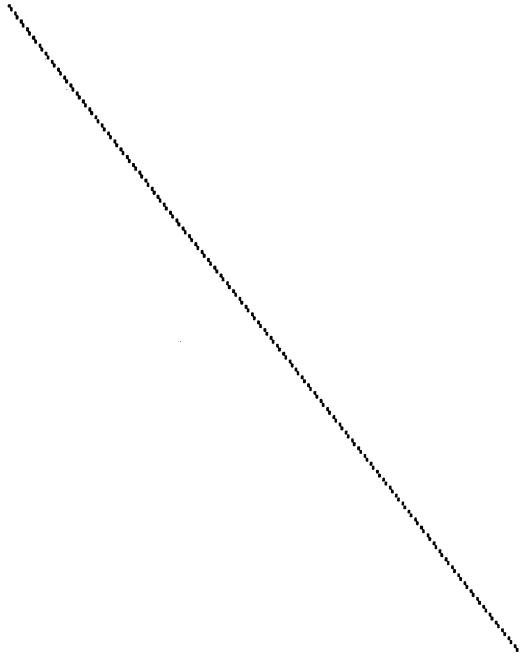


Figure 3–4 Straight line with a slope of 1 and Y-intercept of 0, drawn by Prog. 3–10.

3–4 PIXEL COLOR

The general considerations regarding the use of color that we discussed in Section 2–4 apply to pixel graphics as well. We have the same reasons for using color whether we are creating pictures with characters or pixels, and we need to observe the same considerations in our choice of color combinations.

In medium-resolution graphics mode we have the following color statement available in our set of graphics commands:

> COLOR B,P — Sets the screen to color B (the background) and selects one of two possible sets of colors, or "palettes," P that is to be used with subsequent graphics commands. Parameters B and P may be numeric constants or expressions. If noninteger, they will be rounded.

Either parameter (B or P) may be omitted. Background B selects colors in the range 0 to 15, as listed with the character color statement in Section 2–4. If B is omitted, we default to a background value 0 (black). The palette P selects one of two color combination options. Any even integer for P is interpreted as a zero value and selects palette 0; any odd integer is taken to be the value 1, which selects palette 1. If we omit P, we automatically get palette 1. Once we have selected a palette, the colors in that palette are available for use in commands such

as LINE and PSET. We then choose the color we want for an individual line or pixel by setting parameter C in these commands to the number code of the desired color. Omitting parameter C gives us color code 3 (the default color). The color codes and their corresponding colors for each palette are:

Color Code C	Corresponding Palette 0 Color	Corresponding Palette 1 Color
0	background color B	background color B
1	green	cyan
2	red	magenta
3	brown	white

As an example of the use of the COLOR command, the following program segment plots a red pixel and a green line on a blue background:

```
250 COLOR 1,0
260 PSET (10,10),2
270 LINE (20,20) – (40,40),1
```

The COLOR B,P command can be used only with medium resolution (SCREEN 1). Any text or characters we put on the screen when in this mode will be in color code 3.

We can use the COLOR statement as many times as we wish within a program to set or change either the background color, the palette, or both. When we execute a COLOR statement with a new background color, the background of the entire screen is set immediately to the color specified. Whenever we change palettes, we change the colors of all previously plotted pixels.

The COLOR command cannot be used in high resolution (SCREEN 2). Only black and white are available, and we specify which we want by setting the value for parameter C in the graphics commands. Allowable values for C are 0 (black) and 1 (white). When we omit C, we get white.

Another color statement that is available (in advanced BASIC only) is the PAINT command. This statement allows us to fill in color areas defined by any boundary.

PAINT (X,Y),CP,CB — Fills in a screen area in color CP up to a boundary color CB, with (X,Y) as any point within the boundary. Parameters X, Y, CP, and CB may be numeric constants or expressions. If noninteger, they will be rounded.

Coordinates X and Y can be stated in absolute or relative form (using STEP). In medium resolution, the paint color CP and the boundary color CB should be within the range 0 to 3, corresponding to the colors available in palette 0 or 1. The allowable values for CP and CB in SCREEN 2 are 0 and 1. Either color parameter can be omitted. If CP is omitted, color code 3 in the chosen palette is used for the

paint color when in SCREEN 1 and color code 1 (white) is used in SCREEN 2. If CB is omitted or is not the color of any boundary, the entire screen is filled with the paint color CP. Also, if the boundary contains any holes, the filler color will leak out and around the boundary. We can use the PAINT command to fill the interiors of figures, the screen area outside of a figure, or the screen area between figures.

3–5 PIXEL PICTURES

Point-plotting and line-drawing commands give us the basic tools for creating graphics displays. With PSET, LINE, COLOR, and PAINT, we can construct figure outlines, silhouettes, three-dimensional shapes, and complex scenes.

A triangle, rectangle, or general polygon of any number of sides is drawn by Prog. 3–11 in various color combinations. Coordinates for the vertices of the figure are input in the order in which they are to be connected and stored in arrays X and Y. The polygon is then displayed by drawing a line from point (X(1), Y(1)) to point (X(2),Y(2)), then to point (X(3),Y(3)), and so on. As the last step, the figure is painted in different colors, starting from a specified interior point.

Program 3–11 illustrates another form of time delay. In this case, execution stops indefinitely at line 240. The program will resume interaction with us when

Program 3–11 General polygon drawing and color.

```
10 'PROGRAM 3-11.  DRAWS POLYGONS IN VARIOUS COLOR COMBINATIONS
20  DIM X(10), Y(10)
30  SCREEN 1: CLS
40  INPUT "NUMBER OF POINTS IN POLYGON"; N
50  FOR EACHPOINT = 1 TO N
60      PRINT "COORDINATES FOR POINT"; EACHPOINT;
70      INPUT X(EACHPOINT), Y(EACHPOINT)
80      IF X(EACHPOINT) < 0 OR X(EACHPOINT) > 319 THEN
                PRINT "OFF SCREEN. TRY AGAIN": GOTO 60
90      IF Y(EACHPOINT) < 0 OR Y(EACHPOINT) > 199 THEN
                PRINT "OFF SCREEN. TRY AGAIN": GOTO 60
100 NEXT
110 INPUT "COORDINATES OF AN INTERIOR POINT"; XINTERIOR, YINTERIOR
120 CLS
130 PRINT "HIT ANY KEY FOR NEXT COLOR COMBINATION"
140 FOR BACKGROUND = 1 TO 15
150     FOR PALETTE = 0 TO 1
160         COLOR BACKGROUND,PALETTE
170         FOR COLORCODE = 1 TO 3
180             PSET (X(1),Y(1)),COLORCODE
190             FOR EACHPOINT = 2 TO N
200                 LINE - (X(EACHPOINT),Y(EACHPOINT)),COLORCODE
210             NEXT
220             LINE - (X(1),Y(1)),COLORCODE
230             PAINT (XINTERIOR,YINTERIOR),COLORCODE,COLORCODE
240             IF INKEY$ = "" THEN 240
250         NEXT COLORCODE
260     NEXT PALETTE
270 NEXT BACKGROUND
280 END
```

we hit any key on the keyboard. The INKEY$ operation is initialized by the system to the null string. When executed, it assumes the value of the key currently being pressed on the keyboard, if any. This type of time delay is particularly useful if the program is producing a series of displays that are being used in conjunction with a report or presentation. It is often helpful to include a "prompt" with the time delay, particularly if we are writing a program to be used by other people. We included the message HIT ANY KEY FOR NEXT COLOR COMBINATION at line 130 in Prog. 3–11 to indicate that the program is in a time delay. INKEY$ is also useful at the end of a program, since it delays the appearance of BASIC's "OK" until a key is pressed. This gives us time to print out or photograph our display.

Another technique for producing solid color areas is illustrated with Prog. 3–12. This gives us a means for filling in areas if we do not have the PAINT command. In this program, any rectangular area on the screen is chosen by specifying opposite corners of the rectangle. Then the interior of the rectangle is painted with a selected color.

Solid-color areas can be produced by drawing horizontal, vertical, or diagonal lines. We can vary the order of drawing lines and choice of color to produce special effects. For example, we could fill in a rectangular area with randomly colored horizontal lines, drawn alternately from the top and bottom and meeting in the middle.

Program 3–12 Painting solid color rectangles.

```
10 'PROGRAM 3-12.  PAINTING IN BOXES
20  SCREEN 1: CLS
30  INPUT "TOP LEFT CORNER OF BOX"; X1, Y1
40  INPUT "BOTTOM RIGHT CORNER OF BOX"; X2, Y2
50  INPUT "CODE FOR BACKGROUND COLOR (0 - 15)"; BACKGROUND
60  INPUT "CODE FOR PALETTE (0 OR 1)"; PALETTE
70  INPUT "CODE FOR BOX COLOR (1 - 3)"; BOXCOLOR
80  CLS
90  COLOR BACKGROUND,PALETTE
100 FOR X = X1 TO X2
110     LINE (X,Y1) - (X,Y2),BOXCOLOR
120 NEXT
130 IF INKEY$ = "" THEN 130
140 END
```

SHADING PATTERNS

We can produce shading in our pictures using patterns formed with characters or pixels. The characters that we get in graphics mode, however, are not always the same as the characters available in text mode (SCREEN 0). In SCREEN 1 or SCREEN 2, ASCII codes 128 through 194 give us blanks and the codes from 195 to 255 make patterns different from the text characters for these codes. Figure 3–5 lists character patterns we can access in graphics mode. These patterns could be used to create a variety of shading and textures in our pictures.

With pixels, we can create additional shading and texture patterns. We could

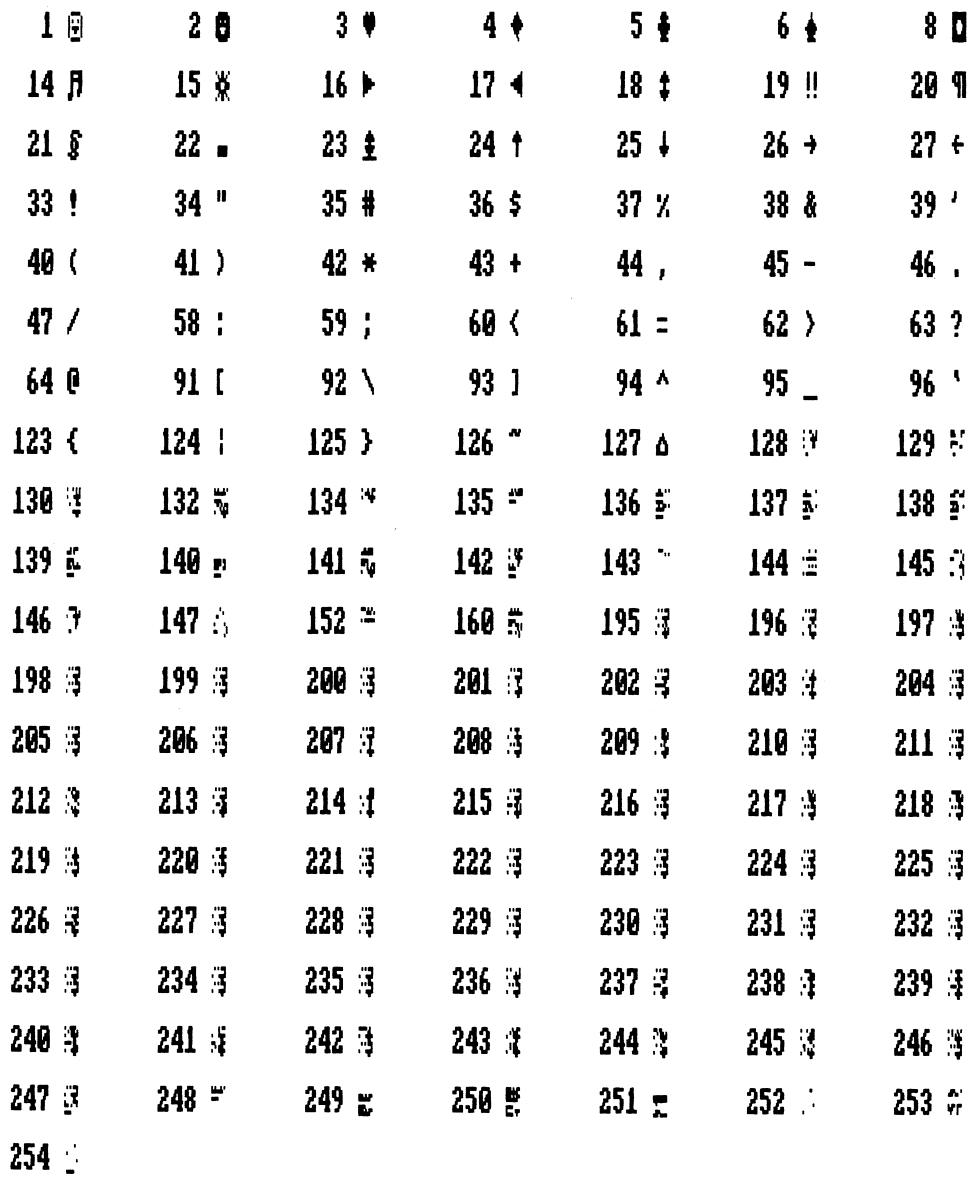


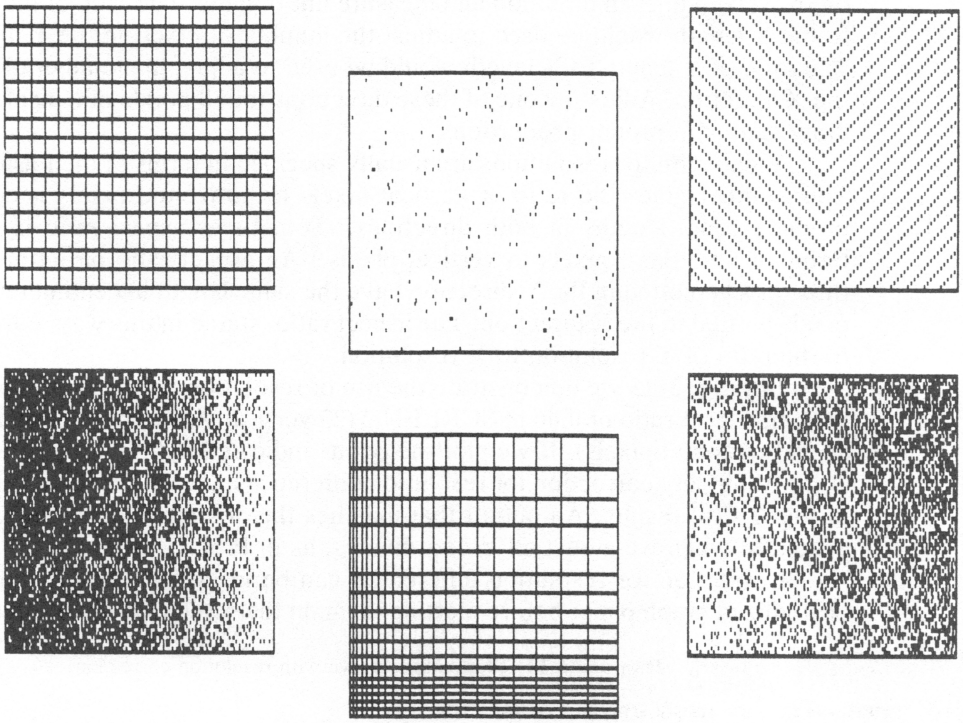
Figure 3–5 Character patterns and their ASCII codes available in graphics mode.

use lines to partially fill in areas, such as with every other line or every third line. We could also draw lines in both directions, spaced to obtain a crosshatched shading. Another possibility is to vary the spacing between the shading lines to produce a gradual dark to light (or light to dark) shading. The spacing between lines could be increased slowly by 1 or 2 units, or we could increase spacing more

rapidly by doubling the previous spacing to get the next spacing length. Areas could also be filled with dots, either symmetrically or randomly spaced. Figure 3–6 shows some shading patterns possible with points and lines.

Color shadings can be developed by combining pixels of different colors. For example, plotting alternating red and brown pixels produces an orange-colored area. Some of the shades possible with palette 0 and a blue background are shown in Figure C of the color insert.

Figure 3–6 Shading patterns possible with pixel graphics.



RESOLUTION RATIOS

Whenever we draw a rectangle with an equal number of pixels in the horizontal and vertical directions, we might expect a square to be displayed. This will not be the case for screens on which the resolution in the X direction is significantly different from that in the Y direction. For example, we might have a monitor on which a 100-pixel horizontal line in SCREEN 1 measures 6 cm long, but a 100-pixel vertical line measures 8 cm. The X resolution is then 100/6, while the Y resolution is 100/8. A 100 by 100 pixel box drawn on the screen will appear as a rectangle, somewhat taller than it is wide. To make the box into a square, we could either plot more points in the X direction or fewer points in the Y direction.

Suppose that we want to change the number of pixels in the Y direction. We can determine the number of pixels needed in the Y direction to make a square by multiplying the original vertical length (100) by the ratio of Y resolution to X resolution (100/8 divided by 100/6, or 3/4). The result is 75, so we need to plot 75 points in the Y direction to make a square. In SCREEN 2, our 100-pixel horizontal line on the same monitor would measure only one-half of its length in SCREEN 1 (3 cm). For this resolution mode, the ratio of Y resolution to X resolution would be one-half the SCREEN 1 value, or 37.5.

To determine **resolution ratios** for any monitor, we can draw a 100-pixel line on the screen in each direction and measure line lengths. The ratio of the X length to the Y length would be used to adjust the number of pixels in the Y directions. The ratio of Y length to X length would be used to adjust the number of pixels in the X direction. Adjusting one of these directions for all points of a display lets us draw figures in proper proportions.

Video monitor resolutions are usually specified in terms of their **aspect ratio**. This number gives the ratio of vertical pixels to horizontal pixels necessary to produce equal lengths in both directions. (Sometimes aspect ratio is stated in terms of horizontal pixels to vertical pixels.) An aspect ratio of 3/4 means that three pixels plotted in the Y direction have the same length in centimeters as four pixels plotted in the X direction. The aspect ratio, stated in this way, corresponds to the ratio of Y resolution to X resolution.

In Prog. 3–13 we demonstrate the use of resolution correction for a monitor with an aspect ratio of 0.46 in SCREEN 2 (23 vertical pixels have the same length as 50 horizontal pixels). If we plot the points indicated by the drawing in Fig. 3–7(a) without any correction for resolution differences, the distorted version shown in Fig. 3–7(b) results. An output that matches the original pattern proportions is produced when we adjust all Y coordinates, as in Prog. 3–13.

Correction for resolution differences can be made by sketching objects on customized graph paper before plotting them on the screen. We make customized

Program 3–13 Star pattern drawn with resolution correction.

```
10 'PROGRAM 3-13. RESOLUTION CORRECTION
20  DIM X(12), Y(12)
30  SCREEN 2: CLS
40  YADJUST = .46                  'ADJUST FOR X-Y RESOLUTION DIFFERENCE
50  FOR POINTER = 0 TO 11
60      READ XINPUT,YINPUT
70      X(POINTER) = XINPUT
80      Y(POINTER) = YINPUT * YADJUST
90  NEXT
100 FOR POINTER = 0 TO 11
110     OTHEREND = (POINTER+4) MOD 12          'CONNECT TO POINT 4 POINTS AWAY
120     LINE (X(POINTER),Y(POINTER)) - (X(OTHEREND),Y(OTHEREND))
130 NEXT
140 DATA 160,0,210,10,250,50,260,100
150 DATA 250,150,210,190,160,200,110,190
160 DATA 70,150,60,100,70,50,110,10
170 IF INKEY$ = "" THEN 170
180 END
```

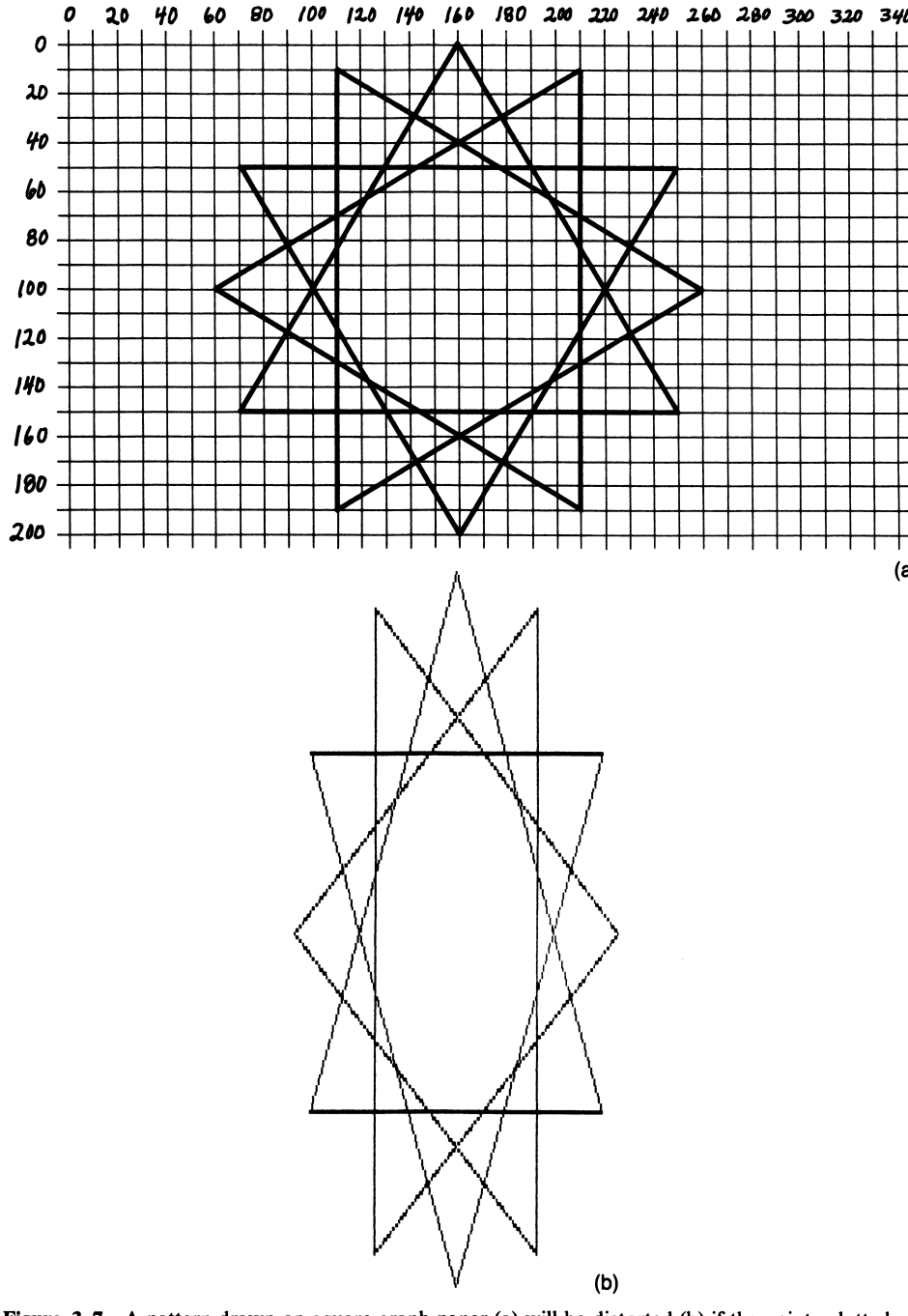


Figure 3–7 A pattern drawn on square graph paper (a) will be distorted (b) if the points plotted are not corrected for resolution differences.

graph paper by printing out a grid drawn on the screen. Appendix A contains customized paper for both SCREEN 1 and SCREEN 2, with the grid lines drawn every five pixels apart.

DRAW STATEMENT

If we have the advanced BASIC interpreter, we can create pictures using the DRAW statement. With this command, it is possible to specify all the parts of a picture in one statement.

DRAW string — Constructs a picture on the screen according to the commands listed in string.

The special commands that we can use in string allow us to draw a series of lines, plot individual points, set colors, and perform some special operations, such as changing sizes or rotating picture parts.

We can draw horizontal and vertical lines by stating the direction and length of the line in string. For example,

```
DRAW "R100"
```

produces a line of length 100, drawn to the right (R). This line will be drawn from the last pixel position used in a program. If no point has yet been used, the initial reference position is taken as the screen center: (160,100) in medium resolution or (320,100) in high resolution. For high resolution, the string "R100" then creates a line from (320,100) to (420,100), if no point has yet been referenced. To get lines drawn in the other directions, we use the DRAW commands L (for left), U (for up), and D (for down). The statement

```
DRAW "R100;U100;L100;D100"
```

gives us a box with 100 pixels on all sides. Semicolons separating the four commands are used for clarity, but they can be omitted.

Variable names can be used to specify the line lengths, as

```
DRAW "R=L1; U=L2; L=L3; D=L4"
```

If each of the variable names L1, L2, L3, and L4 have been assigned the value 100, we get the same box as before. We must use semicolons following any variable names.

Diagonal lines are drawn with the commands E (up and right), F (down and right), G (down and left), and H (up and left). Using these commands, we get a triangle with the following program sequence.

```
10 CLS: SCREEN 1
20 L1 = 100: L2 = 50
30 DRAW "R=L1; H=L2; G=L2;"
```

We can draw lines in any direction with the command M X,Y, which plots points from the last position referenced to position (X,Y). For example, line 30 in the program sequence above could be replaced by

```
30 DRAW "M260,100; M210,50; M160,100"
```

assuming that we are starting from the center of the screen. We cannot use variable names with the M command. A plus sign (+) or minus sign (−) with the X coordinate indicates relative coordinates. With relative coordinate specification, line 30 would be written as

```
30 DRAW "M+100,0; M−50, −50; M − 50, 50"
```

Sometimes we might like to skip over to a new screen position before drawing any further lines in a picture. We can do this by placing B in front of any of the plotting commands. This means move without plotting to the position indicated. The statement

```
DRAW "R25; BR10; R25; BR10; R25"
```

gives us a series of three horizontal lines with 10 units between each line. Our triangle can be drawn from the point (110,125) with

```
30 DRAW "BM110,125; R=L1; H=L2; G=L2;"
```

Here, the BM command sets the reference point to (110,125).

Another available prefix for the plotting commands is N. This prefix has the effect of returning the reference point to the original position. Thus

```
DRAW "NE100; F100"
```

plots two diagonal lines from the same point.

Color is set with the C command. We can specify the color code, following C, as a constant or as a variable. We once more plot the triangle, this time as a solid red figure on a blue background:

```
10 CLS: SCREEN 1
20 COLOR 1,0
30 RED = 2: L1 = 100: L2 = 50
40 DRAW "BM110,125;C=RED;R=L1;H=L2;G=L2;"
50 PAINT (160,100),RED,RED
```

Command C sets the color for all subsequent commands, including additional DRAW statements, until a new color is set. Color values are 0, 1, 2, and 3 in SCREEN 1 (corresponding to palette choices), and 0 and 1 in SCREEN 2.

Program 3–14 gives an example of picture construction using the DRAW statement. The sailboat in Fig. 3–8 is the output.

Picture parts can be made larger or smaller with the S (scale) command. All following commands will then be scaled by the amount specified after S until another scale command is given. The number following S must be an integer in the

Program 3–14 Sailboat constructed with the DRAW statement.

```
10 'PROGRAM 3-14. SAILBOAT USING DRAW STATEMENT
20  SCREEN 1: COLOR 0,0: CLS
30  DRAW "BM60,140R200G30L140H30"          'MAKE BOAT BOTTOM
40  DRAW "BM150,130R110H110D120"           'MAKE LARGE SAIL
50  DRAW "BM150,130L80E80"                 'MAKE SMALL SAIL
60  IF INKEY$ = "" THEN 60
70  END
```

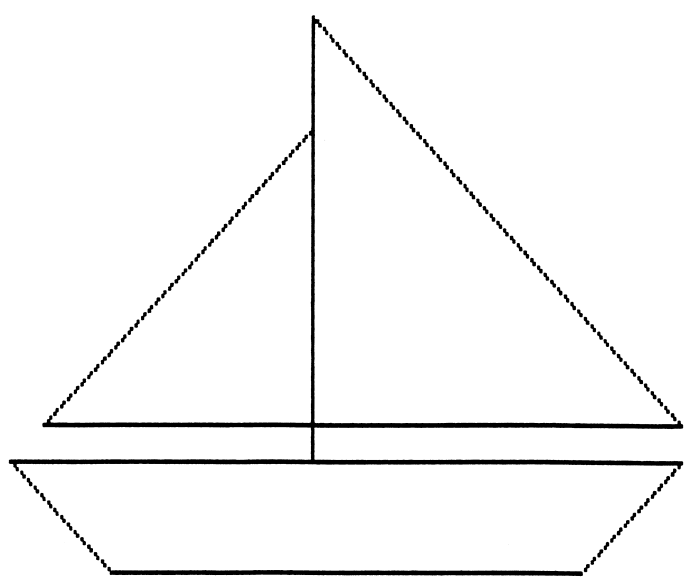
range 1 to 255. This number is divided by 4 to produce a scaling factor for the subsequent plotting commands. The statements

```
160  S$ = "BM125,125;S2;L100"
170  DRAW S$
```

will draw a horizontal line of length 50 from position (125,125) to the left. The scaling factor for this case is 1/2 (2 divided by 4), so that the specified length of 100 is reduced to 50. Scaling factors can be chosen to have values from 1/4 to nearly 64 (as long as we do not enlarge objects beyond the screen size) in steps of 1/4. We will discuss scaling in more detail in Chapter 7. Program 3–15 gives an example of the use of scaling with the DRAW statement. The output is shown in Fig. 3–9.

We have also used the X, or substring command, in Prog. 3–15. This lets us define a particular picture part as a separate string and include it in one or more places within a DRAW statement. In this way, we can set up picture subparts and display the subparts in different places with different scalings. The turrets of the castle of Fig. 3–9 were created in this way.

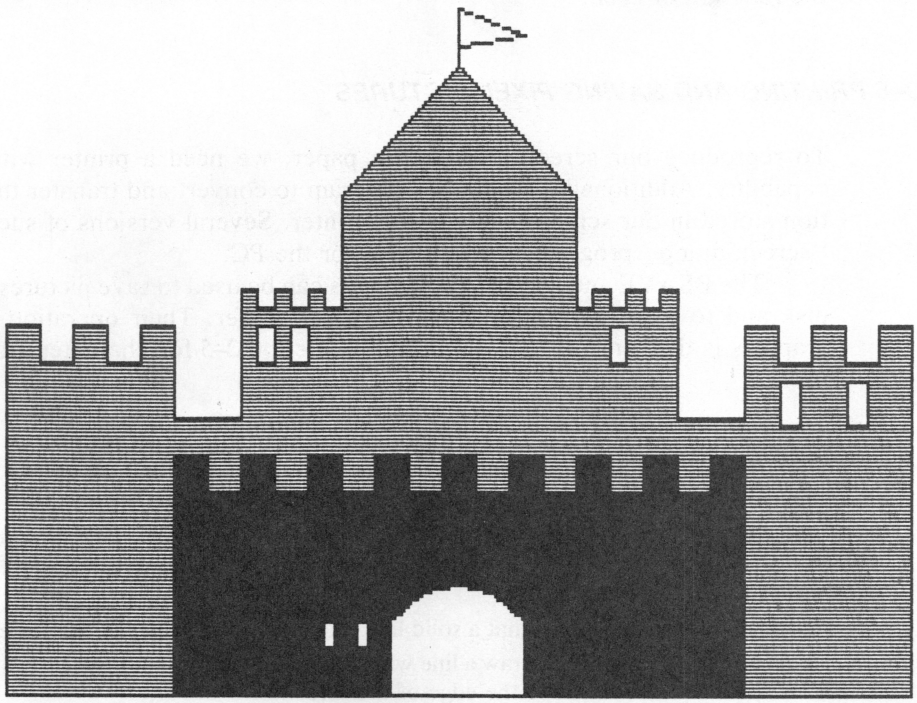
Figure 3–8 Sailboat displayed by Prog. 3–14, using DRAW statements.



Program 3–15 Picture construction (castle) and scaling, using the DRAW statement.

```
10 'PROGRAM 3-15.  DRAW STATEMENTS USED TO DRAW CASTLE
20  SCREEN 1: COLOR 0,0: CLS
30  TURRET$ = "U10 R10 D10 R10"
40  TURRET$ = TURRET$ + TURRET$ + "U10 R10 D10"
50  MAINTURRET$ = TURRET$ + "R10" + TURRET$ + "R10" + TURRET$
60  LEFTSMALLTURRET$ = "S2" + TURRET$ + "S4 R5"
70  RIGHTSMALLTURRET$ = "R5 S2" + TURRET$ + "S4"
80  FLAG$ = "U15; M 180,12; M 160,15"
90  DRAW "BM75,190 L50 U90; X TURRET$; D15 R20 U30; X LEFTSMALLTURRET$;"
100 DRAW "U30 M160,20; X FLAG$; D5 M195,55 D30; X RIGHTSMALLTURRET$;"
110 DRAW "D30 R20 U15 X TURRET$; D90 L220 U55; X MAINTURRET$; D55"
120 DRAW "BM140,190; U20; M150,162; M160,160; M170,162; M180,170; D20"
130 WINDOW$ = "R5 D10 L5 U10"
140 DRAW "BM100,90; X WINDOW$;"
150 DRAW "BM110,90; X WINDOW$;"
160 DRAW "BM205,90; X WINDOW$;"
170 DRAW "BM256,105; S5; X WINDOW$;"
180 DRAW "BM276,105; X WINDOW$;"
190 DRAW "BM130,170; S3; X WINDOW$;"
200 DRAW "BM120,170; X WINDOW$;"
210 PAINT (160,140),3,3
220 PAINT (160,100),2,3
230 IF INKEY$ = "" THEN 230
240 END
```

Figure 3–9 A picture constructed with DRAW statements and the scaling command by Prog. 3–15.



Draw command	Purpose
R	Plot points to the right.
L	Plot points to the left.
U	Plot points up.
D	Plot points down.
E	Plot points diagonally to the right and up.
F	Plot points diagonally to the right and down.
G	Plot points diagonally to the left and down.
H	Plot points diagonally to the left and up.
M X,Y	Plot points to coordinate position (X,Y).
B	Move without plotting (prefix to other commands).
N	Plot, then return to original position (prefix to other commands).
C	Set color.
S	Set scale factor parameter.
A	Set rotation angle.
X string	Specify a substring.

Figure 3–10 Table of commands that can be used with the DRAW statement.

Picture parts can be rotated with a rotation command, A. For now, we will simply note that we can specify rotation angles in steps of 90, 180, or 270 degrees with the numbers 1, 2, or 3 following A. In Chapter 7 we will spend more time discussing rotations and other transformations of pictures.

Figure 3–10 provides a listing of the DRAW commands and prefixes, stating the function of each.

3–6 PRINTING AND SAVING PIXEL PICTURES

To reproduce our screen displays on paper, we need a printer with graphics capability. Additionally, we need a program to convert and transfer the information stored in our screen buffer to the printer. Several versions of such graphics "screen-dump" programs are available for the PC.

The BSAVE and BLOAD commands can be used to save pictures on tape or disk and to load them back into the screen buffer. Their operation with pixel graphics is the same as that discussed in Section 2–5 for character graphics.

PROGRAMMING PROJECTS

3–1. Write a program that clears the screen and draws the word "HELLO" (or any other word) in the center of the screen, using the LINE command. Include a box border around the word.

3–2. Modify Prog. 3–10 so that a solid line is drawn for all values of the slope of the line.

3–3. Modify Prog. 3–10 to draw a line with given slope and Y-intercept from any specified point on the screen to the edge of the screen.

3–4. Using the LINE statement, write a program that inputs any slope, Y-intercept, line length, and starting position and then draws the required line. Provide checks in the program to avoid drawing beyond the screen limits.

3–5. Write a program to produce a screen pattern formed with overlapping, randomly placed rectangles of various sizes. Use the random number generator commands to select the position for one corner of each rectangle and its width and height.

3–6. Modify Project 3–5 to color the interior of each rectangle randomly. The center of each randomly chosen rectangle can be used as the interior point for the PAINT command: (X1+W/2,Y1+H/2), where (X1,Y1) is the location of the upper left corner of the rectangle with width W and height H. Alternatively, we could use the method of Prog. 3–12 to color the interiors.

3–7. Modify Prog. 3–11 so that the program automatically finds an interior point of any given polygon for the PAINT statement. An interior point can be found in several ways. One way is to check each X position along a horizontal screen line with the POINT function to determine when we are inside the polygon boundaries. We can do this by scanning along the next line down from the topmost vertex. That is, if the topmost vertex has a Y value of 10, we check each X position along the horizontal line with Y = 11.

3–8. Modify Prog. 3–12 to fill in the rectangle with dots, where the spacing between dots is specified by input. A random spacing of dots and a random pattern of overlapping rectangles could also be displayed.

3–9. Write a program to produce the shading patterns shown in Fig. 3–6.

3–10. Lay out a figure or scene on graph paper and write a program to display the layout, using the LINE, PSET, and COLOR commands. Provide various shading and color patterns in the figure, and correct for any resolution differences.

3–11. Write a program to produce the layout described in Project 3–10, using the DRAW and PAINT commands.

Chapter 4

Plotting Graphs

Data listed in tables are usually harder to interpret than when presented in graph form. Graphs allow us to grasp the information content of a set of numbers more quickly. They can clearly show various data relationships that are often difficult to pick out in a simple listing of the numbers. We can construct graphs using either the PRINT statement or the special graphics commands introduced in Chapter 2.

4–1 FUNDAMENTALS: DATA TREND GRAPHS

An elementary type of graph is one that shows the general trend or "shape" of the data, with little or no explicit labeling. A trend graph for the data listed in Fig. 4–1 would provide an overall picture of sales fluctuations during the year. We could orient this graph so that sales magnitudes are represented either horizontally or vertically. Figure 4–2 shows these two graph orientations. In Fig. 4–2(a), months would be counted down from the top of the screen, and sales magnitudes would be scaled to fit across the screen, from left to right. Figure 4–2(b) illustrates the case in which months would be counted across from the left of the screen and magnitude scaled vertically.

CHARACTER GRAPHICS METHOD

We can construct a trend graph for the data in Fig. 4–1 by printing characters in screen positions that correspond to relative sales magnitudes. Suppose that we want the data magnitudes represented horizontally on an 80-character screen. We will represent months along every other print line from the top of the screen and use the character positions 25 through 75 along each print line to represent sales

Month	Number of items sold	Month	Number of items sold
Jan	210	Jul	410
Feb	150	Aug	390
Mar	99	Sep	300
Apr	250	Oct	651
May	183	Nov	724
Jun	352	Dec	516

Figure 4–1 Sample sales data table.

magnitudes. The maximum sales data value will then correspond to position 75 on the twenty-first line from the top of the screen (for November), and the minimum sales data value will correspond to position 25 on line 5 (for March).

Scaling each of the data values to be in the interval 25 to 75, is accomplished with the following calculation:

$$\begin{aligned}\text{Print position along a line} = {} & (\text{data value} - \text{minimum data value}) \\ & * \frac{\text{print position range}}{\text{data range}} \\ & + \text{minimum print position}\end{aligned} \tag{4–1}$$

For our sample data, the data range is 724 minus 99, or 625. The print position range is 75 minus 25, or 50, and we have chosen 25 as our minimum print position. Horizontal print position for each of the data values is then found as

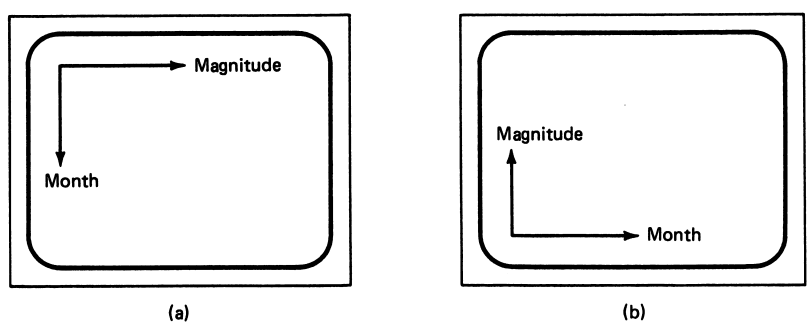
$$\text{Print position} = (\text{data value} - 99) * \frac{50}{625} + 25$$

or

$$P = (\text{data} - 99) * 0.08 + 25$$

Program 4–1 uses this equation to produce the data trend graph of Fig. 4–3. Each

Figure 4–2 The sample sales data magnitudes of Fig. 4–1 can be graphed (a) horizontally or (b) vertically.



Program 4–1 Horizontal data trend graph using character graphics.

```
10 'PROGRAM 4-1.  HORIZONTAL DATA TREND USING CHARACTER GRAPHICS
20      'SALES VALUES ARE SCALED TO LIE BETWEEN COLUMNS 25 - 75.
30      'MONTHS ARE COUNTED DOWN FROM THE TOP, USING EVERY OTHER LINE.
40  SCREEN 0: WIDTH 80: LOCATE ,,0: CLS:
50  RANGERATIO = (75 - 25) / (724 - 99)
60  FOR MONTH = 1 TO 12
70      READ SALES
80      POSITION = INT((SALES - 99) * RANGERATIO + 25 + .5)
90      PRINT TAB(POSITION); "*";
100     IF MONTH <> 12 THEN PRINT: PRINT        'DON'T SCROLL AFTER DECEMBER
110 NEXT MONTH
120 DATA 210,150,99,250,183,352,410,390,300,651,724,516
130 IF INKEY$ = "" THEN 130
140 END
```

print position is calculated and rounded to the nearest whole number using the INT function. The program can be generalized for any set of data and display size by using calculation (4–1) with the maximum and minimum data values and print positions as input.

We have used the command LOCATE ,,0 in statement 40 of Prog. 4–1. This turns the cursor off so that it is invisible. Otherwise, we would see the blinking cursor during the creation of our graph and while we were in the hold loop of line

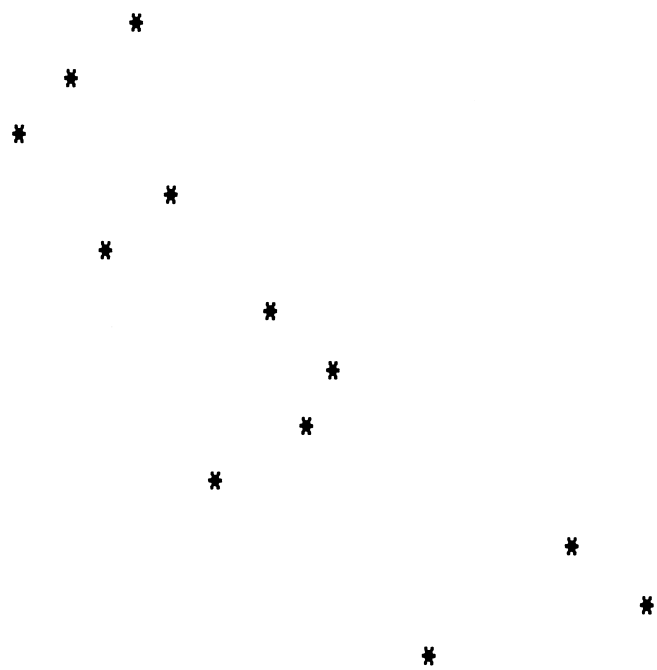


Figure 4–3 Data trend graph produced by Prog. 4–1, with the sales magnitudes of Fig. 4–1 represented horizontally.

130. The cursor is automatically turned back on when execution of the program is finished.

To produce graphs that have magnitudes represented vertically (Fig. 4–2(b)), we make two changes to Prog. 4–1. First, for each data item, we need to determine the proper print line that will correspond to the magnitude of the data value. Second, we select a position along this line based on which month corresponds to the data item. Each data value is represented by printing a character at this row and column position.

For the data of Fig. 4–1 and using WIDTH 40, we now choose months to be in every third column starting at 3 and continuing through 36. Sales magnitudes are plotted using lines 1 through 20. Then the largest magnitude (724) is plotted at location 33 (for November) on the print line at the top of the screen (print line 1). Position along each print line is determined by 3 * M, where M is the number of the month (January = 1, February = 2, and so on). To scale the sales magnitudes onto the 20 print lines, we use the general rule:

$$\begin{aligned}\text{Print line number} = {} & (\text{maximum data value} - \text{data value}) \\ & * \frac{\text{print line range}}{\text{data range}} \\ & + \text{minimum print line number}\end{aligned} \tag{4–2}$$

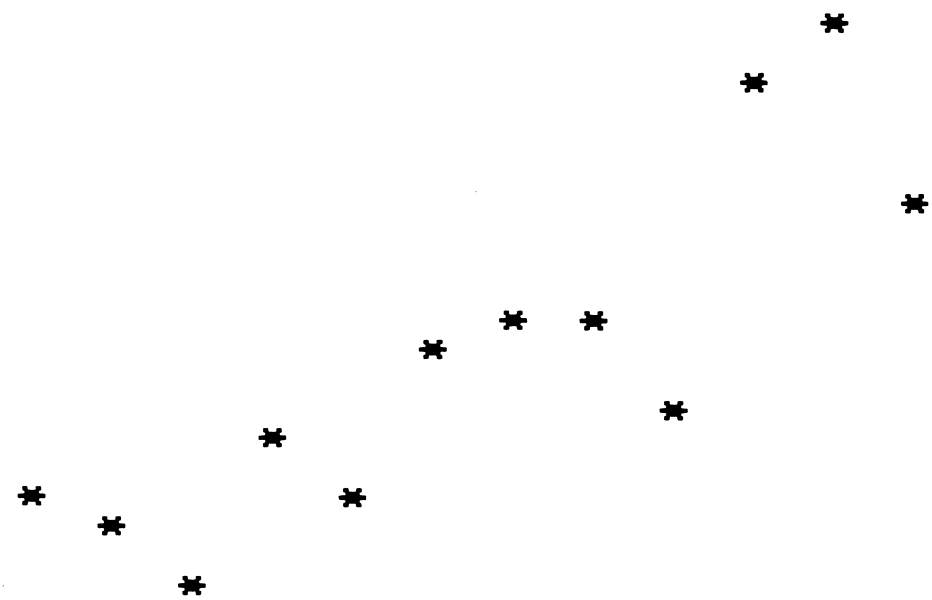


Figure 4–4 Data trend graph produced by Prog. 4–2, with the sales magnitudes of Fig. 4–1 represented vertically.

Program 4–2 Vertical data trend graph using character graphics.

```
10 'PROGRAM 4-2.  VERTICAL DATA TREND USING CHARACTER GRAPHICS
20      'SALES VALES ARE SCALED TO PRINT BETWEEN LINES 1 - 20.
30      'MONTHS USE EVERY 3RD COLUMN STARTING AT 3 AND
40      'CONTINUING ACROSS TO 36.
50  SCREEN 0: WIDTH 40: LOCATE ,,0: CLS
60  RANGERATIO = (20 - 1) / (724 - 99)
70  FOR MONTH = 1 TO 12
80      READ SALES
90      ROW = INT((724 - SALES) * RANGERATIO + 1 + .5)
100     COLUMN = MONTH * 3
110     LOCATE ROW,COLUMN
120     PRINT "*"
130 NEXT MONTH
140 DATA 210,150,99,250,183,352,410,390,300,651,724,516
150 IF INKEY$ = "" THEN 150
160 END
```

In this example, we have chosen a print line range of 19 (or 20 − 1), and a minimum print line number of 1. Figure 4–4 shows the vertical graph produced by Program 4–2. The LOCATE statement is used in this program to position each character on the graph.

Program 4–2 could be generalized to allow a variable graph position. The desired number of lines and column positions along each line would then be entered as input. Data range and the maximum data value would be determined by the program as the data are entered.

PIXEL GRAPHICS METHOD

Using the graphics commands to form data graphs means that we now think of the screen in terms of coordinates instead of print lines and character positions. We can replot the graph produced by Prog. 4–2 using the PSET command to plot pixels instead of characters. With a resolution of 320 by 200, we can position the graph so that we use the pixel rows from 0 to 160 and the pixel columns from 20 to 240. Months will be plotted across the screen at every twentieth pixel, starting with location 20. Data magnitudes will be scaled between the vertical pixels 0 and 160, using the following calculation:

$$\begin{aligned} \text{Y coordinate} = {} & (\text{maximum data value} - \text{data value}) \\ & * \frac{\text{vertical pixel range}}{\text{data range}} \\ & + \text{minimum Y coordinate} \end{aligned} \qquad (4\text{–}3)$$

Program 4–3 produces the resulting vertical data trend graph using pixels. Plotting the data so that magnitudes are represented horizontally is a matter of interchanging the role of the X and Y coordinates, taking screen dimensions into account.

Line segments connecting the data points can be included easily when we plot graphs with pixels. Program 4–4 produces the line graph shown in Fig. 4–5.

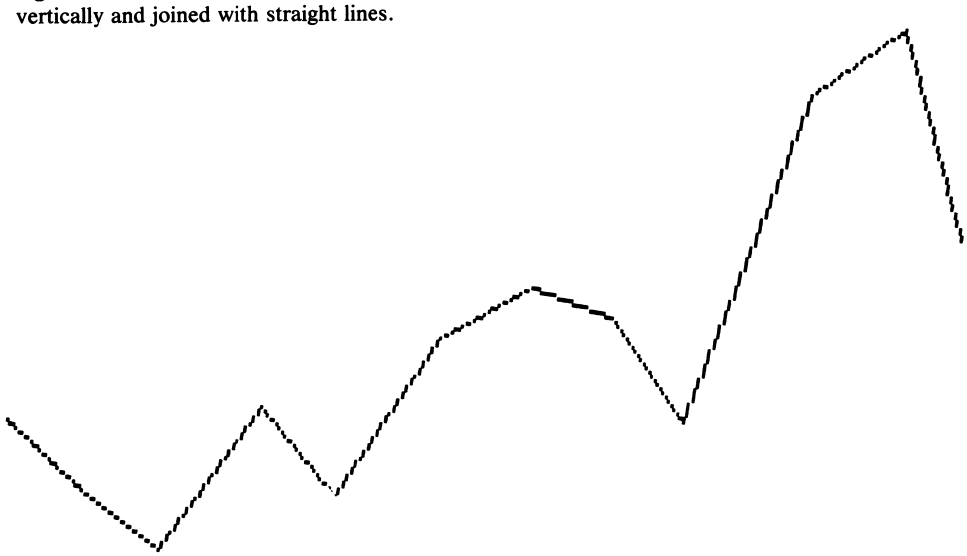
Program 4–3 Vertical data trend graph using point plotting.

```
10 'PROGRAM 4-3.  VERTICAL DATA TREND USING PIXELS
20       'SALES VALUES ARE SCALED TO PIXELS 0 - 160.
30       'MONTHS USE EVERY 20TH PIXEL STARTING AT 20
40       'AND CONTINUING ACROSS TO 240.
50  SCREEN 1: CLS
60  RANGERATIO = (160 - 0) / (724 - 99)
70  X = 20
80  FOR MONTH = 1 TO 12
90       READ SALES
100      Y = INT((724 - SALES) * RANGERATIO + .5)
110      PSET (X,Y)
120      X = X + 20
130 NEXT MONTH
140 DATA 210,150,99,250,183,352,410,390,300,651,724,516
150 IF INKEY$ = "" THEN 150
160 END
```

Program 4–4 Vertical data trend graph using line drawing.

```
10 'PROGRAM 4-4.  DATA CHART WITH CONNECTED LINES
20       'SALES VALUES ARE SCALED TO PIXELS 0 - 160.
30       'MONTHS USE EVERY 20TH PIXEL STARTING AT 20
40       'AND CONTINUING ACROSS TO 240.  SALES
50       'VALUES OF CONSECUTIVE MONTHS ARE CONNECTED.
60  SCREEN 1: CLS
70  RANGERATIO = (160 - 0) / (724 - 99)
80  X = 20
90  READ SALES
100 Y = INT((724 - SALES) * RANGERATIO + .5)
110 PSET (X,Y)
120 FOR MONTH = 2 TO 12
130      X = X + 20
140      READ SALES
150      Y = INT((724 - SALES) * RANGERATIO + .5)
160      LINE - (X,Y)
170 NEXT MONTH
180 DATA 210,150,99,250,183,352,410,390,300,651,724,516
190 IF INKEY$ = "" THEN 190
200 END
```

Figure 4–5 Data trend graph produced by Prog. 4–4, with the sales magnitudes of Fig. 4–1 plotted vertically and joined with straight lines.



4–2 LABELED GRAPHS

The fundamental techniques of the preceding section are useful for quickly plotting simple graphs and displaying data trends. But data trend graphs convey very little quantitative information. Usually, we are interested in determining more precise information from graphs. Labeling of the data point coordinates along the coordinate axes allows us to determine more exact relationships and to interpolate between the data points.

Labeled graphs require some modification to the data scaling equations of (4–1), (4–2), and (4–3). The data range in these equations now corresponds to the labeled graph range. If we plot a data set with a range from −96 to 89 in a graph labeled from −100 to 100, the data range to be used is 200 (the range of the graph). Similarly, the minimum data value would be −100, and the maximum data value would be 100.

Coordinate axes for the X and Y directions can be generated with the ASCII character codes 179, 196, and 197, using the PRINT statement. We have many other character choices, including double lines (code 186) and thicker lines (codes 220 through 223). A labeled graph using single lines is constructed by Prog . 4–5 and displayed in Fig. 4–6.

Program 4–5 Labeled data graph using character graphics.

```
10 'PROGRAM 4-5.  LABELED DATA CHART USING CHARACTER GRAPHICS
20      'SALES VALUES ARE SCALED TO COLUMNS 12 - 76.  MONTHS USE
30      'EVERY PRINT LINE, STARTING FROM THE TOP OF THE CHART.
40      'ASCII CHARACTER CODES 179, 196, AND 197 ARE USED TO MAKE
50      'LINES AND ASCII CODE 219 TO MAKE BARS.
60  SCREEN 0: WIDTH 80: LOCATE ,,0: CLS
70  PRINT TAB(23); "ANNUAL SALES TREND (thousands)"
80  PRINT: PRINT
90  PRINT TAB(12);
100 FOR REPEAT = 1 TO 8
110     PRINT CHR$(197); STRING$(7,196);
120 NEXT
130 PRINT CHR$(197)
140 PRINT TAB(12); CHR$(179); TAB(76); CHR$(179)
150 RANGERATIO = (76 - 12) / (800 - 0)
160 FOR MONTH = 1 TO 12
170     READ MONTHNAME$, SALES
180     POSITION = INT((SALES - 0) * RANGERATIO + 12 +.5)
190     PRINT MONTHNAME$; TAB(12); CHR$(179); TAB(POSITION); CHR$(2);
            TAB(76); CHR$(179)
200 NEXT
210 PRINT TAB(12); CHR$(179); TAB(76); CHR$(179)
220 PRINT TAB(12);
230 FOR REPEAT = 1 TO 8
240     PRINT CHR$(197); STRING$(7,196);
250 NEXT
260 PRINT CHR$(197)
270 PRINT
280 PRINT TAB(11); 0;: FOR REPEAT = 1 TO 8: PRINT STRING$(5," "); REPEAT;: NEXT
290 PRINT TAB(14);: FOR REPEAT = 1 TO 8: PRINT STRING$(5," "); 0;: NEXT
300 PRINT TAB(14);: FOR REPEAT = 1 TO 8: PRINT STRING$(5," "); 0;: NEXT
```

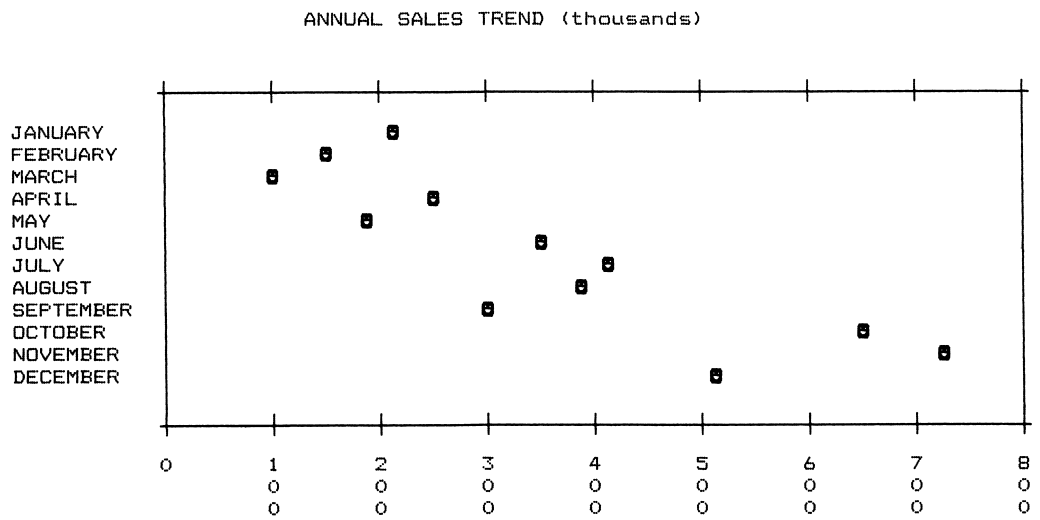
Program 4–5 (cont.)

```
310 DATA "JANUARY",210,"FEBRUARY",150,"MARCH",99
320 DATA "APRIL",250,"MAY",183,"JUNE",352
330 DATA "JULY",410,"AUGUST",390,"SEPTEMBER",300
340 DATA "OCTOBER",651,"NOVEMBER",724,"DECEMBER",516
350 IF INKEY$ = "" THEN 350
360 END
```

Pixel graphics offers a more flexible method for constructing coordinate axes by drawing lines. A labeled graph is produced with this method by Prog. 4–6. The resulting output is shown in Fig. 4–7.

In constructing labeled graphs, we should observe the following guidelines. Labeling should be simple and to the point. Too much labeling can clutter the graph and detract from its effectiveness to convey information. For clear labeling, larger letters and numbers are more effective than small print. If possible, identifying labels should be placed on the lines or in the areas they are meant to identify instead of placing them in separate tables or legends. Divisions for the coordinate axis referencing magnitudes should be chosen in easily comprehended steps, such as multiples of 10 rather than multiples of 8. Including a zero point aids in interpretation. The divisions should be spaced and labeled with tic marks to make interpolation between data points easy. We should also construct data lines to be thicker or more intense than the coordinate axes and grid lines. These ideas were taken into consideration in the construction of the graph in Fig. 4–7.

Figure 4–6 Labeled graph with coordinate axes. Output by Prog. 4–5, using character graphics.



Program 4–6 Labeled graph using line drawing.

```
10 'PROGRAM 4-6.  LABELED DATA CHART WITH CONNECTING LINES
20      'SALES VALUES (in the range of 0 - 800) ARE SCALED TO PIXELS
30      '27 - 155.  MONTHS USE EVERY 40 PIXELS, STARTING AT 148.
40  DIM X(12), Y(12)
50  SCREEN 2: CLS
60  PRINT TAB(31); "ANNUAL SALES FIGURES"
70  LINE (128,27) - (128,155)
80  LINE (608,27) - (608,155)
90     'MAKE NOTCHES FOR SALES MAGNITUDES & CHART LINES
100 FOR Y = 27 TO 155 STEP 16
110     LINE (128,Y) - (608,Y)
120 NEXT
130     'LABEL THE NOTCHES
140 LOCATE 3,12: PRINT "Thousands"
150 ROW = 20
160 FOR SALESAMOUNT = 0 TO 800 STEP 100
170     LOCATE ROW,13: PRINT USING "###"; SALESAMOUNT
180     ROW = ROW - 2
190 NEXT
200     'MAKE NOTCHES FOR MONTHS
210 FOR COLUMN = 19 TO 74 STEP 5
220     LOCATE 20,COLUMN: PRINT "+";
230 NEXT
240 PRINT TAB(19);"J    F    M    A    M    J    J    A    S    O    N    D"
250 PRINT TAB(19);"A    E    A    P    A    U    U    U    E    C    O    E"
260 PRINT TAB(19);"N    B    R    R    Y    N    L    G    P    T    V    C";
270 LOCATE 11,3: PRINT "MONTHLY"
280 LOCATE 13,3: PRINT " SALES"
290     'SCALE SALES VALUES
300 RANGERATIO = (156 - 28) / (800 - 0)
310 XPOSITION = 148
320 FOR MONTH = 1 TO 12
330     X(MONTH) = XPOSITION
340     XPOSITION = XPOSITION + 40
350     READ SALES
360     Y(MONTH) = INT((800 - SALES) * RANGERATIO + 28 + .5)
370 NEXT
380     'DRAW SALES LINES
390 PSET (X(1),Y(1))
400 FOR MONTH = 2 TO 12
410     LINE - (X(MONTH),Y(MONTH))
420 NEXT
430     'MAKE LINE DOUBLE THICKNESS
440 PSET (X(1),Y(1) - 1)
450 FOR MONTH = 1 TO 12
460     LINE - (X(MONTH),Y(MONTH) - 1)
470 NEXT
480 DATA 210,150,99,250,183,352,410,390,300,651,724,516
490 IF INKEY$ = "" THEN 490
500 END
```

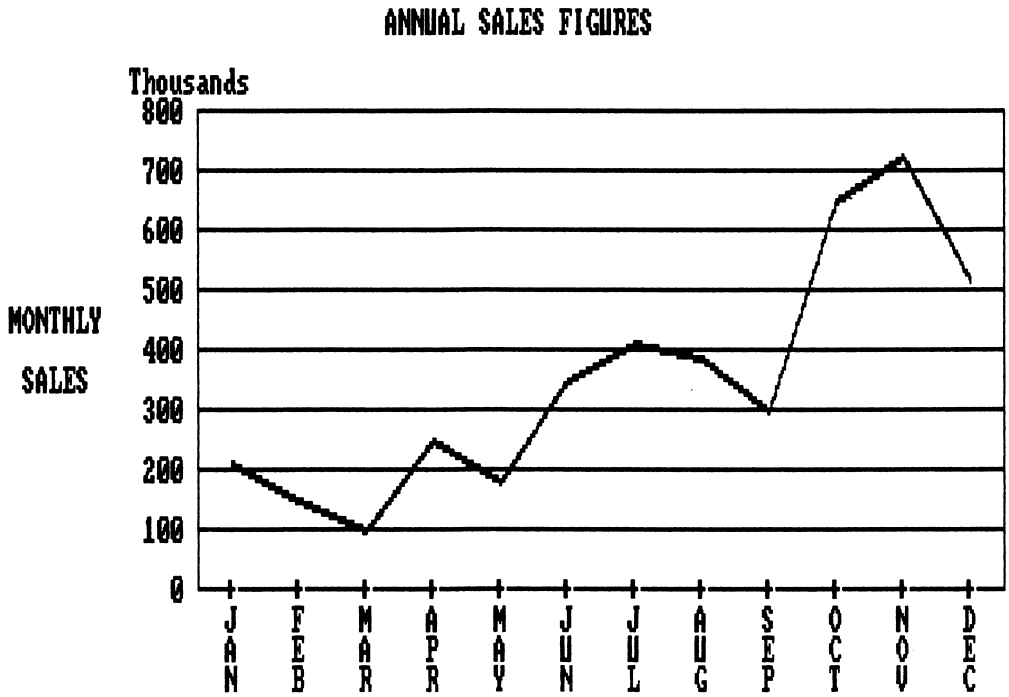


Figure 4–7 Labeled graph with coordinate axes. Output by Prog. 4–6, using pixel graphics methods.

4–3 BAR GRAPHS: COLOR AND SHADING

A useful technique for making data graphs more easily interpreted is the plotting of data magnitudes as "bars" instead of points. This technique is illustrated in Prog. 4–7, using PRINT statement methods. Figure 4–8 shows the resulting bar graph. We used character codes 177 and 219 to get bars with alternate shadings. This helps to distinguish the different bars. We could use other character codes,

Program 4–7 Labeled bar graph using character graphics.

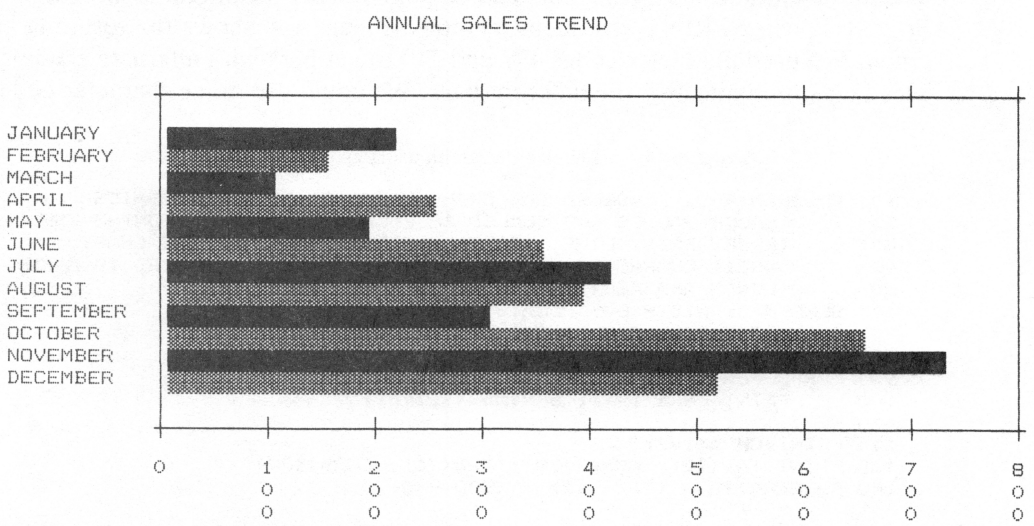
```
10 'PROGRAM 4-7.  LABELED BAR GRAPH USING CHARACTER GRAPHICS
20       'SALES VALUES ARE SCALED TO COLUMNS 12 - 76.  MONTHS USE
30       'EVERY PRINT LINE, STARTING FROM THE TOP OF THE CHART.
40       'ASCII CHARACTER CODES 179, 196, AND 197 ARE USED TO MAKE
50       'LINES AND ASCII CODE 219 TO MAKE BARS.
60  SCREEN 0: WIDTH 80: LOCATE ,,0: CLS
70  PRINT TAB(28); "ANNUAL SALES TREND"
80  LOCATE 4,12
90  FOR REPEAT = 1 TO 8
100      PRINT CHR$(197); STRING$(7,196);
110 NEXT
120 PRINT CHR$(197)
130 PRINT TAB(12); CHR$(179); TAB(76); CHR$(179)
140 RANGERATIO = (76 - 12) / (800 - 0)
```

Program 4–7 (cont.)

```
150 CODE = 177
160 FOR MONTH = 1 TO 12
170     READ MONTHNAME$, SALES
180     POSITION = INT((SALES - 0) * RANGERATIO + 12 +.5)
190     PRINT MONTHNAME$; TAB(12); CHR$(179);
200     IF CODE = 177 THEN CODE = 219: GOTO 220          'USE ALTERNATE
210     IF CODE = 219 THEN CODE = 177                    'SHADING PATTERNS
220     FOR COLUMN = 13 TO POSITION
230         PRINT CHR$(CODE);
240     NEXT
250     PRINT TAB(76); CHR$(179)
260 NEXT
270 PRINT TAB(12); CHR$(179); TAB(76); CHR$(179)
280 PRINT TAB(12);
290 FOR REPEAT = 1 TO 8
300     PRINT CHR$(197); STRING$(7,196);
310 NEXT
320 PRINT CHR$(197)
330 PRINT
340 PRINT TAB(11); 0;: FOR REPEAT = 1 TO 8: PRINT STRING$(5," "); REPEAT;: NEXT
350 PRINT TAB(14);: FOR REPEAT = 1 TO 8: PRINT STRING$(5," "); 0;: NEXT
360 PRINT TAB(14);: FOR REPEAT = 1 TO 8: PRINT STRING$(5," "); 0;: NEXT
370 DATA "JANUARY",210,"FEBRUARY",150,"MARCH",99
380 DATA "APRIL",250,"MAY",183,"JUNE",352
390 DATA "JULY",410,"AUGUST",390,"SEPTEMBER",300
400 DATA "OCTOBER",651,"NOVEMBER",724,"DECEMBER",516
410 IF INKEY$ = "" THEN 410
420 END
```

such as 176 and 178, for shading, or we could plot solid bars in white and high-intensity white. With the Color/Graphics board, we could produce bars of alternate colors (see Fig. E of the insert). If there is enough room on the graph, we can leave some space between the bars to help distinguish them.

Figure 4–8 Labeled bar graph produced by Prog. 4–7, using character graphics.



Program 4–8 Labeled bar graph using line drawing.

```
10 'PROGRAM 4-8. LABELED VERTICAL BAR CHART USING PIXEL GRAPHICS
20      'SALES VALUES (in the range of 0 - 800) ARE SCALED TO
30      'PIXELS 28 - 156.  MONTHS USE EVERY 40 PIXELS, STARTING
40      'AT 136.
50  DIM X(12), Y(12)
60  SCREEN 2: CLS
70  PRINT TAB(31); "ANNUAL SALES FIGURES"
80     'MAKE NOTCHES FOR SALES MAGNITUDES
90  FOR Y = 28 TO 156 STEP 8
100     LINE (125,Y) - (131,Y)
110 NEXT
120     'LABEL THE NOTCHES
130 LOCATE  3,12: PRINT "Thousands"
140 ROW = 20
150 FOR SALESAMOUNT = 0 TO 800 STEP 100
160     LOCATE ROW,13: PRINT USING "###"; SALESAMOUNT
170     ROW = ROW - 2
180 NEXT
190     'MAKE NOTCHES FOR MONTHS
200 FOR COLUMN = 19 TO 74 STEP 5
210     LOCATE 21,COLUMN: PRINT "+";
220 NEXT
230 LINE (128,24) - (608,163),,B
240 PRINT TAB(19);"J    F    M    A    M    J    J    A    S    O    N    D"
250 PRINT TAB(19);"A    E    A    P    A    U    U    U    E    C    O    E"
260 PRINT TAB(19);"N    B    R    R    Y    N    L    G    P    T    V    C";
270 LOCATE 11,3: PRINT "MONTHLY"
280 LOCATE 13,3: PRINT " SALES"
290     'DRAW CHART BARS
300 RANGERATIO = (156 - 28) / (800 - 0)
310 BEGINBAR = 136
320 FOR MONTH = 1 TO 12
330     READ SALES
340     Y = INT((800 - SALES) * RANGERATIO + 28 + .5)
350     LINE (BEGINBAR,Y) - (BEGINBAR+24,156),,BF
360     BEGINBAR = BEGINBAR + 40                    'NEXT BAR IS 40 PIXELS OVER
370 NEXT
380 DATA 210,150,99,250,183,352,410,390,300,651,724,516
390 IF INKEY$ = "" THEN 390
400 END
```

An example of the use of pixel graphics in the construction of bar graphs is given in Prog. 4.8. This pixel bar graph is plotted in Fig. 4–9. We have made the width of the bars in this graph greater than the spacing between the bars. This is a good practice, as narrower bars are usually less effective.

Color in bar graphs can be used to provide greater clarity or to improve the appearance of the graph. The choice of color combinations should be carefully considered, as discussed in Section 2–4. Inclusion of too many colors or clashing colors can actually decrease the effectiveness of the graph. Using colors in pixel graphs means that we must create the graph in medium resolution, which gives us double-wide characters. Since the labeling will now be bigger, we will have to allow more of our screen space for the character strings when we use color, and the area available for plotting a data set will be reduced. A color version of Fig. 4–9 is shown as Fig. F of the color insert.

Various shading patterns can be used with color or black-and-white graphs. As with colors, we should select shading patterns that do not detract from the

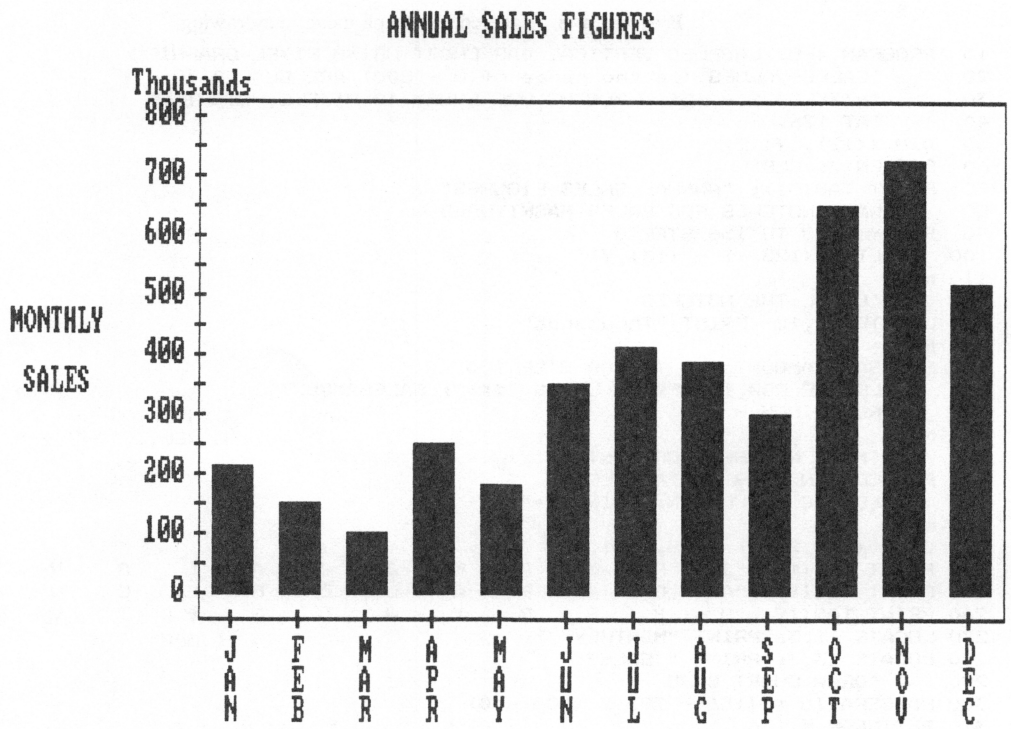


Figure 4–9 Labeled bar graph produced by Prog. 4–8, using pixel graphics.

graph's effectiveness. Bizarre or clashing patterns should be avoided. If adjacent areas are to be shaded, graduating the shading used from darkest to lightest is most effective. Program 4–9 produces a graph shaded according to this scheme, as shown in Fig. 4–10.

Bar charts can be constructed using a combination of pixel and character methods. We draw the graph outline using the LINE (or DRAW) statement and

Program 4–9 Shaded bar graph using pixel graphics.

```
10 'PROGRAM 4-9. SHADED COLUMN CHART USING PIXEL GRAPHICS
20      'SALES VALUES (in the range of 0 - 800) ARE SCALED TO
30      'PIXELS 28 - 156.  EACH QUARTER USES 64 PIXELS ACROSS,
40      'STARTING AT 160.
50  SCREEN 2: CLS
60  PRINT TAB(28); "QUARTERLY SALES BY REGION"
70  LINE (128,24) - (608,162),,B          'MAKE BOX FOR CHART
80  FOR Y = 36 TO 156 STEP 8              'MAKE NOTCHES FOR LABELING
90      LINE (125,Y) - (131,Y)
100 NEXT
110 ROW = 20
120 FOR SALESAMOUNT = 0 TO 30 STEP 10             'LABEL THE NOTCHES
130     LOCATE ROW,13: PRINT USING "##"; SALESAMOUNT
140     ROW = ROW - 5
```

Program 4–9 (cont.)

```
150 NEXT
160     'LABEL QUARTERS
170 LOCATE 23,22: PRINT " FIRST       SECOND        THIRD       FOURTH"
180 PRINT TAB(22); "QUARTER      QUARTER      QUARTER      QUARTER";
190 LOCATE 12,3: PRINT "SALES"
200 LOCATE 13,1: PRINT "(millions)"
210 LOCATE 8,67: PRINT "WEST"                        'LABEL REGIONS
220 LOCATE 13,67: PRINT "SOUTH"
230 LOCATE 18,67: PRINT "MIDWEST"
240     'CONSTRUCT BARS, ONE FOR EACH QUARTER
250 RANGERATIO = (156 - 36) / (30 - 0)
260 XLEFT = 160
270 XRIGHT = XLEFT + 64
280 FOR QUARTER = 1 TO 4
290     YBOTTOM = 156
300     'ADJUSTMENT IS TO FIX VALUE YTOP (THE SCALED SALES MAGNITUDE)
310     'FOR EACH REGION.  ADJUSTMENT IS NECESSARY SINCE THE SHADED
320     'AREAS REPRESENTING QUARTERLY SALES FOR EACH REGION ARE STACKED
330     'ON TOP OF EACH OTHER.
340     ADJUSTMENT = 0
350     FOR DISTRICT = 1 TO 3
360         READ SALES
370         'CONVERT TO MILLIONS
380         SALES = SALES / 1000000!
390         YTOP = INT((30 - SALES) * RANGERATIO + 36 + .5)
400         'ADJUST YTOP BY ADJUSTMENT - THE AREA THAT HAS ALREADY
410         'BEEN TAKEN UP BY PREVIOUS REGIONS SALES MAGNITUDES.
420         YTOP = YTOP - ADJUSTMENT
430         LINE (XLEFT,YTOP) - (XRIGHT,YBOTTOM),,B
440         'FILL IN UPPER RIGHT TRIANGLE
450         FOR X1 = XRIGHT TO XLEFT STEP -DISTRICT * 3
460             X = X1
470             Y = YTOP
480             PSET (X,Y)
490             Y = Y + 1
500             X = X - 1
510             IF Y <= YBOTTOM AND X > XLEFT THEN 480
520         NEXT
530         'FILL IN LOWER LEFT TRIANGLE
540         FOR Y1 = YTOP TO YBOTTOM STEP DISTRICT * 3
550             Y = Y1
560             X = XRIGHT
570             PSET (X,Y)
580             Y = Y + 1
590             X = X - 1
600             IF Y < = YBOTTOM AND X > XLEFT THEN 570
610         NEXT
620         'FIND ADJUSTMENT THAT WILL BE NEEDED FOR NEXT YTOP
630         ADJUSTMENT = 156 - YTOP
640         YBOTTOM = YTOP
650     NEXT
660     'ADVANCE ACROSS TO NEXT QUARTER'S COLUMN
670     XLEFT = XLEFT + 96
680     XRIGHT = XLEFT + 64
690 NEXT
700 DATA 7000000,11000000,4000000
710 DATA 8800000,10500000,7000000
720 DATA 4000000,12000000,8500000
730 DATA 7000000,11333000,10500000
740 IF INKEY$ = "" THEN 740
750 END
```

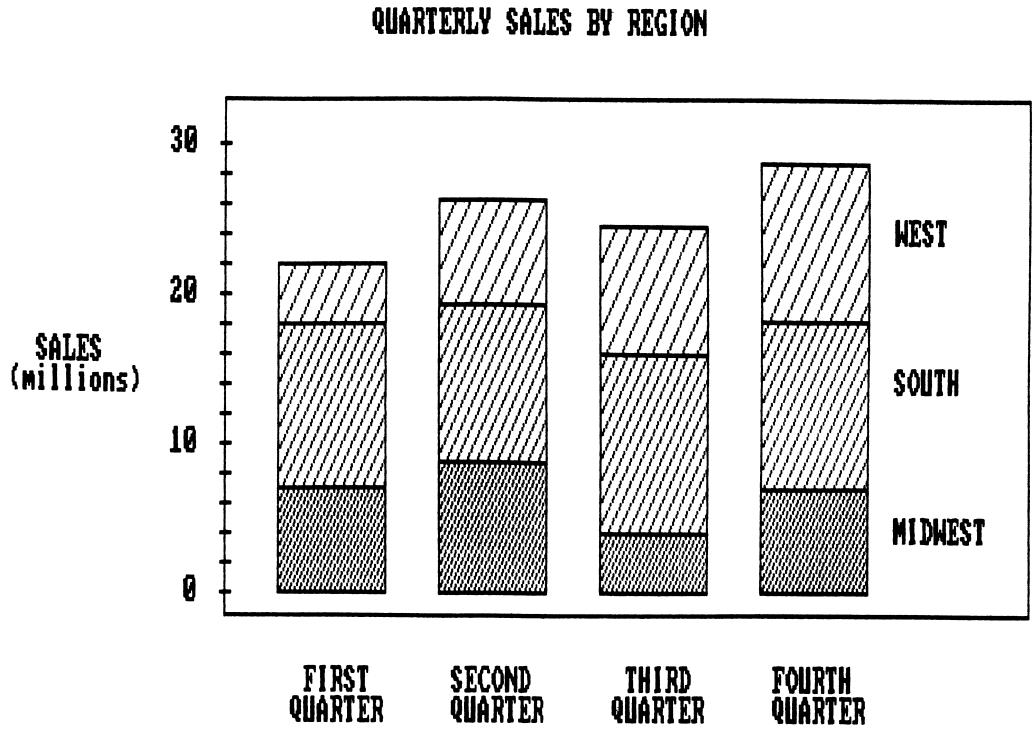


Figure 4–10 Bar graph with shading, output by Prog. 4–9.

shade the bar interiors using ASCII codes. Since each character is composed of either an 8 by 8 pixel grid (Color/Graphics) or a 9 by 14 grid (Monochrome), we would need to have our pixel lines set up along the character grid boundaries. Otherwise, the characters will overlap the lines. With the Color/Graphics option, we must draw our lines at pixel coordinates that are multiples of 8 units apart. These coordinate positions are at the values 0, 8, 16, 32, and so on, in both the X and Y directions.

PROGRAMMING PROJECTS

4–1. Write a program to display a trend graph with data magnitudes plotted horizontally. Use the PRINT statement and the relationship in (4–1) to position the graph on the screen. Input to the program will include the minimum and maximum print positions and the minimum and maximum data values. Data range, print position range, and print position for each data point will then be calculated. Allow for any number of data points.

4–2. Write a program to display a trend graph with data magnitudes plotted vertically, using the PRINT statement. Include a routine to determine the minimum and maximum data values from the input data set, containing an arbitrary number of data points. Input the minimum and maximum print line numbers and calculate print line range and data range. Position each data point according to the relationship (4–2).

4–3. Modify Prog. 4–7 to produce a vertical bar chart with the PRINT statement methods. Separate the bars and use a single character for a shading pattern. The ASCII code for the shading character is to be set by input.

4–4. Write a program to plot a pixel data trend graph horizontally, with small, colored rectangles placed at each data position. Input will include minimum and maximum pixel positions. The minimum and maximum data values will be determined from the input data set. Calculations will include the data range, horizontal pixel range, and X coordinate for each data value.

4–5. Modify the program of Project 4–4 to join the plotted data values (rectangles) with straight lines to produce a "curve" plot. Also draw coordinate axes with labeling.

4–6. Modify Prog. 4–9 to produce a horizontal bar graph. Allow colors and shading patterns to be selected as input.

4–7. Set up a program to output a labeled, vertical bar chart. Use character codes to shade the bar interiors. Calculated bar boundaries should be rounded to the nearest character boundary to avoid overlapping characters with the bar outlines.

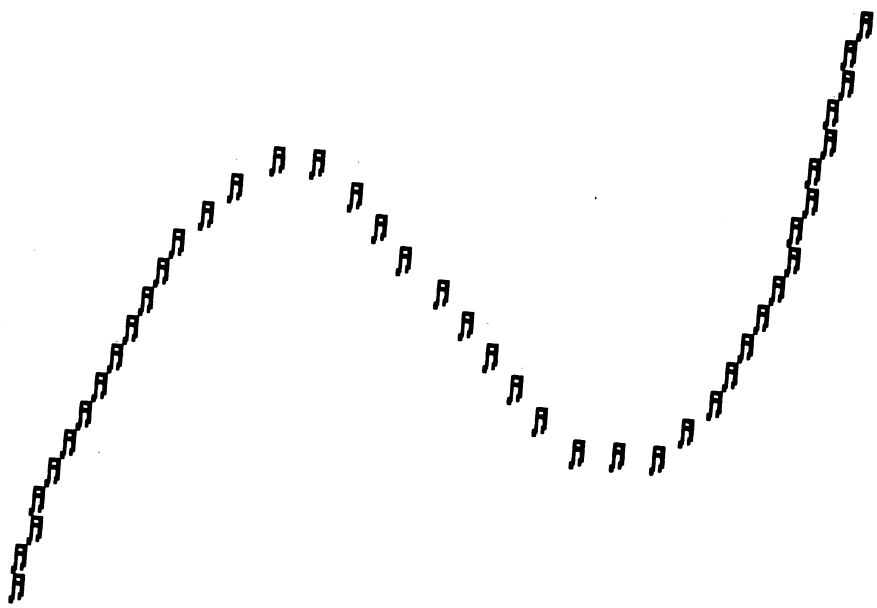
4–8. Write a program to interactively create a graph as the data are input. That is, the points or bars will be displayed as each data point is entered. Minimum and maximum data values can be entered first, together with the pixel range desired.

Chapter 5

Drawing Curves

Curved lines can be incorporated into our pictures and graphs using basic methods introduced in the preceding chapters. A PRINT statement approximation of the shape of a curved line is shown in Fig. 5–1. We determine character print positions along the curve path from the equation for the curve or from a plot of the

Figure 5–1 Printed characters can be used to approximate the general shape of a curve.



curve on graph paper. Since the PRINT statement method has limited effectiveness for displaying curves, our discussion will concentrate on pixel graphics methods.

Pixel graphics provides a means for more accurate representation of curves. We can approximate a curve shape with straight line segments or closely spaced points. Figure 5–2 shows the effect of varying the line segment length on the appearance of a curve. As the number of line segments included between the arc endpoints is increased, the curve appears smoother. However, the more line segments we use, the more time the system will take to create the display. In some applications, such as animation, we want the display to be produced rapidly. So we might accept fewer points and line segments in the approximation of a curve in order to attain greater speed in the execution of the program.

5–1 CIRCLES

The most frequently encountered curve is the circle. This curve is useful as a fundamental component in building pictures and in displaying pie charts (Section 5–4).

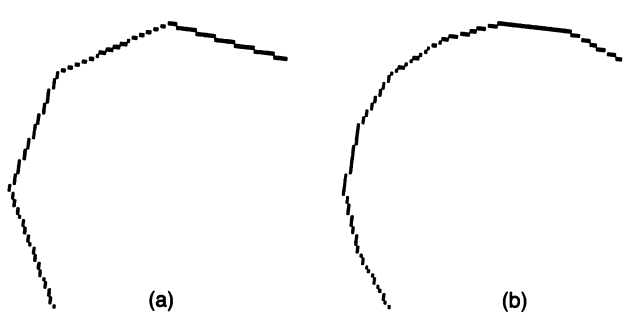
To plot a circle on a video screen, we need to specify its position and size (Fig. 5–3). Position is specified as the coordinates (XC,YC) of the center of the circle. Size is given by the radius R. We produce a circle by specifying these parameters in the CIRCLE statement or in circle equations.

CIRCLE STATEMENT

A circle-drawing command is available to us in advanced BASIC. In its simplest form, we can plot a circle on the screen with the statement

```
CIRCLE (XC,YC),R
```

Figure 5–2 Approximating a curve: (a) with 4 straight line segments and (b) with 8 straight line segments.



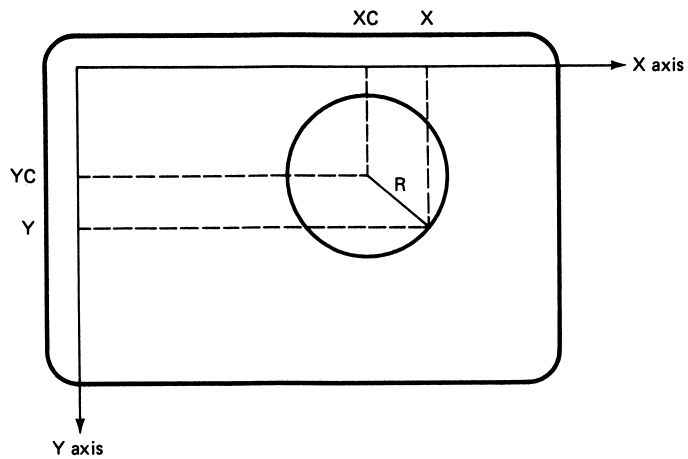


Figure 5–3 Circle with radius R and center coordinates (XC,YC).

Coordinates for the circle center (XC,YC) can be given in absolute or relative form, and the radius must be a positive number. The following program segment will produce a line of 13 circles diagonally down the screen, from left to right:

```
10  CLS: SCREEN 1
20  CIRCLE (30,10),10
30  FOR K = 1 TO 12
40        CIRCLE STEP (20,15),10
50  NEXT K
```

The last screen point referenced with the CIRCLE command is the circle center. This is the point referenced by subsequent DRAW statements or statements involving relative coordinates.

Additional parameters can be included in the CIRCLE command. We draw circles in a palette color by adding the color code after the radius. The statement

```
CIRCLE (100,100),50,1
```

will draw a circle with center at (100,100), radius 50, and color code 1 (green in palette 0, cyan in palette 1). If C is omitted, the circle will be drawn in color code 3 when in medium resolution.

We can also draw a circular arc with the CIRCLE command. This is accomplished by giving the values for the beginning and ending angles of the arc right after the color code specification. These angle values must be expressed in radians, and they are measured counterclockwise from a horizontal line through the circle center, as shown in Fig. 5–4. Radian angles vary from a value of 0 at the horizontal to 6.283185 (2 * PI) after one complete revolution back to the horizontal. For example,

```
CIRCLE (160,100),75,,1.57080,4.71239
```

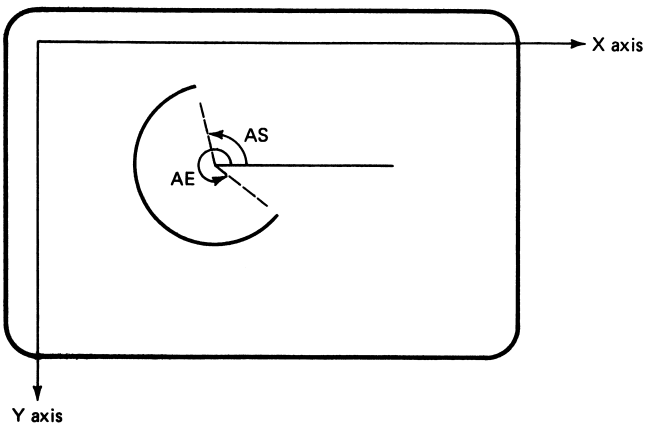


Figure 5–4 A circular arc drawn with the CIRCLE statement from a starting angle AS to an ending angle AE. Angles are measured in a counterclockwise direction from the horizontal.

draws the left half of a circle with a radius of 75 in the center of the screen. The radian values from 0 to 2 * PI correspond to the degree range 0 to 360. Thus the circle segment above starts at 90 degrees and ends at 270 degrees. If either the start or stop angle is omitted, a zero value is assumed for the parameter omitted.

On some monitors, a circle drawn with the CIRCLE command will appear flattened in either the horizontal or vertical direction. In this case, we have an ellipse instead of a circle. This is due to resolution differences in the X and Y directions. A final parameter may be included to adjust for resolution differences between monitors (or to produce ellipses). If we omit this parameter, a value of 5/6 is assumed in medium resolution and a value of 5/12 is assumed in high resolution. This results in a "circle" plotted with a radius R in the X direction and a radius of R * 5/6 (or R * 5/12) in the Y direction. To adjust for resolution differences on any monitor, we set this parameter to the aspect ratio for that monitor. For instance, the statement

```
CIRCLE (160,100),75,,,,1
```

will produce a circle on monitors for which the X resolution is equal to the Y resolution.

We can summarize the parameter options available with the CIRCLE command as

CIRCLE (XC,YC),R,C,AS,AE,ASP — Plots a circular (or elliptical) arc with center position (XC,YC), radius R, color C, beginning angle AS, ending angle AE, and aspect ratio ASP. Each parameter may be a numeric constant or expression. If noninteger, parameters are rounded.

Parameters AS and AE specify starting and ending angles for circular arcs. These angles may be specified in the range −2 * PI to 2 * PI. Negative angles produce circular arcs with the arc endpoints connected to the circle center. The parameter

Program 5–1 Picture (man in the moon) constructed with circular arcs, using the CIRCLE command.

```
10 'PROGRAM 5-1. MAN IN THE MOON FROM DIFFERENT-SHAPED ARCS
20  SCREEN 2: CLS
30  CIRCLE (250,100),200,,4.9,1.4,.37              'OUTER CURVE
40  CIRCLE (250,100),90,,5.1,5.73,.8499999         'CHIN
50  CIRCLE (250,100),90,,.2,1.2,.8499999           'FOREHEAD
60  CIRCLE (330,110),16,,2.2,5.2,.4                'NOSE
70  CIRCLE (320,90),20,,4.8,.5,.7                  'BRIDGENOSE
80  CIRCLE (250,100),90,,5.85,6.05,.8499999        'MOUTH-TO-NOSE
90  CIRCLE (400,110),35,,1.9,4.7,.4                'CHEEK
100 CIRCLE (345,115),45,,4.4,5.4,.35               'MOUTH TOP
110 CIRCLE (335,126),35,,4.5,6.05,.35              'MOUTH BOTTOM
120 CIRCLE (360,90),9                              'EYE
130 PAINT (355,110),1,1                            'FILL IN INTERIOR
140 CIRCLE (400,110),35,0,1.9,4.7,.4               'CHEEK
150 CIRCLE (360,90),18,0,.3,2.5,.9                 'EYEBROW
160 IF INKEY$ = "" THEN 160
170 END
```

ASP allows us to specify an aspect ratio to correct for resolution differences or to produce elliptical arcs.

An example of the use of the CIRCLE command is given in Prog. 5–1 for a monitor with an aspect ratio of 0.92 in medium resolution (0.46 in high resolution). We construct the picture shown in Fig. 5–5 with circular arcs. The PAINT command is used to fill the figure interior.

POINT-PLOTTING METHODS

For applications involving the movement of objects along circular paths, we will need to know how to plot points along such paths. Also, these methods allow us to produce circles if we do not have advanced BASIC.

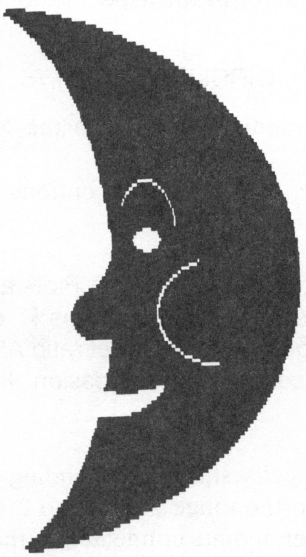


Figure 5–5 A picture drawn with circular arcs by Prog. 5–1, using the CIRCLE command.

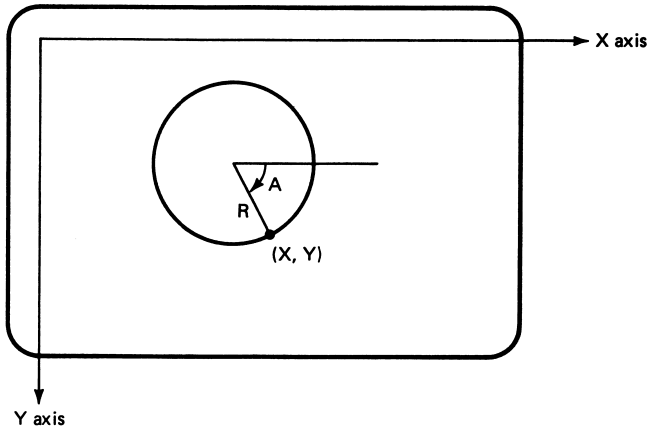


Figure 5–6 Coordinate positions (X,Y) along a circular path are determined from the radius R and values of the angle A, measured clockwise from the horizontal.

Circle equations can be stated in several forms. A convenient form is to determine successive coordinate positions (X,Y) along the circle boundary from the value of the angle measured clockwise from a horizontal line (Fig. 5–6). Using this angle, we can calculate X and Y values from XC, YC, R, and A as

$$X = XC + R * COS(A)$$
$$Y = YC + R * SIN(A) \qquad (5\text{–}1)$$

As with the CIRCLE command, angles must be specified in radians.

We can set up a circle-generating program, using the calculations in (5–1), to display a circle approximated with either straight line segments or closely spaced pixels. Program 5–2 accepts the parameters XC, YC, R, and the number of points to be plotted as input and produces a "circle" formed with straight line segments. The output is shown in Fig. 5–7 for two input sets.

Including fewer line segments along the boundary will speed up the drawing of circles, but the individual lines may become more noticeable. Then we have a polygon appearance instead of a circle (Fig. 5–7(a)). To modify our circle program for resolution differences, we multiply the term R * SIN(A) by the ratio of Y resolution to X resolution.

Plotting pixels along the circumference and omitting the connecting line segments will also speed up circle drawing. This saves the time of drawing the line segments, and the eye tends to fill in the curved path between pixels (Fig. 5–8) if the pixels are not plotted too far apart. The more pixels we include, the better the approximation. We obtain the best possible approximation to a circle when the pixels are as close as we can plot them. In this case the pixels are plotted at the adjacent grid points closest to the desired circular path. The distance between adjacent grid points is 1 unit in the horizontal and vertical directions. Therefore, the angular separation (in radians) of two grid points on a circle can be

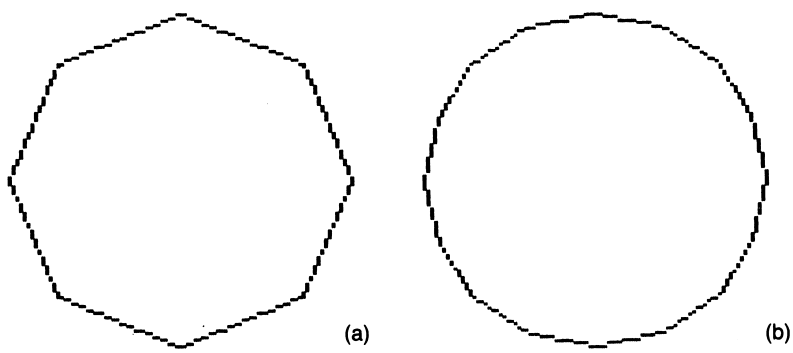
Program 5–2 Circle generator using line drawing and angular increments.

```
10 'PROGRAM 5-2. CIRCLE GENERATOR.
20  SCREEN 0: WIDTH 80: CLS
30  INPUT "ENTER COORDINATES FOR CENTER OF CIRCLE"; XCENTER,YCENTER
40  IF XCENTER < 0 OR XCENTER > 319 OR YCENTER < 0 OR YCENTER > 199 THEN 200
50  INPUT "ENTER RADIUS OF CIRCLE"; RADIUS
60  IF RADIUS < 0 THEN 200
70  IF XCENTER + RADIUS > 319 OR XCENTER - RADIUS < 0 OR YCENTER + RADIUS > 199
      OR YCENTER - RADIUS < 0 THEN 200
80  INPUT "ENTER NUMBER OF POINTS TO BE PLOTTED"; TOTALPOINTS
90  DA = 6.28318 / TOTALPOINTS          '6.28318 IS RADIAN EQUIVALENT
100 SCREEN 1: CLS                       'FOR 360 DEGREES
110 XFIRST = XCENTER + RADIUS: YFIRST = YCENTER
120 FOR ANGLE = DA TO 6.28318 STEP DA
130     XSECOND = XCENTER + RADIUS * COS(ANGLE)
140     YSECOND = YCENTER + RADIUS * SIN(ANGLE) * .9199999    '.92 IS RESOLUTION
150     LINE (XFIRST,YFIRST) - (XSECOND,YSECOND)                'CORRECTION
160     XFIRST = XSECOND: YFIRST = YSECOND              'SAVE NEW ENDPOINT
170 NEXT
180 LINE (XFIRST,YFIRST) - (XCENTER+RADIUS,YCENTER)
190 GOTO 240
200 PRINT "COORDINATE OUT OF RANGE.  ENTER X TO EXIT OR R TO REPEAT."
210 INPUT C$
220 IF C$ = "X" THEN 240
230 IF C$ = "R" THEN 30
240 IF INKEY$ = "" THEN 240
250 END
```

approximated as the inverse of the radius of the circle, as shown in Fig. 5–9. This approximation works well for most cases. A smaller angular step size will ensure that the circle contains no gaps, but then we duplicate the calculation and plotting of some of the points.

Another time-saver we can employ is to take the symmetry of the circle into account. We do not have to calculate every point on the curve individually. The top half has the same shape as the bottom half; the left half has the same shape as the right. This means that each X value on the circle corresponds to two Y values, and each Y value corresponds to two X values, as illustrated in Fig. 5–10. Taking this idea a step further, we can get four more points on the circle by interchanging

Figure 5–7 Circle approximated with (a) 8 straight line segments and (b) 16 line segments, by Prog. 5–2.



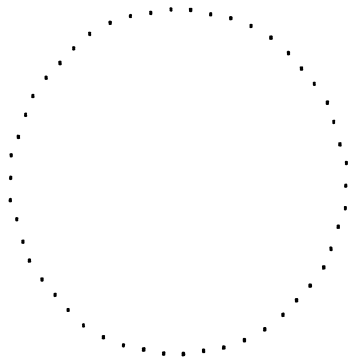


Figure 5–8 Circle plotted with pixels.

all the X and Y coordinates. That is, if (X,Y) is a point on the circle, then (Y,X) is also on the circle. This means that we only have to calculate the points along one-eighth of the circular path (a 45-degree segment). All of the remaining points on the full circle can be obtained from these points. Figure 5–11 shows the eight points that can be plotted on a circle by calculating only the position of the one point at coordinates (9,2). The circle in this figure is centered at the origin of a coordinate system. For a circle centered at position (XC,YC), we add XC to all X-coordinate values and we add YC to all Y-coordinate values. This moves the circle from the origin to the desired position. In Chapter 7 we discuss the movement of displayed objects from one location to another in greater detail.

Program 5–3 illustrates the plotting method for circle generation that calculates only the points in the interval from 0 to 45 degrees. The remaining points are obtained by symmetry. We could eliminate the four point-plotting statements at the beginning of Prog. 5–3 by starting the loop with the value A = 0. This would result in fewer program statements, but each of the four initial points would then be plotted twice. Some other points may be plotted twice in any case. This occurs for larger values of R and is the result of the rounding process in the PSET command. Also, this program will leave some small gaps in those circles with small values of R. This is again due to the rounding of the coordinate values to be plotted.

There are additional improvements we can make in the speed of circle-drawing programs, but the methods we have discussed will be satisfactory for most applications. We will look at some additional techniques when we consider rotations in Chapter 7 and animation in Chapter 8.

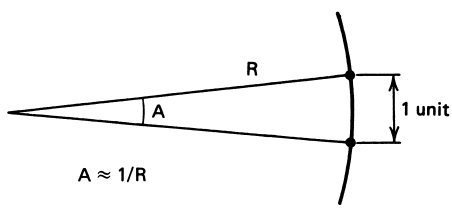


Figure 5–9 Relation between angular separation A (in radians) of two points plotted one unit apart and the radius R of a circle passing through the two points.

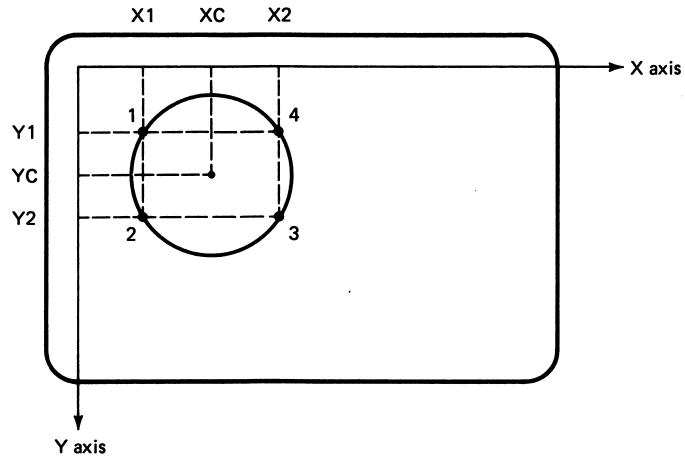
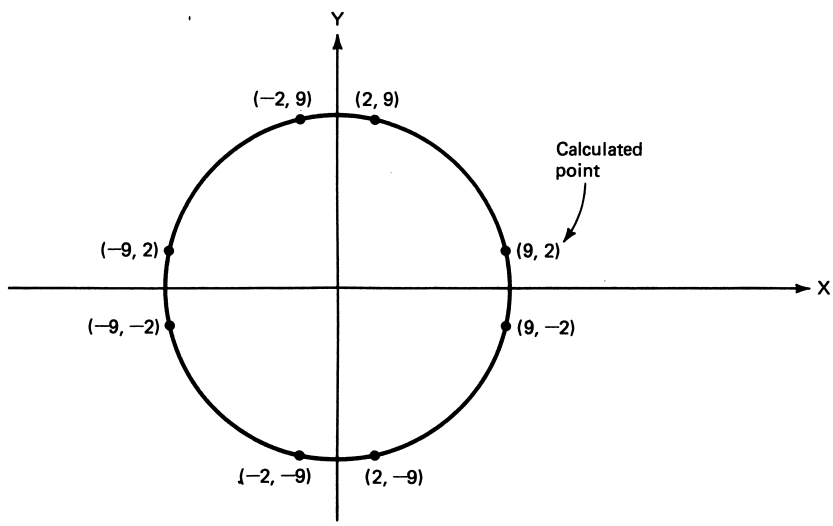


Figure 5–10 Symmetry of a circle. Points 1 and 2 have the same X value; points 1 and 4 have the same Y value.

We can fill the interiors of circles with colors or shading patterns by drawing lines between the boundary points. For solid colors, we can use the PAINT command available in advanced BASIC. If we draw the individual lines ourselves, we can plot them vertically, horizontally, or diagonally. They can be closely spaced for a solid color or spread out to make different shading patterns. We can even draw the lines in random order and in various colors to provide special effects.

Figure 5–11 If the pixel at location (9,2) is calculated to be on a circle, then all eight points shown can be plotted.



Program 5-3 Circle generator using point plotting and angular increments.

```
10 'PROGRAM 5-3. CIRCLE GENERATOR.
20      'CALCULATES POINTS ON THE CIRCLE FROM 0 TO 45
30      'DEGREES AND PLOTS ALL SYMMETRIC POINTS.
40  SCREEN 0: WIDTH 80: CLS
50  INPUT "ENTER COORDINATES FOR CENTER OF CIRCLE"; XCENTER,YCENTER
60  IF XCENTER < 0 OR XCENTER > 319 OR YCENTER < 0 OR YCENTER > 199 THEN 330
70  INPUT "ENTER RADIUS OF CIRCLE"; RADIUS
80  IF RADIUS < 0 THEN 330
90  IF XCENTER + RADIUS > 319 OR XCENTER - RADIUS < 0 OR YCENTER + RADIUS > 199
      OR YCENTER - RADIUS < 0 THEN 330
100 SCREEN 1: CLS
110 PSET (XCENTER+RADIUS,YCENTER)
120 PSET (XCENTER-RADIUS,YCENTER)
130 PSET (XCENTER,YCENTER+RADIUS * .9199999)
140 PSET (XCENTER,YCENTER-RADIUS * .9199999)
150 DA = 1 / RADIUS
160 RADIAN45 = 45 * 3.141593 / 180      'FIND RADIAN EQUIVALENT FOR 45 DEGREES
170 FOR ANGLE = DA TO RADIAN45 STEP DA
180     DX = RADIUS * COS(ANGLE)
190     DY = RADIUS * SIN(ANGLE)
200     GOSUB 230
210 NEXT
220 GOTO 360
230     'PLOT ALL SYMMETRIC POINTS
240 PSET (XCENTER+DX,YCENTER+DY * .9199999)
250 PSET (XCENTER-DX,YCENTER+DY * .9199999)
260 PSET (XCENTER+DX,YCENTER-DY * .9199999)
270 PSET (XCENTER-DX,YCENTER-DY * .9199999)
280 PSET (XCENTER+DY,YCENTER+DX * .9199999)
290 PSET (XCENTER-DY,YCENTER+DX * .9199999)
300 PSET (XCENTER+DY,YCENTER-DX * .9199999)
310 PSET (XCENTER-DY,YCENTER-DX * .9199999)
320 RETURN
330 INPUT "COORDINATE OUT OF RANGE.  ENTER X TO STOP OR R TO REPEAT."; C$
340 IF C$ = "X" THEN 370
350 IF C$ = "R" THEN 50
360 IF INKEY$ = "" THEN 360
370 END
```

To create shading patterns, we draw interior lines with endpoints on the circle boundary. We can calculate these coordinate endpoints with equations (5–1) or with the following equation, which eliminates the need for an angle specification:

$$(X - XC) \wedge 2 + (Y - YC) \wedge 2 = R \wedge 2 \qquad (5\text{–}2)$$

For any X value chosen between XC − R and XC + R, we would calculate the Y endpoints as Y1 = YC − SQR(R ^ 2 − (X − XC) ^ 2) and Y2 = YC + SQR(R ^ 2 − (X − XC) ^ 2). A vertical line can then be drawn from (X,Y1) to (X,Y2). Horizontal lines are obtained by choosing Y values between YC − R and YC + R and then calculating the X endpoints for each Y value from (5–2) as X1 = XC − SQR(R ^ 2 − (Y − YC) ^ 2) and X2 = XC + SQR(R ^ 2 − (Y − YC) ^ 2). This method can be useful for shading circles, but it is not a good way to draw the circle boundary. It leaves gaps unless we join the pixels with line segments or use an X (or Y) increment less than 1. But eliminating the gaps by taking small coordinate

steps results in plotting many of the pixels more than once. To compensate for resolution differences, we either adjust Y1 and Y2 by multiplying the SQR term in the calculations by the aspect ratio, or we adjust X1 and X2 by multiplying this term by the inverse of the aspect ratio.

5–2 OTHER CURVES

Although the circle is the curve we will use most often, various other curves have frequent applications in graphics. We can display these other curves with methods similar to those employed for circles. Pixels on the curve (or the character print positions) can be calculated from the equations for the curve and adjusted for the resolution difference, if necessary. We can then plot the pixels at convenient locations on the screen and join these pixels with line segments. For some curves, we may be able to take symmetry or other considerations into account to reduce computation.

The curves discussed in this section can be used to graphically model the applications areas cited or, in some cases, to provide a curve fit to data tables. Curve-fitting methods (such as the least-squares method) allow a smooth curve representation to be plotted for a set of tabular data points.

ELLIPTICAL CURVES

An elliptical curve may be thought of as a variation of the circle, although in the strict sense the circle is a special case of an ellipse. If we stretch a circle in one direction (say, the X direction), we have an ellipse. Equations for the ellipse can be written in the form

$$\begin{aligned} X &= XC + RX * COS(A) \\ Y &= YC + RY * SIN(A) \end{aligned} \tag{5–3}$$

In these equations, A is the angle measured in radians from the horizontal in a clockwise direction (Fig. 5–12). If RX > RY, the ellipse is longer in the X direction. If RY > RX, the ellipse is longer in the Y direction. We have a circle for the case RX = RY.

Elliptical curves are useful in many areas of graphics modeling. The orbits of satellites are elliptical. Some machine and equipment parts have elliptical shapes. A three-dimensional view of a cylinder will show the ends of the cylinder as ellipses when viewed at an angle. Having the capability to readily display ellipses increases our flexibility to produce a broad range of graphics applications displays.

The programs to generate circles in Section 5–1 can be modified to produce either circles or ellipses. We can accomplish this by replacing the equations in (5–1) with the more general equations in (5–3). In advanced BASIC, we can use the

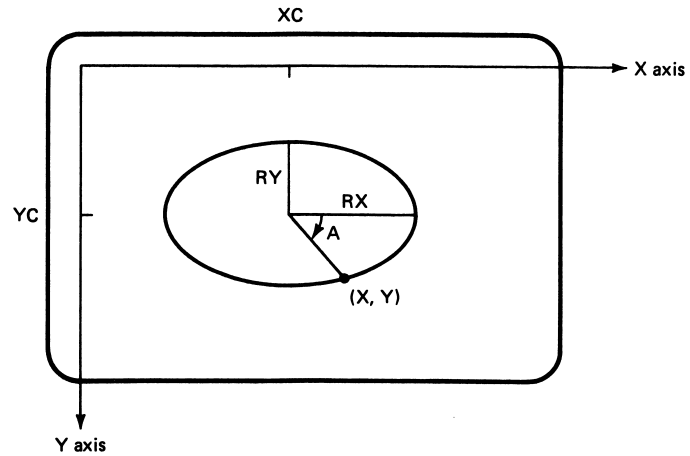


Figure 5–12 An ellipse plotted from equations (5–3) with RX > RY and center at coordinates (XC,YC).

CIRCLE command to get an ellipse by altering the value of the parameter ASP. To get an ellipse with radius RX in the X direction and radius RY in the Y direction, we multiply ASP by the ratio RY/RX and set R either to RX or to RY depending on whether this product is less than 1 or greater than 1. Thus if RY is twice the size of RX, we multiply ASP by 2. Then, if 2 * ASP is greater than 1, we set R to RY. Otherwise, we set R to the value of RX.

SINE CURVES

We can write the general equation for a sine curve as

$$Y = H * SIN(W * X + D) \qquad (5–4)$$

Figure 5–13 shows a plot of the sine curve drawn on a conventional coordinate reference system. The frequency W specifies the number of oscillations (or cycles) of the curve for a given range of X. The parameter D specifies the displacement (shift) of the curve to the right or left. When D is set to a value of zero, we have the standard sine curve, and D = PI/2 produces the standard cosine curve. Figure 5–14 plots three cycles of the sine curve for parameter values H = 50, W = 2 * PI/50, D = 0 over the range X = 0 to X = 150. This curve is produced by Prog. 5–4.

Output from Prog. 5–4 is obtained by taking unit steps in the X direction, calculating the Y values, and joining the resulting points with line segments. This program does not consider the symmetry of the function. However, every one-quarter cycle of a sine curve can be repeated from the points between X = −D/W and X = (PI/2 − D)/W: the "first quadrant." That is, if we know that (X,Y) is a point on the sine curve in the first quadrant, then the following points are also on

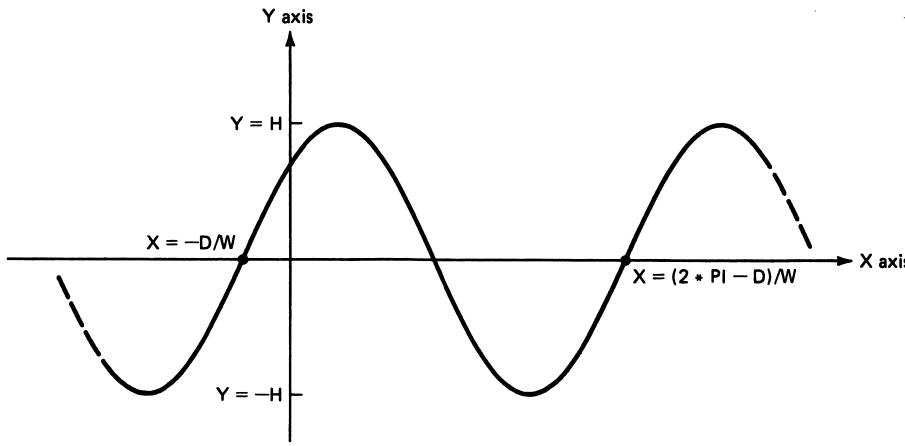


Figure 5–13 Standard sine curve of equation (5–4). One cycle of the curve is plotted between X values of −D/W and (2*PI-D)/W, while the Y coordinate oscillates between a maximum value of H and a minimum value of −H.

the curve for the first complete cycle: (PI − X,Y), (3 * PI/2 − X,−Y), (2 * PI − X, −Y). These symmetry points can be plotted repeatedly through as many cycles as we wish to display without having to recompute values from equation (5–4).

Sine curves are useful in graphics applications involving repeated motion. These applications include simulating voice patterns, music, the vibrations of a spring, the bouncing of a ball, or the swing of a pendulum. In the case of a spring or ball we also have to account for friction. The amplitude of the motion decreases with each cycle. We can model this decrease in amplitude by multiplying the sine function in (5–4) by the exponential function EXP(−K * X). The constant K determines the rate at which the amplitude decreases. A value of 0.1 for K will decrease the amplitude by a factor of approximately 1/2 after one cycle.

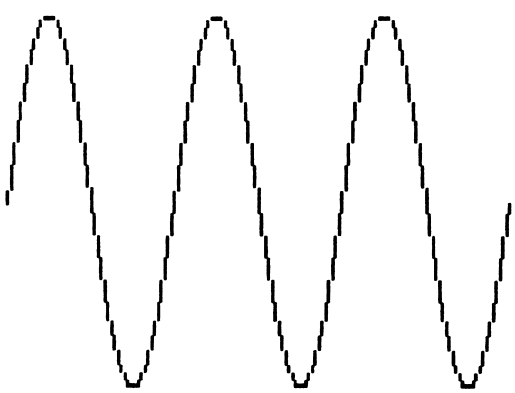


Figure 5–14 Three cycles of a sine curve plotted by Prog. 5–4.

Program 5–4 Plotting a sine curve.

```
10 'PROGRAM 5-4. SINE CURVES.
20      'GENERATES SINE CURVES CENTERING THE CURVE VERTICALLY
30      'ON THE SCREEN. SINCE Y VALUES OF THE SINE CURVE ARE
40      'BOTH POSITIVE AND NEGATIVE, CALCULATED Y VALUES MUST
50      'BE ADJUSTED TO EXTEND UP OR DOWN FROM THE VERTICAL
60      'CENTER LINE OF THE SCREEN (Y=100), AS IN LINES 260-270.
70  SCREEN 0 : WIDTH 80: CLS
80  PRINT "PROGRAM GENERATES SINE CURVES USING"
90  PRINT
100 PRINT "      Y = H * SIN (W * X + D)        "
110 PRINT
120 PRINT "ENTER HEIGHT OF THE CURVE (H), FREQUENCY (W),"
130 PRINT "DISPLACEMENT (D).  H CAN BE NO GREATER THAN ONE-HALF"
140 PRINT "THE HEIGHT OF THE SCREEN (99)"
150 INPUT H, W, D
160 IF H > 199 / 2 THEN 120   'H CAN BE NO MORE THAN 1/2 THE SCREEN HEIGHT
170 INPUT "ENTER MINIMUM AND MAXIMUM VALUES OF X"; XLEFT,XRIGHT
180 SCREEN 1: CLS
190 FOR X = XLEFT TO XRIGHT
200     Y = H * SIN(W * X + D)
210     IF Y >= 0 THEN Y = 100 - Y
220     IF Y < 0 THEN Y = 100 + ABS(Y)
230     IF X > XLEFT THEN LINE - (X,Y) ELSE PSET (X,Y)
240 NEXT
250 IF INKEY$ = "" THEN 250
260 END
```

POLYNOMIAL CURVES

This class of curves contains an essentially endless list of equations. These equations all have the same basic structure and include the straight line and parabola. The straight line equation can be written as

$$Y = C1 * X + C2 \qquad (5–5)$$

where the constants C1 and C2 are fixed numbers that specify the slope and Y-intercept of the line. These numbers are called the coefficients of the equation. The straight line is classified as a polynomial of degree 1. Adding terms to this equation that contain higher integer powers of X produces polynomials of higher degree. A polynomial of degree 2 (a parabola) is written as

$$Y = C1 * X \wedge 2 + C2 * X + C3 \qquad (5–6)$$

Parabolas may be used to approximate the shape of data tables (for data trends or interpolative information) or to model the motions of objects. The path (trajectory) of a ball thrown across some distance describes a parabola. The ball rises to some maximum height, then drops back to the ground. Height is measured by the Y coordinate; horizontal distance is measured by the X coordinate. Maximum height for the ball will occur when the X coordinate has the value

$$X = -C2/(2 * C1) \qquad (5–7)$$

Figure 5–15 plots the parabolic trajectory for a set of coefficients input to Prog. 5–

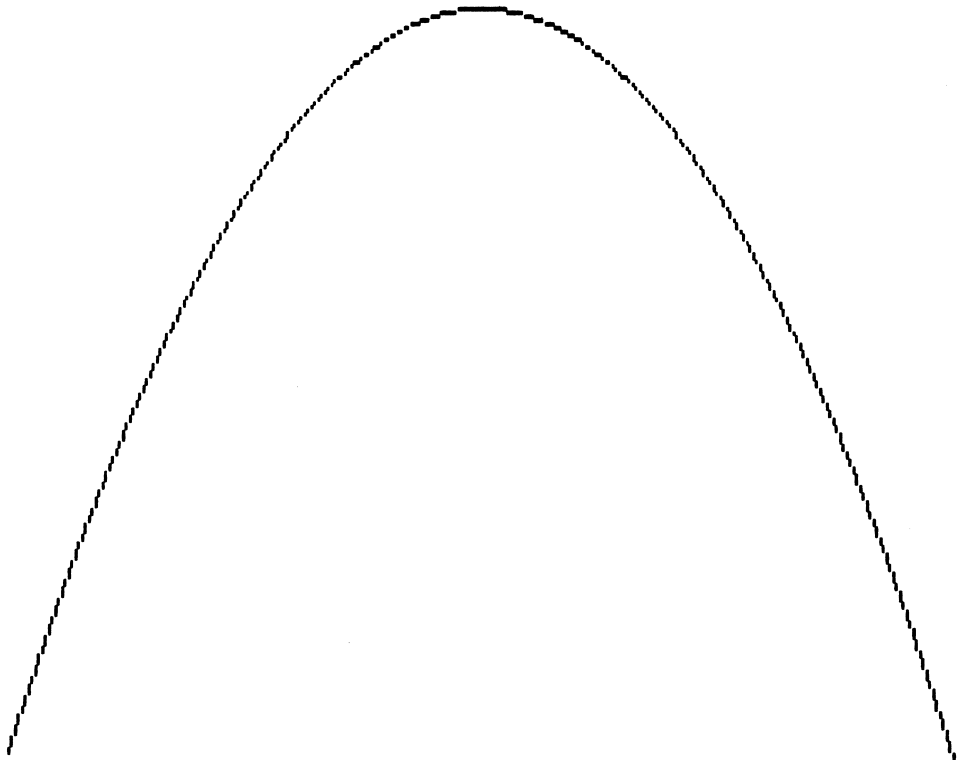


Figure 5–15 Parabolic curve plotted by Prog. 5–5.

5. This curve is symmetric about the X value given in equation (5–7), so that Prog. 5–5 calculates points for only one-half the X range.

Program 5–5 outputs any given parabola, specified by coefficients C1, C2, and C3. Depending on the value of the coefficient C1, a parabola will either increase to a maximum Y value or decrease to a minimum Y value at the midpoint X value (5–7). The Y value at the midpoint will be a maximum if $C1 < 0$, or a minimum if $C1 > 0$. This program plots the parabola so that the midpoint of the curve is plotted at the middle of the top of the screen (for $C1 < 0$), or at the middle of the bottom of the screen (for $C1 > 0$).

Higher-order polynomial curves can be plotted by expanding Prog. 5–5 to accept the degree and coefficients required for these equations:

$$Y = C(1) * X \wedge N + C(2) * X \wedge (N - 1) + \cdots + C(N - 1) * X + C(N) \quad (5–8)$$

Polynomial equations of various degree N can be useful for data fitting. For a table of data points, we may be able to draw a smooth curve through the data with, say, a third- or fourth-degree polynomial. Plotting a polynomial may require some experimenting with the data range. We might first list the points on the curve for the X range of interest, then determine an appropriate scaling to produce the curve in a certain area of the screen.

Program 5–5 Plotting a parabola.

```
10 'PROGRAM 5-5. PARABOLIC CURVES FROM POLYNOMIAL EQUATIONS.
20     'PLOTS THE CENTER SECTION OF A PARABOLA (i.e., where the
30     'curve turns around). THE VERTEX IS CALCULATED, ADJUSTED
40     'TO LIE AT A Y VALUE OF 0 OR AT 199 (DEPENDING ON THE
50     'PARTICULAR CURVE), AND PLOTTED MIDWAY ALONG THE X AXIS.
60     'Y VALUES ALONG THE LEFT HALF OF THE CURVE ARE CALCULATED (USING
70     'DECREASING VALUES OF X) AND PLOTTED, ALONG WITH THE SYMMETRIC
80     'POINT ON THE RIGHT HALF OF THE CURVE. LINES ARE DRAWN BETWEEN
90     'ADJACENT POINTS TO GIVE A CONTINUOUS CURVE.
100 SCREEN 0: WIDTH 80: CLS
110 PRINT "THIS PROGRAM PLOTS A PARABOLIC CURVE FROM THE EQUATION"
120 PRINT: PRINT "    Y = C1 * X ** 2 + C2 * X + C3": PRINT
130 PRINT "IF C1 IS LESS THAN 0, THE CURVE WILL GO UP TO SOME MAXIMUM"
140 PRINT "POINT AND THEN COME DOWN. IF C IS GREATER THAN 0, THE CURVE"
150 PRINT "WILL GO DOWN TO SOME MINIMUM POINT AND THEN COME BACK UP."
160 PRINT: PRINT "ENTER VALUES FOR C1, C2, C3. (C1 may not be zero)"
170 INPUT C(1), C(2), C(3)
180 IF C(1) = 0 THEN PRINT "0 entered for C1": GOTO 160
190     'FIND X VALUE OF VERTEX
200 X = -C(2) / (2 * C(1))
210     'FIND Y VALUE OF VERTEX
220 YVERTEX = C(1) * X ^ 2 + C(2) * X + C(3)
230     'IS VERTEX THE MINIMUM OR MAXIMUM VALUE FOR THIS CURVE?
240     'IF MINIMUM, Y1 = 199. IF MAXIMUM, Y1 = 0.
250 IF C(1) < 0 THEN Y1 = 0
260 IF C(1) > 0 THEN Y1 = 199
270 XCENTER = 160                'XCENTER IS MIDWAY ACROSS SCREEN
280 XLEFT1 = XCENTER: XRIGHT1 = XCENTER
290 XLEFT2 = XCENTER: XRIGHT2 = XCENTER
300 SCREEN 1: CLS
310     'CALCULATE POINTS TO THE LEFT OF THE VERTEX
320 X = X - 1
330     'WITH THIS NEW X VALUE, SOLVE THE POLYNOMIAL EQUATION FOR Y.
340     'EVALUATE THE POLYNOMIAL USING THE FOLLOWING LOOP. CAN BE USED
350     'TO SOLVE POLYNOMIALS OF ANY DEGREE.
360 DEGREE = 3
370 Y = C(1)
380 FOR K = 2 TO DEGREE
390     Y = Y * X + C(K)
400 NEXT
410     'ADJUST THE VALUE OF Y BEFORE PLOTTING ON SCREEN
420 IF C(1) < 0 THEN Y2 = YVERTEX - Y
430 IF C(1) > 0 THEN Y2 = 199 - (Y - YVERTEX)
440 XLEFT2 = XLEFT2 - 1
450 XRIGHT2 = XRIGHT2 + 1
460     'ARE ALL POINTS STILL ON THE SCREEN?
470 IF Y2 < 0 OR Y2 > 199 OR XLEFT2 < 0 OR XLEFT2 > 319 OR XRIGHT2 < 0 OR
    XRIGHT2 > 319 THEN 530
480 LINE (XLEFT1,Y1) - (XLEFT2,Y2)
490 LINE (XRIGHT1,Y1) - (XRIGHT2,Y2)
500 Y1 = Y2
510 XLEFT1 = XLEFT2: XRIGHT1 = XRIGHT2
520 GOTO 320
530 IF INKEY$ = "" THEN 530
540 END
```

NORMAL CURVES

The normal, or Gaussian, curve (sometimes called the bell curve) has the equation

$$Y = EXP(-0.5 * (X - M) \wedge 2/S \wedge 2)/(S * SQR(2 * PI)) \qquad (5\text{–}9)$$

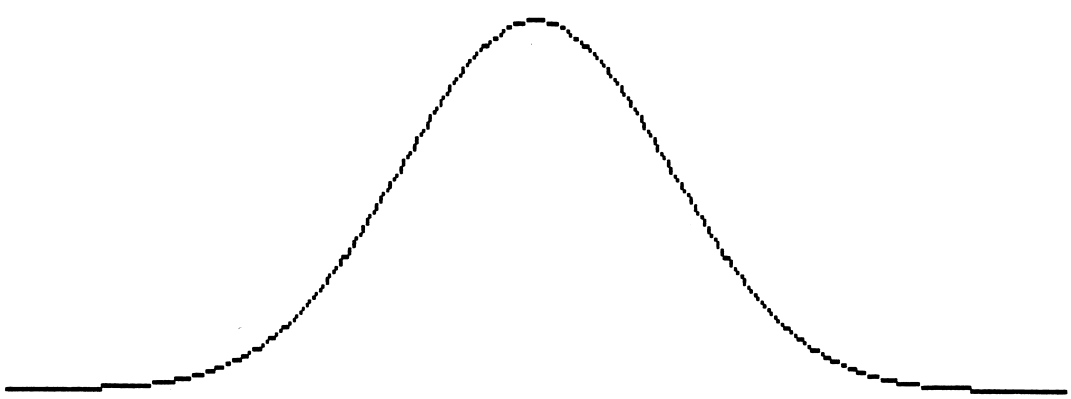
where Y is the probability of having a particular X value in a data set. M is the average (or mean) for all the X values, and S is the standard deviation.

Equation (5–9) is an example of a probability distribution. There are many other probability distribution curves, but the normal curve is of primary importance because many commonly occurring phenomena approximately follow this probability distribution. The probability that an employee in a large organization will have a specified salary X can be estimated as Y in this equation. The value X = M is the mean salary of all the employees, and the value of S provides a measure of the spread of salaries above and below this mean. Approximately 68 percent of the employees will have salaries within 1 standard deviation of the mean (M − S to M + S), and about 99 percent will have salaries within the range of 3 standard deviations of the mean (M − 3 * S to M + 3 * S). Other applications of the normal curve include displaying the probability distribution of the lifetimes of electrical or mechanical parts, variations in sizes of manufactured items, height of U.S. women in a given age range, rate of return on stocks, or daily temperature variations in some city.

Figure 5–16 illustrates the shape of the normal probability curve. This figure was obtained from Prog. 5–6, which accepts a data set as input, calculates M and S, and plots the normal curve. Values for M and S are calculated by standard equations. Normal curves are symmetrical about the mean (X = M), so that Prog. 5–6 need only compute points over one-half of the curve. The curve is centered on the screen with Y values scaled between 50 and 150 over 4 standard deviations.

We have surveyed, in this section, the curves most widely used. There are many other curves that can be useful in a particular graphics application. Bezier and B-spline curves are useful in displaying three-dimensional surfaces, as for automobile and aircraft body design. Legendre and Bessel functions can be used in modeling physical systems, such as atomic and molecular structure, temperature distributions, or gravitational fields. The Poisson and hypergeometric proba-

Figure 5–16 Normal probability curve plotted with M = 51.8 and S = 29.2 by Prog. 5–6.



Program 5–6 Plotting a normal curve.

```
10 'PROGRAM 5-6. PLOTS NORMAL CURVE DERIVED FROM SAMPLE DATA (UP TO
20     '500 VALUES). CURVE IS CENTERED ACROSS X AXIS, AND SCALED TO
30     'LIE BETWEEN Y VALUES OF 50 AND 150. POINTS ALONG LEFT SIDE
40     'OF CURVE ARE CALCULATED AND PLOTTED. SYMMETRIC POINTS ARE
50     'PLOTTED ON RIGHT SIDE OF CURVE.
60      '*****************************************************************
70  DIM D(500)
80  SCREEN 0: WIDTH 80: CLS
90  PI = 3.14159
100 XCENTER = 160                  'XCENTER IS HALFWAY ACROSS SCREEN
110 INPUT "ENTER THE NUMBER OF SAMPLE VALUES"; VALUESCOUNT
120     'FIND THE AVERAGE
130 PRINT "ENTER THE VALUES ONE AT A TIME"
140 TOTAL = 0
150 FOR K = 1 TO VALUESCOUNT
160     INPUT D(K)
170     TOTAL = TOTAL + D(K)
180 NEXT
190 MEAN = TOTAL / VALUESCOUNT
200     'CALCULATE THE STANDARD DEVIATION
210 TOTALDIFF = 0
220 FOR K = 1 TO VALUESCOUNT
230     TOTALDIFF = TOTALDIFF + (D(K) - MEAN) ^ 2
240 NEXT
250 VARIANCE = TOTALDIFF / VALUESCOUNT
260 STANDEV = VARIANCE ^ .5
270     'FIND CENTER POINT (MAXIMUM) OF CURVE
280     'Y IS AT ITS MAXIMUM WHEN X = M
290 X = MEAN
300 YVERTEX = EXP(-.5 * (X - MEAN) ^ 2 / VARIANCE) / (STANDEV * SQR(2 * PI))
310     'THE RANGE OF CURVE POINTS (0 - YVERTEX) IS SCALED TO
320     '50 - 150 ALONG THE Y AXIS
330 YSCALING = (150 - 50) / (YVERTEX - 0)
340     'AND SCALED TO 0 - 319 ALONG THE X AXIS
350 XSCALING = (319 - 0) / (8 * STANDEV)
360 XLEFT1 = XCENTER: XRIGHT1 = XCENTER
370 Y1 = 50
380 CONSTANT = STANDEV * SQR(2 * PI)        'FIND CONSTANT PART OF EQUATION
390     'CALCULATE POINTS ALONG LEFT SIDE OF CURVE. PLOT BOTH SIDES.
400 SCREEN 1: CLS
410 FOR X = MEAN-1 TO MEAN-4*STANDEV STEP -1
420     Y = EXP(-.5 * (X - MEAN) ^ 2 / VARIANCE) / CONSTANT
430     'ADJUST Y TO LIE BETWEEN 50 - 150
440     Y2 = (YVERTEX - Y) * YSCALING + 50
450     'ADJUST X TO LIE BETWEEN 0 - 319
460     XLEFT2 = XCENTER - ((MEAN - X) * XSCALING)
470     XRIGHT2 = XCENTER + ((MEAN - X) * XSCALING)
480     LINE (XLEFT1,Y1) - (XLEFT2,Y2)
490     LINE (XRIGHT1,Y1) - (XRIGHT2,Y2)
500     XLEFT1 = XLEFT2: XRIGHT1 = XRIGHT2              'SAVE THESE POINTS
510     Y1 = Y2
520 NEXT
530 IF INKEY$ = "" THEN 530
540 END
```

bility distributions are useful for modeling statistical applications. These models include the simulation of customer lines with different numbers of tellers at a bank, or graphing various alternative selections of project teams from a group of employees.

5–3 PICTURES WITH CURVES

Programs in this section provide picture-drawing examples using curves. With Prog. 5–7, we produce the picture of Fig. 5–17, drawn with short straight line segments from a graph paper layout. Figure 5–18 shows a fire truck containing circles, a spiral, and a normal curve, drawn by Prog. 5–8. Program 5–9 outputs graphics "art" using curves. This program demonstrates some of the many possibilities using trigonometric functions. The resulting patterns are shown in Fig. 5–19.

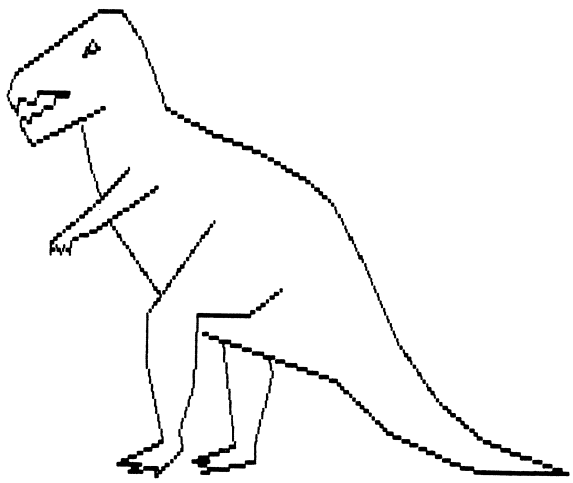


Figure 5–17 Picture drawn from a graph paper layout by Prog. 5–7, using straight line segments to approximate curves.

Program 5–7 Dinosaur drawn with curves approximated by short line segments.

```
10 'PROGRAM 5-7. DINOSAUR USING SHORT LINE SEGMENTS TO GIVE CURVES.
20     'POINTS WERE DERIVED FROM GRAPH PAPER DRAWING.
30  SCREEN 2: CLS
40       'DRAW ONE PART OF THE PICTURE AT A TIME
50       'READ HOW MANY POINTS IN THIS PICTURE PART
60  YADJUST = .9199999
70  READ POINTCOUNT
80  IF POINTCOUNT = 0 THEN 400
90  READ X,Y
100 Y = Y * YADJUST                    'CORRECT FOR RESOLUTION DIFFERENCE
110 PSET (X,Y)
120 FOR EACHPOINT = 2 TO POINTCOUNT
130      READ X,Y
140      Y = Y * YADJUST
150      LINE - (X,Y)
160 NEXT
170 GOTO 70
180      'data for main part of body
190 DATA 38,223,60,180,71,173,65,174,63,179,64,180,62,185,62,187,60
200 DATA 192,60,194,57,200,57,202,55,183,55,182,59,172,58
210 DATA 171,62,165,57,170,50,220,31,233,31,245,40,260,60,290,67,315,72
220 DATA 345,80,360,87,380,105,400,127,425,145,452,155,502,164,450,164
230 DATA 434,162,415,158,395,153,362,138,323,132,283,125
240      'data for large leg
250 DATA 16,289,92,250,120,250,135,260,157,233,163,247,163,240,165
```

Program 5–7 (cont.)

```
260 DATA 257,164,256,166,272,159,270,153,278,140,280,120,285,120
270 DATA 310,120,330,112
280     'data for small leg
290 DATA 11,295,127,303,158,278,162,288,162,282,163
300 DATA 291,164,283,165,315,162,315,157,325,137,323,132
310     'data for arm
320 DATA 11,238,77,220,87,192,99,192,102,195,100
330 DATA 198,102,200,100,203,102,205,98,218,96,255,83
340     'data to fill in body
350 DATA 3,258,114,245,105,228,93
360 DATA 3,222,86,215,77,210,65
370     'data for eye
380 DATA 4,220,42,210,45,215,40,220,42
390 DATA 0
400 IF INKEY$="" THEN 400
410 END
```

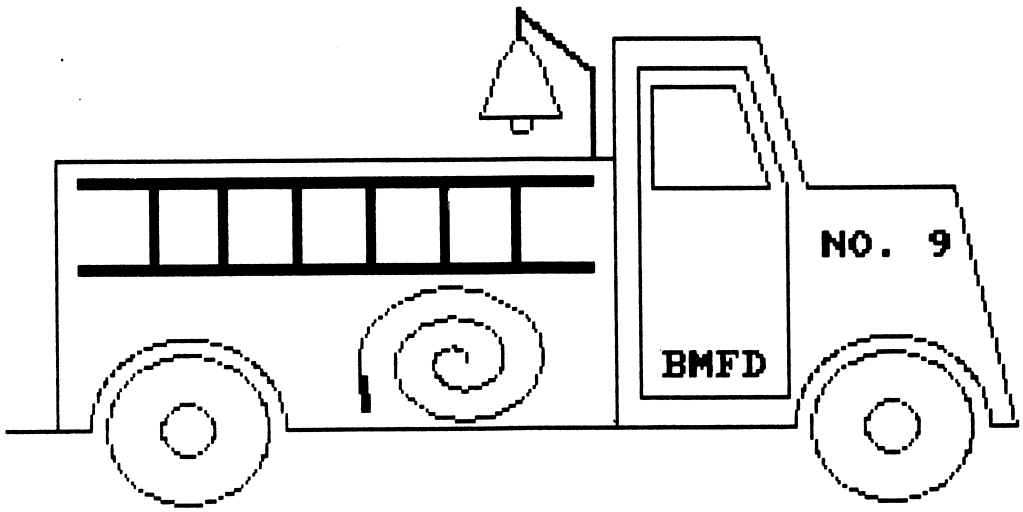


Figure 5–18 Picture drawn by Prog. 5–8, using curve equations.

Program 5–8 Fire truck drawn with curve equations.

```
10 'PROGRAM 5-8. FIRETRUCK WITH STRAIGHT LINES, CIRCLES, ARCS, BELL CURVE
20     'AND SPIRAL.
30  SCREEN 1: CLS
40  READ X, Y: PSET (X,Y)                    'OUTLINE
50  FOR EACHPOINT = 1 TO 8
60      READ X,Y: LINE - (X,Y)
70  NEXT
80  FOR EACHPART = 1 TO 2                    'DOOR AND WINDOW
90      READ POINTCOUNT
100     READ X, Y: PSET (X,Y)
110     FOR EACHPOINT = 1 TO POINTCOUNT
120         READ X, Y: LINE - (X,Y)
130     NEXT
140 NEXT
150     'FINISH BODY, LADDER, BELL STAND
```

Program 5–8 (cont.)

```
FOR EACHLINE = 1 TO 13
    'READ COORDINATES, THICKNESS, DIRECTION OF THICKNESS
    READ X1, Y1, X2, Y2, THICKNESS, DIRECTION$
    IF DIRECTION$ = "X" THEN 240
    FOR K = 0 TO THICKNESS - 1                'THICK IN Y DIRECTION
        LINE (X1,Y1+K) - (X2,Y2+K)
    NEXT
    GOTO 270
    FOR K = 0 TO THICKNESS - 1                'THICK IN X DIRECTION
        LINE (X1+K,Y1) - (X2+K,Y2)
    NEXT
NEXT
YADJUST = .9199999                            'MAKE WHEELWELLS
FOR EACHWHEEL = 1 TO 2
    READ XCENTER, YCENTER, RADIUS
    CIRCLE (XCENTER,YCENTER),RADIUS,,0,3.14159,YADJUST
    LINE (XCENTER-RADIUS+1,YCENTER) - (XCENTER+RADIUS-1,YCENTER),0
NEXT
FOR EACHCIRCLE = 1 TO 4                       'MAKE TIRES AND HUBCAPS
    READ XCENTER,YCENTER,RADIUS
    CIRCLE (XCENTER,YCENTER),RADIUS,,,,YADJUST
NEXT
RADIUS = 1.3                                  'MAKE HOSE
ANGLE = .01                                'ANGLE TO USE IN MAKING SPIRAL
READ XCENTER, YCENTER
X = XCENTER + RADIUS * COS(-ANGLE) * 1.4
Y = YCENTER + RADIUS * SIN(-ANGLE)
PSET (X,Y)
ANGLE = ANGLE + .1
RADIUS = RADIUS + .13
X = XCENTER + RADIUS * COS(-ANGLE) * 1.4
Y = YCENTER + RADIUS * SIN(-ANGLE)
LINE - (X,Y)
IF RADIUS < 22 THEN 440
LOCATE 15,33: PRINT "NO. 9"                   'LABELS
LOCATE 19,27: PRINT "BMFD"
READ MEAN, SDEV                               'MAKE BELL
YVERTEX = .398942 / SDEV
YSCALING = (90-60) / YVERTEX
XSCALING = (186-159) / (3.4*SDEV)
XLEFT1 = 166: XRIGHT1 = 166
Y1 = 60
FOR X = MEAN-1 TO MEAN-1.7*SDEV STEP -1
    Y = (.398942 / SDEV) * EXP(-.5 * (X - MEAN) ^ 2 / (SDEV * SDEV))
    Y2 = (YVERTEX - Y) * YSCALING + 60
    XLEFT2 = 166 - (MEAN - X) * XSCALING
    XRIGHT2 = 166 + (MEAN - X) * XSCALING
    LINE (XLEFT1,Y1) - (XLEFT2,Y2)
    LINE (XRIGHT1,Y1) - (XRIGHT2,Y2)
    XLEFT1 = XLEFT2: XRIGHT1 = XRIGHT2
    Y1 = Y2
NEXT
LINE (XLEFT1,Y1) - (XRIGHT1,Y1)
CIRCLE (166,Y1+2),3
DATA 12,164,313,164,295,100,251,100,237,60,194,60,194,92,27,92,27,164
DATA 5,201,68,201,156,245,156,245,100,233,68,201,68
DATA 4,205,73,205,100,240,100,230,73,205,73
DATA 194,60,194,164,1,X,34,97,187,97,3,Y,34,120,187,120,3,Y
DATA 55,97,55,120,3,X,76,97,76,120,3,X,98,97,98,120,3,X
DATA 120,97,120,120,3,X,142,97,142,120,3,X,163,97,163,120,3,X
DATA 187,92,187,68,2,X,187,68,165,52,2,Y,165,52,165,60,2,X
```

Program 5–8 (cont.)

```
770 DATA 117,150,118,159,3,X
780 DATA 66,164,29,276,164,29,66,164,24,276,164,24,66,164,8,276,164,8
790 DATA 147,146
800 DATA 1,7
810 IF INKEY$ = "" THEN 810
820 END
```

Program 5–9 Art patterns with curves.

```
10 'PROGRAM 5-9. CURVE PATTERNS
20  SCREEN 1: COLOR 0,0: CLS
30      '************************ CLOVER ***********************************
40  COLCODE = 2
50  XCENTER = 160: YCENTER = 100
60  FOR RADIUS = 20 TO 50 STEP 15
70      PSET (XCENTER,YCENTER),COLCODE
80      FOR ANGLE = 0 TO 6.28318 STEP 1/RADIUS
90          R1 = RADIUS * SIN(2 * ANGLE)
100         X = XCENTER + R1 * COS(ANGLE)
110         Y = YCENTER + R1 * SIN(ANGLE)
120         LINE - (X,Y),COLCODE
130     NEXT
140 NEXT
150     '****************** SIDE FIGURES (CARDIOIDS) ********************
160 COLCODE = 1
170 XLEFT = 60              'CENTER POINT OF LEFT FIGURE IS XLEFT,YCENTER
180 YCENTER = 100
190 XRIGHT = 260            'CENTER POINT OF RIGHT FIGURE IS XRIGHT,YCENTER
200 FOR RADIUS = 15 TO 25 STEP 10
210     XLEFT1 = XLEFT
220     XRIGHT1 = XRIGHT
230     YLEFT1 = YCENTER
240     YLEFT3 = YCENTER
250     YRIGHT1 = YCENTER
260     YRIGHT3 = YCENTER
270     FOR ANGLE = 0 TO 3.14159 STEP 1/RADIUS
280         R1 = RADIUS * SIN(ANGLE / 2)
290         DX = R1 * COS(ANGLE)
300         DY = R1 * SIN(ANGLE)
310         XLEFT2 = XLEFT + DX
320         YLEFT2 = YCENTER + DY
330         YLEFT4 = YCENTER - DY
340         LINE (XLEFT1,YLEFT1) - (XLEFT2,YLEFT2),COLCODE
350         LINE (XLEFT1,YLEFT3) - (XLEFT2,YLEFT4),COLCODE
360         XRIGHT2 = XRIGHT - DX
370         YRIGHT2 = YCENTER + DY
380         YRIGHT4 = YCENTER - DY
390         LINE (XRIGHT1,YRIGHT1) - (XRIGHT2,YRIGHT2),COLCODE
400         LINE (XRIGHT1,YRIGHT3) - (XRIGHT2,YRIGHT4),COLCODE
410         XLEFT1 = XLEFT2
420         YLEFT1 = YLEFT2
430         YLEFT3 = YLEFT4
440         XRIGHT1 = XRIGHT2
450         YRIGHT1 = YRIGHT2
460         YRIGHT3 = YRIGHT4
470     NEXT
480 NEXT
490     '************************ FLOWER PATTERNS ***********************
500 READ XCENTER,YCENTER,RADIUS,PETALS,COLCODE
510 IF XCENTER = 0 THEN 720
520 GOSUB 550
```

Program 5–9 (cont.)

```
530 GOTO 500
540 GOTO 720
550 '########################## MAKE FLOWER PATTERNS ####################
560 PSET (XCENTER + RADIUS,YCENTER),COLCODE
570 FOR ANGLE = 0 TO 6.28318 STEP 1/RADIUS
580     R1 = RADIUS * COS(PETALS * ANGLE)
590     X = XCENTER + R1 * COS(ANGLE)
600     Y = YCENTER + R1 * SIN(ANGLE)
610     LINE - (X,Y),COLCODE
620 NEXT
630 RETURN
640 '############################################################################
650 DATA 200,30,15,7,1
660 DATA 100,170,15,6,3
670 DATA 60,25,15,8,2
680 DATA 40,150,14,5,2
690 DATA 230,169,20,8,3
700 DATA 230,169,30,4,2
710 DATA 0,0,0,0,0,0,0,0
720 IF INKEY$ = "" THEN 720
730 END
```

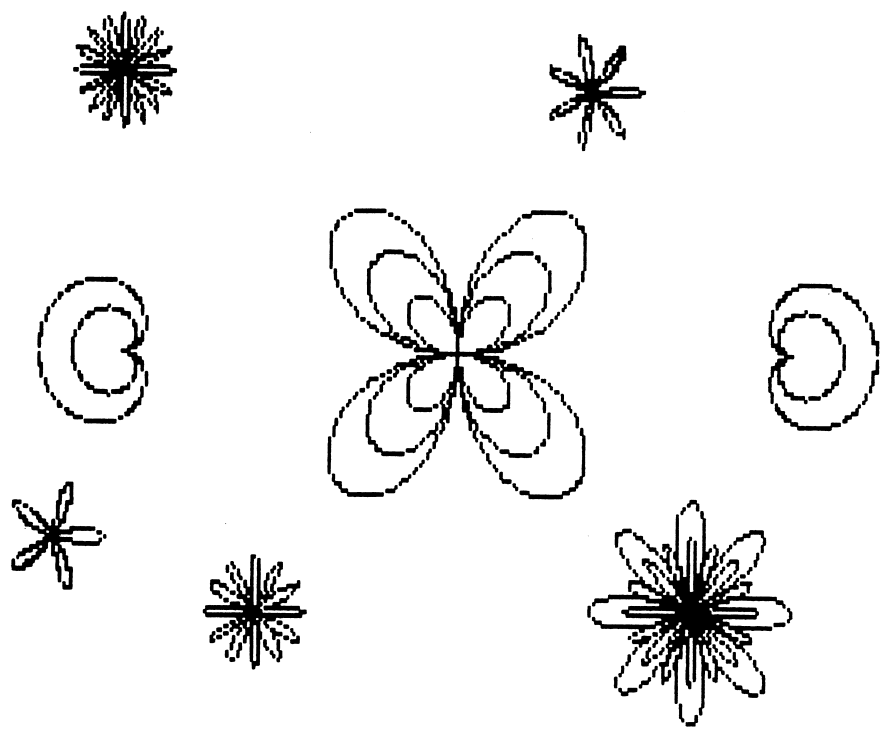


Figure 5–19 Graphics art patterns produced by Prog. 5–9.

5–4 GRAPHS AND PIE CHARTS

Curves are useful for many types of graphs and charts. The programs in this section illustrate graphing techniques using curves.

GRAPHS

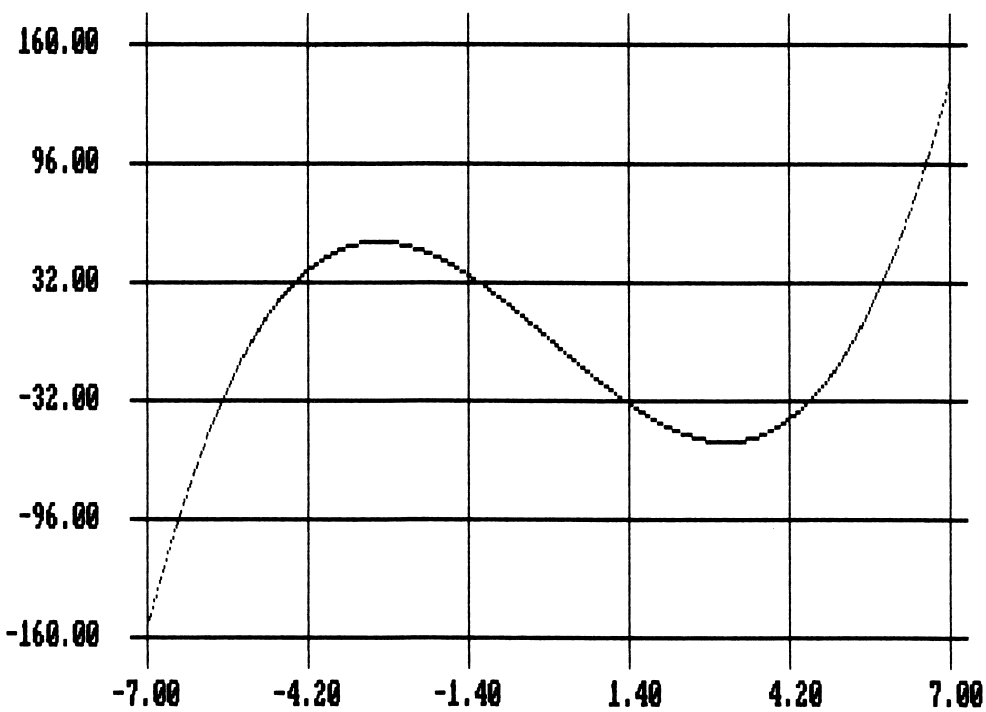
We can display graphs using the points given in data tables or using points calculated from equations. To graph a table of data values, we could plot the points and connect them with straight line segments. We could also use a curve-fitting method to approximate the shape of the data. Curves can be drawn to fit a data set with an analytic technique (such as the least-squares method), or we could use an interactive method that sketches a curve from keyboard instructions or light-pen input. To graph an equation, we calculate coordinates from the equation and either plot closely spaced points or straight line segments connecting the points.

An example of plotting any specified curve equation is given by Prog. 5–10. Graph dimensions for the selected curve are input to the program. Figure 5–20 plots the output of Prog. 5–10 for a third-degree polynomial.

PIE CHARTS

A circle-drawing algorithm is the main ingredient in a program to make pie charts. Program 5–11 illustrates this application. Input to the program includes the name and relative size (data value) for each division of the chart. Positioning of a

Figure 5–20 Graph of the function X ^ 3 − 27 * X, output by Prog. 5–10.



Program 5–10 General graph plotting using any input equation.

```
10 'PROGRAM 5-10. PLOTS ANY EQUATION.
20     'ALLOWS USER TO TYPE IN EQUATION AT LINE 360 AND TO ENTER
30     'MINIMUM AND MAXIMUM X AND Y VALUES FOR WHICH THE EQUATION
40     'IS TO BE PLOTTED. PROGRAM DRAWS A GRID USING PIXELS 74-574
50     'ON THE X AXIS AND 12-188 ON THE Y AXIS.
60  SCREEN 0: WIDTH 80: CLS
70  INPUT "DO YOU WANT INSTRUCTIONS? (TYPE Y OR N)"; I$
80  IF I$ = "n" OR I$ = "N" THEN 170
90  PRINT "THIS PROGRAM DISPLAYS THE GRAPH OF ANY EQUATION DRAWN ON A"
100 PRINT "GRID ON THE SCREEN. THE EQUATION MUST BE ENTERED AT LINE 360."
110 PRINT "YOUR EQUATION MUST USE VARIABLE Y AS THE DEPENDENT VARIABLE"
120 PRINT "AND X AS THE INDEPENDENT VARIABLE (E.G., Y = 6 * X + 20)."
130 PRINT "ONCE YOUR EQUATION HAS BEEN ENTERED AND YOU TYPE RUN, YOU"
140 PRINT "WILL BE ASKED TO SPECIFY THE MINIMUM AND MAXIMUM X COORDINATES"
150 PRINT "FOR WHICH YOU WANT THE EQUATION PLOTTED, AS WELL AS THE MINIMUM"
160 PRINT "AND MAXIMUM Y VALUES THAT SHOULD APPEAR ON THE GRID."
170 INPUT "DOES LINE 360 CONTAIN YOUR EQUATION? (TYPE Y OR N)"; E$
180 IF E$ = "Y" OR E$ = "y" THEN 230
190 IF I$ = "N" OR I$ = "n" THEN 680    'DON'T PRINT ANY INSTRUCTIONS
200 PRINT "TYPE LINE NUMBER 360 FOLLOWED BY YOUR EQUATION."
210 PRINT "HIT RETURN AND TYPE RUN."
220 GOTO 680
230     'GRAPHS EQUATION ON A GRID
240 PRINT "ENTER MINIMUM AND MAXIMUM X VALUES FOR WHICH"
250 PRINT "THE EQUATION SHOULD BE PLOTTED."
260 INPUT XLEFT, XRIGHT
270 XDIFF = XRIGHT - XLEFT
280 PRINT "ENTER THE MINIMUM AND MAXIMUM Y VALUES THAT SHOULD BE ON THE GRID"
290 INPUT YBOTTOM, YTOP
300 YDIFF = YTOP - YBOTTOM
310 GOSUB 420           'DRAW GRID
320     'DRAW EQUATION
330 YA = (180 - 20) / YDIFF     'YA SCALES CALCULATED Y TO GRID POSITION
340 FOR XGRID = 84 TO 564
350     X = XLEFT + (XGRID - 84) / 484 * XDIFF
360     Y = X ^ 3 - 27 * X
370     YGRID = 180 - (Y - YBOTTOM) * YA
380     IF XGRID < 84 OR XGRID > 564 OR YGRID < 20 OR YGRID > 180 THEN 670
390     PSET (XGRID,YGRID)
400 NEXT
410 GOTO 670
420     'DRAWS GRID WITH LABELS
430 SCREEN 2: CLS
440 FOR Y = 20 TO 180 STEP 32
450     LINE (74,Y) - (574,Y)
460 NEXT
470 FOR X = 84 TO 564 STEP 96
480     LINE (X,12) - (X,188)
490 NEXT
500     'FROM BOTTOM OF SCREEN TO TOP, LABEL THE GRID WITH
510     'SUCCESSIVE FIFTHS OF THE VERTICAL RANGE OF THE GRID
520 ROW = 23
530 FOR K = 0 TO 5
540     LOCATE ROW,1: LABEL = YBOTTOM + YDIFF * K / 5
550     PRINT USING "####.##";LABEL
560     ROW = ROW - 4
570 NEXT
580     'FROM LEFT TO RIGHT, LABEL THE GRID WITH SUCCESSIVE
590     'FIFTHS OF THE HORIZONTAL RANGE OF THE GRID
600 COLUMN = 7
```

Program 5–10 (cont.)

```
610 FOR K = 0 TO 5
620     LOCATE 25,COLUMN: LABEL = XLEFT + XDIFF * K / 5
630     PRINT USING "####.####";LABEL;
640     COLUMN = COLUMN + 12
650 NEXT
660 RETURN
670 IF INKEY$ = "" THEN 670
680 END
```

division name on the chart is accomplished by locating the angle corresponding to the centerline for the appropriate sector. If the sector is on the right side of the pie chart, the name starts on this bisecting line 4 units beyond the circumference. If the sector is on the left, the name ends on the bisecting line. We locate label starting positions by converting pixel coordinates to character print positions. This is done by dividing the X coordinate by the number of horizontal pixels in a character and by dividing the Y coordinate by the number of vertical pixels in a character. Figure 5–21 shows a pie chart produced by Prog. 5–11. If we have cassette BASIC, we can substitute one of the algorithms from Section 5–1 for the CIRCLE statements in this program.

We could revise Prog. 5–11 to automatically draw the pie sections by using negative angles in the CIRCLE statement. This would eliminate the extra LINE statements that draw the lines from the circle center to the circumference. For example, lines 190, 260, 270, and 280 can be replaced by

```
260 CIRCLE (XC,YC),R,,-BEFORE,-ANGLE,0.92
```

This would take care of completely drawing the pie slice, but we still need to find a point near each slice to use in positioning labels. Since statements 330 and 340 are based on angles measured clockwise from the horizontal, while the CIRCLE command uses counterclockwise angles, we need to change statement 320 to

```
320 BISECT = 6.28318 - (BEFORE + ANGLE)/2
```

This identifies the correct bisect angle and completes the alterations necessary to construct the pie chart with labels, using only the CIRCLE command.

To be effective, pie charts should be drawn with no more than five or six slices. Shading and color selections for pie charts are similar to those for bar graphs. Shading patterns should be simple and graduated from dark to light around the chart. Labels should be on or close to the areas they are meant to identify.

The kind of graph or chart we choose for displaying data information can have a great influence on the effectiveness of the presentation. Pie charts are best used to convey information about percentages. We could also use a bar chart for percentages, with each bar divided into percentage sections. Line graphs and bar charts are good choices for conveying information about data quantities, such as sales amounts. We can graph sales amounts over time or relative to some other

Program 5–11 Pie chart constructed with the CIRCLE command.

```
10 'PROGRAM 5-11. PIECHART.
20 DIM SECTION$(8), VALUE(8)
30 SCREEN 0: WIDTH 80: CLS
40 INPUT "CENTER COORDINATES FOR PIECHART"; XCENTER,YCENTER
50 INPUT "RADIUS"; RADIUS
60 IF XCENTER+RADIUS > 319 OR XCENTER-RADIUS < 0 OR YCENTER+RADIUS > 199
      OR YCENTER-RADIUS < 0 THEN 580
70 INPUT "TITLE OF CHART"; TITLE$
80 INPUT "NUMBER OF DIVISIONS (UP TO 8)"; N
90 PRINT "ENTER NAME AND VALUE FOR EACH DIVISION"
100 TOTAL = 0
110     'INPUT DATA. FIND TOTAL OF ALL VALUES
120 FOR K = 1 TO N
130     INPUT SECTION$(K), VALUE(K)
140     TOTAL = TOTAL + VALUE(K)
150 NEXT
160 SCREEN 1: CLS
170 COLUMN = 20 - INT(LEN(TITLE$) / 2 + .5)      'CENTER TITLE
180 LOCATE 1,COLUMN: PRINT TITLE$
190 CIRCLE (XCENTER,YCENTER),RADIUS,,,,.9199999
200 BEFORE = 0          'BEFORE IS ANGLE THAT DETERMINED PRECEDING LINE
210 FOR K = 1 TO N
220         'LINE TO PLOT IS BASED ON THE PERCENTAGE OF THE
230         'CIRCLE EQUAL TO VALUE(K) / TOTAL PLUS THE
240         'PRECEDING DIVISIONS
250     ANGLE = BEFORE + 6.28318 * VALUE(K) / TOTAL
260     XP = XCENTER + RADIUS * COS(ANGLE)
270     YP = YCENTER + RADIUS * SIN(ANGLE) * .9199999
280     LINE (XCENTER,YCENTER) - (XP,YP)
290         'PUT LABEL ON DIVISION
300         'FIND A POINT 4 UNITS OUTWARD FROM THE CENTER
310         'POINT OF THIS DIVISION'S ARC
320     BISECT = BEFORE + (ANGLE - BEFORE) / 2          'BISECT IS THE ANGLE
330     XLABEL = XCENTER + (RADIUS + 4) * COS(BISECT)   'WHOSE LINE WOULD
340     YLABEL = YCENTER + (RADIUS + 4) * SIN(BISECT)   'HALVE THIS DIVISION
350         '(XLABEL,YLABEL) IS THE POINT USED TO ANCHOR LABEL
360         'USE THE POINT AS START OF LABEL IF IT'S ON RIGHT
370         'SIDE OF CIRCLE, AS END OF LABEL IF IT'S ON LEFT,
380         'AS MIDPOINT IF IT'S ON TOP OR BOTTOM OF CIRCLE
390         'POINT IS START OF LABEL
400     IF XLABEL > XCENTER + 10 THEN 500
410         'POINT IS END OF LABEL
420     IF XLABEL < XCENTER - 10 THEN 470
430         'OTHERWISE POINT IS MIDPOINT OF LABEL. ADJUST XLABEL BY
440         'ONE-HALF THE NUMBER OF PIXELS NEEDED FOR LABEL
450     XLABEL = XLABEL - LEN(SECTION$(K)) / 16
460     GOTO 500
470         'POINT IS END OF LABEL. MOVE BACK BY THE NUMBER
480         'OF PIXELS REQUIRED FOR LABEL
490     XLABEL = XLABEL - LEN(SECTION$(K)) * 8
500         'CONVERT THE PIXEL LOCATION (XLABEL,YLABEL) TO THE CLOSEST
510         'CORRESPONDING PRINT POSITION
520     ROW = INT(YLABEL / 8) + 1
530     COLUMN = INT(XLABEL / 8) + 1
540     LOCATE ROW,COLUMN: PRINT SECTION$(K);
550     BEFORE = ANGLE          'UPDATE BEFORE TO REFLECT DOING THIS DIVISION
560 NEXT
570 GOTO 590
580 PRINT "COORDINATE OUT OF RANGE"
590 IF INKEY$="" THEN 590
600 END
```

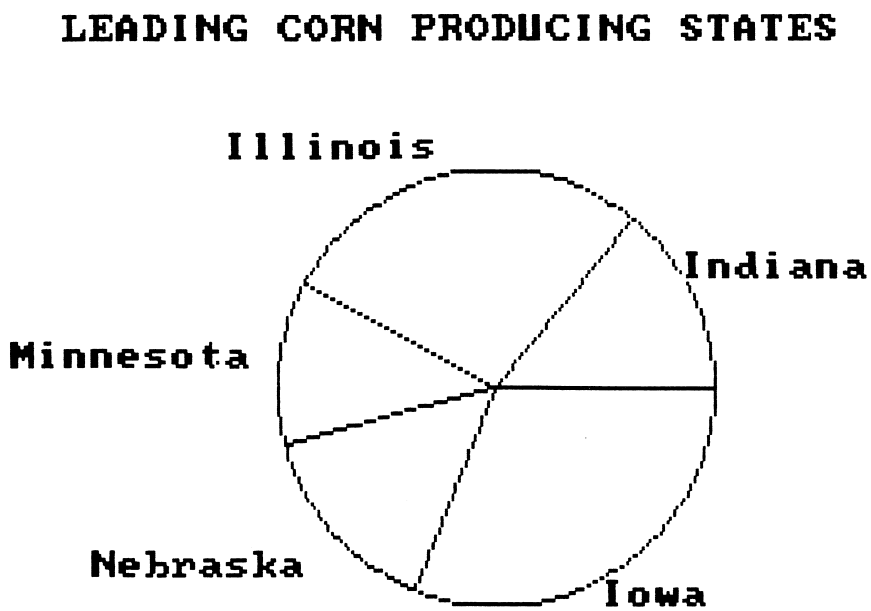


Figure 5–21 Pie chart output of Prog. 5–11.

parameter, such as sales region. In general, either the horizontal or vertical axis could be chosen for sales regions, but time is usually best represented on the horizontal axis. More complex data relationships may be graphed by including more than one curve in a graph, overlapping the bars of a bar graph, or using a three-dimensional graph.

PROGRAMMING PROJECTS

5–1. Modify Prog. 5–2 to input the angular step size for plotting a circle, instead of the number of points. This will produce regular polygons for larger angular step sizes, and we begin to approximate circles as the angle chosen decreases. We get a triangle for an input angular step size of 120 degrees and a square for an input of 90 degrees.

5–2. Using equations (5–1), write a program to display a circular arc. Input will be the radius, center coordinates, and either the arc coordinate endpoints or the beginning and ending angles for the arc. This program can then be used to produce crescent (or moon) shapes, as in Prog. 5–1, without using the CIRCLE command.

5–3. Write a program to produce a solid color circle without the PAINT statement. Paint the interior with any chosen color by drawing diameter lines across the circle in that color. Using equations (5–1), the endpoints of any diameter line are 180 degrees apart. Special effects can be obtained by varying the color as each diameter is drawn.

5–4. The interior of a circle can be filled with dots of variable spacing by plotting points

from the center out to the circumference. Write a program to paint a circle in this way using equations in (5–1) and varying the radius from zero to R, with a step size set by input.

5–5. Write a program to shade the interior of a circle using spaced horizontal lines. Endpoint X values for the lines are to be calculated from equation (5–2), with the Y values varying from YC − R to YC + R. Step size for Y values is to be set by input. (By interchanging the role of X and Y, we could shade with vertical lines or combine the two lines to provide a crosshatched shading.)

5–6. Write a program to display any specified ellipse. Input will be XC, YC, RX, RY, and the aspect ratio.

5–7. Write a program to display an ellipse with interior shading, using any of the methods in Projects 5–3, 5–4, and 5–5.

5–8. Sketch a figure involving circular arcs on graph paper, then write a program to display this figure.

5–9. Modify Prog. 5–4 to produce a sine-curve graph with labeled axes. Reduce the amount of computation by taking advantage of the symmetry of the sine function. The program should use equation (5–4), with any values for H, W, and D as input. Scale and plot three cycles of the curve from X = −D/W to X = (6 * PI − D)/W.

5–10. The program outlined in Project 5–9 can be modified to display a damped sine function by multiplying the sine function by EXP(−K * X) for each value of X. Plot the resulting curve for X = 0 to X = (10 * PI − D)/W for any positive input value of K.

5–11. Write a program to display a set of data points as small colored circles, with each circle centered on the coordinates of a data point. Include labeled coordinate axes in the display, and connect the circles with straight lines.

5–12. Write a program to interactively draw a curve to fit a set of plotted data points. The program should allow the curve to be sketched around the data points through keyboard input, or with other interactive devices.

5–13. Write a program to display a set of data points and the parabolic curve that most closely fits the data set. For a set of N input data points, (X(1), Y(1)), (X(2), Y(2)), . . . , (X(N),Y(N)), the coefficients of the parabola [C1, C2, C3 in equation (5–6)] can be determined using the least-squares method. The following set of equations is to be solved simultaneously for C1, C2, and C3:

$$\Sigma Y(I) = C1 * \Sigma X(I) \wedge 2 + C2 * \Sigma X(I) + C3 * N$$
$$\Sigma X(I) * Y(I) = C1 * \Sigma X(I) \wedge 3 + C2 * \Sigma X(I) \wedge 2 + C3\ \Sigma X(I)$$
$$\Sigma(X(I) \wedge 2) * Y(I) = C1 * \Sigma X(I) \wedge 4 + C2 * \Sigma X(I) \wedge 3 + C3 * \Sigma X(I) \wedge 2$$

where the symbol Σ (sigma) means sum over all values of I from 1 to N.

5–14. Lay out a figure or scene on graph paper and write a program to display the layout, using various curve functions to approximate the outline. Fill in the display with color or shading patterns.

Chapter 6

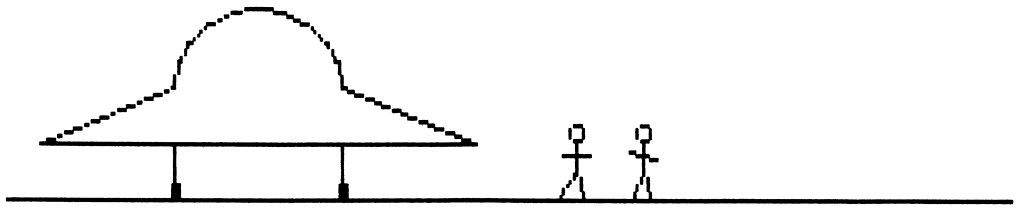
Interactive Techniques

Our programs for constructing pictures and graphs have been set up to produce predefined displays, with the character and pixel positions specified in DATA statements or equations. We can also produce pictures and graphs interactively. With interactive methods, we can exercise some spontaneity by "sketching" patterns on the screen or selecting program options from menus. The interactive devices we can use include the keyboard, light pens, paddles or joysticks, and graphics tablets.

6–1 MENUS

An effective way of providing input to many graphics programs is through an interactive dialogue. Using this technique, the program can carry on a limited conversation with us at certain stages of the processing by asking what we would like to do next. A list of processing options presented by a program is referred to as a menu. Any time we set up a generalized graphics program that allows several processing options, we can use a menu to select the options. The form of the menu will depend on the type of interactive input device to be used.

A menu can fill the screen, or a menu can be placed to one side, the top, or the bottom. If we fill the screen with the menu, we need to erase it after the selection is made. Placing the menu in a smaller part of the screen allows us to include the picture together with the menu, as illustrated in Fig. 6–1. In this case, we can leave the menu on the screen after the selection has been made. This is helpful if the options are to be repeatedly offered for selection, since the display does not have to be redrawn after each menu selection. After all selections have been made, the menu can be erased. Presenting the menu and display together is



DO YOU WANT TO: (1) EXPLORE PLANET,
(2) BOARD SPACESHIP, (3) BLAST OFF?

Figure 6–1 Menu and picture displayed together.

also useful when we want to list multiple menus. A drawback to displaying menus and a picture or graph together is that our effective screen size is reduced by the amount of area the menus occupy.

6–2 KEYBOARD METHODS

For keyboard input, a numerical or alphabetical listing of options, as in Fig. 6–2 or Fig. 6–3, is simple and effective. Typing the number or letter of an option causes the program to branch to the appropriate module that will accomplish that option (such as performing the goodness of fit test when we type in the letter G). Any character string or key on the keyboard could be used to make a menu selection, but numbering or lettering the items is a good choice for most applications.

Screen positions for the location of characters or pixels can also be selected interactively from the keyboard. We can choose character types, character or pixel colors, line lengths, or any other parameters during program execution. A means for accomplishing this type of interaction is with the INKEY$ operation. This variable name stores a character selected from the keyboard. To provide interactive input to a program during various stages of execution, we can set up statements that test INKEY$ to see what key we have pressed at each stage. Depending on the key chosen, we can carry out one type of operation or another.

Simple sketching is illustrated with Prog. 6–1. We first input the coordinates for the starting screen position and the colors to be used. Then a line of pixels is plotted in a direction specified by the four arrows on the numeric keypad portion of the keyboard. These keys return a two-character string, and we must check the rightmost character in the string to determine which arrow key was pressed. Keys B and D are used to indicate whether we want to move over to a new screen position or continue to draw lines. When we hit B, this means that we want to move without drawing. Pixels are then plotted and erased as we move, so that our present position is always indicated. Pressing D puts us in the draw routine. This program can be used to sketch pictures, graphs, and charts, or to sketch a curve to

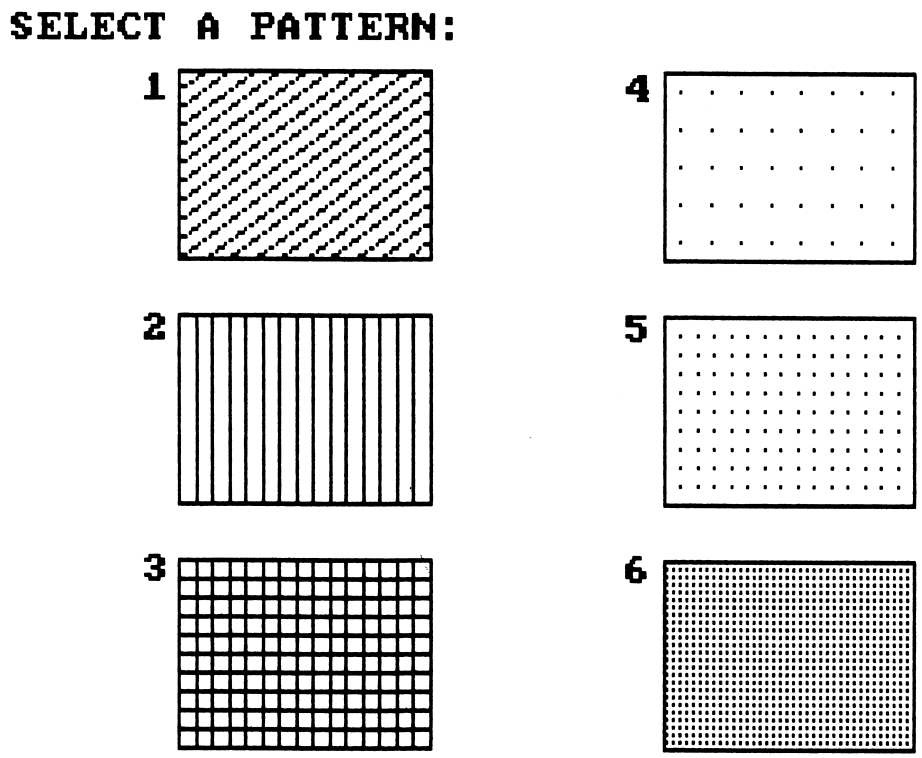


Figure 6–2 A menu to select shading patterns by typing in a number.

fit a set of data values. Any line can be erased by tracing over the line after pressing B. We could also include draw options that allow us to sketch diagonal lines, in addition to horizontal and vertical lines.

Another type of interactive drawing procedure is given in Prog. 6–2. Here we construct pictures or graphs by selecting straight line segments. Each specified

Figure 6–3 A menu of processing options chosen by selecting a letter.

```
WHICH CALCULATION NEXT?

   M:   MEAN AND VARIANCE CALCULATION
   L:   LEAST SQUARES LINEAR CURVE FIT
   Q:   QUADRATIC CURVE FIT
   C:   CUBIC CURVE FIT
   G:   GOODNESS OF FIT TEST
```

Program 6–1 Interactive sketching.

```
10 'PROGRAM 6-1. INTERACTIVELY SKETCHES PICTURES.
20       'USES INKEY$ TO MONITOR THE KEYBOARD AND RECEIVE INSTRUCTIONS.
30       'USER CAN DRAW (USING CURSOR CONTROL KEYS) OR JUST MOVE "PEN"
40       'TO NEW PART OF THE SCREEN. WHEN MOVING TO NEW PART OF THE SCREEN
50       '(WITHOUT DRAWING), DRAWING$ IS SET TO "N", AND POINTS INDICATED
60       'BY THE CURSOR CONTROL KEYS ARE SET AND THEN IMMEDIATELY ERASED.
70  SCREEN 0: WIDTH 80: CLS
80  PRINT "THIS PROGRAM LETS YOU SKETCH PICTURES INTERACTIVELY, USING"
90  PRINT "THE KEYBOARD TO CONSTRUCT YOUR DISPLAY.  THE PROGRAM"
100 PRINT "ASKS YOU TO CHOOSE A BACKGROUND COLOR AND A DRAWING COLOR FOR"
110 PRINT "YOUR PICTURE, AND A PAIR OF COORDINATES AT WHICH TO START"
120 PRINT "DRAWING.  USE THE CURSOR CONTROL KEYS TO DRAW UP, DOWN, LEFT,"
130 PRINT "OR RIGHT.  TO PICK UP YOUR PEN AND MOVE IT TO SOME OTHER PART"
140 PRINT "OF THE SCREEN, PRESS "B" AND THEN THE CURSOR CONTROL KEYS.  TO"
150 PRINT "RESUME DRAWING, PRESS "D" AND CONTINUE WITH THE CURSOR CONTROL KEYS"
160 PRINT "PRESS "S" WHEN YOU ARE FINISHED SKETCHING."
170 PRINT
180 PRINT "0 - BLACK     4 - RED         8 - GRAY          12 - LIGHT RED      "
190 PRINT "1 - BLUE      5 - MAGENTA     9 - LIGHT BLUE    13 - LIGHT MAGENTA "
200 PRINT "2 - GREEN     6 - BROWN      10 - LIGHT GREEN   14 - YELLOW         "
210 PRINT "3 - CYAN      7 - WHITE      11 - LIGHT CYAN    15 - INTENSE WHITE "
220 PRINT
230 INPUT "COLOR CHOICE FOR BACKGROUND"; BACKGROUND
240 IF BACKGROUND >= 0 AND BACKGROUND <= 15 THEN 260
250 PRINT "INVALID COLOR CHOICE. TRY AGAIN": GOTO 230
260 PRINT
270 PRINT "       1 - GREEN       4 - CYAN     "
280 PRINT "       2 - RED         5 - MAGENTA "
290 PRINT "       3 - BROWN       6 - WHITE    "
300 PRINT
310 INPUT "COLOR CODE FOR FOREGROUND"; FORE
320 IF FORE >= 1 AND FORE <= 6 THEN 340
330 PRINT "INVALID COLOR CHOICE. TRY AGAIN": GOTO 310
340 IF FORE = 1 OR FORE = 2 OR FORE = 3 THEN PALETTE = 0 ELSE PALETTE = 1
350 IF FORE = 1 OR FORE = 4 THEN DRAWCOLOR = 1
360 IF FORE = 2 OR FORE = 5 THEN DRAWCOLOR = 2
370 IF FORE = 3 OR FORE = 6 THEN DRAWCOLOR = 3
380 PRINT
390 PRINT "HIT THE DOWN  ARROW TO EXTEND YOUR LINE DOWNWARD"
400 PRINT "HIT THE  UP   ARROW TO EXTEND YOUR LINE UPWARD"
410 PRINT "HIT THE RIGHT ARROW TO EXTEND YOUR LINE TO THE RIGHT"
420 PRINT "HIT THE LEFT  ARROW TO EXTEND YOUR LINE TO THE LEFT"
430 PRINT
440 INPUT "INPUT STARTING COORDINATES"; X,Y
450 SCREEN 1: COLOR BACKGROUND, PALETTE: CLS
460 LINE (0,0) - (319,199),DRAWCOLOR,B          'DRAW BOX AROUND SCREEN EDGES
470 IF X < 0 OR X > 319 OR Y < 0 OR Y > 199 THEN 580
480 PSET (X,Y),DRAWCOLOR
490 IF DRAWING$ = "N" THEN FOR J=1 TO 50: NEXT: PRESET (X,Y)     'MOVING TO
500 A$ = INKEY$: IF A$ = "" THEN 500                             'NEW POSITION
510 IF RIGHT$(A$,1) = CHR$(80) THEN Y = Y + 1: GOTO 480          'DOWN
520 IF RIGHT$(A$,1) = CHR$(72) THEN Y = Y - 1: GOTO 480          'UP
530 IF RIGHT$(A$,1) = CHR$(77) THEN X = X + 1: GOTO 480          'RIGHT
540 IF RIGHT$(A$,1) = CHR$(75) THEN X = X - 1: GOTO 480          'LEFT
550 IF A$ = "D" OR A$ = "d" THEN DRAWING$ = "Y": GOTO 500
560 IF A$ = CHR$(83) THEN 580
570 GOTO 470
580 IF INKEY$ = "" THEN 580
590 END
```

Program 6–2 Interactive picture design using lines.

```
10 'PROGRAM 6-2.  INTERACTIVE PICTURE CONSTRUCTION WITH LINES
20      'INTERACTIVELY DRAWS LINES. ALLOWS USER TO INPUT
30      'STARTING AND ENDING COORDINATES OF EACH LINE AS
40      'WELL AS DESIRED COLOR FOR THE LINE. DRAWS PICTURE
50      'AND THEN ALLOWS USER TO KEEP OR ERASE EACH LINE.
60      '*************************************************************
70  DIM X1(20), Y1(20), X2(20), Y2(20), LINECOLOR(20)
80  SCREEN 1: CLS
90  COUNT = 1     'COUNT IS NUMBER OF LINES
100     'INPUT COORDINATES OF EACH LINE
110 PRINT "ENTER -1,-1 TO QUIT"
120 INPUT "FIRST POINT OF LINE"; X, Y
130 IF X = -1 AND Y = -1 THEN 420
140 IF X >= 0 AND X <= 319 AND Y >= 0 AND Y <= 199 THEN 160
150 PRINT "FIRST POINT OUT OF RANGE. TRY AGAIN": GOTO 120
160 X1(COUNT) = X: Y1(COUNT) = Y
170 INPUT "SECOND POINT OF LINE"; X, Y
180 IF X = -1 AND Y = -1 THEN 420
190 IF X >= 0 AND X <= 319 AND Y >= 0 AND Y <= 199 THEN 210
200 PRINT "SECOND POINT OUT OF RANGE. TRY AGAIN": GOTO 170
210 X2(COUNT) = X: Y2(COUNT) = Y
220     'CHOOSE COLOR OF LINE
230 INPUT "COLOR CODE FOR LINE (1, 2, OR 3)"; COLCODE
240 IF COLCODE = 1 OR COLCODE = 2 OR COLCODE = 3 THEN 260
250 PRINT "COLOR CHOICE INVALID. TRY AGAIN": GOTO 230
260 LINECOLOR(COUNT) = COLCODE
270 GOSUB 360           'DRAW PICTURE
280     'SHOULD LINE BE KEPT OF DISCARDED?
290 INPUT "TYPE K OR E - KEEP OR ERASE LAST LINE"; A$
300 IF A$ = "K" OR A$ = "k" OR A$ = "E" OR A$ = "e" THEN 320
310 PRINT "ENTER K OR E ONLY. TRY AGAIN": GOTO 290
320 IF A$ = "E" OR A$ = "e" THEN COUNT = COUNT - 1      'DISCARD LINE
330 GOSUB 360
340 COUNT = COUNT + 1                   'GET READY FOR NEXT LINE
350 GOTO 100
360     'DRAWS PICTURE
370 CLS
380 FOR EACH = 1 TO COUNT
390     LINE (X1(EACH),Y1(EACH)) - (X2(EACH),Y2(EACH)),LINECOLOR(EACH)
400 NEXT
410 RETURN
420 GOSUB 360
430 IF INKEY$ = "" THEN 430
440 END
```

line is displayed for our approval. Menu options allow us to erase the line if we are not satisfied with its position, to select the next line of the display, or to end.

6–3 LIGHT PENS

Figure 6–4 shows a light pen selecting menu options. A light pen is a pencil-shaped device that attaches to the six-pin connector on the Color/Graphics board and is used to detect light emitted from points on the screen. Emitted light comes from the glow of the phosphor coating on the screen as the electron beam sweeps

Figure 6–4 Light pen being used to make a menu selection.

across it. If an activated light pen is pointed toward a spot on the screen as the electron beam lights up that spot, the coordinates of the point are stored in memory. Since the electron beam sweeps across each point on the screen about 30 times every second, the detection of a lighted spot by the light pen is essentially instantaneous.

Activating a light pen is accomplished in different ways. Some pens have a push-tip that is activated by pressing the tip against the screen. Others have a button on the side of the pen that must be pressed to activate. A third type is activated by touching a metal band near the tip. Pen commands are incorporated into BASIC programs to test for an activated light pen. When a light pen is determined to be "on" by the program, the current screen coordinates of the pen can be used for menu selection, for plotting a point or character, for drawing a line, or for positioning displayed objects.

Menu selection using a light pen is accomplished in a program by testing the coordinates returned by the pen to determine which item is being selected. The item selected is the one that contains the pen coordinates within its defined boundary. In Fig. 6–5, suppose that the coordinates recorded by the pen are close to the words ADD SET OF TEST SCORES. By testing the Y value of these coordinates, we can determine that the middle activity has been selected. Items listed in a menu, such as in Fig. 6–5, should be widely separated so that coordinate positions can be clearly determined to be in only one area. Menu items can be

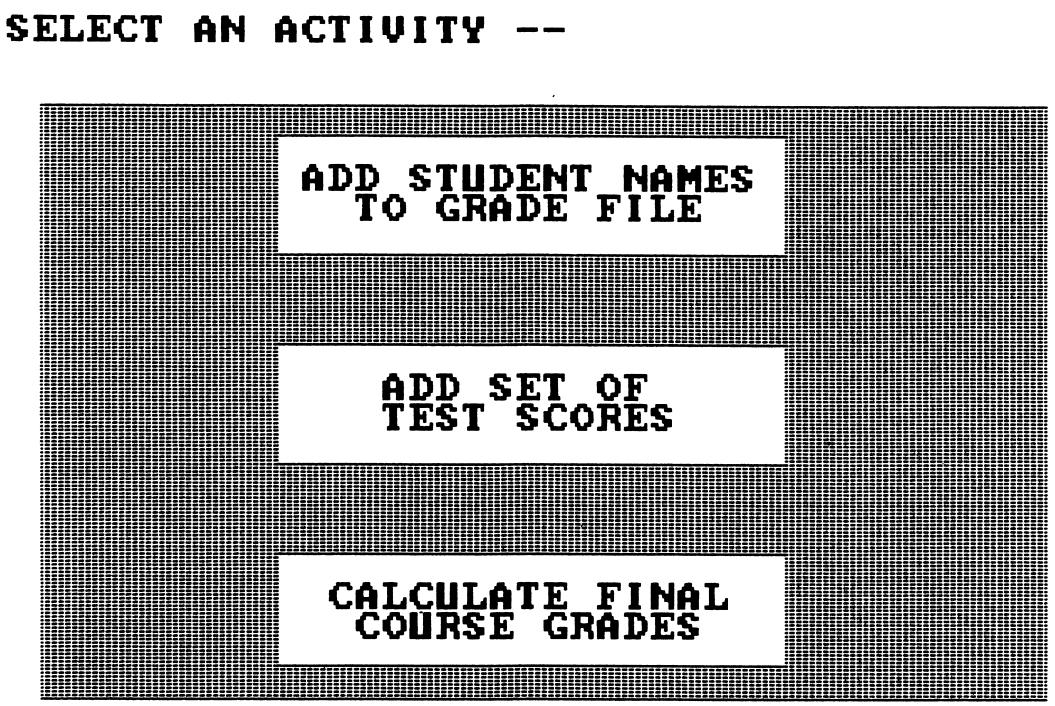


Figure 6–5 By examining the screen coordinates recorded by an activated light pen, the program determines which activity was selected.

listed as words, as color rectangles for color selection, or as shapes for object selection.

We can use a light pen in ways other than as a menu selection device. Programs can be set up to plot a continuous stream of coordinates or a series of straight lines as an activated light pen is moved around the screen. This allows us to sketch pictures and graphs on the screen, although a steady hand is needed to display objects accurately. We can use this same program to display a picture traced onto the screen with a light pen from a layout on transparent paper. The program could then store coordinates for this picture in a display file, using BSAVE. This saved display can then be viewed at a later time or it could be modified in some way. We could also use a light pen to plot a specified figure (rectangle, polygon, circle) at any selected screen position.

To record screen coordinates with a light pen, we first tell the system that we will be using a pen. This is done with the statement

PEN ON — Informs the system that a light pen is to be used, and allows later pen commands to be executed.

We must state the PEN ON command before we can use any other pen

statements. Since this command causes some extra bookkeeping by the system (storing screen coordinates and checking the pen status), we should put it into our programs at a point just before the other pen commands are to be used. This will improve program execution speed. Also, when we no longer need pen input in a program, we can stop the system from periodically checking (polling) the pen status with:

PEN OFF — Informs the system that the light pen will no longer be used.

Once we have turned the pen operations on (with PEN ON), we can make use of the activated pen coordinates with the function

PEN(N) — Returns the pen status and screen coordinates. Parameter N may be a numeric expression or constant in the range 0 to 9. If noninteger, N will be rounded.

PEN(1) returns the X coordinate and PEN(2) returns the Y coordinate of the position where the pen was last activated. We can get the pen position (X and Y) as it is continuously activated and moved around with PEN(4) and PEN(5). In a similar way, we get character row and column positions with values of N in the range 6 through 9. Values 6 and 7 for N give us the row and column numbers, respectively, where the pen was last activated. Values 8 and 9 for N return continuous row and column positions as an activated pen is moved across the screen. PEN(0) returns a value of −1 at the instant we activate the pen and a value of 0 at all other times. PEN(3) will give a value of −1 as long as we continue to activate the pen; otherwise, it is 0. Thus, to select screen positions, we would use the values 1 and 2 or 6 and 7 for N. To sketch or paint on the screen, we use the values 4 and 5 or 8 and 9 for N. The values 0 and 3 provide a means for testing pen status.

Program 6–3 is an example of menu selection using a light pen. This program

Program 6–3 Menu selection using a light pen.

```
10 'PROGRAM 6-3. LIGHT PEN SELECTION FROM A MENU
20      'DISPLAYS MENU OF PROCESSING ACTIVITIES AVAILABLE
30      'IN THE PROGRAM. ALLOWS SELECTION OF AN ACTIVITY
40      'THROUGH THE USE OF THE LIGHT PEN
50  SCREEN 1: COLOR 1,1: CLS
60      'DISPLAY MENU
70  LOCATE 1,1: PRINT "SELECT AN ACTIVITY --"
80  LINE (80,32) - (232,64),2,B
90  LINE (80,88) - (232,120),2,B
100 LINE (80,144) - (232,174),2,B
110 LINE (10,24) - (309,183),2,B
120 PAINT (20,40),1,2           'PAINT AROUND THE MENU BOXES IN CYAN
130 LOCATE 6,12: PRINT "ADD STUDENT NAMES"
140 LOCATE 7,14: PRINT "TO GRADE FILE"
```

Program 6–3 (cont.)

```
150 LOCATE 13,15: PRINT "ADD SET OF"
160 LOCATE 14,15: PRINT "TEST SCORES"
170 LOCATE 20,13: PRINT "CALCULATE FINAL"
180 LOCATE 21,14: PRINT "COURSE GRADES"
190     'READ PEN CHOICE
200 PEN ON
210 IF PEN(0) <> -1 THEN 210     'CHECK TO SEE IF PEN IS ACTIVATED
220 X = PEN(1): Y = PEN(2)       'WHEN IT IS, SAVE COORDINATES
230     'WAS PEN IN CORRECT AREA OF SCREEN?
240 IF X < 80 OR X > 232 OR Y < 32 OR Y > 174 THEN 220
250 PEN OFF
260 IF Y > 32 AND Y < 64 THEN GOSUB 320: GOTO 380
270 IF Y > 88 AND Y < 120 THEN GOSUB 340: GOTO 380
280 IF Y > 144 AND Y < 174 THEN GOSUB 360: GOTO 380
290     'CHOICE WAS IN-BETWEEN BOXES. ASK FOR RE-ENTRY
300 LOCATE 1,1: PRINT "TRY AGAIN. POINT PEN INSIDE A BOX"
310 GOTO 200
320     'CODE TO ADD STUDENT NAMES TO GRADE FILE
330 RETURN
340     'CODE TO ADD A SET OF TEST SCORES
350 RETURN
360     'CODE TO CALCULATE FINAL COURSE GRADES
370 RETURN
380 END
```

displays the menu of Fig. 6–5 and selects the next display routine according to the pen coordinates. The program loops over the pen input statements until a valid set of coordinates are obtained. Since the light pen needs to detect light from the screen in order to record coordinates, we have used a background color other than black. This lets the pen record any screen positions within the box outlines. As an alternative, we could set up a black background and use PEN(7) (or PEN(1) or PEN(4)) to check for the character or pixel row selection. The white characters or pixels then provide the light needed by the pen.

Program 6–4 presents a painting menu. A picture outline is drawn on the

Program 6–4 Picture coloring using a painting menu and a light pen.

```
10 'PROGRAM 6-4. CHOOSING COLORS WITH LIGHT PEN
20     'DRAWS TRAIN OUTLINE. ALLOWS USER TO CHOOSE COLORS
30     'FROM A MENU AND FILLS IN SELECTED AREAS WITH THE
40     'COLOR
50  SCREEN 0: COLOR 7,1,1: WIDTH 40: LOCATE ,,0: CLS
60  LOCATE 3
70  PRINT "POINT PEN AT THE COMBINATION OF COLORS"
80  PRINT "  YOU WOULD LIKE TO USE FOR COLORING -"
90  FOR ROW = 7 TO 12              'MAKE MENU FOR COLOR CHOICE
100     LOCATE ROW,11
110     COLOR 3
120     PRINT CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219);
130     COLOR 5
140     PRINT CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219);
150     COLOR 7
160     PRINT CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219);
170 NEXT
180 FOR ROW = 17 TO 22
```

Program 6–4 (cont.)

```
190     LOCATE ROW,11
200     COLOR 2
210     PRINT CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219);
220     COLOR 6
230     PRINT CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219);
240     COLOR 4
250     PRINT CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219);
260 NEXT
270     'CHOOSE PALETTE
280 PEN ON
290 IF PEN(0) <> -1 THEN 290     'HAS PEN BEEN ACTIVATED?
300 X = PEN(1): Y = PEN(2)
310 IF X < 80 OR X > 224 OR Y < 48 OR Y > 176 THEN 300
320 PEN OFF
330     'WHICH PALETTE WAS SELECTED?
340 IF Y > 48 AND Y < 96 THEN PALETTE = 1: SOUND 500,10: GOTO 390
350 IF Y > 128 AND Y < 176 THEN PALETTE = 0: SOUND 500,10: GOTO 390
360     'OTHERWISE CHOICE WAS IN-BETWEEN BOXES. ASK FOR RE-ENTRY
370 LOCATE 3: PRINT "TRY AGAIN - POINT PEN CAREFULLY"
380 GOTO 280
390 COLOR 0,7,7: CLS
400 LOCATE 1,1: PRINT "NOW POINT PEN AT ONE MORE COLOR -"
410 IF PALETTE = 0 THEN PRINT "(DON'T CHOOSE GREEN, RED, OR BROWN)"
       ELSE PRINT "(DON'T CHOOSE CYAN, MAGENTA, OR WHITE)"
420 BOXCOLOR = 0
430 FOR EACHROW = 0 TO 2                 'MAKE MENU FOR COLOR CHOICE
440     PRINT: PRINT
450     FOR BOXHEIGHT = 1 TO 5
460         PRINT "    ";
470         FOR BLOCK = 0 TO 4
480             BOXCOLOR = EACHROW * 5 + BLOCK
490             COLOR BOXCOLOR
500             PRINT CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219)+CHR$(219);
510             PRINT "  ";          'SKIP OVER TWO SPACES
520         NEXT
530         PRINT
540     NEXT
550 NEXT
560     'READ CHOICE FROM PEN
570 PEN ON
580 IF PEN(0) <> -1 THEN 580
590 ROW = PEN(6): COL = PEN(7)
600 PEN OFF
610 IF ROW > 4  AND ROW < 10 THEN ROW = 1: GOTO 640
620 IF ROW > 11 AND ROW < 17 THEN ROW = 2: GOTO 640
630 IF ROW > 18 AND ROW < 24 THEN ROW = 3
640 IF COL > 3  AND COL < 9  THEN COL = 1: GOTO 710
650 IF COL > 10 AND COL < 16 THEN COL = 2: GOTO 710
660 IF COL > 17 AND COL < 23 THEN COL = 3: GOTO 710
670 IF COL > 24 AND COL < 30 THEN COL = 4: GOTO 710
680 IF COL > 31 AND COL < 37 THEN COL = 5
690 LOCATE 1,1: PRINT "TRY AGAIN - POINT PEN CAREFULLY"
700 GOTO 570
710 SOUND 400,10: BACKGROUND = (ROW - 1) * 5 + COL - 1
720     'DRAW TRAIN
730 SCREEN 1: COLOR BACKGROUND,PALETTE          'GO TO CHOSEN COLORS
740 ENGINE$ = "BM315,135;D40L7;BL26;L33;BL26;L7U65R45D25R55"
750 BOXCAR$ = "BM200,140;D35L7;BL26;L19;BL26;L7U35R85"
760 CABOOSE$ = "BM100,125;D5R5D5L5D40L7;BL26;L19;BL26;L7U40L5U5R5U5R85"
770 TRAIN$ = ENGINE$ + BOXCAR$ + CABOOSE$
780 DRAW TRAIN$
```

Program 6–4 (cont.)

```
790 CIRCLE (35,175),13: CIRCLE (80,175),13
800 CIRCLE (135,175),13: CIRCLE (180,175),13
810 CIRCLE (236,175),13: CIRCLE (295,175),13
820 LINE (221,115) - (255,140),,B
830 LINE (25,130) - (88,150),,B
840 LINE (100,170) - (114,170)
850 LINE (200,170) - (215,170)
860     'DRAW COLOR CHOICES
870 LINE (15,20) - (65,50),,B
880 FOR EACHCOLOR = 1 TO 3
890     LINE (15+EACHCOLOR*80,20) - (15+EACHCOLOR*80+50,50),EACHCOLOR,BF
900 NEXT
910     'COLOR TRAIN
920 PEN ON
930 IF PEN(0) <> -1 THEN 930
940 X = PEN(1): Y = PEN(2)
950 IF X < 15 OR X > 305 OR Y < 20 OR Y > 50 THEN 930
960 IF X > 15 AND X < 65 THEN DRAWCOLOR = 0: GOTO 1020
970 IF X > 95 AND X < 145 THEN DRAWCOLOR = 1: GOTO 1020
980 IF X > 175 AND X < 225 THEN DRAWCOLOR = 2: GOTO 1020
990 IF X > 255 AND X < 305 THEN DRAWCOLOR = 3: GOTO 1020
1000 LOCATE 1,1: PRINT "TRY AGAIN": FOR K=1 TO 30: NEXT
1010 LOCATE 1,1: PRINT "          ": GOTO 920
1020 SOUND 600,10
1030 PEN ON
1040 IF PEN(0) <> -1 THEN 1040
1050 X = PEN(1): Y = PEN(2)
1060 SOUND 800,10
1070 PAINT (X,Y),DRAWCOLOR,3
1080 GOTO 910
1090 END
```

screen with a chosen background and palette. We paint areas of the picture by first pointing the pen at a color box and then pointing the pen at a position within an area to be painted. The picture outline and paint menu are shown in Fig. 6–6.

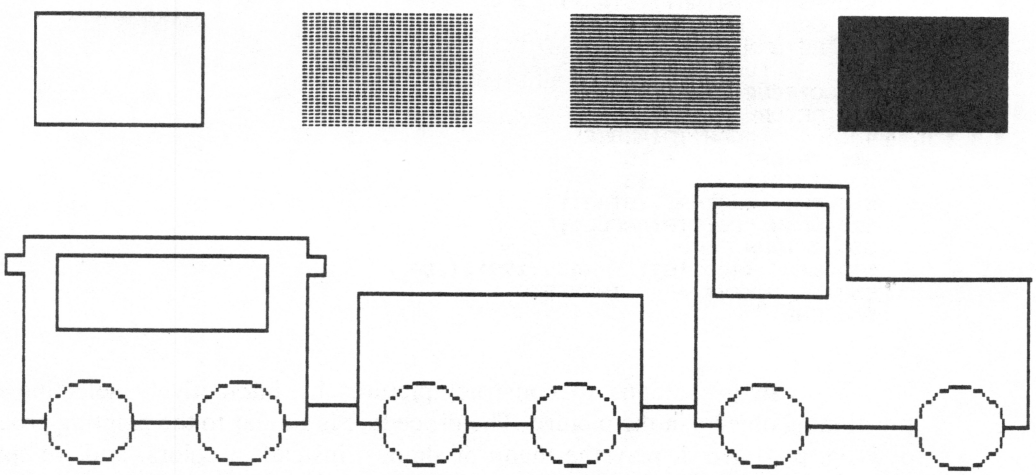


Figure 6–6 Painting menu and picture output by Prog. 6–4.

Program 6–5 Interactive picture construction using a light pen.

```
10 'PROGRAM 6-5. INTERACTIVE PICTURE CONSTRUCTION WITH LIGHT PEN
20      'DRAWS A MENU OF SHAPES DOWN THE LEFT SIDE OF THE SCREEN.
30      'ACCEPTS INPUT FROM THE LIGHT PEN TO CHOOSE A SHAPE AND
40      'FOR PLACEMENT OF THE SHAPE.
50  SCREEN 2: CLS
60  LINE (0,0) - (639,199),1,BF
70  LINE (80,0) - (80,199),0
80  LINE (0,50) - (80,50),0
90  LINE (0,100) - (80,100),0
100 LINE (0,150) - (80,150),0
110 BOX$ = "R50D25L50U25"
120 TRIANGLE$ = "R50H25G25"
130 DRAW "C0BM15,10;XBOX$;BM15,140;XTRIANGLE$;"
140 CIRCLE (40,75),25,0
150 LINE (16,168) - (64,190),0,BF
160 LOCATE 23,4: PRINT "STOP";
170 LINE (400,189) - (639,199),0,BF
180 LOCATE 25,53
190 PRINT "USE PEN TO CHOOSE A SHAPE ";
200 PEN ON
210 IF PEN(0) <> -1 THEN 210
220 X = PEN(1): Y = PEN(2)
230 IF X > 80 THEN 210
240 PEN OFF
250 IF Y < 50 THEN SHAPE = 1: SOUND 400,10: GOTO 290
260 IF Y > 50 AND Y < 100 THEN SHAPE = 2: SOUND 500,10: GOTO 290
270 IF Y > 100 AND Y <150 THEN SHAPE = 3: SOUND 600,10: GOTO 290
280 IF Y > 150 THEN SOUND 300,10: GOTO 540
290     'FIND WHERE TO PLACE THE SHAPE
300 LOCATE 25,53
310 PRINT "USE PEN TO PLACE THE SHAPE";
320 PEN ON
330 IF PEN(0) <> -1 THEN 330
340 X = PEN(1): Y = PEN(2)
350 IF X < 80 THEN 330
360 PEN OFF
370 ON SHAPE GOSUB 390,450,480
380 GOTO 180
390     'DRAW BOX
400 XSTART = X - 25     'LET PEN COORDINATES BE CENTER OF SHAPE
410 YSTART = Y - 13
420 PSET (XSTART,YSTART)
430 DRAW "C0;XBOX$;"
440 RETURN
450     'DRAW CIRCLE
460 CIRCLE (X,Y),25,0
470 RETURN
480     'DRAW TRIANGLE
490 XSTART = X - 25
500 YSTART = Y + 13
510 PSET (XSTART,YSTART)
520 DRAW "C0;XTRIANGLE$;
530 RETURN
540 LINE (400,189) - (639,199),1,BF
550 IF INKEY$ = "" THEN 550
560 END
```

We can use menus to construct pictures by interactively selecting and positioning objects in the picture. The procedure is similar to the painting process of Prog. 6–4. We display the menu of shapes, instead of colors, and we select shapes and positions with pen input. This menu is shown in Fig. 6–7, as output by Prog. 6–5. Since the program is written in SCREEN 2 and the pen needs light to

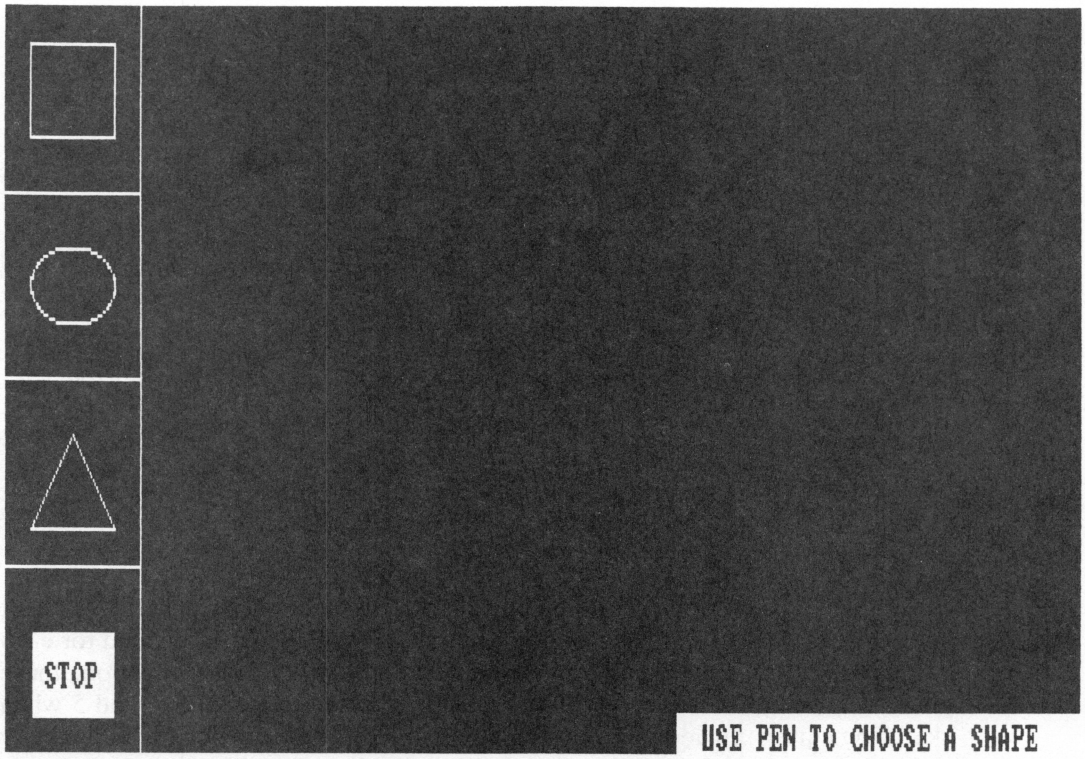


Figure 6–7 Menu to be used for picture construction with a light pen (Prog. 6–5).

record positions, we have painted the screen in white and drawn our menu shapes in black.

So far we have used menus to make selections from a list of items. Another type of menu is one that allows us to select any number within a larger interval, say 0 to 100. This type of menu can be displayed as a line (either horizontal or vertical). We make selections within the desired interval by pointing a light pen at positions along the line. Coordinate line positions selected are then converted to numbers within the interval. For example, by pointing a pen at the left end of a horizontal line, we could get the value 0. Similarly, we could get the value 100 by pointing the pen at the right end of the line. The middle of the line would then represent 50, and so on for the other points on the line.

A light pen can be used to do interactive sketching, as in Prog. 6–6. By continuously activating the pen, we sketch a figure outline as we move the pen around the screen. Sketching programs can be devised to draw with different width lines, to erase unwanted lines, and to move the pen without drawing a line. We could also change PEN(4) and PEN(5) to PEN(6) and PEN(7) and change the line-drawing statement to PRINT CHR$(AC). Then we would have a program that sketched using the character whose ASCII code is AC.

Statement 90 in Prog. 6–6 clears the storage areas for the pen coordinates. Without this statement, the last pen coordinates from the previous run of the

Program 6–6 Interactive sketching with a light pen.

```
10 'PROGRAM 6-6. INTERACTIVE SKETCHING WITH THE LIGHT PEN
20  SCREEN 1: COLOR 1,1: CLS
30      'MAKE STOP BOX
40  LINE (0,176) - (48,199),1,BF          'MAKE BLUE BOX
50  LINE (4,180) - (44,195),0,BF          'MAKE BLACK BOX INSIDE BLUE
60  LOCATE 24,2: PRINT "STOP";
70      'READ FIRST POINT FROM PEN
80  PEN ON
90  CLEER = PEN(0)                        'CLEAR OUT PRIOR PEN ACTIVITY
100 IF PEN(0) <> -1 THEN 100              'HAS PEN BEEN ACTIVATED?
110 X = PEN(1): Y = PEN(2)
120 PSET (X,Y),2                          'SET THE FIRST POINT
130     'READ ADDITIONAL POINTS
140 IF PEN(3) <> -1 THEN 140
150 IF PEN(4) < 48 AND PEN(5) > 176 THEN 190    'WE'RE IN THE STOP BOX
160 X = PEN(4): Y = PEN(5)
170 LINE - (X,Y),2
180 GOTO 140
190 LINE (0,176) - (48,199),0,BF          'COLOR IN STOP BOX WITH BLUE
200 PEN OFF
210 IF INKEY$ = "" THEN 190
220 END
```

program may be carried over to the present run. Interactive sketching with a light pen requires a steady hand. Since the pen must be individually activated for each coordinate point when we use parameters 1 and 2 in the PEN function, these parameters can produce somewhat better results than parameters 4 and 5 when sketching curves. In addition, the resolution of a pen may be such that only every eighth horizontal point can be recorded, so that line sketching may be a better choice than pixel plotting.

Two more pen statements are available in advanced BASIC. We use the following statement as a means for subroutine branching whenever we activate a light pen.

ON PEN GOSUB LN — Performs a GOSUB branch to line number LN when a light pen is activated.

After an ON PEN statement is encountered, the pen status will be checked continually as long as the program is executing. Whenever the pen is activated, there will be an immediate GOSUB branch to the line number (LN) specified in the ON PEN statement. Return from the subroutine is to the next statement that the program was about to execute when the pen was activated—unless we use a RETURN N statement at the end of the subroutine. We could set up a similar branching procedure using PEN(0) or PEN(3), but it would be a more complicated process to program. After every program statement, we would have to check the pen status with the PEN function. The ON PEN statement is useful for applications where we want to interrupt the normal program processing for a timely execution of a special routine. In a game-playing program, ON PEN could

be used to make the light pen into a switch to control the movement of some displayed object at any time during the program execution.

We suspend pen activity (if we have advanced BASIC) with

PEN STOP — Informs the system that light pen operations are to be suspended.

No further light pen operations can be performed after PEN STOP, but the system will continue to monitor the pen status. If the light pen has been activated after PEN STOP and a subsequent PEN ON is encountered, the system immediately executes any ON PEN statement we may have in the program.

6–4 JOYSTICKS AND PADDLES

These input devices connect to the Game Control Adapter board and may be used in either text or graphics mode. We can attach one or two joysticks or up to four paddles to the PC. Multiple joysticks or paddles are connected at the 15-pin connector of the Game board through an intermediate adapter plug. Paddles provide a one-dimensional input (back and forth or up and down) with a rotating knob. Joysticks give a two-dimensional input equivalent to two paddles by moving a little control stick around. One type of joystick is shown in Fig. 6–8.

Joystick or paddle input is entered into our programs with the STICK function:

STICK(N) — Returns the screen coordinates corresponding to joystick or paddle position. Parameter N may be a numeric expression or constant in the range 0 to 3. If noninteger, N will be rounded.

With paddles, each of the four values of N (0, 1, 2, and 3) causes STICK to return the knob setting on paddle A, B, C, or D, respectively. With joysticks, STICK(0) and STICK(1) return the X and Y screen position of joystick A. The other two options (N = 2 and N = 3) with the STICK function yield X and Y coordinates for joystick B. We use STICK in much the same way as the PEN function.

The range of values for X and Y that are returned by STICK depend on the particular joystick or paddle we are using. We can adapt these ranges to any screen intervals with methods similar to those we used in Chapter 4 for graphing data ranges. We first test a joystick to determine the minimum and maximum screen X values it can produce. Designating these values as XJMIN and XJMAX, we map the joystick range into any other screen range from, say, XMIN to XMAX with

$$X = XMIN + (XJ - XJMIN) * (XMAX - XMIN)/(XJMAX - XJMIN) \quad (6\text{–}1)$$

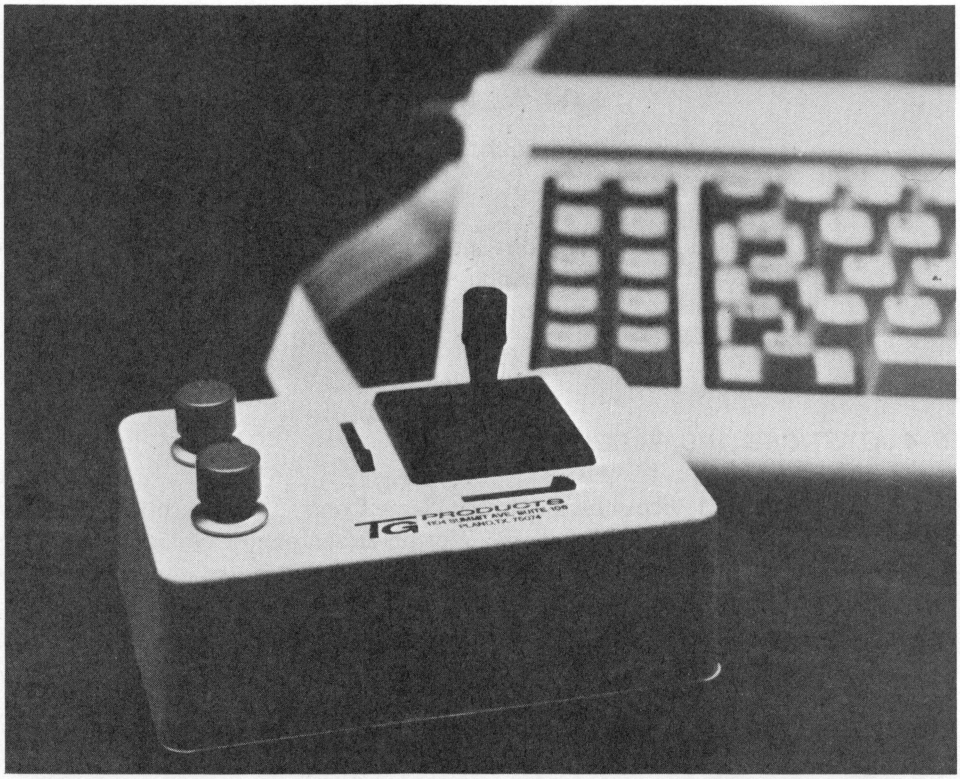


Figure 6–8 A joystick, showing the control stick and buttons.

In this calculation, X is the scaled screen position for a particular value of XJ returned by the joystick through the STICK function. For example, if the X range of joystick A is 3 to 160 and we map this across the screen width (0 to 319), then a value of 100 for XJ = STICK(0) will give us an X screen position of 197. Similarly, we get screen Y positions from a joystick value of YJ with the calculation

$$Y = YMIN + (YJ - YJMIN) * (YMAX - YMIN)/(YJMAX - YJMIN) \quad (6–2)$$

The screen range from YMIN to YMAX corresponds to a joystick range from YJMIN to YJMAX.

Equations (6–1) and (6–2) are used in Prog. 6–7 to produce a sketching routine using a joystick. Starting position is the center of the screen, which corresponds to the joystick center position. Joystick range and desired screen range are input. Program 6–7 then draws lines across the screen as the joystick is moved from one position to another. In a game application, we could use the joysticks in this way to move an object around the screen. We will discuss object translation and animation in detail in Chapters 7 and 8. Another way to draw lines with a joystick is to use the buttons as a means for signaling the program when to record a joystick position as the next line endpoint. Although the joystick can be

used for interactive sketching, as demonstrated in Prog. 6–7, it is not as good as the other interactive devices for this purpose. It is more effective in menu selection and for animating objects.

Buttons on joysticks or paddles are monitored through the following commands.

STRIG ON — Informs the system that joystick buttons are to be used.

STRIG OFF — Informs the system that joystick buttons will no longer be used.

STRIG(N) — Returns the button status of the joysticks. Parameter N may be a numeric expression or constant in the range 0 to 3. If noninteger, N will be rounded.

The STRIG function records button status on joystick A when N is set to either 0 or 1, and button status on joystick B is recorded when N is either 2 or 3. We use a value of 0 for N when we want to keep track of button activation any time during program execution. A value of 1 is used for N when we want to know whether a button is being pressed right at that instant. We will get a value of −1 from STRIG(0) if we have pushed the button on joystick A at any time since the last occurrence of STRIG(0); otherwise, we get a zero value. STRIG(1) returns a value of −1 if we are currently pressing button A and returns a value of 0 if we are not pressing it right then. The functions STRIG(2) and STRIG(3) correspond to STRIG(0) and STRIG(1) operations, respectively, for joystick B.

An example of the use of joystick buttons is given in Prog. 6–8. Here we draw lines by positioning the joystick and then pressing the button when we want the joystick position recorded. A line is then drawn from the last referenced position to the new recorded joystick position. As in programs with a light pen,

Program 6–7 Interactive sketching with a joystick.

```
10 'PROGRAM 6-7. INTERACTIVE SKETCHING WITH JOYSTICKS
20      'SKETCHES FROM CENTER JOYSTICK POSITION TO OTHER POINTS.
30      'SKETCHING IS TERMINATED BY HITTING ANY KEY ON THE KEYBOARD.
40  SCREEN 1: COLOR 1,0: CLS
50  FIRSTPOINT = 1                          'FLAG TO INDICATE FIRST POINT
60  READ XMIN, XMAX, YMIN, YMAX             'SET SCREEN AREA IN WHICH TO DRAW
70  READ XJMIN, XJMAX, YJMIN, YJMAX         'READ MIN AND MAX FOR THIS JOYSTICK
80  XCONST1 = (XMAX - XMIN) / (XJMAX - XJMIN)
90  XCONST2 = XMIN - (XJMIN * XCONST1)
100 YCONST1 = (YMAX - YMIN) / (YJMAX - YJMIN)
110 YCONST2 = YMIN - (YJMIN * YCONST1)
120     'READ POINTS OF JOYSTICK
130 XJ = STICK(0): YJ = STICK(1)
140 X = XCONST2 + (XJ * XCONST1)
150 Y = YCONST2 + (YJ * YCONST1)
160 IF FIRSTPOINT = 0 THEN LINE - (X,Y) ELSE PSET (X,Y): FIRSTPOINT = 0
170 IF INKEY$ = "" THEN 130             'IF NO KEY PRESSED, CONTINUE
180 IF INKEY$ = "" THEN 180             'HOLD PICTURE WITHOUT "OK"
190 DATA 0,319,0,199
200 DATA 3,166,3,174:   'JOYSTICK RANGE
210 END
```

joystick coordinates from the previous run of a program may be saved in STRIG(0), so we clear this area at the start of Prog. 6–8.

Menu selection using joysticks can be carried out with the methods of Prog. 6–8. A pixel object, or a character, could be displayed on the screen and positioned with a joystick at the menu item to be selected. We then signal our selection by pressing the button. Program 6–9 illustrates a joystick method of menu selection. This program uses the "happy face" character (ASCII code 1) in text mode for visually locating screen coordinates, as seen in Fig. 6–9. When we have the character positioned at the correct line, we press the button to make our selection. In graphics mode, we could display and move a small circle, an arrow, or any other object among the menu items.

Additional button commands are available in advanced BASIC. These commands allow us to independently turn on and off each of the two joysticks and to branch to subroutines.

STRIG(N) ON — Informs the system that buttons on joystick N are to be used.

STRIG(N) OFF — Informs the system that buttons on joystick N are no longer to be used.

A value of zero for N corresponds to joystick A, and the value N = 2 corresponds to joystick B. We can cause a subroutine branch any time a button is pressed with

ON STRIG(N) GOSUB LN — Performs a GOSUB branch to line number LN when the button on joystick A (N = 0) or joystick B (N = 2) is pressed.

Program 6–8 Interactive line drawing using a joystick.

```
10 'PROGRAM 6-8. INTERACTIVE SKETCHING USING JOYSTICKS AND BUTTONS
20  SCREEN 2: CLS
30  READ XMIN, XMAX, YMIN, YMAX          'SET AREA OF SCREEN IN WHICH TO DRAW
40  READ XJMIN, XJMAX, YJMIN, YJMAX
50  XCONST1 = (XMAX - XMIN) / (XJMAX - XJMIN)
60  YCONST1 = (YMAX - YMIN) / (YJMAX - YJMIN)
70  XCONST2 = XMIN - (XJMIN * XCONST1)
80  YCONST2 = YMIN - (YJMIN * YCONST1)
90  STRIG ON
100 IF STRIG(0) <> -1 THEN 100            'WAIT FOR BUTTON
110 GOSUB 150                             'CONVERT TO SCREEN POINTS
120 IF FIRSTPOINT = 0 THEN LINE - (X,Y) ELSE PSET (X,Y): FIRSTPOINT = 0
130 IF INKEY$ = "" THEN 100 ELSE 210      'HITTING KEYBOARD KEY ENDS PROGRAM
140     'CONVERT JOYSTICK COOORDINATES TO SCREEN COORDINATES
150 XJ = STICK(0): YJ = STICK(1)
160 X = XCONST2 + (XJ * XCONST1)
170 Y = YCONST2 + (YJ * YCONST1)
180 RETURN
190 DATA 0,639,0,199
200 DATA 3,166,3,175
210 END
```

Program 6–9 Menu selection with a joystick.

```
10 'PROGRAM 6-9. INTERACTIVE MENU SELECTION USING JOYSTICKS & BUTTONS
20  SCREEN 0: COLOR 2,1,4: WIDTH 40: LOCATE ,,0: CLS
30  ROW = 5
40  FOR CHOICE = 1 TO 5          'MAKE MENU
50      LOCATE ROW,12
60      PRINT STR$(CHOICE); ". Menu item "; CHR$(64+CHOICE);
70      ROW = ROW + 4
80  NEXT
90  READ YMIN, YMAX              'SET RANGE OF VALUES TO CHOOSE AMONG
100 READ YJMIN, YJMAX            'READ RANGE OF THIS JOYSTICK
110 STRIG ON
120 CLEER = STRIG(0)             'CLEAR OUT ANY BUTTON READINGS
130 WHILE STRIG(0) <> -1: GOSUB 160: WEND
140 GOTO 260
150     'DRAW IN NEW POSITION
160 HOLD = STICK(0)              'USE STICK(0) TO SAMPLE JOYSTICK
170 YJ = STICK(1)
180 CHOICE = YMIN + ((YJ - YJMIN) * (YMAX - YMIN)) / (YJMAX - YJMIN)
190 CHOICE = INT(CHOICE + .5)
200 LOCATE ROW,10: COLOR 1,1: PRINT CHR$(2);: COLOR 2,1          'ERASE OLD
210 ROW = (CHOICE - 1) * 4 + 5           'CONVERT CHOICE TO ROW ON SCREEN
220 LOCATE ROW,10: PRINT CHR$(2);        'PRINT FACE IN NEW POSITION
230 RETURN
240 DATA 1,5
250 DATA 3,175
260     'CONTINUATION OF PROGRAM
270 ON CHOICE GOTO 500, 1000, 1500, 2000, 2500
```

This command is similar to the ON PEN command. It causes the button status to be checked before each subsequent program statement is executed. When the button is pressed, the subroutine is immediately executed. Return from the subroutine is to the statement that the program was about to execute before we directed it to another part of the program by pushing a button. If we want to return

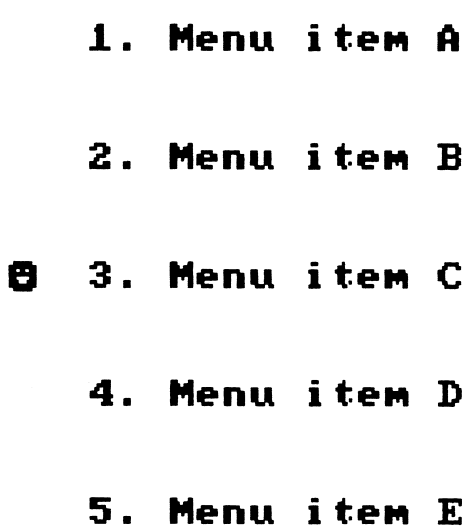


Figure 6–9 Menu for joystick selection, as produced by Prog. 6–9.

to some other part of the program, we use a RETURN LN statement. Subroutine branches can be suspended with

STRIG(N) STOP — Informs the system that button operation is suspended for joystick A (N = 0) or joystick B (N = 2).

Button activity will continue to be monitored after a STRIG(N) STOP statement, but no action will be taken. A subsequent STRIG(N) ON statement, however, will cause an immediate response if we had pressed the button at any time (provided that we have an ON STRIG command in the program).

6–5 GRAPHICS TABLETS

Like light pens or joysticks, graphics tablets are devices for selecting coordinate positions on the video screen. But now we locate screen positions from the tablet surface. The coordinate origin of a tablet is usually set to the lower left corner, but can be relocated to any other tablet position. A coordinate location will be stored in memory by the tablet whenever an activated hand cursor or stylus is placed at that coordinate position on the tablet. We activate a hand cursor or a stylus by pressing a button. Several buttons are often available to provide options, such as returning a single point or a continuous stream of points as we move around the tablet surface. Figure 6–10 shows an example of a hand cursor tablet. Graphics tablets attach to an RS–232 port and require connection to a power supply.

Selecting positions on a tablet is accomplished by lining up cross hairs on the hand cursor over a point on the tablet. Many tablets measure coordinates using voltage differences on a grid of wires in the tablet surface. Each wire has a slightly different voltage, so that we have voltage differences in each direction across the tablet, corresponding to coordinate differences. By activating the stylus or hand cursor, we cause the voltages at that position to be recorded. These voltages are converted to (X,Y) screen coordinates and stored in memory through tablet conversion programs. We can develop our own programs to accomplish this, or we can purchase programs that will load coordinate data from the tablet into memory. Some programs are available to plot automatically the loaded data and perform menu selection.

Interactive graphics uses of the tablet are similar to those of the light pen. The tablet can record coordinate positions much more accurately than the light pen, so that it is a good device for tracing ("digitizing") graphs, charts, or pictures onto the screen. The coordinate positions of the display can then be stored for use later. We can use the tablet to position objects on the screen or to create figures by drawing lines. We can also use the tablet for menu selection with a transparent overlay. The shapes of menu items to be offered for selection are drawn on the overlay, and the overlay is placed on the tablet surface. Then, selecting a coordinate position within the area of the tablet occupied by a particular shape selects that shape. We can similarly design overlays to go with programs that select colors or processing options.

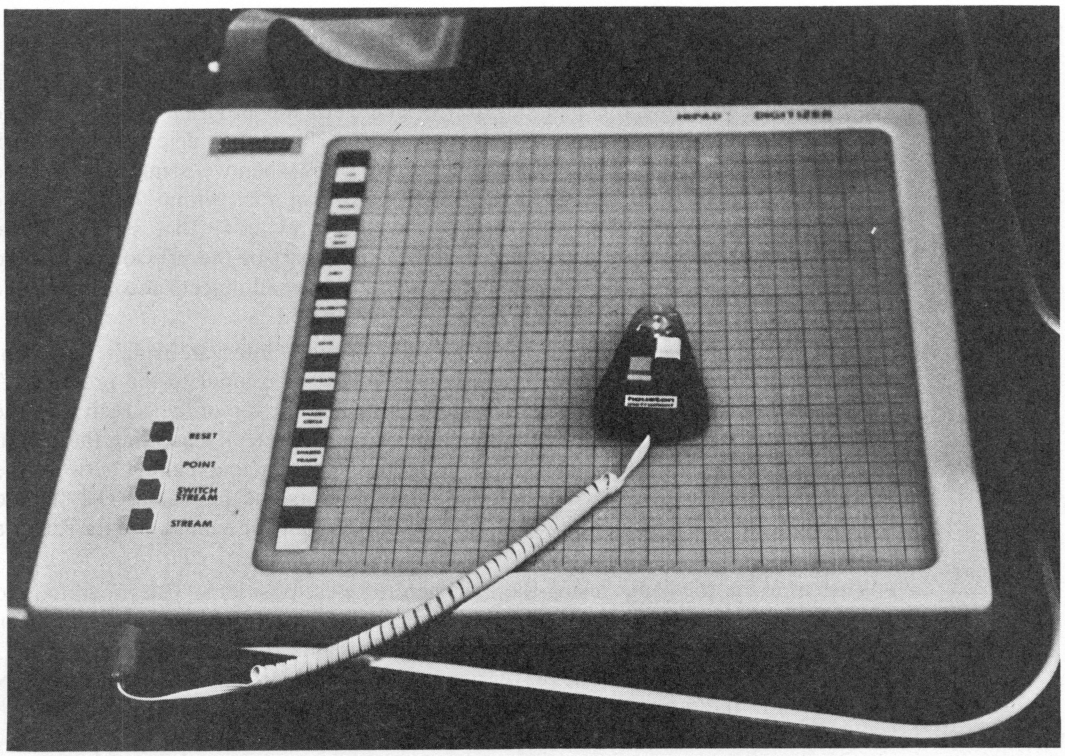


Figure 6–10 Graphics tablet with hand cursor.

PROGRAMMING PROJECTS

6–1. Modify Prog. 6–1 to draw diagonal lines. This can be accomplished by using four additional keys on the keyboard to specify the four directions: up and right, down and right, down and left, and up and left. The same keys that we use with the DRAW statement for these directions (E, F, G, H) could be used for this interactive sketching program, or we can use the cursor control keys.

6–2. Modify Project 6–1 to interactively draw lines with a specified length.

6–3. Write a menu selection program that makes character selections in high resolution (SCREEN 2) using PEN(7). Use a black background with white letters.

6–4. Expand Prog. 6–5 to select the size of the objects as well as their shapes. We can do this by pointing to the new coordinate positions for the three corners of the triangle, the two opposite corners for the box, or the two endpoints for any circle radius.

6–5. Modify the program in Project 6–4 to include straight line segments as a menu option.

6–6. Expand Project 6–4 or 6–5 to erase any selected shape if it is not positioned correctly. One way this can be accomplished is to save the locations and colors of all objects previously drawn. When the last object is to be erased, we clear the screen and redraw the picture without that shape. If the object is to be saved, we add the

location and color information to the arrays that are used to store the picture definition.

6–7. Write a program to "drag" an object around the screen as the position of a light pen is moved. Each time the pen position is changed, the program is to erase the displayed object and redraw it at the new position. This technique can be used with Prog. 6–5 to drag objects into position. Dragging stops when we point the pen at the "SELECT A NEW OBJECT" or "TERMINATE" options within the menu. The final position of an object can then be added to an array that stores picture information. Since the dragging process may erase part of the previously created picture, the program could erase the screen and redraw all objects according to the picture array information after each object is positioned.

6–8. Write a routine to select any number within a specified interval, using a light pen. The starting and ending values for the interval are to be input to the program. A horizontal line, 500 pixels long, is then to be drawn at the top of the screen with the endpoints labeled according to the interval values entered. By pointing the pen at any line position, a value within the interval (corresponding to a number between the two entered values) is selected, and the program prints out this value. The method for determining the values selected is the same as that for scaling a data set to a screen coordinate interval, as discussed in Chapter 4.

6–9. Write a tic-tac-toe game, using the light pen to select positions. The program is to first draw the empty "board." Two players then alternate pointing the pen at an empty square within the board. The program draws an X for the first player and an O for the second until either one player wins or all squares are filled. Use the ON PEN command in the program to select the routines to draw the X and O patterns.

6–10. Modify Prog. 6–7 to drag an object around the screen instead of plotting pixels. As the joystick position is changed, the displayed object is to be erased and redrawn at the new position. Movement of the object can be terminated when a joystick button is pressed.

6–11. Revise Project 6–8 to select numbers using a joystick. Selection is to be made by pointing a small arrow at the desired line position. The arrow can be made to slide back and forth below the line as the joystick is moved back and forth. Position is recorded when the joystick button is pressed.

6–12. Set up a program to interactively construct pictures with joystick input. Use the methods outlined in Project 6–6 (or 6–7). For menu selection, the joystick position can be monitored by causing a character (say, a number) to blink when the joystick position corresponds to that menu item. Pressing the button then makes that menu selection. Once an object is selected, it is positioned with the joystick and drawn when the button is pressed.

6–13. Revise Prog. 6–6 for tablet input.

6–14. Write a program to drag an object around the screen using input from a graphics tablet.

6–15. Revise Prog. 6–5 to construct pictures with a graphics tablet instead of a light pen.

6–16. Revise Project 6–6 so as to accept tablet input instead of using a light pen.

Part III

DISPLAY MANIPULATIONS

We can make our pictures move, we can change their size, or we can selectively erase any part of them. In the following three chapters we will explore basic transformation methods, animation techniques, and clipping procedures.

Chapter 7

Transformations

The transformation methods discussed in this chapter provide basic tools for manipulating displays. With these transformation methods we can move pictures and graphs from one screen location and orientation to another. We can create inserts for a developed display. We can enlarge the size of a picture or graph for clarity or insertion of more detail. We can reduce the size of a display in order to be able to add explanatory information, such as notes or another picture. Transformation methods also provide the basis for animating displays.

These manipulations of our pictures and graphs are brought about through the application of the three fundamental transformations: (1) movement of a displayed object from one screen location to another (translation), (2) enlargement or reduction in the size of a displayed picture or graph (scaling), and (3) changes in the direction of orientation of a graphics representation (rotation).

7–1 CHANGING POSITIONS (TRANSLATION)

Techniques for changing screen position allow us to construct a picture or graph in any part of the screen and then move it to any other screen location. We may want only to rearrange the display or we may want to build up a display a piece at a time from a set of component parts.

Relocating displays from one screen position to another merely requires that we change the screen coordinates of plotted points to correspond to the new screen position. We do this by specifying a displacement, or offset, from the old position to the new position. The transformation calculation necessary for moving a point about the screen from location (X,Y) to a new location (XT,YT) can be

stated in terms of **translation distances** H and V:

$$XT = X + H$$
$$YT = Y + V \qquad (7\text{–}1)$$

A positive value for H indicates displacement of the point horizontally to the right, while a positive value for V indicates displacement vertically toward the bottom of the screen. Negative values for H or V will translate the point to the left or up, respectively.

In translating points, we should use values for H and V that keep the points within the screen boundary. Very small values for H and V should also be avoided. If values for H and V are both less than 0.5, the original points will simply be replotted, which could amount to a significant slowing down of a program.

TRANSLATING PICTURES

Displacing a picture from one screen location to another means that we translate all points of the picture and then redraw all the lines connecting the translated points. To avoid changing the shape of an object, we must displace all points by the same distance. That is, calculations (7–1) are applied with the same H value for all horizontal coordinates and the same V value for all vertical coordinates of the picture. Using a different H or a different V for different points in the picture will distort the original display. We usually want to translate objects without distortion, but deliberate distortions can be the basis for experimenting with design shapes or for game playing.

Coordinate endpoints for each line in a picture can be conveniently stored in an array. For a simple picture, we can store the points in a one-dimensional array in the order that we want them connected. But if we have a more complicated figure, a point may connect to several lines or we may have parts in the picture that are not connected by lines to other parts, as in Fig. 7–1. We could then store coordinate values in two-dimensional arrays. One subscript of the array identifies the picture part, the other subscript tells us which point of that part. Thus, (X(2,1),Y(2,1)) could be used to store the coordinates for the first point of part 2 of the picture (such as the boy in Fig. 7–1). Two-dimensional arrays are used in Prog. 7–1 to store coordinates defining the details of Fig. 7–1. This program lets us translate the picture components to any screen position, as many times as we wish.

Program 7–1 stores the translated coordinates in the original arrays X and Y. If we wanted to save the original position, we could store translated coordinates in different arrays, such as XT and YT. Saving the original position is desirable if we just want to test new positions for visual effect or if we want to display the object in both positions. If there is no reason to keep the original position of a picture, we save storage space by recalculating coordinates in the original arrays X and Y.

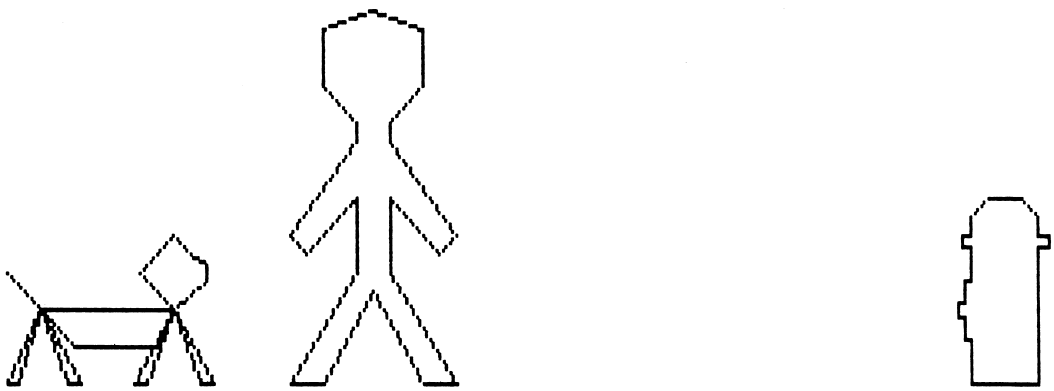
For objects with symmetry or with boundaries calculated from equations, we do not have to add translation distances to each point of the object to relocate it. To move a circle or ellipse, for instance, we only need to translate the figure center and redraw the curve. To relocate a rectangle, we could translate one corner and redraw the rectangle using values for the width and height.

TRANSLATING GRAPHS

The methods and considerations discussed for translating pictures also apply to graphs—all points of a graph are reassigned new values (within the screen limits) and all lines are redrawn at the new location. Labeling could then be added at appropriate places within the new graph, or we could translate the label positions together with the other pixel coordinates of the display.

To translate character labels, we need to express translation distances H and V in terms of character position changes. We get the horizontal shift of a label by dividing H by the number of horizontal points in the character pixel grid. We get the vertical shift by dividing V by the number of vertical points in the character

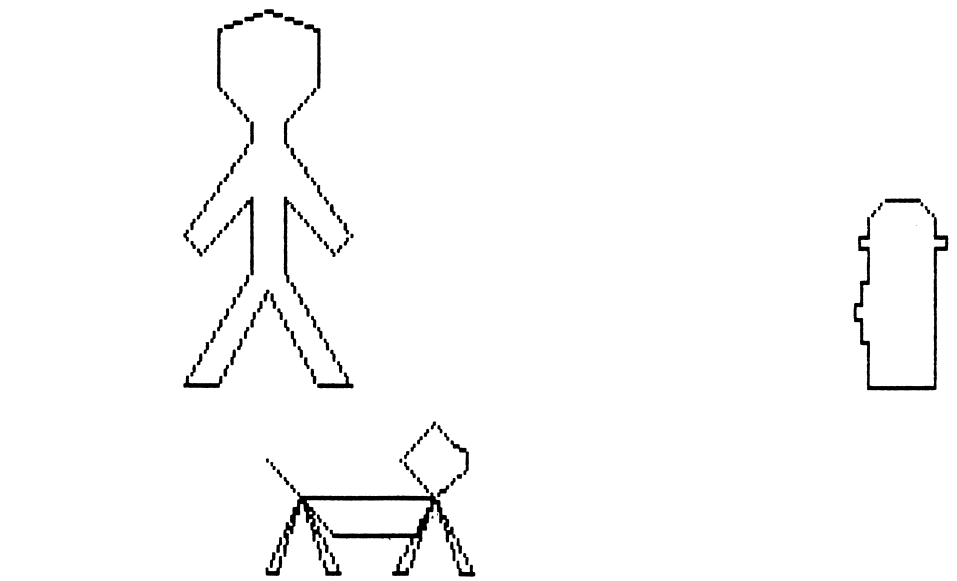
Figure 7–1 Translation of a picture component from the original position (a) to position (b) then to position (c) by Prog. 6–1.



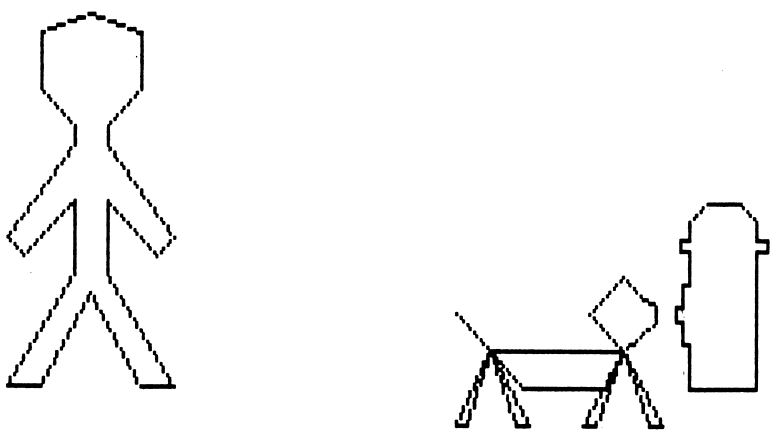
```
1-BOY  2-DOG  3-HYDRANT
PICTURE PART TO MOVE (0 TO END)? ■
```

(a)

Figure 7–1 (cont.)



(b)



(c)

Program 7–1 Translating pictures (boy, dog, and hydrant).

```
10 'PROGRAM 7-1. TRANSLATION OF PICTURE PARTS.
20      'DRAWS PICTURE AND ALLOWS USER TO TRANSLATE PARTS OF
30      'THE DISPLAY TO OTHER LOCATIONS. PICTURE PARTS ARE
40      'STORED IN ARRAYS X AND Y. TRANSLATED POINTS REPLACE
50      'THE ORIGINAL POINTS IN X AND Y.
60  DIM X(5,50), Y(5,50), POINTCOUNT(5)
70  SCREEN 1: COLOR 4,0: CLS
80      '*************** READ PICTURE PARTS AND DRAW ***************
90  PICTUREPART = 0      'PICTUREPART IS PART NUMBER
100 READ XD, YD
110 WHILE XD <> -100                    '-100 IS END OF DATA SIGNAL
120     PICTUREPART = PICTUREPART + 1
130     EACHPOINT = 0
140     WHILE XD => 0                   '-1 IS END OF PICTURE PART SIGNAL
150           EACHPOINT = EACHPOINT + 1
160           X(PICTUREPART,EACHPOINT) = XD
170           Y(PICTUREPART,EACHPOINT) = YD
180           READ XD,YD
190     WEND
200     'STORE # OF ELEMENTS IN PICTUREPART IN POINTCOUNT(PICTUREPART)
210     POINTCOUNT(PICTUREPART) = EACHPOINT
220     READ XD,YD
230 WEND
240     'DRAW PICTURE
250 DRAWCOLOR = 3
260 FOR K = 1 TO PICTUREPART
270     GOSUB 570               'DRAW EACH PART
280 NEXT
290     '****************** CONTROL PROGRAM FLOW ******************
300 GOSUB 430                   'GET CHOICE OF WHICH PART TO MOVE
310 WHILE CHOICE <> 0           'USER ENTERS 0 TO END PROGRAM
320     GOSUB 480               'HOW MUCH TO MOVE?
330     K = CHOICE
340     DRAWCOLOR = 0           'ERASE CURRENT DISPLAY OF CHOSEN PART
350     GOSUB 570
360     GOSUB 510               'RECALCULATE POINTS
370     DRAWCOLOR = 3
380     GOSUB 570               'DISPLAY IN NEW POSITION
390     GOSUB 430               'GET CHOICE OF PART TO MOVE OR QUIT
400 WEND
410 GOTO 760
420     '********** PRINT INSTRUCTIONS FOR TRANSLATING **********
430 LOCATE 22,1: PRINT "1-BOY  2-DOG  3-HYDRANT"
440 INPUT "PICTURE PART TO MOVE (0 TO END)"; CHOICE
450 LOCATE 22,1: PRINT STRING$(78," ");
460 RETURN
470     '****************** HOW MUCH TO MOVE? ******************
480 LOCATE 23,1: INPUT "H AND V AMOUNT TO MOVE"; H, V
490 RETURN
500     '******************* RECALCULATE POINTS *******************
510 FOR J = 1 TO POINTCOUNT(CHOICE)
520     X(CHOICE,J) = X(CHOICE,J) + H
530     Y(CHOICE,J) = Y(CHOICE,J) + V
540 NEXT
550 RETURN
560     '********************* DRAW ROUTINE *************************
570 FOR J = 1 TO POINTCOUNT(K)-1
580     LINE (X(K,J), Y(K,J)) - (X(K,J+1), Y(K,J+1)),DRAWCOLOR
590 NEXT
600 RETURN
```

Program 7–1 (cont.)

```
610 '*************************************************************************
620 DATA 85,70,90,75,105,60,105,80,85,110,95,110,110,85
630 DATA 125,110,135,110,115,80,115,60,130,75,135,70,115,45
640 DATA 115,40,125,30,125,15,110,10,95,15,95,30,105,40,105,45,85,70
650 DATA -1,-1
660 DATA 50,90,62,110,58,110,50,90,42,110,38,110,50,90
670 DATA 45,100,20,100,10,90,22,110,18,110,10,90,3,110
680 DATA 0,110,10,90,0,80,10,90,50,90,40,80,50,70,55,75,60,78,60,82,50,90
690 DATA -1,-1
700 DATA 290,110,290,98,288,98,288,92,286,92,286,88
710 DATA 286,88,288,88,288,82,290,82,290,73,287,73,287,70,290,70,290,65
720 DATA 295,60,305,60,310,65,310,70,313,70,313,73,310,73,310,110,290,110
730 DATA -1,-1
740 DATA -100,-100
750 '*************************************************************************
760 IF INKEY$ = "" THEN 760
770 END
```

pixel grid. Assuming that we have the Color/Graphics option with an 8 by 8 pixel grid, the line number of a label would be displaced by a distance of V/8 and the horizontal position of a label would be shifted by a distance of H/8. If labels are to be translated, we should choose H and V to be multiples of the character grid dimensions. This will keep the labels in the same relative graph positions after translation.

A method for moving a labeled bar graph is given in Prog. 7–2. The LEN function is used to get the length of each string label in checking for possible translation off screen.

Program 7–2 Translating a graph.

```
10 'PROGRAM 7-2. TRANSLATING A LABELED HORIZONTAL BAR GRAPH.
20      'LABELS ARE STORED IN THE ARRAY LABEL$. PRINT POSITION
30      'OF EACH LABEL IS STORED IN ARRAYS ROW AND COLUMN. LABELS
40      'MAY BE UP TO 8 CHARACTERS IN LENGTH AND OCCUPY THE FIRST 8
50      'COLUMNS OF A PRINT LINE. MAGNITUDES ARE SCALED TO USE PIXELS
60      '64 - 160. ENDPOINTS OF THE BAR FOR EACH GRAPH DIVISION ARE
70      'STORED IN ARRAYS X AND Y. ONCE THE GRAPH IS CREATED, IT MAY
80      'BE TRANSLATED TO SOME OTHER LOCATION ON THE SCREEN.
90 '*************************************************************************
100 DIM X(8,2), Y(8,2), LABEL$(9), ROW(9), COLUMN(9)
110 SCREEN 1: COLOR 4,1: CLS
120     '*************** READ DATA FOR GRAPH AND DRAW ***************
130 READ MIN, MAX       'READ MINIMUM, MAXIMUM VALUES TO USE FOR DATA RANGE
140 RANGERATIO = (160 - 64) / (MAX - MIN)
150    'READ DATA FOR GRAPH
160 READ NUMBER      'READ NUMBER OF DIVISIONS
170 FOR K = 1 TO NUMBER
180     READ LABEL$(K), MAGNITUDE
190     ROW(K) = K + 2          'SAVE FIRST 2 ROWS FOR TITLE AND SPACE
200     COLUMN(K) = 1
210     X(K,1) = 64
220     X(K,2) = INT((MAGNITUDE - MIN) * RANGERATIO + 64.5)
230     Y(K,1) = (K + 1) * 8 + 1      'TOP OF BAR
240     Y(K,2) = Y(K,1) + 4           'BOTTOM OF BAR
```

Program 7–2 (cont.)

```
250 NEXT
260 READ LABEL$(NUMBER + 1)     'READ TITLE
270 ROW(NUMBER + 1) = 1
280 COLUMN(NUMBER+1) = 10 - LEN(LABEL$(NUMBER+1)) / 2   'CENTER ON 10
290         'COPY ORIGINAL VALUES TO ARRAYS USED TO HOLD TRANSLATED
300         'POINTS SINCE THESE ARE USED TO DRAW GRAPH
310 GOSUB 580               'DRAW GRAPH
320     '************* PRINT INSTRUCTIONS FOR TRANSLATING ***********
330 LOCATE 20,1: PRINT "ENTER -999,-999 TO END"
340 INPUT "H AND V AMOUNTS TO MOVE"; H,V
350         'ERASE INSTRUCTIONS
360 LOCATE 20,1: PRINT STRING$(80," ");          'ERASE 2 LINES
370 IF H = -999 THEN 800
380 IF H/8 = INT(H/8) AND V/8 = INT(V/8) THEN 410       'MULTIPLE OF 8?
390 LOCATE 20,1: PRINT "USE MULTIPLES OF 8";
400 FOR DELAY = 1 TO 500: NEXT: GOTO 330
410     '****************** RECALCULATE POINTS ***********************
420 FOR K = 1 TO NUMBER
430     ROW(K) = ROW(K) + INT(V / 8)
440     IF ROW(K) < 1 OR ROW(K) > 25 THEN 700
450     COLUMN(K) = COLUMN(K) + INT(H / 8)
460     IF COLUMN(K) < 1 OR COLUMN(K) + LEN(LABEL$(K)) > 40 THEN 700
470     FOR J = 1 TO 2
480         X(K,J) = X(K,J) + H
490         IF X(K,J) < 0 OR X(K,J) > 319 THEN 720
500         Y(K,J) = Y(K,J) + V
510         IF Y(K,J) < 0 OR Y(K,J) > 199 THEN 720
520     NEXT J
530 NEXT
540 ROW(NUMBER + 1) = ROW(K) + INT(V / 8)
550 COLUMN(NUMBER + 1) = COLUMN(K) + INT(H / 8)
560 GOSUB 580
570 GOTO 330
580 '########################## DRAW ROUTINE ####################################
590 CLS
600 LOCATE ROW(NUMBER + 1),COLUMN(NUMBER + 1)
610 PRINT LABEL$(NUMBER + 1)
620 PRINT
630 FOR K = 1 TO NUMBER
640     LOCATE ROW(K),COLUMN(K): PRINT LABEL$(K);
650     'DRAW BAR FOR THIS DIVISION
660     LINE (X(K,1),Y(K,1)) - (X(K,2),Y(K,2)),,BF
670 NEXT
680 RETURN
690 '###############################################################################
700 PRINT "LABEL OFF SCREEN"
710 GOTO 810
720 PRINT "GRAPH POINT OFF SCREEN"
730 GOTO 810
740 '#########################
750 DATA 0,800
760 DATA 6
770 DATA NORTH,500,CENTRAL,700,MIDWEST,300,SOUTH,800,FARWEST,400,OVERSEA,750
780 DATA "REGIONAL SALES"
790 '#########################
800 IF INKEY$ = "" THEN 800
810 END
```

INTERACTIVE TRANSLATIONS

Any of the interactive methods discussed in Chapter 6 can be used to perform translations. We can set up menus that allow us to select all or part of the picture to be translated and the coordinates of the new position. A general translation program could then clear the screen and move the object to the new position by calculating displacements.

With a light pen, we can touch the screen at two positions to accomplish translation. One position is a point on the object and the other position is where we want to move it. This is demonstrated in Prog. 7–3. The displacements H and

Program 7–3 Interactive object translation using a light pen.

```
10 'PROGRAM 7-3. TRANSLATION WITH THE LIGHT PEN
20      'PRESENTS SERIES OF SHAPES. SHAPE IS SELECTED BY LIGHT
30      'PEN AND ERASED. SHAPE IS REDRAWN IN NEW POSITION
40      'INDICATED BY LIGHT PEN. SHAPE MAY BE MOVED ANY
50      'NUMBER OF TIMES.
60  SCREEN 2: CLS
70  LINE (0,0) - (639,199),1,BF  'FILL IN SCREEN WITH WHITE
80  LINE (8,168) - (66,190),0,B          'MAKE STOP BOX
90  LINE (16,174) - (60,184),0,BF
100  LOCATE 23,4: PRINT "STOP";
110      'READ PICTURE PARTS
120 READ SHAPECOUNT
130 FOR EACHSHAPE = 1 TO SHAPECOUNT
140     READ POINTCOUNT(EACHSHAPE)
150     FOR EACHPOINT = 1 TO POINTCOUNT(EACHSHAPE)
160         READ X(EACHSHAPE,EACHPOINT), Y(EACHSHAPE,EACHPOINT)
170     NEXT
180 READ TOPLEFTX(EACHSHAPE), TOPLEFTY(EACHSHAPE)
190 READ BOTTOMRIGHTX(EACHSHAPE), BOTTOMRIGHTY(EACHSHAPE)
200 NEXT
210 STOPBOX = SHAPECOUNT + 1
220 TOPLEFTX(STOPBOX) = 8: TOPLEFTY(STOPBOX) = 168     'SAVE STOP BOX INFO
230 BOTTOMRIGHTX(STOPBOX) = 64: BOTTOMRIGHTY(STOPBOX) = 199
240     'DRAW
250 DRAWCOLOR = 0
260 FOR EACHSHAPE = 1 TO SHAPECOUNT
270     GOSUB 770
280 NEXT
290     'MOVE SHAPES
300 PEN ON
310 IF PEN(0) <> -1 THEN 310
320 X = PEN(1): Y = PEN(2)
330 PEN OFF
340     'WHICH SHAPE ARE WE POINTING AT?
350 EACHSHAPE = 1
360 IF X >= TOPLEFTX(EACHSHAPE) AND X <= BOTTOMRIGHTX(EACHSHAPE) THEN 400
370 EACHSHAPE = EACHSHAPE + 1
380 IF EACHSHAPE <= SHAPECOUNT + 1 THEN 360
390 GOTO 300              'NOT POINTED AT SHAPE -- READ PEN AGAIN
400 IF Y >= TOPLEFTY(EACHSHAPE) AND Y <= BOTTOMRIGHTY(EACHSHAPE) THEN 420
410 GOTO 300              'NOT POINTED AT SHAPE -- READ PEN AGAIN
420     'EACHSHAPE IS NUMBER OF CHOSEN SHAPE
430     'IS IT THE STOP BOX?
440 IF EACHSHAPE = SHAPECOUNT + 1 THEN LINE(8,168) - (64,199),1,BF: GOTO 890
450     'IF NOT, ERASE CURRENT DISPLAY OF SHAPE
460 DRAWCOLOR = 1
470 GOSUB 770                   'DRAW SHAPE IN BACKGROUND TO ERASE
```

Program 7–3 (cont.)

```
480     'WHERE TO MOVE SHAPE?
490 PEN ON
500 IF PEN(0) <> -1 THEN 500     'HAS PEN BEEN ACTIVATED?
510 X = PEN(1): Y = PEN(2)
520 PEN OFF
530     'RECALCULATE POINTS
540 H = X - X(EACHSHAPE,1)       'USE FIRST POINT OF SHAPE AS CURRENT POSITION
550 V = Y - Y(EACHSHAPE,1)
560 EACHPOINT = 1
570 X(EACHSHAPE,EACHPOINT) = X(EACHSHAPE,EACHPOINT) + H
580 Y(EACHSHAPE,EACHPOINT) = Y(EACHSHAPE,EACHPOINT) + V
590     'IS ANY PART OF SHAPE OFF SCREEN?
600 IF X(EACHSHAPE,EACHPOINT) < 0 OR X(EACHSHAPE,EACHPOINT) > 639 OR
       Y(EACHSHAPE,EACHPOINT) < 0 OR Y(EACHSHAPE,EACHPOINT) > 199 THEN 720
610 EACHPOINT = EACHPOINT + 1
620 IF EACHPOINT <= POINTCOUNT(EACHSHAPE) THEN 570
630     'RECALCULATE TOPLEFT AND BOTTOMRIGHT COORDINATES
640 TOPLEFTX(EACHSHAPE) = TOPLEFTX(EACHSHAPE) + H
650 TOPLEFTY(EACHSHAPE) = TOPLEFTY(EACHSHAPE) + V
660 BOTTOMRIGHTX(EACHSHAPE) = BOTTOMRIGHTX(EACHSHAPE) + H
670 BOTTOMRIGHTY(EACHSHAPE) = BOTTOMRIGHTY(EACHSHAPE) + V
680 DRAWCOLOR = 0
690 GOSUB 770            'DRAW SHAPE IN NEW POSITION
700 GOTO 300
710     'SHAPE IS OFF SCREEN. RESTORE THOSE POINTS THAT WERE CHANGED
720 FOR K = EACHPOINT TO 1 STEP -1
730     X(EACHSHAPE,K) = X(EACHSHAPE,K) - H
740     Y(EACHSHAPE,K) = Y(EACHSHAPE,K) - V
750 NEXT
760 GOTO 300
770     '*************** DRAWS SHAPE ************************
780 FOR K = 1 TO POINTCOUNT(EACHSHAPE) - 1
790     LINE (X(EACHSHAPE,K),Y(EACHSHAPE,K)) - (X(EACHSHAPE,K + 1),
            Y(EACHSHAPE,K + 1)),DRAWCOLOR
800 NEXT
810 LINE (X(EACHSHAPE,POINTCOUNT(EACHSHAPE)),Y(EACHSHAPE,POINTCOUNT(EACHSHAPE)))
          - (X(EACHSHAPE,1),Y(EACHSHAPE,1)),DRAWCOLOR
820 RETURN
830 DATA 5
840 DATA 3,125,175,100,195,150,195,100,175,150,195
850 DATA 4,260,175,210,175,210,195,260,195,210,175,260,195
860 DATA 6,363,175,337,175,325,187,337,195,363,195,375,187,325,175,375,195
870 DATA 4,435,175,435,195,495,195,495,175,435,175,495,195
880 DATA 3,550,160,580,195,610,160,550,160,610,195
890 IF INKEY$ = "" THEN 890
900 END
```

V are calculated from the values for the two points selected. This program allows us to continually drag an object around the screen until we select QUIT. Similar methods can be set up for keyboard, graphics tablet, or joystick input.

DRAW STATEMENT TRANSLATIONS

We can translate an object in a very simple way if we have constructed it with DRAW statements. Since the object is drawn from the last point referenced, we just change the reference point and redraw it. For example, changing the

coordinates of the PSET statement in Prog. 7–4 displaces the object when we redraw it.

Program 7–4 Translation using the DRAW statement.

```
10 'PROGRAM 7-4. TRANSLATION WITH THE DRAW STATEMENT
20  SCREEN 1: COLOR 1,0: CLS
30  SHAPE$ = "R50D25L50U25"
40  DRAW "BM90,10;XSHAPE$;"
50  LOCATE 1,1: INPUT "WHERE TO PUT SHAPE (-1,-1 TO END)"; X,Y
60  LOCATE 1,1: PRINT "
70  IF X = -1 THEN 110
80  PSET (X,Y)
90  DRAW SHAPE$
100 GOTO 50
110 END
```

7–2 CHANGING SIZES (SCALING)

Having created a display, we may decide to enlarge it in order to clarify the information presented or to reduce it to fit into a larger graphics picture. The size of a picture or graph is changed by multiplying all distances between points by the amount that we wish to enlarge or reduce the display. If we want to double the size of a rectangle, all line lengths are multiplied by 2; if the size is to be cut in half, all lengths are multiplied by 1/2.

SCALING LINES

A horizontal line with left X coordinate X1 and right X coordinate X2 has length

$$LH = X2 - X1 \tag{7–2}$$

Changing the length of the line means that we multiply LH by a **horizontal scaling factor,** say HS, to produce the new length:

$$\begin{aligned} LHS &= LH * HS \\ &= X2 * HS - X1 * HS \\ &= XS2 - XS1 \end{aligned} \tag{7–3}$$

where XS1 and XS2 are the new X coordinates of the scaled line and HS must be a positive number (HS > 0). The scaling factor HS will produce a longer line if HS > 1. A shorter line will result if HS < 1. If HS = 1, there is no change in the length of the line.

Similarly, a vertical line with length

$$LV = Y2 - Y1 \tag{7–4}$$

can be scaled with a **vertical scaling factor** VS to produce a line of scaled length:

$$\begin{aligned} LVS &= LV * VS \\ &= Y2 * VS - Y1 * VS \\ &= YS2 - YS1 \end{aligned} \tag{7–5}$$

The new Y coordinates of the line endpoints after scaling are called YS1 and YS2. We get a shorter line when VS is between 0 and 1, and we produce a longer line when VS is greater than 1.

Diagonal lines (Fig. 7–2) can be considered in terms of their horizontal and vertical components. We enlarge or reduce them by scaling both the horizontal length component and the vertical length component, using calculations (7–3) and (7–5).

Since scaling a line affects the values of the coordinate endpoints, we usually want to specify where the scaled line is to be redrawn. We could translate the line so that one end is where we want it after scaling and then scale the length with that end fixed. A general approach is to first pick out a reference point that we consider fixed, such as (XF,YF) in Fig. 7–2. This point is called the **fixed coordinate position**. It can be any point on or off the screen. We then get the new endpoints of the line by scaling distances between each original endpoint and the fixed point (XF,YF). Thus the scaled position of point 1 in Fig. 7–2 will become

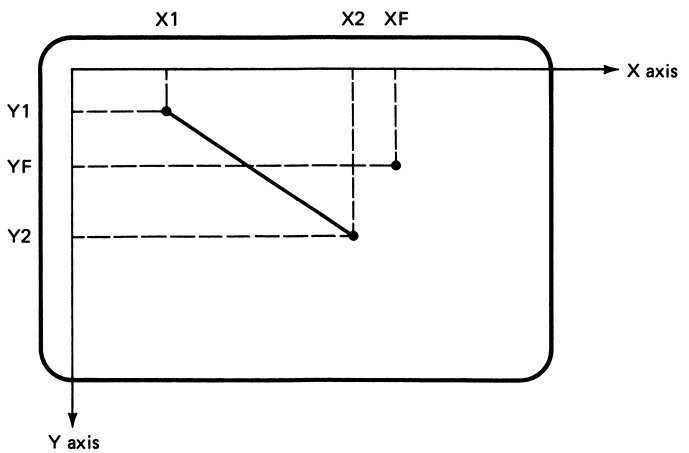
$$\begin{aligned} XS1 &= XF + (X1 - XF) * HS \\ YS1 &= YF + (Y1 - YF) * VS \end{aligned} \tag{7–6}$$

We can rewrite these calculations, and those for point 2, in the form

$$\begin{aligned} XS1 &= X1 * HS + XF * (1 - HS) \\ YS1 &= Y1 * VS + YF * (1 - VS) \\ XS2 &= X2 * HS + XF * (1 - HS) \\ YS2 &= Y2 * VS + YF * (1 - VS) \end{aligned} \tag{7–7}$$

The expressions XF * (1 − HS) and YF * (1 − VS) are each constant, so we only

Figure 7–2 A straight line, with horizontal length component X2 − X1 and vertical length component Y2 − Y1, can be scaled relative to any fixed point (XF,YF).



have to evaluate them once for all coordinates. To enlarge or reduce a diagonal line without changing the slope, we must set HS = VS.

SCALING DISPLAYS

Scaled pictures and graphs are uniformly enlarged or reduced by setting HS and VS to the same value. Relations (7–7) can then be used to calculate new coordinates for all the points of the scaled display. The enlarged or reduced picture is displayed by drawing the connecting lines for these new points. For figures with curved lines it may not be necessary to apply (7–7) to every point of the picture. To scale a circle with radius R, we only need to compute the new coordinates for the center of the circle, using (7–7), and calculate the new radius as R * S. The CIRCLE command or the routines of Section 5–1 can then be used to display the scaled circle.

In some cases we may want to stretch a display in one direction (say, the vertical direction) and maintain the original size in the other direction (horizontal). To turn a square into a rectangle, we could double the height only. This is accomplished by setting the vertical scaling factor VS to 2 for the calculations involving vertical coordinates and by setting the horizontal scaling factor HS to 1 for the calculations involving the horizontal coordinates. This technique can be useful in experimenting with or adjusting the proportions of picture components. We can also use nonuniform scaling to build composite pictures from a set of standard shapes, such as a square which we can scale to a rectangle of any size.

Program 7–5 gives an example of picture scaling. We input the picture

Program 7–5 Scaling picture parts (car).

```
10 'PROGRAM 7-5. SCALING A PICTURE WITH THE LIGHT PEN
20      'DRAWS CAR AND ALLOWS SCALING OF ALL OR PART OF CAR
30  DIM X(5,100), Y(5,100), POINTCOUNT(5)
40  SCREEN 1: COLOR 3,0: CLS
50      '************** CONTROL PROGRAM FLOW **********************
60  GOSUB 180
70  GOSUB 410                       'GET CHOICE OF WHICH PART TO SCALE
80  WHILE CHOICE <> 0
90      GOSUB 460                   'HOW MUCH TO SCALE?
100     DRAWCOLOR = 0
110     IF CHOICE <> 3 THEN K = CHOICE: GOSUB 600
           ELSE FOR K = 1 TO PICTUREPART: GOSUB 600: NEXT
120     GOSUB 490                   'RECALCULATE SCALED POINTS
130     DRAWCOLOR = 2
140     IF CHOICE <> 3 THEN K = CHOICE: GOSUB 600
           ELSE FOR K = 1 TO PICTUREPART: GOSUB 600: NEXT
150     GOSUB 410                   'GET CHOICE OF PART TO SCALE (OR QUIT)
160 WEND
170 GOTO 860
180     '*************** READ PICTURE PARTS AND DRAW ***************
190 PICTUREPART = 0
200 READ XD,YD
210 WHILE XD <> -100
220     PICTUREPART = PICTUREPART + 1
230     EACHPOINT = 0
```

Program 7–5 (cont.)

```
240     WHILE XD => 0            '-1 IS END OF POINTS FOR THIS PART
250             EACHPOINT = EACHPOINT + 1
260             X(PICTUREPART,EACHPOINT) = XD
270             Y(PICTUREPART,EACHPOINT) = YD
280             READ XD,YD
290     WEND
300     'STORE # OF POINTS IS PICTUREPART IN POINTCOUNT(PICTUREPART)
310     POINTCOUNT(PICTUREPART) = EACHPOINT
320     READ XD,YD
330 WEND
340     'DRAW PICTURE
350 DRAWCOLOR = 2
360 FOR K = 1 TO PICTUREPART
370     GOSUB 600
380 NEXT
390 RETURN
400     '******************** PRINT INSTRUCTIONS **********************
410 LOCATE 21,1: PRINT "1-FRONT 2-REAR 3-ALL 0-STOP";
420 LOCATE 22,1: INPUT "PICTURE PART TO SCALE"; CHOICE
430 LOCATE 21,1: PRINT STRING$(80," ");          'ERASE 2 LINES
440 RETURN
450     '******************** HOW MUCH TO SCALE? ******************************
460 LOCATE 21,1: INPUT "H AND V AMOUNTS TO SCALE"; HS,VS
470 LOCATE 21,1: PRINT STRING$(40," ");
480 RETURN
490     '******** RECALCULATE POINTS FOR APPROPRIATE PICTURE PART ********
500 ON CHOICE GOTO 510, 520, 530
510 XFIXED = X(1,2): YFIXED = Y(1,2): GOTO 540
520 XFIXED = X(2,2): YFIXED = Y(2,2): GOTO 540
530 XFIXED = X(1,7): YFIXED = Y(1,7): GOTO 560
540 SCALEPART = CHOICE: GOSUB 660       'DO SCALING
550 GOTO 590
560 FOR SCALEPART = 1 TO PICTUREPART             'SCALE EACH PART
570     GOSUB 660
580 NEXT
590 RETURN
600 '############################ DRAW ROUTINE ################################
610 FOR J = 1 TO POINTCOUNT(K) - 1
620         IF X(K,J) < 0 OR X(K,J) > 319 OR Y(K,J) < 0 OR Y(K,J) > 199
              THEN 880
630         LINE (X(K,J),Y(K,J)) - (X(K,J+1),Y(K,J+1)),DRAWCOLOR
640 NEXT
650 RETURN
660 '########################## SCALING ROUTINE ###############################
670 FOR J = 1 TO POINTCOUNT(SCALEPART)
680     X(SCALEPART,J) = X(SCALEPART,J) * HS + XFIXED * (1 - HS)
690     Y(SCALEPART,J) = Y(SCALEPART,J) * VS + YFIXED * (1 - VS)
700 NEXT
710 RETURN
720 '##############################################################################
730     'FRONT OF CAR
740 DATA 220,90,160,90,160,60,205,60,187,35,160,35
750 DATA 160,30,190,30,210,60,250,65,260,90,250,90
760 DATA 250,100,240,110,230,110,220,100,220,90
770 DATA 230,80,240,80,250,90
780 DATA -1,-1
790     'BACK OF CAR
800 DATA 120,90,160,90,160,60,120,60,135,35,160,35
810 DATA 160,30,130,30,115,60,80,60,75,90,90,90
820 DATA 90,100,100,110,110,110,120,100,120,90,110,80,100,80,90,90,90,100
```

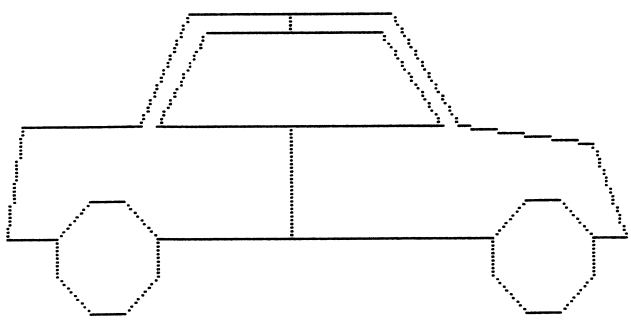
Program 7–5 (cont.)

```
830 DATA -1,-1
840 DATA -100,-100
850 '##########################################################################
860 IF INKEY$ = "" THEN 860
870 GOTO 890
880 PRINT "COORDINATE OUT OF RANGE"
890 END
```

definition and scaling factors, HS and VS. The program then scales the total picture or any selected component of the picture. A sample output is shown in Fig. 7–3.

Graphs are scaled using the same methods as those illustrated in Prog. 7–5. Since labeling cannot be scaled (the size of a character is fixed), we need to consider the placement of labels after scaling the pixel lengths. We can do this manually by examining the graph after scaling, or we can calculate new starting positions for the strings relative to some point on the graph. As an alternative, we could create labels with pixels. Then the labels could be scaled together with all other parts of the graph.

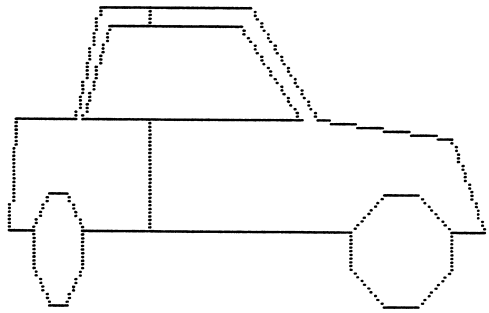
Figure 7–3 Scaling picture components from the original (a), with a reduction of the rear section (b) and an overall reduction (c), by Prog. 7–5.



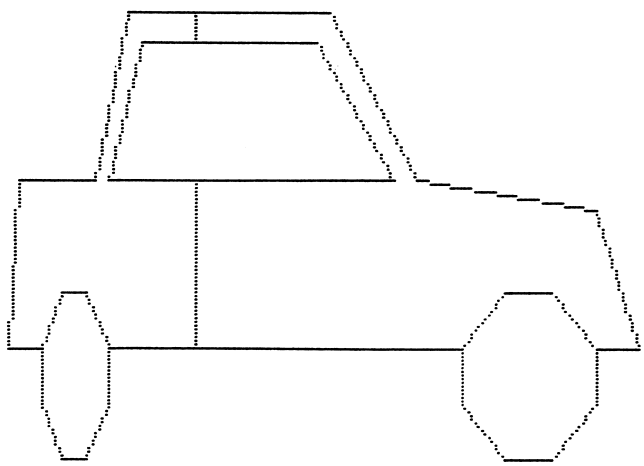
```
1-FRONT 2-REAR 3-ALL 0-STOP
PICTURE PART TO SCALE? ■
```

INTERACTIVE SCALING

Repeated scaling of a picture is accomplished by Prog. 7–5 with input from the keyboard. Menus could be devised for this program that would let us choose scaling factors with a light pen. We could do this with a menu that lists numbers, but this would then restrict scaling factors to only those numbers in the list. Another type of menu we could use for this purpose is a straight line that represents the numerical values from 0 to 20, say. In our program, we would then



```
1-FRONT 2-REAR 3-ALL 0-STOP
PICTURE PART TO SCALE? ■
```



```
1-FRONT 2-REAR 3-ALL 0-STOP
PICTURE PART TO SCALE? ■
```

Figure 7–3 (cont.)

write a routine to convert coordinate values selected along this line to numbers between 0 and 20. Touching the pen to a position near the left end of the line means that we want the scaling factor to be less than 1 (a reduction in size). Touching the pen to the other end sets the scaling factor to 20. For other positions we get intermediate values. Halfway along the line selects the number 10, one-

fourth of the distance from the low end selects 5, and one-third of the distance from the low end selects a scaling factor of 6.67. We could also set up two lines: one for enlargements and the other for reductions.

An interactive scaling method is demonstrated in Prog. 7–6, which is a

Program 7–6 Interactive scaling with a light pen.

```
10 'PROGRAM 7-6. SCALING A PICTURE WITH THE LIGHT PEN
20     'DRAWS CAR AND ALLOWS USER TO SCALE PART OF ALL OF
30     'CAR. SCALED POINTS REPLACE ORIGINAL VALUES IN X AND Y
40     'X AND Y.
50  DIM X(5,100), Y(5,100), POINTCOUNT(5)
60  SCREEN 1: COLOR 1,0: CLS
70      '************** CONTROL PROGRAM FLOW **********************
80  GOSUB 210                    'READ PICTURE PARTS
90  GOSUB 440                    'MAKE MENUS
100 GOSUB 560                    'GET CHOICE OF WHICH PART TO SCALE
110 WHILE CHOICE <> 0
120     GOSUB 700                'HOW MUCH TO SCALE?
130     DRAWCOLOR = 0
140     IF CHOICE <> 3 THEN K = CHOICE: GOSUB 1020
            ELSE FOR K = 1 TO PICTUREPART: GOSUB 1020: NEXT
150     GOSUB 910                'RECALCULATE SCALED POINTS
160     DRAWCOLOR = 2
170     IF CHOICE <> 3 THEN K = CHOICE: GOSUB 1020
            ELSE FOR K = 1 TO PICTUREPART: GOSUB 1020: NEXT
180     GOSUB 560                'GET CHOICE OF PART TO SCALE (OR QUIT)
190 WEND
200 GOTO 1280
210     '*************** READ PICTURE PARTS AND DRAW ***************
220 PICTUREPART = 0
230 READ XD,YD
240 WHILE XD <> -100
250     PICTUREPART = PICTUREPART + 1
260     EACHPOINT = 0
270     WHILE XD => 0            '-1 IS END OF POINTS FOR THIS PART
280             EACHPOINT = EACHPOINT + 1
290             X(PICTUREPART,EACHPOINT) = XD
300             Y(PICTUREPART,EACHPOINT) = YD
310             READ XD,YD
320     WEND
330     'STORE # OF POINTS IS PICTUREPART IN POINTCOUNT(PICTUREPART)
340     POINTCOUNT(PICTUREPART) = EACHPOINT
350     READ XD,YD
360 WEND
370     'DRAW PICTURE
380 DRAWCOLOR = 2
390 FOR K = 1 TO PICTUREPART
400     GOSUB 1020
410 NEXT
420 RETURN
430     '**************** MAKE MENUS FOR LIGHT PEN SELECTION **********
440 LOCATE 18,1: PRINT "PART -";
450 LINE (3,149) - (53,161),1,B: LOCATE 20,2: PRINT "FRONT";
460 LINE (3,165) - (53,177),1,B: LOCATE 22,2: PRINT "REAR";
470 LINE (3,181) - (53,193),1,B: LOCATE 24,3: PRINT "ALL";
480 LOCATE 18,10: PRINT "SCALING FACTOR - "
490 LINE (75,193) - (315,199),1,BF
500 LOCATE 20,10: PRINT ".1            .5            .9";
510 LINE (75,161) - (315,167),1,BF
```

Program 7–6 (cont.)

```
520 LOCATE 24,11: PRINT "1              2              3";
530 LINE (272,2) - (319,20),1,BF
540 LINE (278,6) - (313,16),0,BF: LOCATE 2,36: PRINT "STOP";
550 RETURN
560     '***************** READ PART TO SCALE FROM PEN ********************
570 PEN ON
580 WHILE PEN(0) <> -1          'WAIT FOR PEN TO BE ACTIVATED
590     LOCATE 18,1: PRINT "      ";: FOR DELAY = 1 TO 80: NEXT
600     LOCATE 18,1: PRINT "PART -";: FOR DELAY = 1 TO 80: NEXT
610 WEND
620 X = PEN(1): Y = PEN(2)
630 PEN OFF
640     'HAVE WE HIT THE STOP BOX; OR, IS THE PEN READING INVALID?
650 IF X > 270 AND Y < 25 THEN CHOICE = 0
      ELSE IF X > 53 OR Y < 149 THEN 570
660 IF Y > 179 THEN CHOICE = 3: GOTO 690
670 IF Y > 163 THEN CHOICE = 2: GOTO 690
680 IF Y > 149 THEN CHOICE = 1
690 RETURN
700     '********************* HOW MUCH TO SCALE? ************************
710 PEN ON
720 WHILE PEN(0) <> -1          'WAIT UNTIL PEN IS ACTIVATED
730     LOCATE 18,10:PRINT "                    ";:FOR D=1 TO 100:NEXT
740     LOCATE 18,10:PRINT "SCALING - HORIZONTAL";:FOR D=1 TO 100:NEXT
750 WEND
760 X = PEN(1): Y = PEN(2)
770 PEN OFF
780 IF X < 75 OR Y < 160 THEN 710       'INVALID READ - TRY AGAIN
790 IF Y < 188 THEN HS = (X - 75) * (.8 / 240) + .1
              ELSE HS = (X - 75) * (2 / 240) + 1
800     'READ VERTICAL SCALING FACTOR
810 PEN ON
820 WHILE PEN(0) <> -1          'WAIT UNTIL PEN IS ACTIVATED
830     LOCATE 18,10:PRINT "                    ";:FOR D=1 TO 100:NEXT
840     LOCATE 18,10:PRINT "SCALING - VERTICAL  ";:FOR D=1 TO 100:NEXT
850 WEND
860 X = PEN(1): Y = PEN(2)
870 PEN OFF
880 IF X < 75 OR Y < 160 THEN 810       'INVALID READ - TRY AGAIN
890 IF Y < 188 THEN VS = (X - 75) * (.8 / 240) + .1
              ELSE VS = (X - 75) * (2 / 240) + 1
900 RETURN
910     '******** RECALCULATE POINTS FOR APPROPRIATE PICTURE PART ********
920 ON CHOICE GOTO 930, 940, 950
930 XFIXED = X(1,2): YFIXED = Y(1,2): GOTO 960
940 XFIXED = X(2,2): YFIXED = Y(2,2): GOTO 960
950 XFIXED = X(1,7): YFIXED = Y(1,7): GOTO 980
960 SCALEPART = CHOICE: GOSUB 1080      'DO SCALING
970 GOTO 1010
980 FOR SCALEPART = 1 TO PICTUREPART            'SCALE EACH PART
990     GOSUB 1080
1000 NEXT
1010 RETURN
1020 '############################ DRAW ROUTINE ##############################
1030 FOR J = 1 TO POINTCOUNT(K) - 1
1040         IF X(K,J) < 0 OR X(K,J) > 319 OR Y(K,J) < 0 OR Y(K,J) > 199
                THEN 1300
1050         LINE (X(K,J),Y(K,J)) - (X(K,J+1),Y(K,J+1)),DRAWCOLOR
1060 NEXT
1070 RETURN
1080 '########################## SCALING ROUTINE ############################
```

Program 7–6 (cont.)

```
1090 FOR J = 1 TO POINTCOUNT(SCALEPART)
1100     X(SCALEPART,J) = X(SCALEPART,J) * HS + XFIXED * (1 - HS)
1110     Y(SCALEPART,J) = Y(SCALEPART,J) * VS + YFIXED * (1 - VS)
1120 NEXT
1130 RETURN
1140 '*************************************************************************
1150     'FRONT OF CAR
1160 DATA 220,90,160,90,160,60,205,60,187,35,160,35
1170 DATA 160,30,190,30,210,60,250,65,260,90,250,90
1180 DATA 250,100,240,110,230,110,220,100,220,90
1190 DATA 230,80,240,80,250,90
1200 DATA -1,-1
1210     'BACK OF CAR
1220 DATA 120,90,160,90,160,60,120,60,135,35,160,35
1230 DATA 160,30,130,30,115,60,80,60,75,90,90,90
1240 DATA 90,100,100,110,110,110,120,100,120,90,110,80,100,80,90,90,90,100
1250 DATA -1,-1
1260 DATA -100,-100
1270 '*************************************************************************
1280 IF INKEY$ = "" THEN 1280
1290 GOTO 1310
1300 PRINT "COORDINATE OUT OF RANGE"
1310 END
```

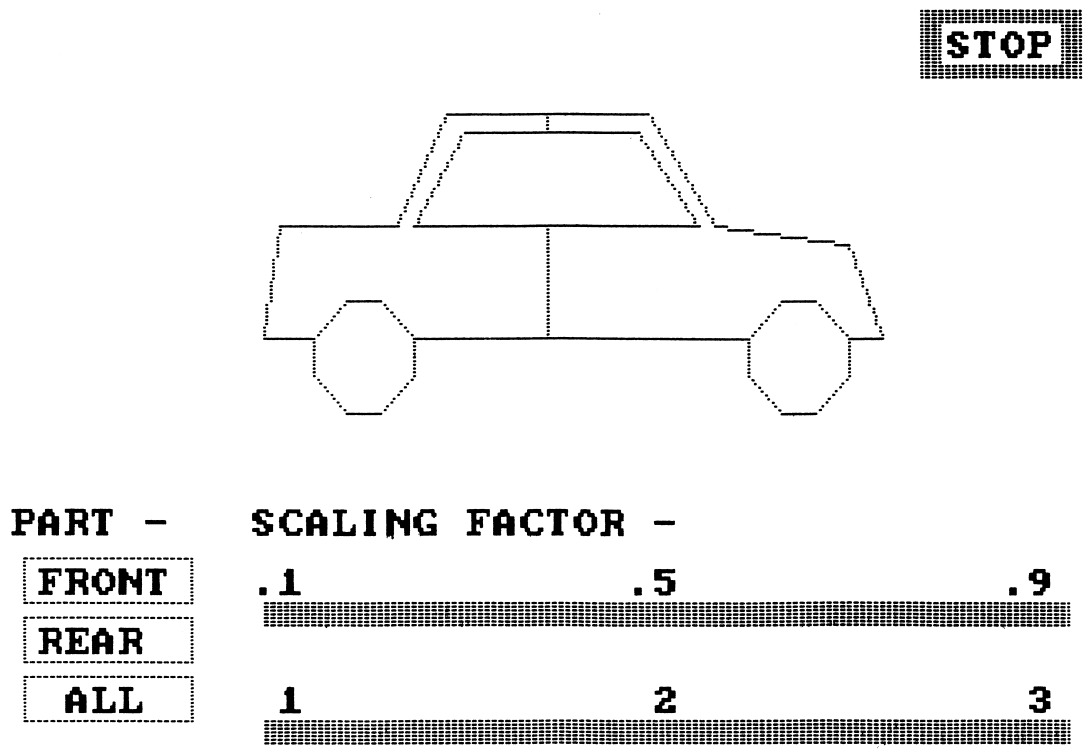


Figure 7–4 Scaling menu used by Prog. 7–6 for light pen input.

Figure A

Figure B

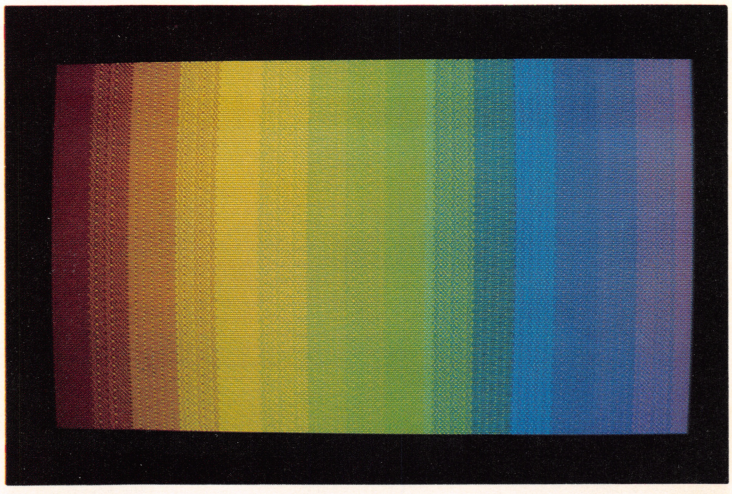


Figure C

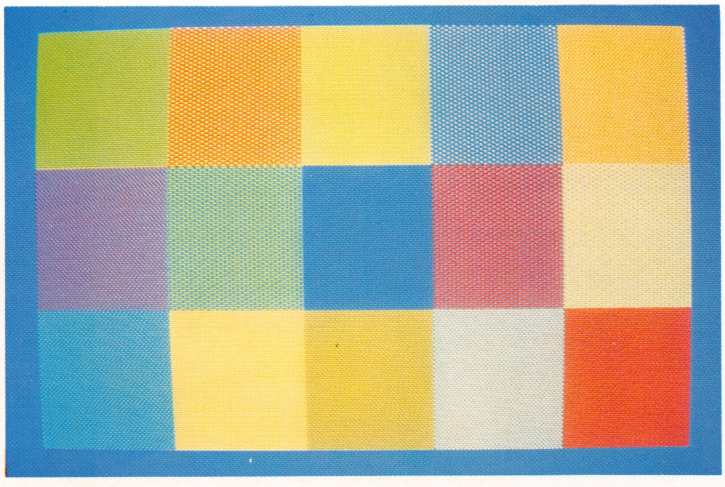


Figure D

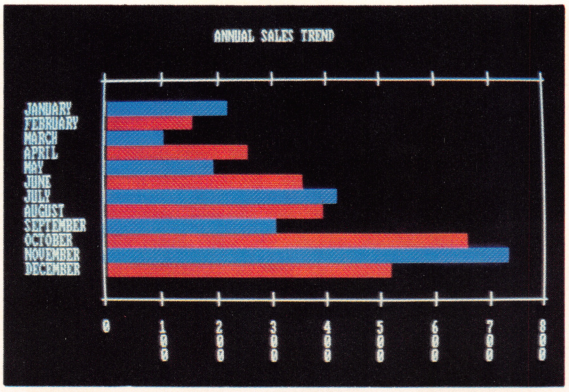


Figure E

Figure F

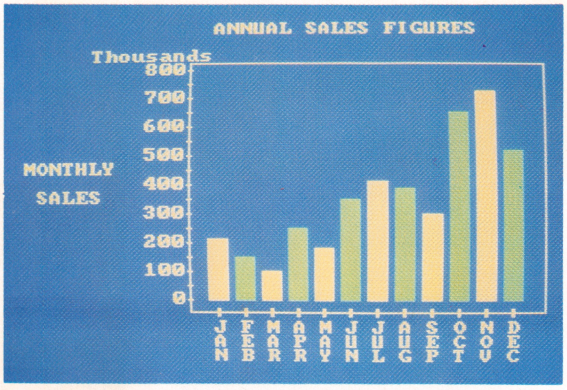


Figure G

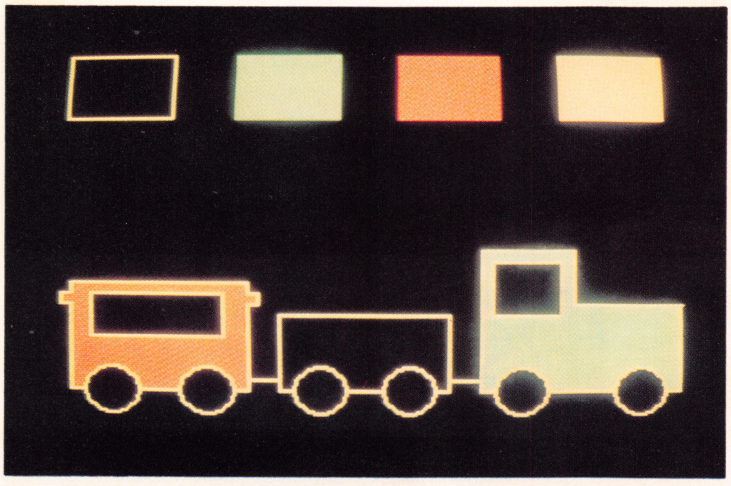
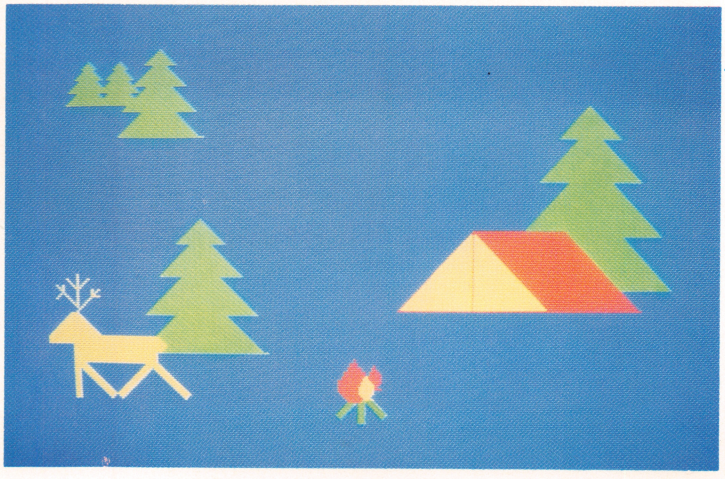


Figure H



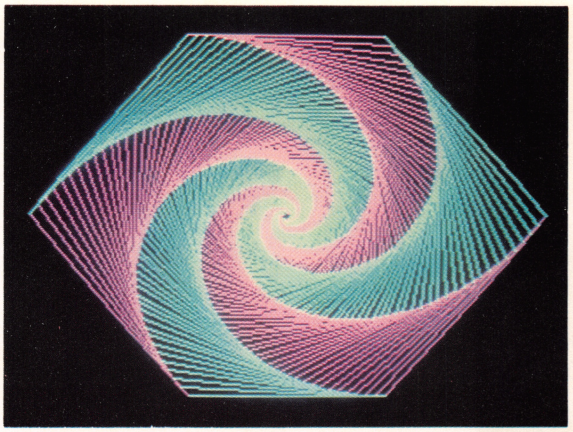


Figure I

Figure J

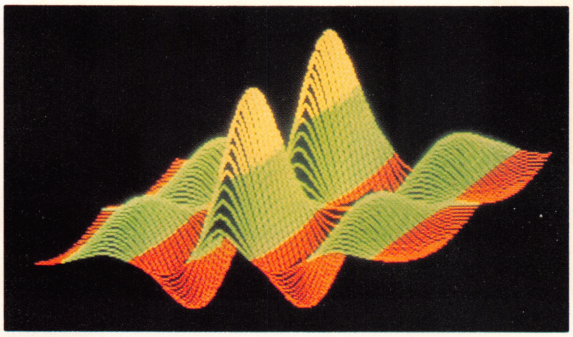


Figure K

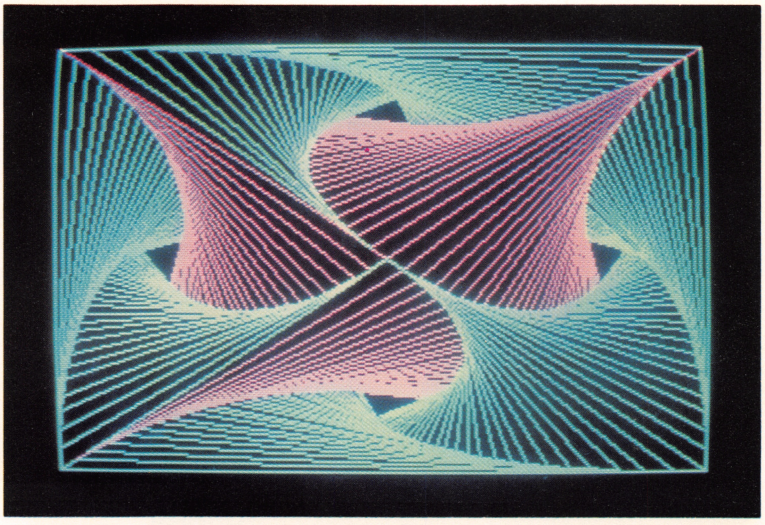


Figure L

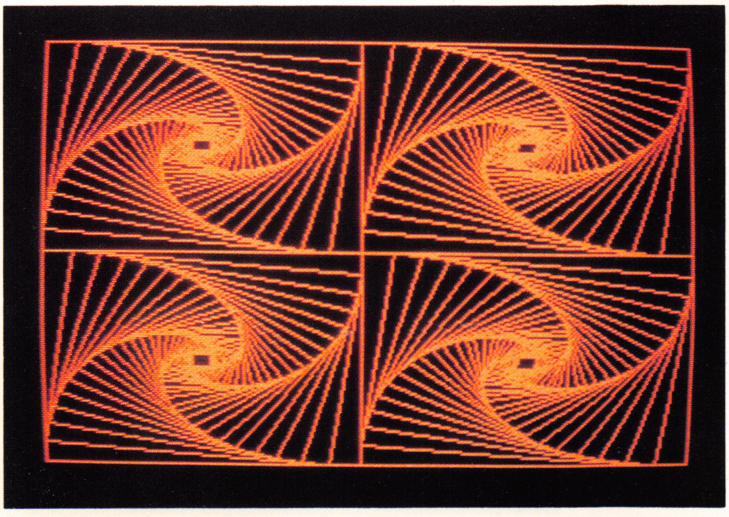


Figure M

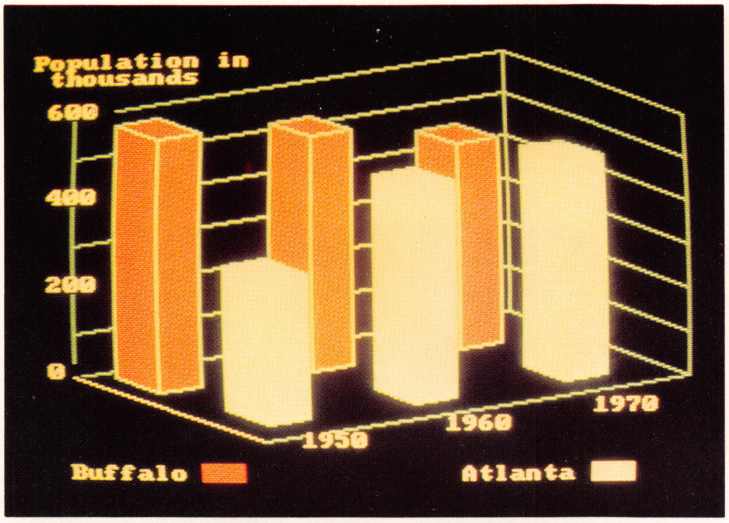


Figure N

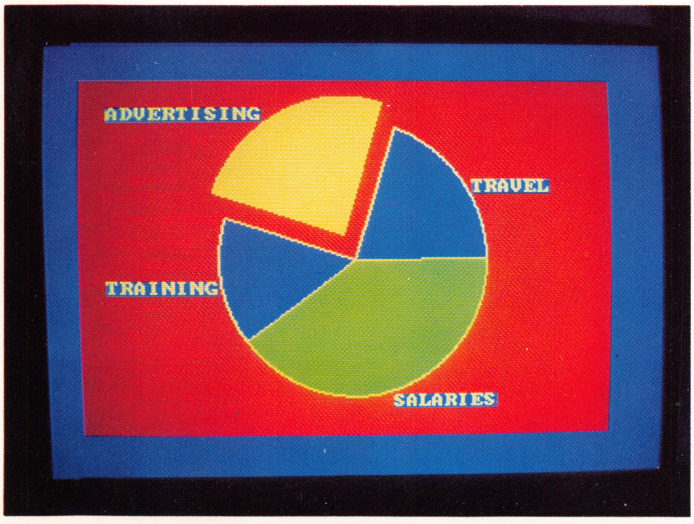


Figure O

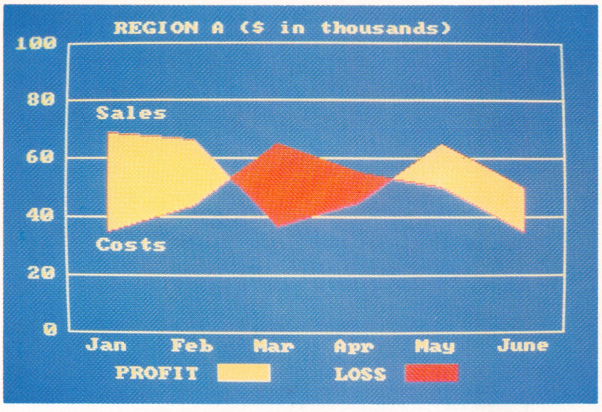
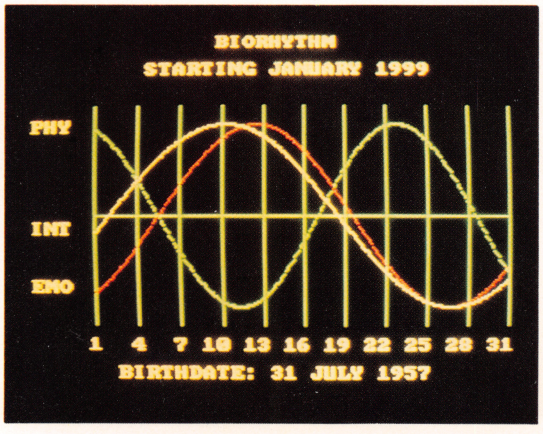


Figure P



modification of Prog. 7–5. The picture component to be scaled and the scaling factor are selected with a light pen. Figure 7–4 shows the screen presentation of the menus. A tablet or joystick could be used in a similar way to select screen positions.

DRAW STATEMENT SCALING

The S command used with the DRAW statement is another means for scaling picture parts. With this command, we can scale objects using scaling factors that are multiples of 1/4 up to a maximum of 63 (0.25, 0.5, 0.75, 1, 1.25, and so on). The scaling factor selected in this range is then multiplied by 4 and inserted with the S command into the picture string definition just before the commands that draw the object we want to scale. If we want to triple the size of an object, we put the command S12 into the DRAW statement just before the commands to actually draw that object.

Any S command inserted into a DRAW statement will scale all the picture parts following S. If we only want to change the size of one object, we put the command S4 immediately after the commands to draw that object. This will scale all the remaining objects in the picture by a factor of 1. The following DRAW statement scales one object within a display in this way:

```
DRAW "BM10,150;S12;R40;H20;G20;BM50,150;S4;R40;H20;G20"
```

Scaling command S12 enlarges the first triangle by a factor of 3, and command S4 sets the second triangle to normal size. These scaling commands scale each line drawn with respect to the last point referenced. For the command S12, the fixed point of the first line drawn is (10,150). The end of this horizontal line is the next fixed point (for the command H20). Similarly, position (50,150) is the first fixed point for the command S4.

Scaling with the DRAW statement is somewhat less flexible than using the scaling equations. We are restricted to scaling factors that are multiples of 1/4, starting with the value 1/4. Also, we have to remember to "undo" our scaling if subsequent objects are not to be scaled. To scale different parts of a single object differently takes some juggling. If we want different horizontal and vertical scaling factors, we either need to include multiple S commands for the object or we need to rearrange the drawing sequence. In either case, this can get messy. So we are better off drawing and scaling objects using other methods when we want different scale factors for different parts of individual objects.

7–3 CHANGING ORIENTATIONS (ROTATION)

In many applications, we would like to be able to change the orientation of a display. We may decide to change a bar graph so that the bars are drawn horizontally instead of vertically. Rotating the display by 90 degrees can be a

convenient technique for accomplishing this without having to reconstruct the graph. In simulation and game playing applications we often want to display rotating objects.

ROTATING A POINT

Figure 7–5 illustrates rotation of a point. The **rotation path** is along a circular arc from position (X,Y) to position (XR,YR) about a **pivot point** (XO,YO). Angle A in this figure specifies the amount of rotation from (X,Y) to (XR,YR). The rotated coordinates (XR,YR) are calculated from the values of A, (XO,YO), and (X,Y) as

$$\begin{aligned} XR &= XO + ((X - XO) * COS(A)) + ((Y - YO) * SIN(A)) \\ YR &= YO + ((Y - YO) * COS(A)) - ((X - XO) * SIN(A)) \end{aligned} \qquad (7\text{–}8)$$

Coordinate values for the pivot point (XO,YO) can be chosen to be at any convenient location—either on the screen or beyond the screen boundaries in any direction. This point is not plotted and is only chosen as a reference to determine the circular path of rotation, as shown in Fig. 7–6.

Rotation angle A is measured in a counterclockwise direction from the starting position (X,Y) of the point. This angle usually will be given a value between zero radians and 2 * PI (6.283185) radians. Other angles may be specified, but they simply repeat the rotations in this range.

The distance that the point moves along the circle path for a specified angle A depends on the distance of (X,Y) from the pivot point (XO,YO). The farther (XO,YO) is from (X,Y), the greater the distance traveled from (X,Y) to (XR,YR). Figure 7–7 shows the relationship between the pivot distance R, the angle A, and the displacement D from (X,Y) to (XR,YR). For small rotation angles, D is approximately equal to the product R * A. If R is 50, then a point could be rotated a distance of about 1 unit (to a neighboring pixel location) with an angle A of 0.02 radian.

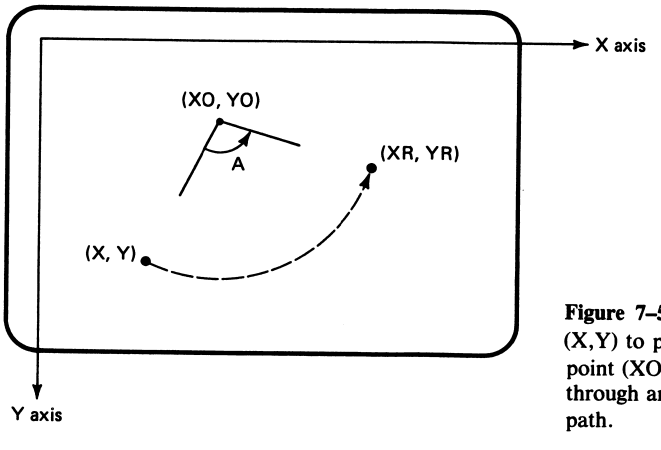


Figure 7–5 Rotating a point at position (X,Y) to position (XR,YR) about the point (XO,YO). The point rotates through an angle A along a circular path.

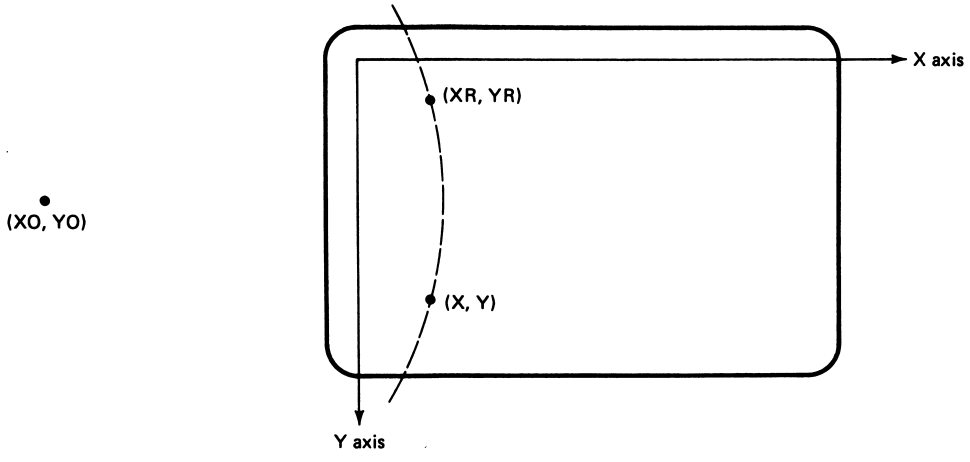


Figure 7–6 Values for the coordinates of the pivot point (XO,YO) can be chosen to be off screen, as well as within the screen coordinate boundaries.

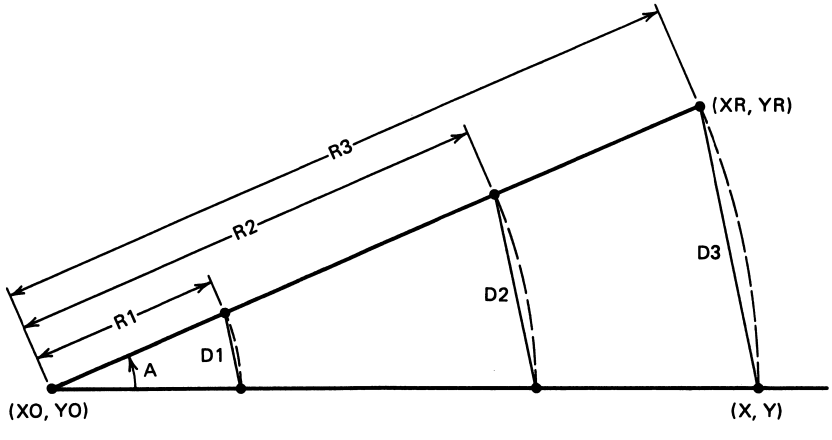


Figure 7–7 For small values of the rotation angle A, the distance D, from (X,Y) to (XR,YR), is approximately equal to the pivot distance R times rotation angle A. As R gets larger, D gets larger, for the same rotation angle A.

ROTATING DISPLAYS

To change the orientation of a displayed object, we first select the pivot point. Then the rotated coordinate positions of all points in the object are calculated with relations (7–8). Finally, the screen is cleared and the object lines are redrawn using the rotated coordinates.

Program 7–7 rotates a picture about any selected pivot point (XO,YO) through any specified rotation angle A, provided that the rotated position is on the

screen. Output of the program for rotation angles of 90 degrees and 180 degrees are shown in Fig. 7–8. The pivot point for rotation in this example was chosen at the center of the figure.

Rotating a display can cause the picture or graph to be distorted if the X resolution of the system is different from the Y resolution. We adjust for

Program 7–7 Rotation of a picture (clown).

```
10 'PROGRAM 7-7. ROTATING A PICTURE.
20     'DRAWS PICTURE AND ALLOWS USER TO INPUT THE ANGLE OF
30     'DESIRED ROTATION. ROTATED POINTS ARE STORED IN XR AND YR.
40  DIM X(7,100), Y(7,100), POINTCOUNT(7)
50  DIM XR(7,100), YR(7,100), POINTCOUNTR(7,100)
60  SCREEN 1: COLOR 1,0: CLS
70      '**************** CONTROL PROGRAM FLOW ************************
80  GOSUB 180              'READ PICTURE PARTS
90  GOSUB 580              'DRAW
100 GOSUB 430              'GET ANGLE TO ROTATE
110 WHILE ANGLE <> -1      'ENTER -1 TO QUIT
120       GOSUB 490        'GET POINT ABOUT WHICH TO ROTATE
130       GOSUB 520        'RECALCULATE POINTS
140       GOSUB 580        'ERASE AND DRAW IN NEW ORIENTATION
150       GOSUB 430
160 WEND
170 GOTO 910
180     '***************** READ PICTURE PARTS *********************
190 PICTUREPART = 0
200 READ XD,YD
210 WHILE XD <> -100
220     PICTUREPART = PICTUREPART + 1
230     EACHPOINT = 0
240     WHILE XD => 0                              '-1 IS END OF POINTS FOR PART
250             EACHPOINT = EACHPOINT + 1
260             X(PICTUREPART,EACHPOINT) = XD
270             Y(PICTUREPART,EACHPOINT) = YD * 5/6      'RESOLUTION CORRECTION
280             READ XD,YD
290     WEND
300     'STORE # OF POINTS IN PICTUREPART IN POINTCOUNT(PICTUREPART)
310     POINTCOUNT(PICTUREPART) = EACHPOINT
320     READ XD,YD
330 WEND
340     'COPY ORIGINAL VALUES TO ARRAYS USED TO HOLD ROTATED
350     'POINTS, SINCE DRAW ROUTINE USES THESE ARRAYS
360 FOR EACHPART = 1 TO PICTUREPART
370     FOR EACHPOINT = 1 TO POINTCOUNT(EACHPART)
380         XR(EACHPART,EACHPOINT) = X(EACHPART,EACHPOINT)
390         YR(EACHPART,EACHPOINT) = Y(EACHPART,EACHPOINT)
400     NEXT
410 NEXT
420 RETURN
430     '******************** PRINT INSTRUCTIONS ********************
440 LOCATE 22,1: PRINT " ANGLE TO ROTATE FROM ORIGINAL POSITION";
450 LOCATE 23,1: INPUT " (0-360, OR -1 TO END)"; ANGLE
460 LOCATE 22,1: PRINT STRING$(80," ");
470 RETURN
480     '************* GET POINT ABOUT WHICH TO ROTATE ********************
490 LOCATE 21,1
500 INPUT " ABOUT WHAT POINT"; XO,YO
510 RETURN
520     '****************** RECALCULATE POINTS ****************************
```

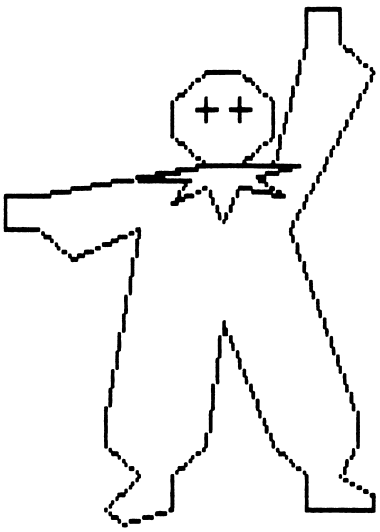
Program 7–7 (cont.)

```
530 ANGLE = ANGLE * 3.14159 / 180     'CHANGE DEGREES TO RADIANS
540 FOR P = 1 TO PICTUREPART             'ROTATE EACH PART
550     GOSUB 670
560 NEXT
570 RETURN
580 '***************************** DRAW ROUTINE *********************************
590 CLS
600 FOR P = 1 TO PICTUREPART
610     FOR E = 1 TO POINTCOUNT(P) - 1
620         IF XR(P,E) < 0 OR XR(P,E) > 319 OR YR(P,E) < 0 OR YR(P,E) > 199
               THEN 890
630         LINE (XR(P,E),YR(P,E)) - (XR(P,E+1),YR(P,E+1))
640     NEXT
650 NEXT
660 RETURN
670 '***************************** ROTATE ROUTINE *******************************
680 FOR E = 1 TO POINTCOUNT(P)
690     XR(P,E) = XO + (X(P,E)-XO)*COS(ANGLE) + (Y(P,E)-YO)*SIN(ANGLE)*6/5
700     YR(P,E) = YO + (Y(P,E)-YO)*COS(ANGLE) - (X(P,E)-XO)*SIN(ANGLE)*5/6
710 NEXT
720 RETURN
730 '************************************************************************
740 DATA 160,60,170,50,170,40,160,30,150,30,140,40,140,50
750 DATA 150,60,178,60,165,63,173,70,160,67,155,78,150,67
760 DATA 140,72,145,65,130,65,150,60,100,70,90,70,90,80,100
770 DATA 80,110,90,130,80,120,140,120,150,130,160,120,170,125
780 DATA 175,140,170,140,160,150,150,155,110,170,150,180,160,180
790 DATA 170,200,170,200,165,190,160,195,150,195,140,175,80,200
800 DATA 30,190,20,190,10,180,10,180,20,170,60
810 DATA -1,-1
820 DATA 147,42,153,42,-1,-1
830 DATA 150,38,150,45,-1,-1
840 DATA 157,42,163,42,-1,-1
850 DATA 160,38,160,45
860 DATA -1,-1
870 DATA -100,-100
880 '************************************************************************
890 PRINT "COORDINATE OUT OF RANGE";
900 GOTO 920
910 IF INKEY$ = "" THEN 910
920 END
```

resolution differences in Prog. 7–7 by multiplying the term (Y − YO) * SIN(A) on line 690 by the ratio of X resolution to Y resolution and multiplying the term (X − XO) * SIN(A) on line 700 by the ratio of Y resolution to X resolution. This is equivalent to rotating the display along an elliptical path, which looks circular on the screen.

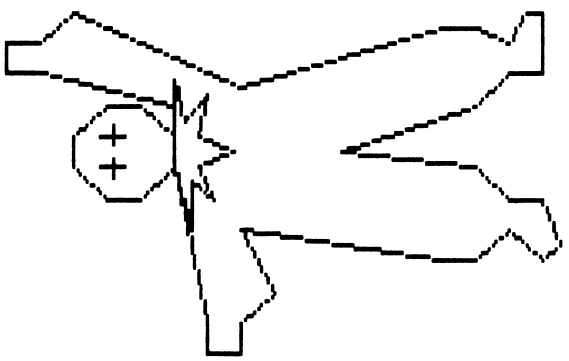
INTERACTIVE ROTATIONS

We can organize programs to select rotation transformation parameters with a light pen, tablet, or joystick in much the same way that we did with scaling. A menu consisting of a straight line can be used to represent angles from 0 to 2 * PI radians. Coordinate values selected along the line by a light pen, say, are then



ANGLE TO ROTATE FROM ORIGINAL POSITION
(0-360, OR -1 TO END)? ■

(a)

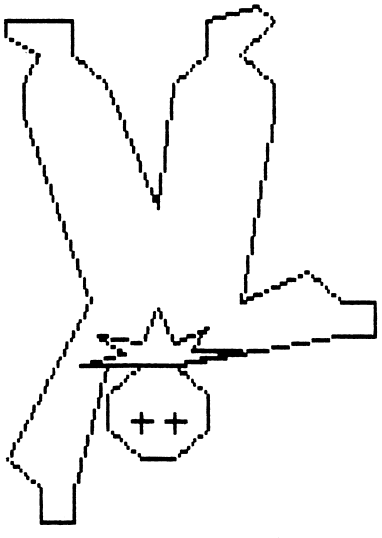


ANGLE TO ROTATE FROM ORIGINAL POSITION
(0-360, OR -1 TO END)? ■

(b)

Figure 7–8 Rotating a picture from the original position (a) through 90 degrees (b) and through 180 degrees (c) by Prog. 7–7.

Figure 7–8 (cont.)



```
ANGLE TO ROTATE FROM ORIGINAL POSITION
(0-360, OR -1 TO END)? ■
```

(c)

interpreted as angles within this range. We can select the objects to be rotated and the pivot points by pointing the pen to positions on the screen. In Section 7–4 we will consider how we might incorporate interactive methods for rotation into our programs.

DRAW STATEMENT ROTATIONS

Objects produced by a DRAW statement can be rotated with the A command. We have three rotation angle options. The command A1 gives us a rotation angle of 90 degrees (PI/2 radians), A2 provides an angle of 180 degrees (PI radians), and A3 selects 270 degrees (3 * PI/2 radians). We also have the command A0, which sets the rotation angle at 0 degrees. This allows us to rotate only selected parts of a picture, by canceling previous rotation specifications. To rotate only one object in a picture, we specify the appropriate rotation command before we draw that object, then specify the command A0 for the rest of the objects in the picture. Pivot points for rotation are taken as the last points referenced before a line is

drawn. For example,

```
DRAW "BM100,100;A2;R50;H25;G25"
```

rotates the triangle through 180 degrees about the point (100,100).

A rotation command affects all subsequent DRAW commands up to the next rotation command. Each of the rotation commands takes the last point referenced as the pivot point for that rotation. In addition, rotation angles of 90 degrees and 270 degrees are automatically adjusted for resolution differences, assuming an aspect ratio of 3/4. If we want other aspect ratios or rotation angles different from 90, 180, or 270 degrees, we use the rotation equations (7–8).

7–4 COMBINED TRANSFORMATIONS AND PICTURE CONSTRUCTION

A general interactive transformation program that combines translation, scaling, and rotation is given in Prog. 7–8. The program illustrates a method for setting up interactive menus for the three transformations. It also shows how we can use

Program 7–8 Interactive picture construction using a shape menu and keyboard input.

```
10 'PROGRAM 7-8. BUILDING PICTURES FROM COMPONENT PARTS.
20      'DISPLAYS MENUS ON SCREEN LISTING CHOICES FOR
30      'SHAPES AND TRANSFORMATIONS.  EACH TIME A
40      'SHAPE IS SELECTED, ANY NUMBER AND COMBINATION
50      'OF TRANSFORMATIONS CAN BE APPLIED.
60      '*****************************************************************
70 SCREEN 2: CLS
80 YA = 5/12              'YA AND XA ARE RESOLUTION ADJUSTMENTS
90 XA = 12/5
100 GOSUB 460             'MAKE MENUS
110 AA = 0        'AA IS ACCUMULATED ANGLE THAT A CIRCLE HAS BEEN ROTATED
120      '************************** INPUT CHOICES **********************
130 LOCATE 1,1: PRINT "        ";: FOR DELAY=1 TO 300: NEXT
140 LOCATE 1,1: PRINT "SELECT:";: FOR DELAY=1 TO 300: NEXT
150 A$ = INKEY$           'A$ IS CHOICE OF SHAPE
160 IF A$ = "" OR A$ < "1" OR A$ > "4" THEN 130          'INVALID CHOICE
170 RO = VAL(A$) * 4
180 N = 0                 'N INDICATES WHETHER CURRENT DISPLAY OF SHAPE
190 LOCATE 23,1: PRINT "SELECT:";           'SHOULD BE ERASED OR NOT
200 LOCATE RO,1: PRINT " ";: FOR DELAY=1 TO 300: NEXT
210 LOCATE 23,1: PRINT "        ";
220 LOCATE RO,1: PRINT A$: FOR DELAY=1 TO 300: NEXT
230 B$ = INKEY$           'B$ IS CHOICE OF TRANSFORMATION
240 IF B$ = "" OR B$<>"T" AND B$<>"S" AND B$<>"R" AND B$<>"E" AND
        B$<>"N" AND B$<>"Q" THEN 190
250 IF B$ = "Q" THEN 2310
260 IF B$ <> "N" THEN 310
270 RESTORE                       'GO BACK TO BEGINNING OF DATA
280 GOSUB 600                     'READ DATA POINTS
290 GOTO 110
300     '******** ERASING? ********
310 IF B$ = "E" THEN C = 0: ON VAL(A$) GOSUB 680, 710, 770, 990: RESTORE:
            GOSUB 600 : GOTO 110
320     '****************************
330 LOCATE 23,1: PRINT "        ";        'ERASE MENU TO MAKE ROOM FOR OTHER
```

Program 7–8 (cont.)

```
340 LOCATE 24,1: PRINT STRING$(79," ");          'TRANSFORMATIONS INSTRUCTIONS
350     '****** TRANSLATING? ******
360 IF B$ = "T" THEN GOSUB 1310: C = 0: ON VAL(A$) GOSUB 1050, 1110, 1190, 1240
370     '******** SCALING? ********
380 IF B$ = "S" THEN GOSUB 1670: C = 0: ON VAL(A$) GOSUB 1350, 1420, 1520, 1580
390     '******** ROTATING? ********
400 IF B$ = "R" THEN GOSUB 2220: C = 0: ON VAL(A$) GOSUB 1720, 1840, 2020, 2080
410     '************************************************
420 N = 1                   'FROM NOW ON, ERASE OLD PICTURE OF SHAPE
430 GOSUB 560               'REDISPLAY TRANSFORMATION MENU
440 GOTO 190
450 '
460 '####################### DISPLAY MENUS ########################
470 LOCATE 1,1: PRINT "SELECT:";
480 FOR K=1 TO 4:LOCATE K*4,1:PRINT RIGHT$(STR$(K),1);".";:NEXT
490 C = 3
500 GOSUB 600               'READ DATA POINTS
510 GOSUB 690               'DRAW LINE
520 GOSUB 720               'DRAW BOX
530 AA = 0
540 GOSUB 780               'DRAW CIRCLE
550 GOSUB 1000              'DRAW TRIANGLE
560 LOCATE 23,1: PRINT "SELECT:"; STRING$(50," ");
570 LOCATE 24,1: PRINT "T -TRANSLATE   S -SCALE   R -ROTATE";
580 LOCATE 24,39: PRINT "E -ERASE   N -NEXT OBJECT   Q -QUIT";
590 RETURN
600     '************ READ DATA POINTS ******************
610 READ XP,YP,XQ,YQ
620 READ XU,YU,XV,YV,XW,YW,XX,YX
630 READ XC,YC,RX,RY
640 READ XR,YR,XS,YS,XT,YT
650 RETURN
660 '
670 '####################### DRAW ROUTINES ########################
680     '*************** DRAW LINE ************
690 LINE (XP,YP) - (XQ,YQ),C
700 RETURN
710     '***************** DRAW BOX ************
720 LINE (XU,YU) - (XV,YV),C
730 LINE (XV,YV) - (XW,YW),C
740 LINE (XW,YW) - (XX,YX),C
750 LINE (XX,YX) - (XU,YU),C
760 RETURN
770     '************** DRAW CIRCLE **********
780 IF RX = RY THEN CIRCLE (XC,YC),RX,C: GOTO 980
790 IF RX < RY THEN R = RX ELSE R = RY
800 IF AA <> 0 THEN 890
810 FOR A = 0 TO 1.5708 STEP 1/R
820     DX = RX * COS(A): DY = RY * SIN(A)
830     PSET (XC+DX,YC+DY*YA),C
840     PSET (XC+DX,YC-DY*YA),C
850     PSET (XC-DX,YC+DY*YA),C
860     PSET (XC-DX,YC-DY*YA),C
870 NEXT
880 GOTO 980
890 CX = SIN(AA) * XA           'CALCULATE CONSTANT PART OF EQUATION
900 CY = SIN(AA) * YA
910 FOR A = 0 TO 3.14159 STEP 1/R
920     X = RX * COS(A): Y = RY * SIN(A) * YA
930     XH = X
940     X = X * COS(AA) + Y * CX: Y = Y * COS(AA) - XH * CY
```

Program 7–8 (cont.)

```
950     PSET (XC+X,YC+Y),C
960     PSET (XC-X,YC-Y),C
970 NEXT A
980 RETURN
990     '************ DRAW TRIANGLE ***********
1000 LINE (XR,YR) - (XS,YS),C: LINE - (XT,YT),C: LINE - (XR,YR),C
1010 RETURN
1020 '
1030 '############################## TRANSLATE ###################################
1040     '************ TRANSLATE LINE ***********
1050 IF N <> 0 THEN GOSUB 680   'IF N IS 0 SHAPE IS STILL IN MENU-DON'T ERASE
1060 XP = XP + HT: YP = YP + VT
1070 XQ = XQ + HT: YQ = YQ + VT
1080 C = 3: GOSUB 680                    'DRAW
1090 RETURN
1100     '************ TRANSLATE BOX ************
1110 IF N <> 0 THEN GOSUB 710
1120 XU = XU + HT: YU = YU + VT
1130 XV = XV + HT: YV = YV + VT
1140 XW = XW + HT: YW = YW + VT
1150 XX = XX + HT: YX = YX + VT
1160 C = 3: GOSUB 710                    'DRAW
1170 RETURN
1180     '********** TRANSLATE CIRCLE **********
1190 IF N <> 0 THEN GOSUB 770
1200 XC = XC + HT: YC = YC + VT
1210 C = 3: GOSUB 770                    'DRAW
1220 RETURN
1230     '********** TRANSLATE TRIANGLE *******
1240 IF N <> 0 THEN GOSUB 990
1250 XR = XR + HT: YR = YR + VT
1260 XS = XS + HT: YS = YS + VT
1270 XT = XT + HT: YT = YT + VT
1280 C = 3: GOSUB 990                    'DRAW
1290 RETURN
1300     '************ INSTRUCTIONS ************
1310 LOCATE 23,1: INPUT "HORIZONTAL AND VERTICAL DISTANCE TO TRANSLATE"; HT,VT
1320 RETURN
1330 '
1340 '############################## SCALING #####################################
1350     '************* SCALE LINE *************
1360 IF N <> 0 THEN GOSUB 680
1370 XF = (XP + XQ) / 2: YF = (YP + YQ) / 2
1380 XP = XP * HS + XF * (1 - HS): YP = YP * VS + YF * (1 - VS)
1390 XQ = XQ * HS + XF * (1 - HS): YQ = YQ * VS + YF * (1 - VS)
1400 C = 3: GOSUB 680             'DRAW
1410 RETURN
1420     '*************** SCALE BOX ************
1430 IF N <> 0 THEN GOSUB 710
1440 XF = (XU + XV + XW + XX) / 4
1450 YF = (YU + YV + YW + YX) / 4
1460 XU = XU * HS + XF * (1 - HS): YU = YU * VS + YF * (1 - VS)
1470 XV = XV * HS + XF * (1 - HS): YV = YV * VS + YF * (1 - VS)
1480 XW = XW * HS + XF * (1 - HS): YW = YW * VS + YF * (1 - VS)
1490 XX = XX * HS + XF * (1 - HS): YX = YX * VS + YF * (1 - VS)
1500 C = 3: GOSUB 710             'DRAW
1510 RETURN
1520     '************* SCALE CIRCLE ***********
1530 IF N <> 0 THEN GOSUB 770
1540 RX = RX * HS
```

Program 7–8 (cont.)

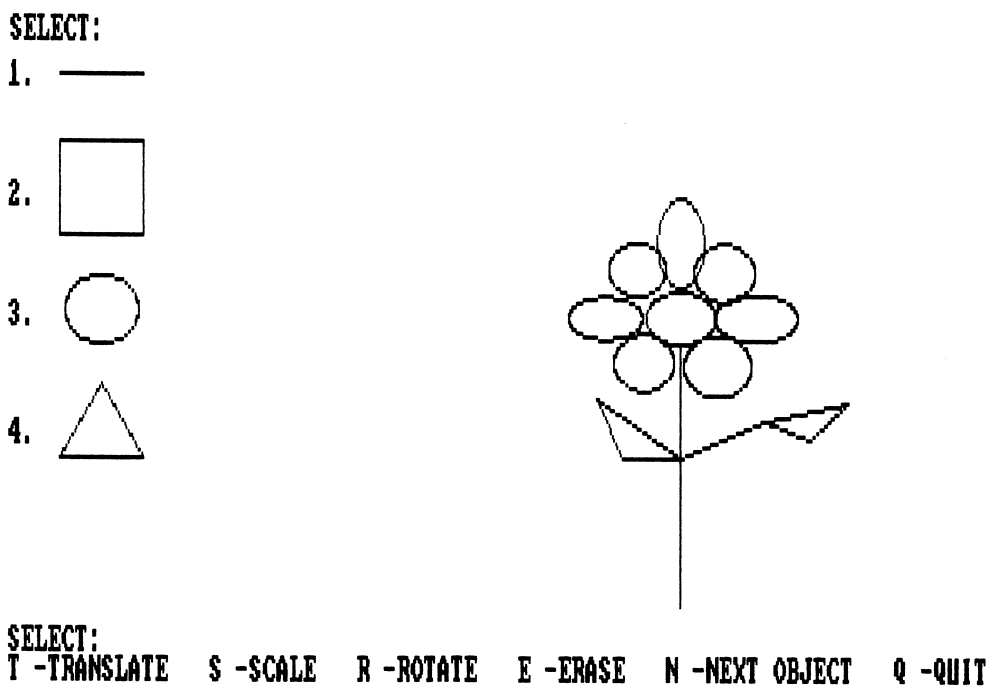
```
1550 RY = RY * VS
1560 C = 3: GOSUB 770          'DRAW
1570 RETURN
1580     '************ SCALE TRIANGLE *********
1590 IF N <> 0 THEN GOSUB 990
1600 XF = (XR + XS + XT) / 3
1610 YF = (YR + YS + YT) / 3
1620 XR = XR * HS + XF * (1 - HS): YR = YR * VS + YF * (1 - VS)
1630 XS = XS * HS + XF * (1 - HS): YS = YS * VS + YF * (1 - VS)
1640 XT = XT * HS + XF * (1 - HS): YT = YT * VS + YF * (1 - VS)
1650 C = 3: GOSUB 990          'DRAW
1660 RETURN
1670     '************ INSTRUCTIONS ***********
1680 LOCATE 23,1: INPUT "ENTER X AND Y SCALING FACTORS"; HS,VS
1690 RETURN
1700 '
1710 '############################## ROTATION ############################
1720     '************** ROTATE LINE ************
1730 IF N <> 0 THEN GOSUB 680
1740 XO = (XP + XQ) / 2
1750 YO = (YP + YQ) / 2
1760 XH = XP                    'HOLD VALUE OF XP FOR USE IN Y CALCULATION
1770 XP = XO + (XP - XO) * COS(AR) + (YP - YO) * SIN(AR) * XA
1780 YP = YO + (YP - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
1790 XH = XQ
1800 XQ = XO + (XQ - XO) * COS(AR) + (YQ - YO) * SIN(AR) * XA
1810 YQ = YO + (YQ - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
1820 C = 3: GOSUB 680           'DRAW
1830 RETURN
1840     '*********** ROTATE BOX ***************
1850 IF N <> 0 THEN GOSUB 710
1860 XO = (XU + XV + XW + XX) / 4
1870 YO = (YU + YV + YW + YX) / 4
1880 XH = XU                    'HOLD CURRENT VALUE OF XU
1890 XU = XO + (XU - XO) * COS(AR) + (YU - YO) * SIN(AR) * XA
1900 YU = YO + (YU - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
1910 XH = XV
1920 XV = XO + (XV - XO) * COS(AR) + (YV - YO) * SIN(AR) * XA
1930 YV = YO + (YV - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
1940 XH = XW
1950 XW = XO + (XW - XO) * COS(AR) + (YW - YO) * SIN(AR) * XA
1960 YW = YO + (YW - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
1970 XH = XX
1980 XX = XO + (XX - XO) * COS(AR) + (YX - YO) * SIN(AR) * XA
1990 YX = YO + (YX - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
2000 C = 3: GOSUB 710           'DRAW
2010 RETURN
2020     '************** ROTATE CIRCLE ******************
2030 IF N <> 0 THEN AA = AS: GOSUB 770
2040 AA = AR + AS
2050 C = 3: GOSUB 770           'DRAW
2060 AS = AA
2070 RETURN
2080     '*********** ROTATE TRIANGLE **********
2090 IF N <> 0 THEN GOSUB 990
2100 XO = (XR + XS + XT) / 3: YO = (YR + YS + YT) / 3
2110 XH = XR
2120 XR = XO + (XR - XO) * COS(AR) + (YR - YO) * SIN(AR) * XA
2130 YR = YO + (YR - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
2140 XH = XS
```

Program 7–8 (cont.)

```
2150 XS = XO + (XS - XO) * COS(AR) + (YS - YO) * SIN(AR) * XA
2160 YS = YO + (YS - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
2170 XH = XT
2180 XT = XO + (XT - XO) * COS(AR) + (YT - YO) * SIN(AR) * XA
2190 YT = YO + (YT - YO) * COS(AR) - (XH - XO) * SIN(AR) * YA
2200 C = 3: GOSUB 990
2210 RETURN
2220     '********** INSTRUCTIONS ****************
2230 LOCATE 23,1: INPUT "NUMBER OF DEGREES TO ROTATE"; AR
2240 AR = AR * 3.14159 /180          'CONVERT AR TO RADIANS
2250 RETURN
2260 '*****************************************************************************
2270 DATA 30,27,80,27
2280 DATA 80,45,80,70,30,70,30,45
2290 DATA 55,90,22,22
2300 DATA 30,130,80,130,55,110
2310 END
```

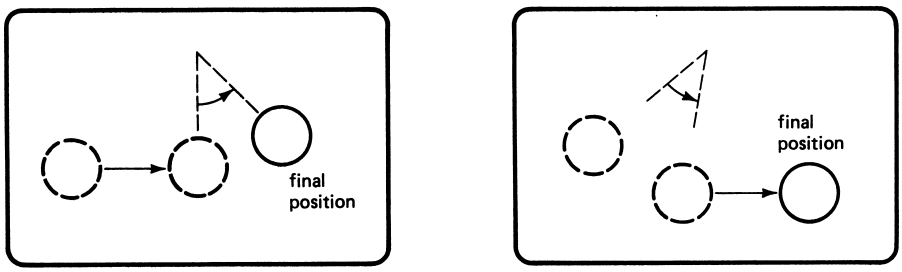
these methods for constructing a display a piece at a time from a set of standard shapes. Using keyboard input, we select a line, triangle, box, or circle and position it on the screen with a specified orientation and size. An example of the screen display that we can make is shown in Fig. 7–9. The menu of shapes is at the

Figure 7–9 Sample output from Prog. 7–8, using a combination of transformations on selected shapes for interactive picture construction.



left of the screen, and the menu for transformations is at the bottom of the screen. Once the picture is constructed, we could scale it to the full screen size and save it on disk or tape. We can use similar methods for constructing and positioning graphs and charts.

The order in which we perform translation and rotation can affect the final displayed position of an object. As shown in Fig. 7–10, when we rotate an object about a point external to the object, the final transformed position will depend on whether we translate before or after we rotate.



(a) Translation & rotation (b) Rotation & translation

Figure 7–10 Final position of an object after translation and rotation (about a point external to the object) depends on the order of these transformations. In (a), translation is performed first; in (b), translation is performed last.

PROGRAMMING PROJECTS

7–1. Write a program to translate a circle to any input screen location by translating the circle center and redrawing the circle at the new position.

7–2. Modify the program of Project 7–1 to scale the circle (relative to its center) and paint the interior after it is translated. Parameters for translating, scaling, and coloring the circle are to be set by input.

7–3. Revise Prog. 7–3 to translate any character string from one screen location to another. No part of the character string should be translated off screen. The program could either reject off-screen translation or split the string in some way to fit on the screen.

7–4. Modify Prog. 7–3 to accept coordinate inputs from a joystick (or tablet), instead of from a light pen.

7–5. Write a program that will scale and display any input polygon, taking the polygon center (the centroid) as a fixed point. The centroid X coordinate is calculated as the average value of the X coordinates of the vertices (that is, we add all the X coordinates and divide by the total number of vertices). The centroid Y coordinate is calculated as the average value of the Y coordinates of the vertices.

7–6. Modify Prog. 7–6 for joystick (or tablet) input.

7–7. Write a program to scale an input figure with respect to any specified direction. The direction is to be specified by an angle A, measured from the horizontal, as shown in

Fig. 7–11. This scaling can be accomplished by rotating the object counterclockwise through the angle A, applying the scaling transformations with HS = S1 and VS = S2, then rotating the object back (through a rotation angle of −A) to its original position. The parameters A, S1, and S2 are to be determined as input.

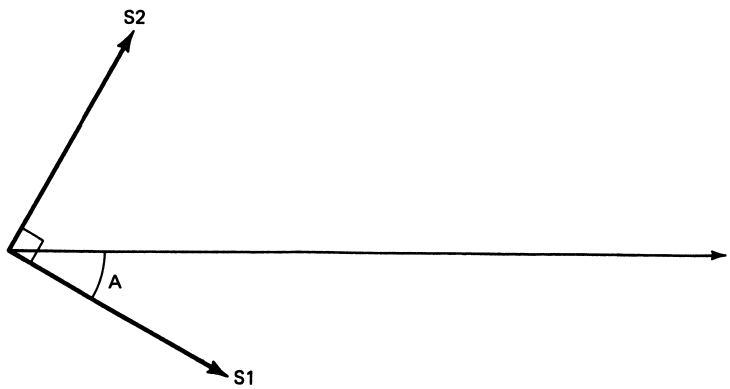


Figure 7–11 Scaling directions specified by an angle A for scaling factors S1 and S2.

7–8. Write a scaling program for bar graphs that will automatically reposition string labels after scaling. Determine the new starting position of a label by scaling the distance from the fixed point to the original starting position.

7–9. Expand Project 7–8 to perform any combination of translation, scaling, and rotation on an input bar graph or line graph.

7–10. Revise Prog. 7–8 to save the positions and orientations of all objects in the constructed picture so that the picture can be redrawn if an object is erased. Also allow each object to be painted in a chosen color.

7–11. Modify Project 7–10 for light pen, joystick, or tablet input.

7–12. Write a program that will perform any combination of transformations (translation, scaling, rotation) on a word drawn with pixels in large letters. Stretching the letters along a diagonal (as outlined in Project 7–7) slants the letters.

Chapter 8

Animation

Animated displays are the result of repeated transformations. The technique used here is to display an object, apply transformations, erase the original, and display the transformed object. When this procedure is repeated several times, we have motion. This is the way animated cartoons and movies are produced. The artist draws each filmstrip frame with slight changes in positions and sizes of objects. Viewing the frames rapidly, one after the other, produces movement. We use this same method to animate both character pictures and pixel pictures.

8–1 CHARACTER ANIMATION

To move a character around the screen, we adapt the transformation methods discussed in Chapter 7 to row and column changes. A character is moved horizontally across the screen by keeping the row number fixed and incrementing the column number. At each step we erase the previously displayed character and put it back on the screen in the translated position. Program 8–1 gives an example of this motion. We start a "happy face" character (ASCII code 2) moving to the right, one column at a time, from the left edge of the screen. When the character reaches the right side of the screen, we reverse its motion. The program repeatedly bounces the character back and forth across the screen.

Statement 50 in Prog. 8–1 displays the character in red on a blue background. Statement 70 erases the character by displaying it in the background color (blue) at the same location. Then the column number is changed to the next position and the process repeated. We can reduce the flicker in animation by erasing an object just before we display it in the next position. This keeps the object on the screen for the maximum time, while we compute coordinates for the next position. A time-delay statement, as included in Prog. 8–1, also helps to

Program 8–1 Bouncing a character horizontally.

```
10 'PROGRAM 8-1. BOUNCING HAPPY FACE BACK & FORTH ACROSS SCREEN
20  SCREEN 0: COLOR 4,1,7: WIDTH 80: LOCATE ,,0: CLS
30  BOUNDARY1 = 1: BOUNDARY1 = 80: CHANGE = 1
40  FOR COLUMN = BOUNDARY1 TO BOUNDARY2 STEP CHANGE
50      LOCATE 10,COLUMN: COLOR 4,1: PRINT CHR$(2);
60      FOR DELAY = 1 TO 10:NEXT
70      LOCATE 10,COLUMN: COLOR 1,1: PRINT CHR$(2);
80      FOR DELAY = 1 TO 10:NEXT
90  NEXT
100 IF BOUNDARY2 = 80 THEN BOUNDARY1 = 79: BOUNDARY2 = 1 : CHANGE = -1
        ELSE BOUNDARY1 = 2 : BOUNDARY2 = 80: CHANGE = 1
110 GOTO 40
120 END
```

reduce flicker by leaving the character on the screen for a longer time before erasing it. We get the best animation effect when the object is left on the screen for a short while, then erased and immediately redrawn. The movement of the character can be speeded up by reducing the delay time or by increasing the translation increment. Changes in column number could be made in steps of 2 or steps of 5, rather than steps of 1.

We can vary the path of motion in many ways. Instead of bouncing the character of Prog. 8–1 back and forth, we could just always move it to the right. It would then seem to move off the screen on the right, circling back to appear again on the left. In a game application, we could randomly choose a new row number each time it starts back across the screen. As another variation on this motion, boundaries could be set up at different screen positions, and we could bounce the object off these boundaries. We could also move objects along vertical, diagonal, or curved paths. The row and column numbers for each successive position would be determined from the path equation. We will consider these various types of motion in more detail with pixel objects.

THE SCREEN FUNCTION

If we wanted to bounce the character of Prog. 8–1 off a wall, we could continually test the character position and turn it around when it gets to the wall. Instead of a wall, other objects could be in the path of our moving character. We can determine when we have a potential collision with any object in two ways. One way is to check positions to determine when we reach the row and column location of another object. This is a straightforward coordinate test for stationary objects, like walls, but when we have several moving objects it is difficult to keep track of all the coordinate positions. With our PC, we can simply ask the system, through the SCREEN function, to identify either the type or color attributes of any characters in our path.

The SCREEN function can be used with three parameters:

SCREEN (R,C,CA) — Gives the ASCII code or the color attributes of the character at row R and column C, depending on the value of the color attribute parameter CA.

Color attribute, CA, is optional. If we omit CA, the SCREEN function will give us the ASCII code value of the character at position R,C. We could then determine whether we were about to have any clashes with other objects by looking at values of the SCREEN function at each row and column position before we moved our object to that location. As long as SCREEN has the value 32 (blank), there is nothing in the way. Including parameter CA allows us to test for color attributes instead of character types.

When CA is assigned any nonzero value, the SCREEN function gives us an integer in the range 0 to 255. If SCREEN is bigger than 127, the character at location R,C is blinking. The value of the remainder after dividing SCREEN by 16 is the foreground color set for that location. We get the background color of the location by dividing SCREEN by 128 and subtracting the foreground color from the remainder. That is, we calculate foreground and background colors as

```
FORECOLOR = SCREEN (R,C,CA) MOD 16
BACKCOLOR = SCREEN (R,C,CA) MOD 128 - FORECOLOR
```

In these calculations, CA must be nonzero and the MOD operator performs modulo arithmetic: returning the remainder after division by 16 or 128. With the Monochrome option, foreground colors in the range 8 through 15 indicate high intensity values for the color codes 0 through 7.

Foreground and background color attributes for each screen character location are stored in an "attribute byte" for that location. When we state a COLOR command and then clear the screen, we set the specified color codes into the attribute bytes for all screen locations. These are the color attributes returned by the SCREEN function for any position, whether or not we have actually placed a character in that position. If we do not set the color attributes in this way, they are set by the system to the default values: white on black. Color attributes for individual locations can be changed by specifying the desired colors in a COLOR statement and then placing characters in those locations.

We make use of the SCREEN function in Prog. 8–2 to bounce a block (ASCII code 219) vertically between two boundaries. Character code 205 is used to form the boundaries. When the block reaches one of the bounding walls, we

Program 8–2 Bouncing a character block vertically, using a SCREEN function character code test.

```
10 'PROGRAM 8-2. BOUNCING CHARACTER UP AND DOWN
20      'SCREEN FUNCTION IS USED TO DO BOUNDARY TEST
30  SCREEN 0: COLOR 3,4,3: WIDTH 40: LOCATE ,,0: CLS
40  LOCATE  5,1: PRINT STRING$(40,205);
50  LOCATE 20,1: PRINT STRING$(40,205);
60  CHANGE = 1
70  ROW = 6
80  IF SCREEN(ROW + CHANGE,20) = 205 THEN CHANGE = -CHANGE
90  COLOR 4,4: LOCATE ROW,20: PRINT CHR$(219);
100 ROW = ROW + CHANGE
110 COLOR 1,4: LOCATE ROW,20: PRINT CHR$(219);
120 GOTO 80
130 END
```

Program 8–3 Multiple object (airplane and block) animation, using a SCREEN function color code test.

```
10 'PROGRAM 8-3. SHOOT THE AIRPLANE!
20      'SPACE BAR IS USED TO SHOOT A MISSILE AT THE AIRPLANE AS
30      'IT MOVES ACROSS THE SKY. SCREEN FUNCTION IS USED TO
40      'DETERMINE IF THE MISSILE IS ABOUT TO COLLIDE WITH THE
50      'PLANE.
60  SCREEN 0: COLOR 7,0,0: WIDTH 80: LOCATE ,,0: CLS
70  PLANEROW = 2: PLANECOLUMN = 1'STARTING POSITIONS
80  MISSILEROW = 21: MISSILECOLUMN = 39
90  WHILE A$ <> "Q" AND A$ <> "q"
100     COLOR 0: CLS    'SET ATTRIBUTE OF UNUSED SCREEN AREAS TO 0
110     COLOR 7: GOSUB 260        'DRAW PLANE
120     IF PLANECOLUMN < 70 THEN PLANECOLUMN = PLANECOLUMN + 2
           ELSE PLANECOLUMN = 1
130     A$ = INKEY$
140     IF A$ = " " THEN FIRE$ = "YES"
150     IF FIRE$ <> "YES" THEN 240
160     MISSILEROW = MISSILEROW - 2: MISSILECOLUMN = MISSILECOLUMN + 2
170     IF MISSILEROW < 1 OR MISSILECOLUMN > 80 THEN FIRE$ = "NO":
           MISSILEROW = 21: MISSILECOLUMN = 39: GOTO 240
180           'IF THE FOREGROUND OF THE MISSILE'S NEXT POSITION
190           'IS 7 (WHITE) WE MUST BE AT THE PLANE!!
200           'FOREGROUND OF NON-PLANE AREAS WOULD BE 0 (BLACK)
210           'SINCE IN LINE 60 WE SET FOREGROUND TO BLACK AND THEN
220           'CLEARED THE SCREEN
230     IF SCREEN(MISSILEROW,MISSILECOLUMN,1) MOD 16 = 7 THEN GOSUB 310
           ELSE LOCATE MISSILEROW,MISSILECOLUMN: PRINT CHR$(254);
240 WEND
250 GOTO 340
260    'MAKE AIRPLANE
270 LOCATE PLANEROW  ,PLANECOLUMN: PRINT "  " + CHR$(220) + "   " + CHR$(219);
280 LOCATE PLANEROW+1,PLANECOLUMN: PRINT "  " + STRING$(6,219) + CHR$(254);
290 LOCATE PLANEROW+2,PLANECOLUMN: PRINT "  " + CHR$(223) + "   " + CHR$(219);
300 RETURN
310 COLOR 23: PRINT "BULLSEYE!!!";: FOR HOLD = 1 TO 3000: NEXT
320 PLANECOLUMN = 1: FIRE$ = "NO": MISSILEROW = 21: MISSILECOLUMN = 39
330 RETURN
340 END
```

reverse its motion by setting DR = −DR. The row increment DR is initially set to 1, but we could speed up the motion with a larger value. We test the SCREEN function for ASCII code 205 to determine when we are about to hit a wall.

Objects formed with several characters are animated with methods similar to those used with single characters. We just have a bit more bookkeeping. In Prog. 8–3, an airplane starts at the left and flies across the screen. When it reaches the right side of the screen, we start it over again at the left. Figure 8–1 shows two positions along the flight path. As an added feature, we shoot a block diagonally up the screen from left to right by pressing the space bar. If the block and airplane collide, they blink and the word BULLSEYE is printed. We test for a collision using the SCREEN function, which tells us the foreground color of the next position along the block's path. Statement 60 sets the stored foreground color of all screen positions to black, and statement 70 specifies that the displayed airplane positions (drawn by statements 170 through 200) will be white. This allows us to identify the airplane position (white foreground) from the remainder of the screen

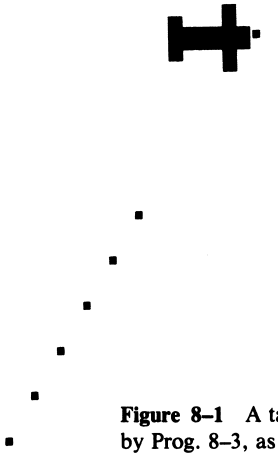


Figure 8–1 A target airplane animated by Prog. 8–3, as a block is fired at it.

(black foreground). We then have a collision when the SCREEN function returns a foreground value of 7.

Moving compound objects around is a slower process since we have more characters to erase and display at each step. Faster animation is possible using the text pages that we have available on our PC.

TEXT PAGES WITH THE SCREEN STATEMENT

In text mode, we can use the SCREEN statement to set up and display multiple screen storage areas. These storage areas are referred to as pages. We can create a character display on one page while another page is being shown on the screen. Then we can put the page we have just created on the screen. This page switching gives us a faster way to animate objects. Pages are set up and displayed with the command

```
SCREEN ,,AP,VP
```

where AP designates the page number we want to construct (the "active" page) and VP designates the page number to be shown on the screen (the "visual" page). In WIDTH 40, we can work with eight pages, numbered 0 through 7. Using WIDTH 80, we have four pages, numbered 0 through 3. The statement SCREEN ,,1,0 would display page number 0 while allowing us to construct a scene on page 1.

Pages are stored in the screen buffer area of the Color/Graphics board. This buffer contains 16K bytes of memory. Since we have 1000 character screen positions in WIDTH 40, we need 1000 bytes to store ASCII codes for a full screen. We also need one byte for the color attributes of each character position. Thus,

2000 bytes are needed for each screen page, allowing us a total of eight pages in the screen buffer. For WIDTH 80, we have twice as many screen positions, so the 16K screen buffer can hold only half as many pages.

Use of text pages in animating an object formed with characters is demonstrated with Prog. 8–4. By drawing the object in successively displaced column positions, we make it move across the screen. We also draw it with different body postures from one page to the next, so that we have a creeping inchworm (Fig. 8–2).

Program 8–4 Animation of an object (worm) using text pages.

```
10 'PROGRAM 8-4. CREEPING WORM ON MULTIPLE SCREENS
20  SCREEN 0: COLOR 2,0,0: WIDTH 80: LOCATE ,,0: CLS
30  ROW = 20: COLUMN = 1
40  A$ = INKEY$
50  SCREEN ,,1,0: CLS
60  WHILE A$ = ""
70      GOSUB 280          'DRAW POSITION 1
80      SCREEN ,,2,1: CLS
90      FOR DELAY = 1 TO 400: NEXT
100     IF COLUMN < 65 THEN COLUMN = COLUMN + 8 ELSE COLUMN = 1
110     GOSUB 310          'DRAW POSITION 2
120     SCREEN ,,3,2: CLS
130     FOR DELAY = 1 TO 300: NEXT
140     IF COLUMN < 68 THEN COLUMN = COLUMN + 7 ELSE COLUMN = 1
150     GOSUB 380          'DRAW POSITION 3
160     SCREEN ,,0,3: CLS
170     FOR DELAY = 1 TO 300: NEXT
180     IF COLUMN < 68 THEN COLUMN = COLUMN + 5 ELSE COLUMN = 1
190     GOSUB 310          'DRAW POSITION 2
200     SCREEN ,,1,0: CLS
210     FOR DELAY = 1 TO 300: NEXT
220     IF COLUMN < 65 THEN COLUMN = COLUMN + 7 ELSE COLUMN = 1
230     A$ = INKEY$
240 WEND
250 SCREEN ,,0,0           'GO BACK TO FIRST SCREEN
260 GOTO 460
270     'DRAWS POSITION #1
280 LOCATE ROW,COLUMN: PRINT STRING$(8,219);: COLOR 14,0: PRINT CHR$(223);
290 LOCATE ROW-1,COLUMN:PRINT STRING$(8,32)+CHR$(220);:COLOR 2,0
300 RETURN
305     'DRAWS POSITION #2
310 LOCATE ROW,COLUMN
320 PRINT STRING$(2,CHR$(219)) + "    " + STRING$(2,CHR$(219));
330 COLOR 14,0: PRINT CHR$(223);: COLOR 2,0
340 LOCATE ROW-1,COLUMN: PRINT " " + CHR$(219) + "    " + CHR$(219) + " ";
350 COLOR 14,0: PRINT CHR$(220);:COLOR 2,0
360 LOCATE ROW-2,COLUMN: PRINT "  " + STRING$(3,CHR$(219)) + " ";
370 RETURN
375     'DRAWS POSITION #3
380 LOCATE ROW,COLUMN: PRINT CHR$(219) + "    " + CHR$(219);
390 COLOR 14,0: PRINT CHR$(223);: COLOR 2,0
400 LOCATE ROW-1,COLUMN: PRINT " " + CHR$(219) + " " + CHR$(219) + " ";
410 COLOR 14,0: PRINT CHR$(220);: COLOR 2,0
420 LOCATE ROW-2,COLUMN: PRINT " " + CHR$(219) + " " + CHR$(219);
430 LOCATE ROW-3,COLUMN: PRINT " " + CHR$(219) + " " + CHR$(219);
440 LOCATE ROW-4,COLUMN: PRINT "  "  + CHR$(219);
450 RETURN
460 END
```

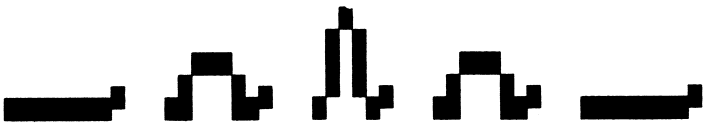


Figure 8–2 Inchworm creeping across the screen, as animated by Prog. 8–4 using text pages.

8–2 PIXEL ANIMATION CONCEPTS

Animating objects that are drawn with pixels is a process similar to character animation. But now we work with pixel coordinates instead of row and column numbers and with graphics commands instead of PRINT and LOCATE statements. We plot the object at a given coordinate position, determine the next position from the transformation equations, erase the pixel object, and replot it in the new coordinate position.

STRAIGHT-LINE MOTION

As a simple example of animation along a straight line, let us consider moving a single pixel horizontally across the screen. If we want to move the pixel from left to right, we can start it at some location, say (X1,Y), and stop it at some position (X2,Y), where X2 > X1. The animation process then consists of a series of translations from X1 to X2, one unit at a time. At each step, we erase the previously plotted point (X,Y) and plot the next point (X + 1, Y). Motion to the left is accomplished by decreasing the X coordinate by one unit at each step. Combining these two motions, we can bounce the point back and forth between X1 and X2. In Prog. 8–5 we illustrate this motion between vertical boundaries drawn at positions XLEFT and XRIGHT, which are specified as input. The pixel initially moves to the right with a positive unit increment (DX = 1). When it reaches the point (XRIGHT −1, Y), we reverse the increment (DX = −DX) to

Program 8–5 Bouncing a pixel between vertical boundaries.

```
10 'PROGRAM 8-5. BOUNCING A POINT BETWEEN VERTICAL BOUNDARIES.
20  SCREEN 0: COLOR 0,7,7: WIDTH 80: CLS
30  INPUT "ENTER X VALUES FOR LEFT WALL AND RIGHT WALL"; XLEFT,XRIGHT
40  IF XLEFT >= XRIGHT OR XLEFT < 0 OR XRIGHT > 319 THEN 160
50  DX = 1
60  X = XLEFT + INT((XRIGHT - XLEFT) / 2)     'START POINT MIDWAY BETWEEN WALLS
70  Y = 100
80  SCREEN 1: COLOR 0,1: CLS
90  LINE (XLEFT,50) - (XLEFT,150): LINE (XRIGHT,50) - (XRIGHT,150)
100     '*******************************
110 PSET (X,Y)
120 IF X + DX = XLEFT OR X + DX = XRIGHT THEN DX = -DX
130 PRESET (X,Y)
140 X = X + DX                                    'MOVE POINT
150 GOTO 110
160 END
```

Program 8–6 Bouncing a point inside a box using unit increments.

```
10 'PROGRAM 8-6. BOUNCING A POINT WITHIN A BOX.
20      'DX = DY = 1.
30  SCREEN 1: COLOR 3,0: CLS
40  INPUT "ENTER X VALUES FOR LEFT AND RIGHT WALL"; XLEFT,XRIGHT
50  IF XLEFT >= XRIGHT OR XLEFT < 0 OR XRIGHT > 319 THEN 220
60  INPUT "ENTER Y VALUES FOR TOP AND BOTTOM OF BOX"; YTOP,YBOTTOM
70  IF YTOP >= YBOTTOM OR YTOP < 0 OR YBOTTOM > 199 THEN 220
80  DX = 1: DY = 1
90  X = XLEFT + INT((XRIGHT - XLEFT) / 2)
100 Y = YTOP + INT((YBOTTOM - YTOP) / 2)      'START THE POINT IN MIDDLE OF BOX
110 CLS
120     '********************** DRAW BOX ****************************
130 LINE (XLEFT,YTOP) - (XRIGHT,YBOTTOM),2,B
140     '********************** BOUNCE POINT ************************
150 PSET (X,Y),2
160 IF X-1=XLEFT OR X+1=XRIGHT THEN DX = -DX     'IF WE'RE ONE UNIT AWAY FROM
170 IF Y-1=YTOP OR Y+1=YBOTTOM THEN DY = -DY     'BOX SIDE, REVERSE DIRECTION
180 PRESET (X,Y)          'ERASE CURRENT POINT
190 X = X + DX: Y = Y + DY                'CALCULATE NEW POINT
200 GOTO 150
210     '************************************************************
220 PRINT "ERROR IN CHOICE OF BOX WALLS"
230 END
```

make it negative and to move the pixel to the left. At (XLEFT + 1, Y), we reverse the increment again. The program repeats this motion indefinitely and appears to bounce the pixel off the boundary walls, since we reverse the motion 1 unit before it gets to either wall. If we allowed the pixel to reach the wall position at either XLEFT or XRIGHT, we would erase part of the wall when we reversed the motion.

This same motion can be accomplished in a vertical direction, using horizontal boundaries and an increment DY in the Y direction. To get motion in any other direction, we increment both the X and Y coordinates. This process is similar to drawing a straight line, except that we now erase each plotted point before we plot the next one. We can specify the path of motion in various ways. We could choose endpoints for the path, we could use the equation for the line (slope and Y-intercept), or we could select any X and Y increments (DX and DY). In the first two cases, we need to set values for DX and DY so that the ratio DY/DX is equal to the slope of the line.

An example of diagonal straight-line motion is given in Prog. 8–6. Here we bounce a point around inside a box. We select a box size and begin by moving the point diagonally down the screen to the right, with increments of 1 unit for both coordinates. When the point encounters a side of the box it changes direction, as shown in Fig. 8–3. If the point bounces off a vertical side, the X increment changes sign (DX = −DX). If the point bounces off a horizontal side, the Y increment changes sign (DY = −DY). Both increments change sign at a corner.

We can speed up the bouncing pixel in Prog. 8–6 by selecting larger values for DX and DY. Choosing these increments equal to 5 moves the pixel around five times faster. Choosing one increment larger than the other (such as DX > DY)

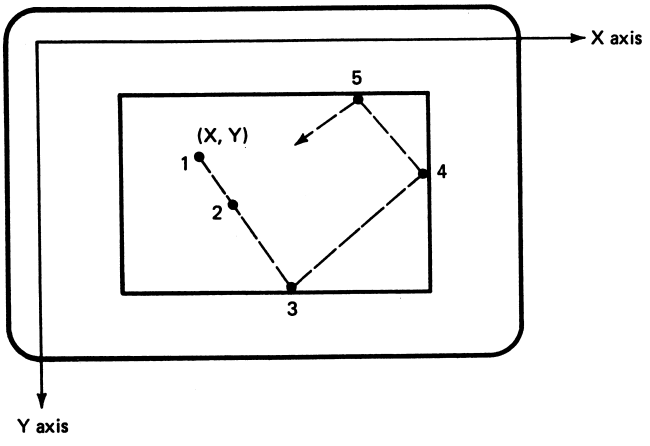


Figure 8–3 Path of a bouncing pixel inside a box. Starting from coordinates (X,Y) at position 1, the pixel will have coordinates (X+DX,Y+DY) at position 2. At position 3, the direction of travel changes by setting DY = −DY. The direction of motion is changed again at position 4 with DX = −DX, and at position 5 with DY = −DY.

moves the pixel faster in the direction with the larger increment. We can, as an alternative, slow the pixel motion in one direction by giving the increment for that direction a value less than one (say, 0.5). Changing the magnitude of the increments during program execution can speed up and slow down the motion as the pixel bounces around. If only one increment magnitude is changed as the pixel rebounds from a wall, we get a skidding or "spin" effect.

Program 8–6 can be modified to work with any values for either increment, DX or DY, by changing the rebound test. The direction of motion of the pixel must be reversed whenever either the X coordinate or Y coordinate would be incremented through a wall of the box. We can make this change in Prog. 8–6 by replacing lines 160 and 170 with the following:

```
160 IF DX > 0 AND X + DX >= XRIGHT THEN DX = -DX
      ELSE IF DX < 0 AND X + DX <= XLEFT THEN DX = -DX
170 IF DY > 0 AND Y + DY >= YBOTTOM THEN DY = -DY
      ELSE IF DY < 0 AND Y + DY <= YTOP THEN DY = -DY
```

The preceding program segment has the effect of rebounding the pixel before it gets to the walls of the box when either DX or DY is greater than 1. To get a more realistic bounce, we can plot the pixel near the wall before reversing its direction of motion. We could do this by always choosing the increments and box size so that the distance across the box in either direction is an integral multiple of the increment for that direction. This is probably a little too restrictive for many applications. A more general solution to this problem is to project the path of the pixel to the wall and determine the intersection point. Then we can produce a display that rebounds the pixel at the walls for any increment chosen.

Figure 8–4 depicts the path of pixel motion toward a vertical boundary of a box. Starting from position (X,Y), the intersection point (XI,YI) on this boundary is determined from the equation for the straight line path:

$$YI = M * XI + (Y - M * X) \qquad (8\text{–}1)$$

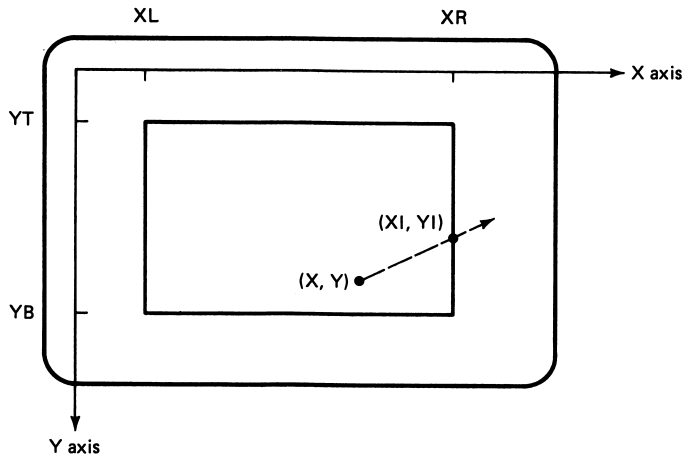


Figure 8–4 A pixel at position (X,Y), traveling along the diagonal path indicated inside a box, will intersect the right boundary at position (XI,YI).

with XI = XR at the right boundary and XI = XL at the left boundary. The slope, M, of the line is calculated from the coordinate increments as

$$M = DY/DX \tag{8–2}$$

Equations (8–1) and (8–2) can be used to determine intersection positions for any boundary and any of the possible directions of travel. For intersection with the top or bottom of the box, XI is calculated from (8–1) as XI = X + (YI − Y)/M, where either YI = YT or YI = YB (Fig. 8–4). At each of these rebound positions, we plot the pixel 1 unit inside the boundary and reverse the direction of motion. An application of this method for a bouncing ball is given in Prog. 8–7.

A moving circle provides the basis for many animation applications involving a bouncing ball. The methods discussed for points can be used to bounce a circle around as if it were a big point. We move the circle center and test for intersection of the circle boundary with the turnaround positions. A bouncing ball inside a box is displayed by Prog. 8–7 using the techniques discussed for

Program 8–7 Bouncing a ball inside a box.

```
10 'PROGRAM 8-7. BOUNCING BALL WITHIN A BOX.
20      'DX = DY = 5. IF BALL IS GOING TO GO BEYOND A BOUNDARY,
30      'FIND INTERSECTION OF BALL & BOUNDARY THEN DRAW BALL.
40      'REVERSE BALL'S DIRECTION, AND CONTINUE.
50      '*****************************************************************
60  SCREEN 0: COLOR 4,7,7: CLS
70  INPUT "ENTER X VALUES FOR LEFT AND RIGHT WALL"; XLEFT,XRIGHT
80  IF XLEFT >= XRIGHT OR XLEFT < 0 OR XRIGHT > 319 THEN 600
90  INPUT "ENTER Y VALUES FOR TOP AND BOTTOM OF BOX"; YTOP,YBOTTOM
100 IF YTOP >= YBOTTOM OR YTOP < 0 OR YBOTTOM > 199 THEN 600
110 DX = 5: DY = 5                  'BALL TRAVELS 5 UNITS IN EACH STEP
120 R = 3                           'R IS RADIUS OF BALL
130 XNEW = XLEFT + INT((XRIGHT - XLEFT) / 2)     'START BALL IN MIDDLE OF BOX
```

Program 8–7 (cont.)

```
140 YNEW = YTOP + INT((YBOTTOM - YTOP) / 2)
150 SCREEN 1: COLOR 1,0: CLS
160     'DRAW BOX
170 LINE (XLEFT,YTOP) - (XRIGHT,YBOTTOM),2,B
180     '*************************** BOUNCE BALL ****************************
190 PAINT (X,Y),0,1                         'ERASE CURRENT BALL POSITION
200 CIRCLE (X,Y),R,0
210 CIRCLE (XNEW,YNEW),R,1                  'DRAW NEW POSITION
220 PAINT (XNEW,YNEW),3,1                   'FILL IN BALL AREA
230 X = XNEW: Y = YNEW                      'SAVE CURRENT POSITION IN X AND Y
240 BX = 0                                  'BX AND BY ARE SWITCHES TO INDICATE
250 BY = 0                                  'WHICH WALL WE'RE GOING TO HIT
260 SLOPE = DY / DX                         'SLOPE IS SLOPE OF BALL'S PATH
270     '*****************************************************************
280     'WILL WE HIT A VERTICAL WALL?
290 IF DX > 0 AND X + DX + R >= XRIGHT THEN BX = 1: XNEW = XRIGHT - R - 1
    ELSE IF DX < 0 AND X + DX - R <= XLEFT THEN BX = 1: XNEW = XLEFT + R + 1
300     'WILL WE HIT A HORIZONTAL WALL?
310 IF DY > 0 AND Y + DY + R >= YBOTTOM THEN BY = 1: YNEW = YBOTTOM - R - 1
    ELSE IF DY < 0 AND Y + DY - R <= YTOP THEN BY = 1: YNEW = YTOP + R + 1
320     '*****************************************************************
330     'ARE WE BOUNCING OFF NO WALLS, AN X WALL, A Y WALL, OR BOTH WALLS?
340 IF BX = 0 AND BY = 0 THEN 390           'NOT BOUNCING
350 IF BX = 0 AND BY = 1 THEN 430           'BOUNCING OFF Y
360 IF BX = 1 AND BY = 0 THEN 470           'BOUNCING OFF X
370 IF BX = 1 AND BY = 1 THEN 510           'BOUNCING OFF BOTH (IN A CORNER)
380 '
390 '########################## NOT BOUNCING ##############################
400 XNEW = X + DX
410 YNEW = Y + DY
420 GOTO 190
430 '########################## BOUNCE OFF Y WALL #########################
440 XNEW = (YNEW - Y) / SLOPE + X
450 DY = - DY
460 GOTO 190
470 '########################## BOUNCE OFF X WALL #########################
480 YNEW = (XNEW - X) * SLOPE + Y
490 DX = -DX
500 GOTO 190
510 '########################## GOING INTO A CORNER #######################
520     'WHICH WALL WOULD IT HIT FIRST?
530 IF ABS(XNEW - X) < ABS(YNEW - Y) THEN 470       'BOUNCE OFF X
540 IF ABS(YNEW - Y) < ABS(XNEW - X) THEN 430       'BOUNCE OFF Y
550     'BALL IS EQUAL DISTANCE FROM X AND Y WALLS ON EACH SIDE OF CORNER
560 DX = -DX
570 DY = -DY
580 GOTO 190
590 '#########################################################################
600 PRINT "ERROR IN CHOICE OF BOX WALLS"
610 END
```

rebounding points. The ball is rebounded from a wall by finding the intersection of the ball's path with the wall and turning the ball around at a point 1 unit inside the wall. A yellow ball is displayed at each position using the PAINT command.

Motion of an arbitrarily shaped object is accomplished with the same basic techniques that we have used to move a point or circle. We display all the parts of the object, erase the object, and redraw all the parts in a new position. Repeating

Program 8–8 Bouncing a line vertically.

```
10 'PROGRAM 8-8. POGO STICK
20  SCREEN 0: COLOR 1,7,7: CLS
30  INPUT "Y VALUES FOR TOP AND BOTTOM BOUNDARIES OF BOUNCE"; YTOP,YBOTTOM
40  IF YTOP < YBOTTOM AND YTOP >= 0 AND YBOTTOM <= 199 THEN 70
50  PRINT "BAD BOUNDARIES. CHOOSE YT < YB AND BOTH BETWEEN 0 AND 199";
60  GOTO 30
70  INPUT "AMOUNT FOR CHANGING Y VALUES OF LINE"; DY
80  X = 160                          'CENTER LINE ACROSS SCREEN
90  Y1 = YTOP + INT((YBOTTOM - YTOP) / 2) - 20  'ENDPOINTS ARE 20 UNITS UP AND
100 Y2 = YTOP + INT((YBOTTOM - YTOP) / 2) + 20  'DOWN FROM CENTER OF BOUNDARIES
110 SCREEN 1: COLOR 5,1: CLS
120     '*************************************************************************
130 LINE (X,Y1) - (X,Y2)
140 IF DY > 0 THEN 180           'LINE IS GOING DOWN
150     'OTHERWISE LINE IS GOING UP (DY IS NEGATIVE)
160 IF Y1 + DY <= YTOP THEN DY = -DY       'REVERSE DIRECTION
170 GOTO 210
180     'LINE IS GOING DOWN
190 IF Y2 + DY >= YBOTTOM THEN DY = -DY       'REVERSE DIRECTION
200     'ERASE CURRENT LINE, CALCULATE NEW POSITION, GOTO DRAW
210 LINE (X,Y1) - (X,Y2),0
220 Y1 = Y1 + DY
230 Y2 = Y2 + DY
240 GOTO 130
250 END
```

this sequence over and over again produces object motion. Animating an object involves some additional considerations when we want to rotate or scale the object or when we want to move different parts of the object differently.

A line drawn between points (X,Y1) and (X,Y2) is bounced up and down between two fixed boundaries by Prog. 8–8. The motion of this pogo stick is

Program 8–9 Animation by scaling (box).

```
10 'PROGRAM 8-9. MOVING TOWARD US THROUGH SCALING
20  SCREEN 2: CLS
30      'READ DATA POINTS OF OBJECT
40  READ POINTCOUNT
50  FOR EACHPOINT = 1 TO POINTCOUNT
60      READ X(EACHPOINT), Y(EACHPOINT)
70  NEXT
80  READ XSCALING,YSCALING,XFIXED,YFIXED
90  ONSCREEN = 1
100 WHILE ONSCREEN
110     DRAWCOLOR = 1
120     GOSUB 180                  'DRAW SHAPE
130     DRAWCOLOR = 0
140     GOSUB 180                  'ERASE SHAPE
150     GOSUB 240                  'RECALCULATE POINTS
160 WEND
170 GOTO 370
180     'DRAWS SHAPE
190 FOR EACH = 1 TO POINTCOUNT - 1
200     LINE (X(EACH),Y(EACH)) - (X(EACH+1),Y(EACH+1)),DRAWCOLOR
```

Program 8–9 (cont.)

```
210 NEXT
220 LINE (X(POINTCOUNT),Y(POINTCOUNT)) - (X(1),Y(1)),DRAWCOLOR
230 RETURN
240     'RECALCULATE POINTS
250 XCONSTANT = XFIXED * (1 - XSCALING)
260 YCONSTANT = YFIXED * (1 - YSCALING)
270 FOR EACHPOINT = 1 TO POINTCOUNT
280     X(EACHPOINT) = X(EACHPOINT) * XSCALING + XCONSTANT
290     IF X(EACHPOINT) < 0 OR X(EACHPOINT) > 639 THEN 370
300     Y(EACHPOINT) = Y(EACHPOINT) * YSCALING + YCONSTANT
310     IF Y(EACHPOINT) < 0 OR Y(EACHPOINT) > 199 THEN 370
320 NEXT
330 RETURN
340 DATA 4
350 DATA 630,0,638,0,638,3,630,3
360 DATA 1.2,1.2,639,0
370 END
```

speeded up or slowed down with choices for DY in the range 1 to 20. This program never lets the line reach or overshoot the boundaries set at the top (YT) and at the bottom (YB).

We can animate the vertical line horizontally by incrementing the X coordinate instead of the Y coordinate. In this case, we might bounce it off vertical walls positioned at XL and XR. Incrementing both coordinates moves the line diagonally. We could then bounce the line around on straight paths much as we did with a single pixel. Horizontal lines, or lines drawn at any angle, are moved about with similar methods. We move both endpoints of the line with the same increments, DX and DY. The line moves faster for larger increment values and changes direction whenever we change the sign of one or both increments.

For an object formed with multiple lines, we can store all the vertex coordinates in an array. Moving this polygon around then means moving all the vertex points and redrawing the polygon at each position along the path of motion. We could also define the object with DRAW statements and animate it repeatedly by changing the reference point for these statements. The more lines in the object, the more time it takes to erase and redraw the object at each step. With advanced BASIC we can use the PUT statement, examined in Section 8–3, to speed up animation. Without this statement, we must erase and redraw each individual line in an object using the methods previously discussed.

Scaling an object provides a means for simulating motion toward or away from us. A box is made to move toward us with Prog. 8–9, as shown in Fig. 8–5. The box is repeatedly scaled relative to a point near the upper right corner of the screen, causing it to move to the left as it enlarges. Again, the more lines in the object, the slower the animation process. Faster methods for animating objects into the distance or toward us are discussed in Section 8–3.

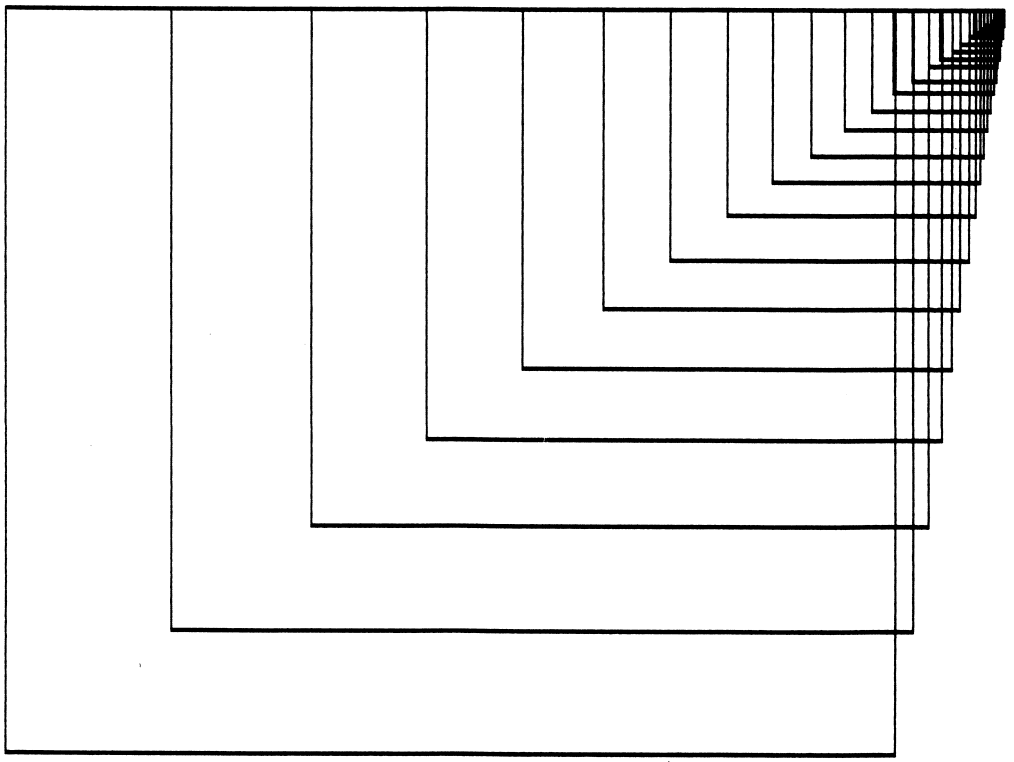


Figure 8–5 Positions of a box as it moves toward us, as scaled and animated by Prog. 8–9.

THE POINT FUNCTION

In applications where we have potential collisions between moving objects, we can use the POINT function to determine when two objects bump into each other. This gives us an easier way to handle collisions when we have multiple objects on the screen, particularly if several of them are moving. Otherwise we have to keep track of all object positions.

POINT (X,Y) — Gives the color code of the pixel at coordinates (X,Y).

Values of the POINT function will be −1, 0, 1, 2, or 3 in medium resolution and −1, 0, or 1 in high resolution. A value of −1 indicates that coordinate position (X,Y) is not within the screen limits.

The POINT function is used in Prog. 8–10 to test for impact between a ball and the walls of a maze (Fig. 8–6). Motion of the ball is directed by use of the cursor control keys on the numeric keypad, in unit steps to the right, left, up, or down. If collision with a wall is detected (POINT=2), the PC's speaker beeps.

Program 8–10 Animating a ball through a maze, using the POINT function to test for wall collisions.

```
10 'PROGRAM 8-10. GETTING THROUGH MAZE WITH THE SCREEN FUNCTION
20  SCREEN 1: COLOR 1,1: CLS
30      'MAKE MAZE
40  LINE (150,95) - (200,100),1,BF        'MAKE INTERIOR BOX
50  R = 1
60  UA = 5                'UA,RA,LA,DA ARE AMOUNTS TO DRAW UP,RIGHT,DOWN,LEFT
70  PSET(150,100)         'START BY THE CENTER
80  DRAW "C2"
90  FOR TIME = 1 TO 8
100     RA = UA + 45: DA = UA + 10: LA = UA + 55
110     DRAW "U=UA; R=RA; D=DA; L=LA;"
120     UA = UA + 20
130 NEXT
140     'DRAW BALL
150 X = 70: Y = 175
160 CIRCLE (70,175),R,1
170 A$ = INKEY$: IF A$ = "" THEN 170
180 B$ = A$
190 WHILE A$ <> "Q" AND A$ <> "q" AND ITHACA = 0        'QUIT OR INTERIOR?
200     CIRCLE (X,Y),R,0                  'ERASE CURRENT BALL
210     'IF THE COLOR OF THE NEXT POINT IS BACKGROUND OR MAZE INTERIOR BOX
         THEN ADVANCE TO POINT AND DRAW CIRCLE. OTHERWISE WE'RE ON WALL
220     IF RIGHT$(B$,1) = CHR$(80) THEN IF POINT(X,Y+1+R) < 2 THEN
            Y=Y+1:GOTO 260 ELSE BEEP:GOTO 270
230     IF RIGHT$(B$,1) = CHR$(72) THEN IF POINT(X,Y-1-R) < 2 THEN
            Y=Y-1:GOTO 260 ELSE BEEP:GOTO 270
240     IF RIGHT$(B$,1) = CHR$(77) THEN IF POINT(X+1+R,Y) < 2 THEN
            X=X+1:GOTO 260 ELSE BEEP:GOTO 270
250     IF RIGHT$(B$,1) = CHR$(75) THEN IF POINT(X-1-R,Y) < 2 THEN
            X=X-1:GOTO 260 ELSE BEEP:GOTO 270
260     IF POINT(X+R,Y) = 1 THEN ITHACA = 1
270     CIRCLE (X,Y),R,1                  'DRAW NEW BALL
280     IF POINT(X,Y) = 1 THEN ITHACA = 1
290     A$ = INKEY$
300     IF A$ <> "" THEN H = ASC(RIGHT$(A$,1))          'NEW, VALID KEY?
310     IF H = 80 OR H = 72 OR H = 77 OR H = 75 THEN B$ = A$:H = 0
320 WEND
330 IF ITHACA THEN PRINT "HOORAY!!"
340 END
```

When color code 1 is detected, the ball has reached the maze center. The program then prints HOORAY and ends.

MOTION ALONG CURVED PATHS

Animating an object along a specified curved path is accomplished by determining positions from the path equation. The motion is begun at a specified starting position and terminated at a specified stopping point. We can move objects along circular or elliptical paths with the circle-drawing techniques discussed in Section 5–1, or with the rotation methods of Section 7–3. To get smooth motion around a circle, we want the pixel positions spaced evenly along the path. This means that we would use trigonometric equations for calculating coordinate positions, with an equal angular distance between positions. Large angular increments between points (say, 30 degrees) are used to produce faster motion. Varying the radius of a

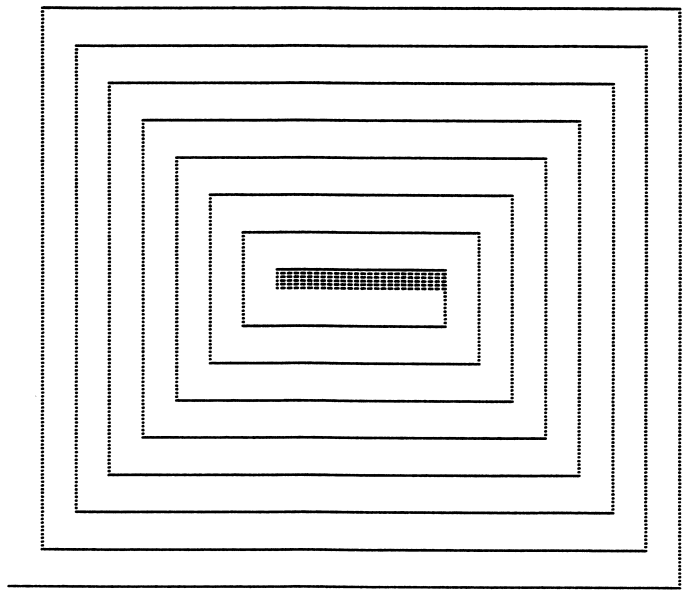


Figure 8–6 Maze pattern produced by Prog. 8–10 as a ball is moved toward the center.

circle results in a spiraling motion. Rotations along circular paths provide a basis for modeling satellite orbits, the solar system, or machine parts. Program 8–11 moves one end of a horizontal line in a circle. The line moves back and forth and up and down, as shown in Fig. 8–7, simulating motion of a horizontal bar attached to a rotor.

Simulation of a bouncing ball is displayed by Prog. 8–12. This program approximates the motion of a ball that is dropped from some height H and then bounces across the screen. Each bounce is decreased a little in height as the ball travels across the screen (Fig. 8–8). A SIN function is used to obtain the up-and-down motion, and the EXP function is used to decrease the amplitude. Since we

Program 8–11 Moving a line in a circle.

```
10 'PROGRAM 8-11. CIRCULAR MOVEMENT OF A HORIZONTAL LINE.
20  SCREEN 1: CLS
30  XC = 160: YC = 100
40  R = 50                'R IS RADIUS
50     'CALCULATE X1 POINTS AT EVERY 15 DEGREES ALONG THE CIRCLE
60  DA =  15 * 3.14159 / 180             'CONVERT 15 TO RADIANS
70       '*************************************************************
80  FOR ANGLE = DA TO 6.28318 STEP DA
90      LINE (X,Y) - (X + 80,Y),0        'ERASE CURRENT LINE
100     X = XC + R * SIN(ANGLE): Y = YC + R * COS(ANGLE)
110     LINE (X,Y) - (X + 80,Y)          'LINE IS 80 UNITS LONG
120 NEXT
130 GOTO 80
140 END
```

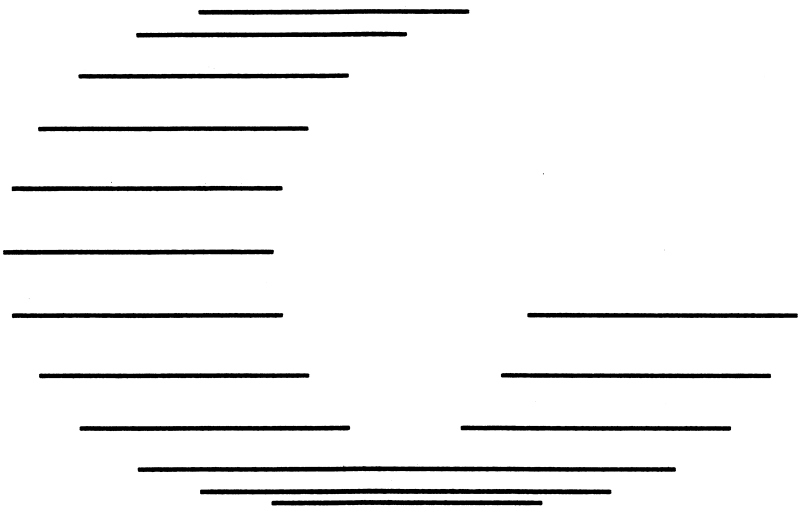


Figure 8–7 Positions of a line with left endpoint moving in a circle, as output by Prog. 8–11. The horizontal line starts at the right, moves down and to the left, and moves around the circular path back to the starting position.

want the motion to begin at coordinates (0,H), we set D = PI/2 in the SIN function. Next we choose a distance of 40 between bounces, so that several bounces can be displayed across the screen. We do this by setting W = PI/40. Finally, we want to display the ball exactly when it hits the ground at each bounce. This means that the increment for X must be chosen so that it divides evenly into 40. For this example, the X increment is set to 4, and the ball is

Program 8–12 Bouncing motion of a dropped ball.

```
10 'PROGRAM 8-12. BOUNCING BALL DROPPED FROM SOME HEIGHT.
20      'PROGRAM SIMULATES MOVEMENT OF A BALL DROPPED FROM
30      'SOME HEIGHT.
40  SCREEN 1: CLS
50  INPUT "BALL IS DROPPED FROM WHAT HEIGHT"; HEIGHT
60  W = 3.14159 / 40              'DISTANCE FROM BOUNCE TO BOUNCE IS 40
70  D = 90 * 3.14159 / 180        'DISPLACE BY 90 DEGREES (EXPRESSED AS RADIANS)
80  K = .01                       'K IS DAMPING FACTOR
90  CLS
100 LINE (0,199) - (319,199)              'DRAW GROUND
110     '***************** DROP BALL AND BOUNCE *****************
120 FOR XNEW = 0 TO 319-10 STEP 4           '4 EVENLY DIVIDES INTO 40
130     YNEW = HEIGHT * SIN(W * XNEW + D) * EXP(-K * XNEW)
140     YNEW = 199 - ABS(YNEW) - 3
150     CIRCLE (X,Y),2,0          'ERASE CURRENT POSITION
160     CIRCLE (XNEW + 10,YNEW),2     'DRAW NEW POSITION
170     X = XNEW + 10       'STORE CURRENT POSITION IN X AND Y
180     Y = YNEW
190 NEXT
200 END
```

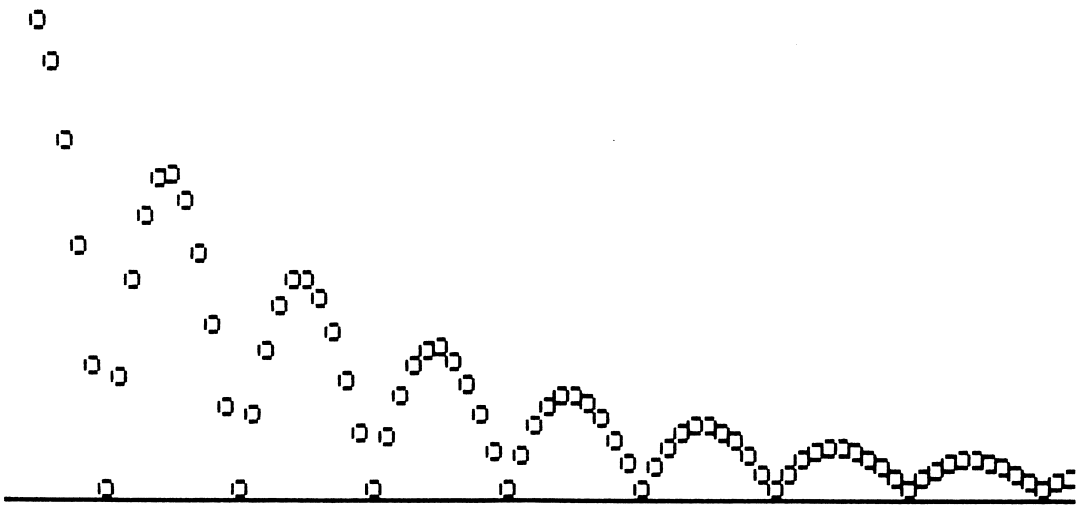


Figure 8–8 The motion of a ball bouncing from left to right, displayed by Prog. 8–12.

displayed each time it bounces at X = 20, 60, 100, and so on. The final display is offset 10 pixels to the right so that motion does not begin at the edge of the screen.

A circle moving along a parabola provides a more accurate simulation of the path of motion of a ball tossed into the air. The distance that the ball travels is determined by how fast it is initially thrown into the air and the angle of this projection, measured from the horizontal, as shown in Fig. 8–9. From the values of the projection speed S and projection angle A, we can calculate the range R and maximum height HT as

$$R = S * S * SIN(2 * A)/G$$
$$HT = ((S * SIN(A)) \wedge 2)/(2 * G) \qquad (8\text{–}3)$$

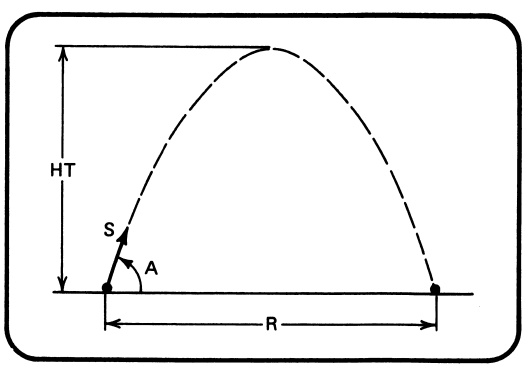


Figure 8–9 An object tossed into the air with initial speed S at an angle A will rise to height HT and land a distance R away from the starting point. Both the range R and height HT are calculated from values for the projection speed S and projection angle A.

where G is the acceleration due to gravity (980 cm/(sec * sec)). For a fixed value of S, we get maximum range at an angle of 45 degrees. Height, HT, is largest when we throw the ball straight up (A = 90 degrees). Positions along this curve are obtained by varying X from 0 to R and calculating the corresponding Y values from the equation

$$Y = C1 * X \wedge 2 + C2 * X + YO \tag{8–4}$$

where YO is any starting value we choose on the screen, and the constants C1 and C2 are determined as

$$\begin{aligned} C1 &= G/(2 * (S * COS(A)) \wedge 2) \\ C2 &= -TAN(A) \end{aligned} \tag{8–5}$$

Program 8–13 animates a circle along a parabolic path. Coordinates for the starting position (XO,YO) are specified as input. The projection angle A and initial speed S are also determined as input. Angle A is limited to values between 0 and 90 degrees, so that the motion is to the right (Fig. 8–9). Values for S in the range 200 to 600 produce curves within the screen limits for SCREEN 1. Other ranges for S can be set up by changing the value of G. We can vary the initial and final positions of the circle in this program to simulate other similar types of motion.

Program 8–13 Animating a ball along a parabolic path.

```
10 'PROGRAM 8-13. MOVING ALONG A PARABOLIC CURVE
20 SCREEN 0: WIDTH 80: CLS
30 INPUT "ENTER COORDINATES OF START POSITION"; XSTART,YSTART
40 IF XSTART >= 0 AND XSTART <= 319 AND YSTART >= 0 AND YSTART <= 199 THEN 60
50 PRINT "RE-ENTER START POSITION": GOTO 30
60 INPUT "ENTER ANGLE (0 - 90)"; ANGLE
70 ANGLE = ANGLE * 3.14159 / 180        'CONVERT A TO RADIANS
80 INPUT "ENTER SPEED (100 - 600)"; SPEED
90 GRAVITY = 980
100 RANGE = SPEED * SPEED * SIN(2 * ANGLE) / GRAVITY
110                         'WILL WHOLE CURVE FIT ON SCREEN?
120 IF XSTART + RANGE <= 319 THEN 160
130                         'IF NOT, ENTER NEW VALUES
140 PRINT "RE-ENTER ANGLE AND SPEED":GOTO 60
150                         'HEIGHT IS HEIGHT OF CURVE
160 HEIGHT = ((SPEED * SIN(ANGLE)) ^ 2) / (2 * GRAVITY)
170                         'WILL WHOLE CURVE FIT ON SCREEN?
180 IF HEIGHT > 0 AND HEIGHT <= 199 THEN 220
190                         'IF NOT, ENTER NEW VALUES
200 PRINT "RE-ENTER ANGLE AND SPEED": GOTO 60
210                         'CALCULATE COEFFICIENTS OF EQUATION
220 C1 = GRAVITY / (2 * (SPEED * COS(ANGLE)) ^ 2)
230 C2 = - TAN(ANGLE)
240 SCREEN 1: CLS
250     '******************** MOVE BALL ALONG CURVE ************************
260 FOR X = 0 TO RANGE STEP 2
270     Y = C1 * X ^ 2 + C2 * X + YSTART
280     CIRCLE (X + XSTART,Y),3,3,,,.9199999
290     CIRCLE (X + XSTART,Y),3,0,,,.9199999
300 NEXT
310 END
```

Tossing an object from the top of a building or hill means that we start and stop the motion at different Y values. To simulate the path of an object dropped from a moving airplane, the object begins at maximum height, HT, and falls to the ground.

In some cases we would like to have an object turn as it moves along a curved path. Figure 8–10 shows positions and orientations of a line (with an arrow tip) whose left endpoint traverses a parabolic path as the line turns to stay tangent to the curve at each step. This motion could simulate the appearance of an arrow shot into the air. To get this motion, we need to change the slope of the line as it moves. For the parabola equation (8–4), the slope of a line tangent to the curve is calculated from the value of the X position:

$$M = 2 * C1 * X + C2 \tag{8–6}$$

The coordinates (X1,Y1) of the other end of the line can be determined from this slope and the line length L, as indicated in Fig. 8–11. From this diagram we obtain the relationships for the sides of the triangle with hypotenuse L as

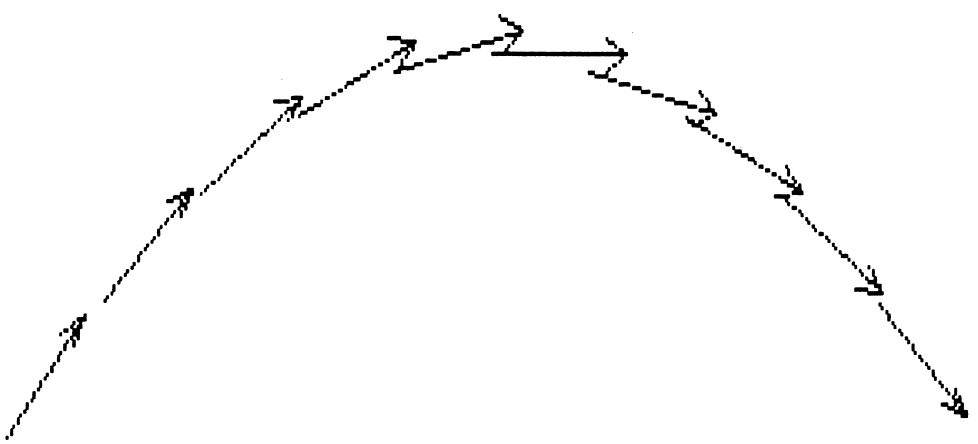
$$\begin{aligned} X1 - X &= L * COS(AS) \\ Y1 - Y &= L * SIN(AS) \end{aligned} \tag{8–7}$$

These relationships allow us to compute the position (X1,Y1) from values for X, Y, and AS. Angle AS is the angle that the line makes with the horizontal, and it is calculated from the slope M:

$$AS = ATN(M) \tag{8–8}$$

The motion of the arrow in Fig. 8–10 along a parabola is produced by Prog. 8–14, using equations (8–7) and (8–8). We draw in the tip of the arrow at each position with short lines attached to the point (X1,Y1). These short lines have

Figure 8–10 Appearance of a line whose left endpoint moves along a parabola, with the line remaining tangent to the path as it travels left to right, as displayed by Prog. 8–14.



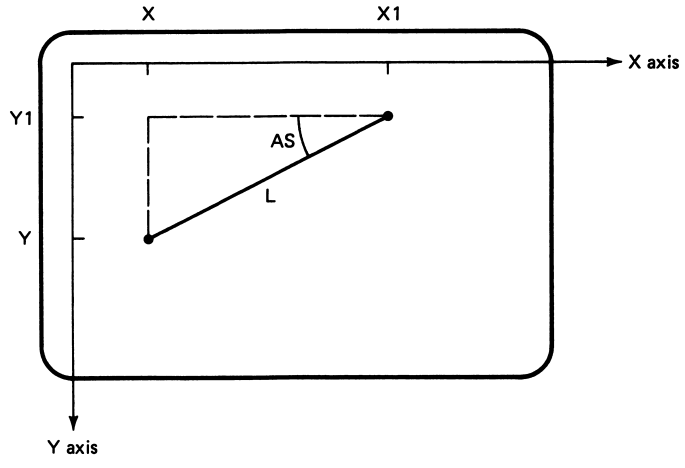


Figure 8–11 Coordinate values (X1,Y1) for one end of a line can be calculated from the values for the line length L, the slope angle AS, and the coordinate values (X,Y) of the other end of the line. The slope angle AS is determined from the value for the slope M as AS = ATN(M).

Program 8–14 Motion of an arrow along a parabolic path.

```
10 'PROGRAM 8-14. ARROW SHOT ALONG A PARABOLIC PATH.
20      'REPEATEDLY DRAWS AND ERASES AN ARROW WHOSE TAIL
30      'IS A POINT ON A PARABOLA. REMAINDER OF THE ARROW
40      'IS FOUND USING THIS TAIL POINT, THE SLOPE OF THE
50      'LINE TANGENT TO THE CURVE AT THIS POINT, AND THE
60      'LENGTH OF THE ARROW
70 '*****************************************************************************
80  SCREEN 0: WIDTH 80: CLS
90  INPUT "ENTER COORDINATES OF START POSITION"; XSTART,YSTART
100 IF XSTART>=0 AND XSTART<=319 AND YSTART>=0 AND YSTARTM<=199 THEN 120
110 PRINT "START POSITION OFF SCREEN": GOTO 90
120 INPUT "ENTER ANGLE (0 - 90)"; ANGLE
130 ANGLE = ANGLE * 3.14159 / 180        'EXPRESS A AS RADIANS
140 INPUT "ENTER SPEED (100 - 600)"; SPEED
150 GRAVITY = 980                 'G IS FORCE OF GRAVITY
160 ARROWLENGTH = 40
170 TIPLENGTH = 8
180     'FIND RANGE AND HEIGHT OF ARROW'S FLIGHT
190 RANGE = SPEED * SPEED * SIN(2 * ANGLE) / GRAVITY
200 IF XSTART + RANGE <= 319 THEN 220
210 PRINT "RE-ENTER ANGLE AND SPEED": GOTO 120
220 HEIGHT = ((SPEED * SIN(ANGLE)) ^ 2) / (2 * GRAVITY)
230 IF HEIGHT > 0 AND HEIGHT <= 199 THEN 250
240 PRINT "RE-ENTER ANGLE AND SPEED": GOTO 120
250     'DETERMINE COEFFICIENTS FOR PARABOLA'S EQUATION
260 C1 = GRAVITY / (2 * (SPEED * COS(ANGLE)) ^ 2)
270 C2 = - TAN(ANGLE)
280 SCREEN 1: CLS
290     '********************* MOVE ARROW *************************
300     'FIND ARROW TAIL POINTS ALONG THE PARABOLA AND DRAW ARROW
310 FOR X = 0 TO RANGE STEP RANGE/10     'PLACE ARROW AT SUCCESSIVE TENTHS
320     Y = C1 * X * X + C2 * X + YSTART
330             'X AND Y ARE THE TAILPOINTS ON THE PARABOLA
340             'FIND OTHER ENDPOINT OF ARROW
```

Program 8–14 (cont.)

```
350     SLOPE = C1 * X * 2 + C2
360     ANGLE1 = ATN(SLOPE)              'INVERSE TANGENT OF SLOPE IS ANGLE1
370     Y1 = Y + ARROWLENGTH * SIN(ANGLE1)
380     X1 = X + ARROWLENGTH * COS(ANGLE1)
390     IF X1 > 319 OR Y1 > 199 THEN 670        'IS OTHER ENDPOINT ON SCREEN?
400     GOSUB 620                               'ERASE ARROW
410             'CALCULATE ARROW TIP
420     SLOPE2 = SLOPE + .75                    'SLOPE OF ONE TIP
430     ANGLE2 = ATN(SLOPE2)
440     X2 = X1 - TIPLENGTH * COS(ANGLE2)
450     Y2 = Y1 - TIPLENGTH * SIN(ANGLE2)
460     SLOPE3 = SLOPE - .75                    'SLOPE OF SECOND TIP
470     ANGLE3 = ATN(SLOPE3)
480     X3 = X1 - TIPLENGTH * COS(ANGLE3)
490     Y3 = Y1 - TIPLENGTH * SIN(ANGLE3)
500     GOSUB 570                               'DRAW ARROW
510     XS = X                                  'SAVE CURRENT POSITION IN
520     YS = Y                                  'XS,YS,X1S,Y1S
530     X1S = X1
540     Y1S = Y1
550 NEXT X
560 GOTO 670
570 '########################## DRAW ARROW ##########################
580 LINE (X,Y) - (X1,Y1)
590 LINE (X1,Y1) - (X2,Y2)
600 LINE (X1,Y1) - (X3,Y3)
610 RETURN
620 '########################## ERASE ARROW ##########################
630 LINE (XS,YS) - (X1S,Y1S),0
640 LINE (X1S,Y1S) - (X2,Y2),0
650 LINE (X1S,Y1S) - (X3,Y3),0
660 RETURN
670 END
```

slopes that are only slightly different from the slope M for the arrow shaft. In this program, we choose one of these slopes to be 0.75 greater than M and the other to be 0.75 less than M. The length of each short line is chosen to be 8 and the arrow shaft length is set to 40.

Motion of a line tangent to any curve can be produced by the methods of Prog. 8–14. We move one end of the line along the curve, compute the slope at each position, determine the slope angle, and calculate the coordinates for the other end of the line using equations (8–7). The slope of a line tangent to a curve at any point is determined from the equation for that curve. As an example, a third degree polynomial equation written as

$$Y = C1 * X \wedge 3 + C2 * X \wedge 2 + C3 * X + C4 \qquad (8\text{–}9)$$

has a slope at position X of

$$M = 3 * C1 * X \wedge 2 + 2 * C2 * X + C3 \qquad (8\text{–}10)$$

The general sine curve

$$Y = H * SIN(W * X + D) \qquad (8\text{–}11)$$

Program 8–15 Spinning a line.

```
10 'PROGRAM 8-15. ROTATING LINE AROUND ITS MIDPOINT.
20  SCREEN 1: CLS
30  XC = 160: YC = 100          'CENTER AND ROTATE LINE ABOUT SCREEN MIDPOINT
40  X1 = 150: X2 = 170          'LINE IS 20 UNITS LONG, CENTERED ON MIDPOINT
50  Y1 =  YC: Y2 = YC
60  YADJUST = 5/6: XADJUST = 6/5        'YA & XA ARE RESOLUTION ADJUSTMENTS
70  ANGLE = 15 * 3.14159 / 180          'EXPRESS 15 DEGREES AS RADIANS
80      'CALCULATE CONSTANT PARTS OF ROTATION EQUATIONS
90  COSANGLE = COS(ANGLE)
100 SINX = SIN(ANGLE) * XADJUST
110 SINY = SIN(ANGLE) * YADJUST
120 XE = XC - XC * COSANGLE - YC * SINX
130 YE = YC - YC * COSANGLE + XC * SINY
140     '****************** ROTATE ENDPOINTS & DRAW ****************
150 X1ROTATED = XE + X1 * COSANGLE + Y1 * SINX
160 Y1ROTATED = YE + Y1 * COSANGLE - X1 * SINY
170 X2ROTATED = XE + X2 * COSANGLE + Y2 * SINX
180 Y2ROTATED = YE + Y2 * COSANGLE - X2 * SINY
190     'ERASE OLD LINE
200 DRAWCOLOR = 0
210 LINE (X1,Y1) - (X2,Y2),DRAWCOLOR
220     'DRAW NEW LINE
230 DRAWCOLOR = 3
240 LINE (X1ROTATED,Y1ROTATED) - (X2ROTATED,Y2ROTATED),DRAWCOLOR
250     'SAVE CURRENT POINTS FOR LATER ERASING
260 X1 = X1ROTATED
270 Y1 = Y1ROTATED
280 X2 = X2ROTATED
290 Y2 = Y2ROTATED
300 GOTO 140
310 END
```

has a slope at position X calculated from the COS function:

$$M = H * W * COS(W * X + D) \qquad (8–12)$$

Moving a line tangent to a circle or an ellipse can be accomplished with the rotation transformations.

We can spin a line about any fixed point by performing repeated rotations. Program 8–15 outputs a line rotating about its midpoint. This type of motion could be used to represent the spokes of a rotating wheel. The rotation equations used in this program have been factored and rewritten in a form that reduces computation time.

Moving lines are useful for animating displays in several ways. We can use a moving line in a game application to represent a paddle or racquet and hit bouncing balls. We can also move the component lines of a picture in different ways to represent more complex motion, as in the simulation of a walking figure.

Interactive input can be used to produce object motion along any arbitrary path. Using the keyboard, paddles, or light pen, we could move objects in any direction with any specified speed. In this way, we could set up interactive simulations or games.

8–3 GET AND PUT GRAPHICS STATEMENTS

If we have advanced BASIC, we can do animation with the GET and PUT graphics statements. These statements give us a means for moving objects around without the need to erase and replot individual lines. The idea behind these two statements is that we can define a shape with GET and then move it from one location to another with a PUT statement. We set up animation programs with the following sequence: (1) draw the object on the screen; (2) using GET, store the color values for all points of the object in an array; and (3) PUT the object defined in this array at various screen positions to produce animation.

Objects drawn on the screen can be stored for later animation with the statement

GET(X1,Y1)–(X2,Y2),ARR — Stores the color values of all points within a rectangular screen area into array ARR. The rectangular area is defined by coordinates (X1,Y1) and (X2,Y2), specifying opposite corners.

Array ARR must be numeric and must be specified large enough to hold the color values for all points within the rectangular area. This minimum size, in bytes, is calculated as

$$\text{MINSIZE} = 4 + H * \text{INT}((W * \text{BITS} + 7)/8)$$

Here, H and W denote height and width of the rectangle in pixels: H = Y2 + 1 − Y1 and W = X2 + 1 − X1. BITS is the number of bits needed to specify the color of each pixel, which is 2 in medium resolution and 1 in high resolution. Four bytes are used in ARR to store the values of W and H. We can use an integer array specification for ARR, which gives us 2 bytes per array element, and we can set the array size to the minimum to conserve storage space. For the statement GET(50,50) − (69,79), OBJECT%, we have a 20 by 30 rectangular area (W = 20 and H = 30). In SCREEN 1, BITS = 2, so that MINSIZE is 154 bytes. Since OBJECT% is an integer array with 2 bytes per element, we need to have at least 77 elements in OBJECT% to store the 20 by 30 screen area. The value for W * BITS is stored in OBJECT%(0), and the value for H is stored in OBJECT%(1). The remaining 75 elements store the picture information.

Once we have stored a shape, we can move it around with

PUT(X,Y),ARR,HOW — Displays the pixel pattern stored in array ARR in a rectangular screen area with (X,Y) as the upper left corner. Parameter HOW specifies the manner in which the pattern is displayed on an existing screen background.

There are several options for the parameter HOW that specify different ways for the pixel pattern in ARR to be superimposed onto the screen. We can choose a

logical operator (XOR, OR, or AND) for HOW, or we can use the commands PSET or PRESET. The PSET or PRESET commands cause PUT to erase any patterns already on the screen in the selected area and directly display either the pattern in ARR or its inverse. The logical operators use Boolean operations to combine the color codes of the points in the array to be PUT onto the screen with the color codes of the points already on the screen within the area specified by (X,Y).

Operation XOR (Exclusive Or) has the effect of "inverting" color codes so that two successive PUT statements first place an object on the screen and then erase it without changing the background scene. Program 8–16 demonstrates animation using GET and PUT with the XOR operation by driving a truck along a road lined with telephone poles, as shown in Fig. 8–12. Line 20 in this program provides 47 array elements, which is the minimum size needed for the rectangular area of the truck.

The truck displayed by Prog. 8–16 appears to be traveling "behind" the poles. This occurs because the XOR operator in the PUT statement produces a black pole when the white truck is superimposed on a white pole. In high resolution, color codes for screen points can only be either 0 (for black) or 1 (for white). When we use XOR in SCREEN 2, two codes combine to produce a white screen pixel (code 1) only when one of the codes is 0 and the other is 1. Otherwise, we get a black screen pixel (code 0). Thus, white on white gives black and white on black (or black on white) gives white. Repeating the PUT operation with XOR then restores the screen pattern to its original colors.

We can produce different animation effects by using the other options for HOW. The OR operator combines color codes in SCREEN 2 to produce white when either or both codes are 1 (white). We get black only when both codes are 0 (black). Using AND in SCREEN 2 would give us white only when both codes are

Program 8–16 Moving a truck along a straight-line path, using PUT with the XOR operator.

```
10 'PROGRAM 8-16. MOVING TRUCK (in back of poles) WITH GET AND PUT STATEMENTS
20  DIM TRUCK%(47)
30  SCREEN 2: CLS
40  LINE (96,100) - (126,110),,BF          'DRAW TRUCK BODY
50  LINE (126,105) - (136,110),,BF
60  CIRCLE (101,110),4                     'DRAW WHEELS
70  CIRCLE (131,110),4
80  GET (96,100) - (136,114),TRUCK%        'STORE OBJECT SHAPE
90  CLS
100 LINE (0,65) - (639,65)                 'DRAW ROAD LINED WITH POLES
110 FOR X = 70 TO 580 STEP 80
120     LINE (X,45) - (X,65)
130 NEXT
140 FOR X = 0 TO 576 STEP 48               'ANIMATE TRUCK
150     PUT (X,50),TRUCK%,XOR              'XOR TRUCK IMAGE ONTO SCREEN
160     FOR DELAY = 1 TO 100: NEXT
170     PUT (X,50),TRUCK%,XOR              'XOR TRUCK IMAGE ONTO SCREEN
180 NEXT
190 GOTO 140
200 END
```

Figure 8–12 Driving a truck across the screen from left to right, as animated by Prog. 8–16.

1 (white), and in any other case we get black. The PSET operation simply transfers color codes stored in ARR to the specified screen area, replacing all existing colors. We get a similar effect with PRESET, except color codes are inverted: white is displayed as black, and black is displayed as white. Program 8–16 can be modified so that the truck appears to move in "front" of the poles by using the OR operator in the PUT statement, instead of using XOR. We achieve this effect by replacing lines 150 through 170 in Prog. 8–16 with the statements

```
150  GET (X,50)-(X+40,64),SAVEBACK%  'SAVE COPY OF BACKGROUND
160  PUT (X,50),TRUCK%,OR            'PUT TRUCK IMAGE ONTO SCREEN
170  FOR DELAY = 1 TO 80: NEXT
175  PUT (X,50),SAVEBACK%,PSET       'COPY BACKGROUND TO SCREEN
```

Now we get a white pattern whenever the white truck is superimposed onto a white pole. But OR does not restore the background as XOR does, so we need to PUT the scenery back into place when the truck moves on (line 175). This also means that we must dimension a second array as SAVEBACK%(47) in line 20.

If parameter HOW is omitted in the PUT statement, the display operation defaults to XOR. This default method restores the original screen colors after two PUTs of a defined pattern to the same rectangular screen area, which is usually what we want in an animation method. However, when one object is superimposed with XOR over another object that is not painted in the background color, both objects change color. A red object overlapped with another red object changes to the background color in the overlap area. A green object that is PUT onto a red object turns brown. The results of the various palette color combinations with XOR are listed in Fig. 8–13. Animated objects retain their colors against the background color (code 0), but change color when passing over other objects. Figure 8–13 also shows the color effects produced by the other two logical operators, OR and AND. A variety of color patterns can be created by animating objects against a multicolored background with OR and AND, but the background will not be restored after two repeated operations as it is with XOR.

We can do animation with only the PSET option of PUT, if we do not have a background to worry about. With this method, color codes of ARR are directly transferred to the screen area. All background objects within the rectangular screen area affected by the PUT statement are simply covered over with the colors defined from the GET operation. One PUT statement can be used to move an object and to erase the object from its last position at the same time. We do this by making the array ARR large enough to fit over both positions, as in Prog. 8–17. Here, the left half of the array PLANE% is filled with the background color, while the object is stored in the right half. In this example, we fly an airplane across the

	Array color	Original screen color 0	1	2	3
XOR operator	0	0	1	2	3
	1	1	0	3	2
	2	2	3	0	1
	3	3	2	1	0
OR operator	0	0	1	2	3
	1	1	1	3	3
	2	2	3	2	3
	3	3	3	3	3
AND operator	0	0	0	0	0
	1	0	1	0	1
	2	0	0	2	2
	3	0	1	2	3

Figure 8–13 Table of color codes produced by PUT operations. The resulting color code at a screen position affected by a PUT statement is a Boolean combination of the original color code at that screen position and the color code of the corresponding point in the array specified in the PUT statement. Boolean operators are XOR, OR, and AND.

screen, erasing it from the previous position as we display it in the next position. The array BLANK% is filled with the background color only, and it is used to erase the last display before we start the motion over again from the left. Figure 8–14 illustrates the structure of the array (PLANE%) used to animate the object.

Program 8–17 Motion of an airplane along a straight path, using PUT with the PSET operation.

```
10 'PROGRAM 8-17. AIRPLANE GET AND PUT WITH PSET
20  DIM PLANE%(98),BLANK%(50)
30  SCREEN 1: COLOR 1,0: CLS
40       'DRAW AIRPLANE
50  LINE (16,100) - (31,103),2,BF
60  LINE (26,90) - (29,113),2,BF
70  LINE (18,96) - (20,107),2,BF
80       'IN PLANE%, SAVE PLANE-SIZED BLANK BLOCK ALONG WITH PLANE IMAGE
90       'ALSO SAVE JUST PLANE-SIZED BLANK BLOCK IN BLANK%
100 GET (0,90) - (31,113),PLANE%
110 GET (0,90) - (15,113),BLANK%
120      'PRESS ANY KEY TO STOP PROGRAM
130 STOPSIGN$ = INKEY$
140 WHILE STOPSIGN$ = ""
150      FOR X = 0 TO 288 STEP 16
160          PUT (X,90),PLANE%,PSET        'PUT ERASES OLD PLANE & MAKES NEW PLANE
170          FOR DELAY = 1 TO 100: NEXT
180      NEXT
190      PUT (304,90),BLANK%,PSET          'ERASE RIGHTMOST DISPLAY OF PLANE
200      STOPSIGN$ = INKEY$
210 WEND
220 END
```

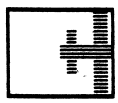


Figure 8–14 Screen display of the area used by array PLANE% in Prog. 8–17 to animate an airplane using the PSET operation in the PUT statement. The left half of the area stored in PLANE% is set to the background color and is used to erase the airplane at the same time that the new position is displayed.

This method of animation can be faster, since we only have one PUT statement at each step. PRESET produces similar results to PSET in the PUT statement, except that color codes 0 to 3 are reversed and displayed as 3 to 0. That is, code 0 will be displayed as code 3, code 1 as code 2, code 2 as code 1, and code 3 as code 0.

Before we can actually begin an animation, we must store the object definitions in arrays. But drawing them on the screen and then storing them with GET produces image flashes at the start of our programs. We can avoid this by assigning the array elements for ARR without displaying the objects first. One way to determine the correct array elements is to run a preliminary program that draws objects, stores them using GET, and then prints out the values of the object arrays. These arrays can then be accessed directly in an animation program that uses only PUT statements. For example, we could replace lines 40 through 110 in Prog. 8–17 with the statements

```
 40  FOR K = 0 TO 98
 45      READ PLANE%(K)
 50  NEXT K

 55  DATA 62,24,0,0,0,-32726,0,0,0
 60  DATA -32726,0,0,0,-32726,0,0,0,-32726
 65  DATA 0,0,0,-32726,0,0,0,-32726,0
 70  DATA 0,42,-32726,0,0,42,-32726,0,0
 75  DATA 42,-32726,0,0,42,-32726,0,512,-21846
 80  DATA -22358,0,512,-21846,-22358,0,512,-21846,-22358
 85  DATA 0,512,-21846,-22358,0,0,42,-32726,0,
 90  DATA 0,42,-32726,0,0,42,-32726,0,0
 95  DATA 42,-32726,0,0,0,-32726,0,0,0
100  DATA -32726,0,0,0,-32726,0,0,0,-32726
105  DATA 0,0,0,-32726,0,0,0,-32726,0
```

The values in these DATA statements correspond to the definition of our red airplane on a blue background inside a 32 by 24 pixel area on the screen. Values for PLANE%(0) and PLANE%(1) are 64 (width multiplied by BITS) and 24 (height). The actual picture specification begins in PLANE%(2), with each array

element corresponding to eight pixel positions across a row. Each new row begins on a byte boundary. In our example, four array elements (or eight bytes) are used for each row. Elements 3, 4, 5, and 6 store colors for the top row; elements 7, 8, 9, and 10 store the second row from the top; and so on.

Since the color information for each row of pixels in our object rectangle is stored starting on a byte boundary, we improve the speed of animations by displaying objects so that screen positions correspond to byte boundaries. We do this by choosing X1 (the coordinate for the top left corner of the rectangular screen area) to be either a multiple of 4, in medium resolution, or a multiple of 8, in high resolution. Then the step size for animating an object with PUT is set to either a multiple of 4 or a multiple of 8.

Another way to use GET and PUT commands in animation is to set up different views, or "frames," of an object with multiple GET statements and then successively PUT the different frames onto the screen to represent the type of motion we want to simulate. For example, motion toward or away from us can be simulated by creating several frames that store different sizes of the object. Each stored frame is scaled from the original object size by a different amount. Objects moving toward us are then displayed so that they get larger. Objects receding into the background will get progressively smaller. We demonstrate this animation technique with Prog. 8–18, using the PUT statement and 12 scaled frames. As shown in Fig. 8–15, the boat sails off into the sunset from a point at the lower right of our screen.

Program 8–18 Animation by scaling, using frames and the GET and PUT commands (sailboat).

```
10 'PROGRAM 8-18. SAILING INTO THE SUNSET.
20      'SCALES SIALBOAT IN RELATION TO A FIXED POINT (-30,70).
30      'REPEATED SCALING MOVES THE BOAT FROM RIGHT TO LEFT AND
40      'SHRINKS IT AS IT RECEDES INTO THE DISTANCE.
50  DIM X(10), Y(10)
60  DIM BOAT1%(886), BOAT2%(716), BOAT3%(572), BOAT4%(485), BOAT5%(405),
        BOAT6%(310), BOAT7%(257), BOAT8%(209), BOAT9%(191), BOAT10%(154),
        BOAT11%(121), BOAT12%(95), BLANK%(886)
70  SCREEN 1: COLOR 1,1: CLS
80  FOR K = 1 TO 9: READ X(K),Y(K): NEXT          'READ DATA POINTS
90  GOSUB 410
100 CLS
110 LINE (0,0) - (319,199),1,B                    'DRAW BORDER
120 LINE (0,60) - (319,60),1                      'DRAW HORIZON
130 PAINT (160,10),1,1                            'PAINT IN SKY
140 CIRCLE (40,60),25,0,0,3.14159                 'DRAW SUN OUTLINE
150 PAINT (40,50),2,0                             'PAINT IN SUN
160 CIRCLE (40,60),25,3,0,3.14159                 'DRAW SUN OUTLINE IN WHITE
170 TIMER = 75
180      'SAIL BOAT INTO SUNSET
190 X = 247
200 GOSUB 350: PUT (X,Y),BOAT1%:  GOSUB 370
210 GOSUB 350: PUT (X,Y),BOAT2%:  GOSUB 370
220 GOSUB 350: PUT (X,Y),BOAT3%:  GOSUB 370
```

Program 8–18 (cont.)

```
230 GOSUB 350: PUT (X,Y),BOAT4%:  GOSUB 370
240 GOSUB 350: PUT (X,Y),BOAT5%:  GOSUB 370
250 GOSUB 350: PUT (X,Y),BOAT6%:  GOSUB 370
260 GOSUB 350: PUT (X,Y),BOAT7%:  GOSUB 370
270 GOSUB 350: PUT (X,Y),BOAT8%:  GOSUB 370
280 GOSUB 350: PUT (X,Y),BOAT9%:  GOSUB 370
290 GOSUB 350: PUT (X,Y),BOAT10%: GOSUB 370
300 GOSUB 350: PUT (X,Y),BOAT11%: GOSUB 370
310 GOSUB 350: PUT (X,Y),BOAT12%
320 GOTO 820
330     '************************************************************
340     'CALCULATE EACH X AND Y DISPLAY POSITION
350 X = X - 20: Y = .1 * X + 60: RETURN
360     'SLOW BOAT DOWN WITH TIMER AND DELAY, THEN ERASE
370 TIMER = TIMER + 50: FOR DELAY = 1 TO TIMER: NEXT
380 PUT (X,Y),BLANK%,PSET: RETURN
390 RETURN
400     '******************** SAIL INTO SUNSET ********************
410 GET (210,97) - (282,189),BLANK%      'GET AREA TO USE FOR ERASING
420 XFIXED = 210: YFIXED = 97
430 HSCALING = .9: VSCALING = .9
440     'CALCULATE CONSTANT PARTS OF SCALING EQUATIONS
450 XCONST = XFIXED * (1 - HSCALING): YCONST = YFIXED * (1 - VSCALING)
460 FOR TIME = 1 TO 12
470     FOR K = 1 TO 9           'SCALE EACH OF 9 POINTS
480         X(K) = INT(X(K) * HSCALING + XCONST + .5)
490         Y(K) = INT(Y(K) * VSCALING + YCONST + .5)
500     NEXT
510     CLS: GOSUB 680           'DRAW NEW POSITION
520             'GET INTO APPROPRIATE ARRAY
530     ON TIME GOTO 540,550,560,570,580,590,600,610,620,630,640,650
540     GET (210,97) - (282,189),BOAT1%: GOTO 660
550     GET (210,97) - (275,180),BOAT2%: GOTO 660
560     GET (210,97) - (269,172),BOAT3%: GOTO 660
570     GET (210,97) - (263,165),BOAT4%: GOTO 660
580     GET (210,97) - (258,158),BOAT5%: GOTO 660
590     GET (210,97) - (253,152),BOAT6%: GOTO 660
600     GET (210,97) - (249,147),BOAT7%: GOTO 660
610     GET (210,97) - (245,142),BOAT8%: GOTO 660
620     GET (210,97) - (242,138),BOAT9%: GOTO 660
630     GET (210,97) - (239,134),BOAT10%: GOTO 660
640     GET (210,97) - (236,130),BOAT11%: GOTO 660
650     GET (210,97) - (233,127),BOAT12%: GOTO 660
660 NEXT
670 RETURN
680 '######################### DRAW SAILBOAT ############################
690 FOR K = 1 TO 8
700     LINE (X(K),Y(K)) - (X(K+1),Y(K+1)),3
710 NEXT
720 XINSAIL = (X(1) + X(2) + X(3)) / 3
730 YINSAIL = (Y(1) + Y(2) + Y(3)) / 3
740 PAINT (XINSAIL,YINSAIL),3,3
750 XINBOAT = (X(4) + X(5) + X(6) + X(7) + X(8) + X(9)) / 6
760 YINBOAT = (Y(4) + Y(5) + Y(6) + Y(7) + Y(8) + Y(9)) / 6
770 PAINT (XINBOAT,YINBOAT),1,3
780 RETURN
790 '############################################################################
800 DATA 250,165,290,165,250,97,250,173,270,183
810 DATA 258,199,215,179,210,157,250,174
820 IF INKEY$ = "" THEN 820
830 END
```

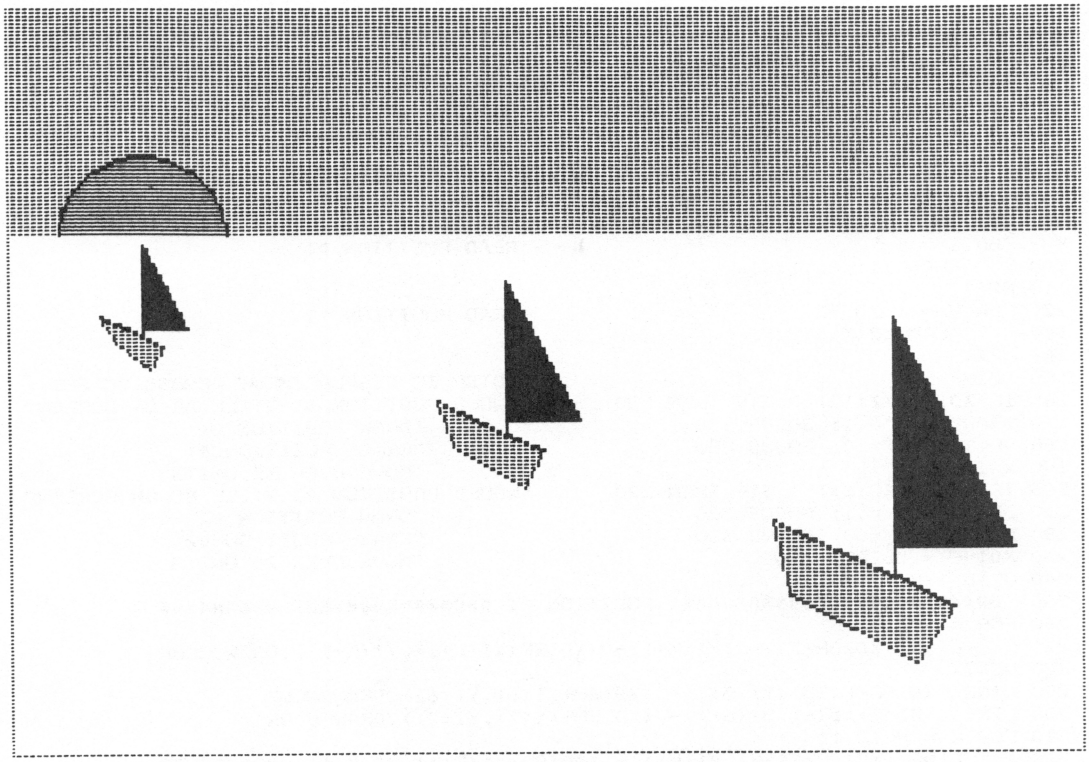


Figure 8–15 Sailing a boat into the sunset. Animation produced by Prog. 8–18 with scaling transformations used to set up the different frames for the motion.

8–4 COMPOUND MOTION

Pictures, for many applications, are animated by moving different parts of the picture at the "same" time. We could have vehicles traveling in different directions, creatures with arms and legs moving, electrical or other networks with "flow" along several paths at once, or complex equipment with multiple moving parts. These motions can be accomplished with coordinated PUT statements, using arrays defining the various shapes, or with the drawing and erasing methods discussed earlier (if we do not have advanced BASIC).

Straightforward drawing and erasing methods, without GET and PUT, produce satisfactory animation for uncomplicated shapes with simple transformation calculations. Translating a small block, for example, by alternating the drawing and erasing commands is about as effective as using the PUT statement. For compound motion, we can set up frames that define different objects or different orientations of a single object and move the frames around. Arrays, storing coordinate endpoints of lines, can be used for the various frames. In Prog. 8–19, we display a running stick figure. Each frame of the motion was plotted on

Program 8–19 Compound motion: running stick figure formed with frames.

```
10 'PROGRAM 8-19. RUNNER
20      'DISPLAYS A RUNNER ALTERNATING BETWEEN 2 POSITIONS (OR
30      'FRAMES) AT LOCATIONS ACROSS THE SCREEN. ARRAYS X1 AND
40      'Y1 HOLD ALL DATA POINTS FOR POSITION #1; X2 AND Y2
50      'HOLD POSITION #2.
60      '*************************************************************
70  DIM X1(15), Y1(15), X2(15), Y2(15)
80  SCREEN 1: CLS
90  FOR K = 1 TO 13                         'READ POSITION #1
100     READ X1(K),Y1(K)
110 NEXT
120 FOR K = 1 TO 12                         'READ POSITION #2
130     READ X2(K),Y2(K)
140 NEXT
150 XDISP = 0                               'XDISP IS DISPLACEMENT ACROSS SCREEN
160 IF XDISP+X1(12) > 319 THEN 520          'WOULD POSITION #1 STILL BE ON SCREEN?
170 DRAWCOLOR = 1: GOSUB 250                        'DRAW POSITION #1
180 DRAWCOLOR = 0: GOSUB 250                        'ERASE POSITION #1
190 XDISP = XDISP + 15                              'MOVE OVER 15 UNITS
200 IF XDISP+X2(11) > 319 THEN 520          'WOULD POSITION #2 STILL BE ON SCREEN?
210 DRAWCOLOR = 1: GOSUB 360                        'DRAW POSITION #2
220 DRAWCOLOR = 0: GOSUB 360                        'ERASE POSITION #2
230 XDISP = XDISP + 20                              'MOVE OVER 20 UNITS
240 GOTO 170
250 '########################## POSITION #1 ##########################
260 FOR K = 1 TO 3
270     LINE (XDISP+X1(K),Y1(K)) - (XDISP+X1(K+1),Y1(K+1)),DRAWCOLOR
280 NEXT
290 LINE (XDISP+X1(5),Y1(5)) - (XDISP+X1(6),Y1(6)),DRAWCOLOR
300 LINE (XDISP+X1(6),Y1(6)) - (XDISP+X1(7),Y1(7)),DRAWCOLOR
310 FOR K = 8 TO 11
320     LINE (XDISP+X1(K),Y1(K)) - (XDISP+X1(K+1),Y1(K+1)),DRAWCOLOR
330 NEXT
340 CIRCLE (XDISP+X1(13),Y1(13)),10,DRAWCOLOR
350 RETURN
360 '########################## POSITION #2 ##########################
370 FOR K = 1 TO 2
380     LINE (XDISP+X2(K),Y2(K)) - (XDISP+X2(K+1),Y2(K+1)),DRAWCOLOR
390 NEXT
400 LINE (XDISP+X2(4),Y2(4)) - (XDISP+X2(5),Y2(5)),DRAWCOLOR
410 LINE (XDISP+X2(5),Y2(5)) - (XDISP+X2(6),Y2(6)),DRAWCOLOR
420 FOR K = 7 TO 10
430     LINE (XDISP+X2(K),Y2(K)) - (XDISP+X2(K+1),Y2(K+1)),DRAWCOLOR
440 NEXT
450 CIRCLE (XDISP+X2(12),Y2(12)),10,DRAWCOLOR
460 RETURN
470 '*************************************************************************
480 DATA 14,150,20,133,15,120,20,93,5,145,25,133,15,120
490 DATA 20,115,10,110,19,92,20,108,30,113,20,83
500 DATA 2,132,25,136,40,93,43,150,50,130,30,120
510 DATA 30,111,22,103,38,95,43,110,58,104,40,83
520 END
```

graph paper, then converted into screen coordinates to represent the running motion for the arms and legs. The two frames for each position of the runner are separately stored in arrays. This allows us to simply translate each frame, alternately, across the screen without recomputing relative positions of the individual figure parts. Figure 8–16 illustrates the resulting motion.

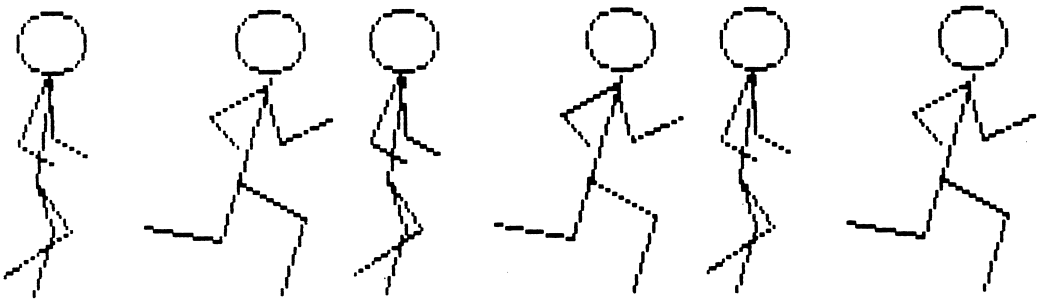


Figure 8–16 Compound motion, showing two frames used to display a running stick figure in Prog. 8–19.

Frames that represent different object orientations can be defined similarly for PUT methods. Suppose that we want to roll a wagon across the screen. With GET, we can set up several arrays, showing the wagon wheels in different rotated positions. Program 8–20 outputs a moving wagon (Fig. 8–17), animated with three frames. The wheels of the wagon rotate as the wagon is translated to the left.

Program 8–20 Compound motion: moving wagon with turning wheels.

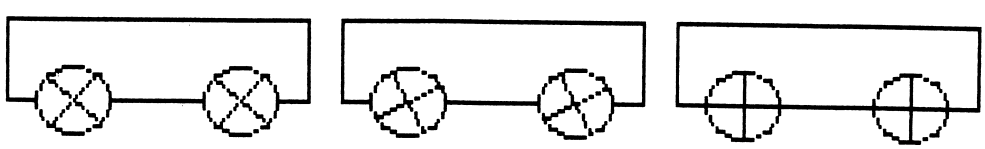
```
10 'PROGRAM 8-20. MOVING WAGON AND WHEELS WITH GET AND PUT
20      'DRAWS WAGON AND THEN MOVES IT FROM RIGHT TO LEFT.
30  DIM X(20), Y(20), POSITION1%(434), POSITION2%(434), POSITION3%(434)
40  SCREEN 1: COLOR 1,1: CLS
50  YADJUST = 5/6: XADJUST = 6/5
60  FOR K = 1 TO 18
70      READ X(K), Y(K)
80  NEXT
90  ANGLE = 25 * 3.14159 / 180          'EXPRESS 25 IN RADIANS
100     'CALCULATE CONSTANT PARTS OF ROTATION EQUATIONS
110 COSA = COS(ANGLE)
120 SINX = SIN(ANGLE) * XADJUST: SINY = SIN(ANGLE) * YADJUST
130     'DRAW VARIOUS POSITIONS OF WAGON AND GET
140 GOSUB 310                       'DRAW WAGON
150 GET (202,95) - (308,126),POSITION1%
160 CLS: GOSUB 420: GOSUB 310
170 GET (202,95) - (308,126),POSITION2%
180 CLS: GOSUB 420: GOSUB 310
190 GET (202,95) - (308,126),POSITION3%
200     'MOVE WAGON ACROSS SCREEN
210 CLS
220 FOR X = 212 TO 32 STEP -48
230     PUT (X,95),POSITION1%,PSET
240     FOR DELAY = 1 TO 100: NEXT
250     PUT (X - 16,95),POSITION2%,PSET
260     FOR DELAY = 1 TO 100: NEXT
270     PUT (X - 32,95),POSITION3%,PSET
280     FOR DELAY = 1 TO 100: NEXT
```

Program 8–20 (cont.)

```
290 NEXT
300 GOTO 620
310 '############################ DRAW ROUTINE ############################
320 FOR K = 1 TO 5
330     LINE (X(K),Y(K)) - (X(K+1),Y(K+1))
340 NEXT
350 LINE (X(7),Y(7)) - (X(8),Y(8))
360 CIRCLE (X(9),Y(9)),11          'WHEEL CENTERS ARE ARRAY ELEMENTS 9 AND 10
370 CIRCLE (X(10),Y(10)),11
380 FOR K = 11 TO 17 STEP 2        'DRAW SPOKES
390     LINE (X(K),Y(K)) - (X(K+1),Y(K+1))
400 NEXT
410 RETURN
420 '######################## ROTATE SPOKES ROUTINE ######################
430 YO = Y(9)                      'Y VALUE OF CENTER OF BOTH WHEELS
440 XO = X(9)                      'X VALUE OF CENTER OF FIRST WHEEL
450 FOR K = 11 TO 14
460     XSAVE = X(K)               'SAVE X(K) FOR USE IN Y(K) CALCULATIONS
470     X(K) = XO + (X(K) - XO) * COSA + (Y(K) - YO) * SINX
480     Y(K) = YO + (Y(K) - YO) * COSA - (XSAVE - XO) * SINY
490 NEXT
500 XO = X(10)                     'X VALUE OF CENTER OF SECOND WHEEL
510 FOR K = 15 TO 18
520     XSAVE = X(K)               'SAVE X(K) FOR USE IN Y(K) CALCULATIONS
530     X(K) = XO + (X(K) - XO) * COSA + (Y(K) - YO) * SINX
540     Y(K) = YO + (Y(K) - YO) * COSA - (XSAVE - XO) * SINY
550 NEXT
560 RETURN
570 '############################################################################
580 DATA 210,117,202,117,202,95,292,95,292,117,284,117,260,117,234,117
590 DATA 222,117,272,117
600 DATA 232,117,212,117,222,108,222,125
610 DATA 282,117,262,117,272,108,272,125
620 END
```

Several moving objects in one display are handled by drawing all the objects at once, computing all transformed positions, and then erasing each object and displaying it in the new position. Each object is defined in a separate array, as demonstrated in Prog. 8–21. This program flies the three airplanes shown in Fig. 8–18 across the screen along randomly selected lines.

Figure 8–17 Compound motion. Animating a wagon with translation and rotating wheels (Prog. 8–20).



Program 8–21 Compound motion: multiple objects (airplanes) moved along random horizontal and vertical paths with PUT statements.

```
10 'PROGRAM 8-21. FLYING THREE AIRPLANES WITH GET AND PUT
20  SCREEN 1: COLOR 1,0: CLS
30  DIM PLANE1%(98), PLANE2%(98), PLANE3%(98),HBLANK%(150), VBLANK%(150)
40  GOSUB 220                                   'GET PLANE SHAPES
50  Y1 = 90: Y2 = 90: X1 = 90                   'INITIAL POSITIONS OF PLANES
60  WHILE INKEY$ = ""
70      Y = 0
80      FOR X = 0 TO 285 STEP 15
90          PUT (X, Y1), PLANE1%, PSET
100         PUT (285 - X, Y2), PLANE2%, PSET
110         PUT (X1, Y), PLANE3%, PSET
120         FOR DELAY = 1 TO 50: NEXT
130         Y = Y + 8
140     NEXT
150     PUT (300,Y1),HBLANK%,PSET               'CLEAR PLANES AT EDGE OF SCREEN
160     PUT (  0,Y2),HBLANK%,PSET
170     PUT (X1,167),VBLANK%,PSET
180     Y1 = INT(RND*173)+1: Y2 = INT(RND*173)+1: X1 = INT(RND*280)+1
190 WEND
200 GOTO 410
210     '************ GETS PLANES FROM SCREEN DRAWINGS ********
220 LINE (15,100)-(30,103),2,BF
230 LINE (25, 90)-(28,113),2,BF
240 LINE (17, 96)-(19,107),2,BF
250     'GET PALNE 1
260 GET (0,90)-(30,113),PLANE1%
270 LINE (285,100)-(300,103),3,BF
280 LINE (287, 90)-(290,113),3,BF
290 LINE (296, 96)-(298,107),3,BF
300     'GET PLANE 2
310 GET (285,90)-(315,113),PLANE2%
320 LINE (100,15)-(103,30),1,BF
330 LINE ( 90,25)-(113,28),1,BF
340 LINE ( 96,17)-(107,19),1,BF
350     'GET PLANE 3
360 GET (90,0)-(113,30),PLANE3%
370     'GET BLANK SPACES TO ERASE AT EDGE OF SCREEN
380 GET (0,0)-(23,15),VBLANK%
390 GET (0,0)-(15,23),HBLANK%
400 RETURN
410 END
```

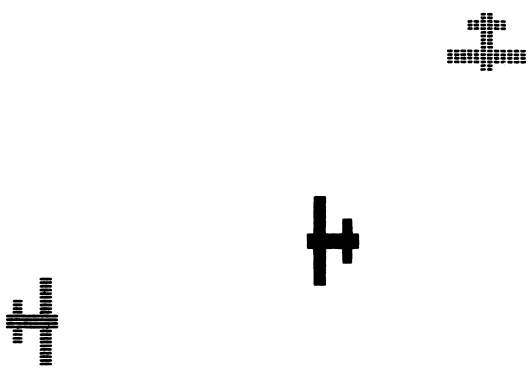


Figure 8–18 Three airplanes animated by Prog. 8–21.

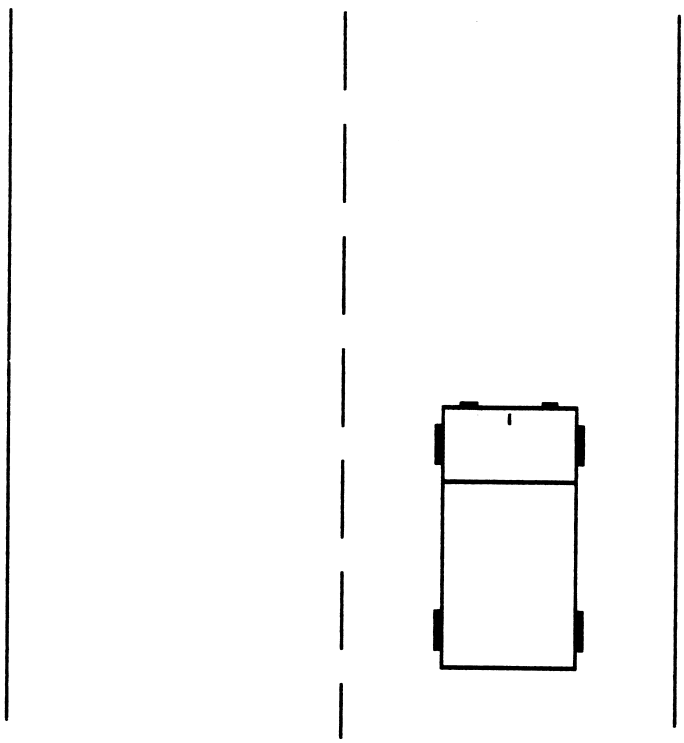
8–5 BACKGROUND MOTION

As the number of computations and lines to be drawn increases, the time required to animate a scene can increase significantly. For complex objects and complicated motions, the animation time can become so long that we lose the sense of motion we were trying to create on the screen. We can speed up the motion by animating only simple shapes that have fewer lines to be drawn, and we can limit the animation to simple types of motion. Another method, which is appropriate for some applications, is to move the background in a scene to give the appearance of motion.

Figure 8–19 illustrates a simple form of background motion. We move the centerline of the road down the screen, so that the car appears to move up. The PUT statement with PSET is used in Prog. 8–22 to produce the two alternating positions of the centerline. Pressing keys on the numeric keypad during program execution speeds up and slows down the motion by changing the value of the time-delay factor TIMER.

Moving the telephone poles in Fig. 8–20 to the right causes the truck to

Figure 8–19 Simulation of the car movement up the screen is produced by moving the center lines of the road down (Prog. 8–22).



Program 8–22 Background motion: moving centerlines on a road.

```
10 'PROGRAM 8-22. MOVING ROAD BACKGROUND WITH GET AND PUT
20  SCREEN 1: CLS
30  DIM R1%(190), R2%(190)
40  GOSUB 180              'DRAW CAR
50  GOSUB 310              'DRAW CENTERLINES AND GET BOTH
60      'MOVE CENTERLINE
70  TIMER = 100
80  WHILE A$ <> "Q" AND A$ <> "q"
90      PUT (200,5),R2%,PSET
100     FOR DELAY = 1 TO TIMER: NEXT
110     PUT (200,5),R1%,PSET
120     FOR DELAY = 1 TO TIMER: NEXT
130     A$ = INKEY$: IF A$ <> "" THEN A = VAL(A$)
140             'SPEED IS CHANGED BY PRESSING KEYS 1 - 9 ON KEYBOARD
150     IF A >= 0 AND A < 10 THEN TIMER = 300 - A * 30          'SET SPEED
160 WEND
170 GOTO 440
180     'DRAWS CAR
190 LINE (100,5) - (100,195)            'DRAW ROAD EDGES
200 LINE (300,5) - (300,195)
210 LINE (230,110)-(270,180),,B
220 LINE (230,130)-(270,130)
230 LINE (250,112)-(250,114)            'HOOD ORNAMENT
240 LINE (228,115)-(229,125),,BF        'WHEELS
250 LINE (228,165)-(229,175),,BF
260 LINE (271,165)-(272,175),,BF
270 LINE (271,115)-(272,125),,BF
280 LINE (235,109)-(240,109)            'HEADLIGHTS
290 LINE (260,109)-(264,109)
300 RETURN
310     'DRAWS CENTERLINES AND LOADS INTO ARRAYS
320 FOR K = 5 TO 155 STEP 30
330     LINE (200,K) - (200,K+20)
340 NEXT
350 LINE (200,185) - (200,195)
360 GET(200,5) - (200,195),R1%          'GET ROAD POSITION #1
370 PUT (200,5),R1%
380 LINE (200,5) - (200,15)
390 FOR K = 25 TO 145 STEP 30
400     LINE (200,K) - (200,K+20)
410 NEXT K
420 GET (200,5) - (200,195),R2%         'GET ROAD POSITION #2
430 RETURN
440 END
```

appear to travel to the left. If the poles are meant to be on the far side of the road, they could be spaced and positioned so that they are never drawn on top of the truck. Otherwise, they will appear in the foreground. As an alternative method, we can draw only part of the background at the points where the truck and poles overlap in the animation. Program 8–23 sets up four views of the telephone poles. Three arrays are used to store the pole as it would be seen in various positions in back of the truck. The other array stores the pole full size. The poles are moved across the screen so as to maintain a constant spacing between them. A time-delay factor TIMER is again used to adjust the speed of the motion.

Many types of background motion are possible. Program 8–24 combines a

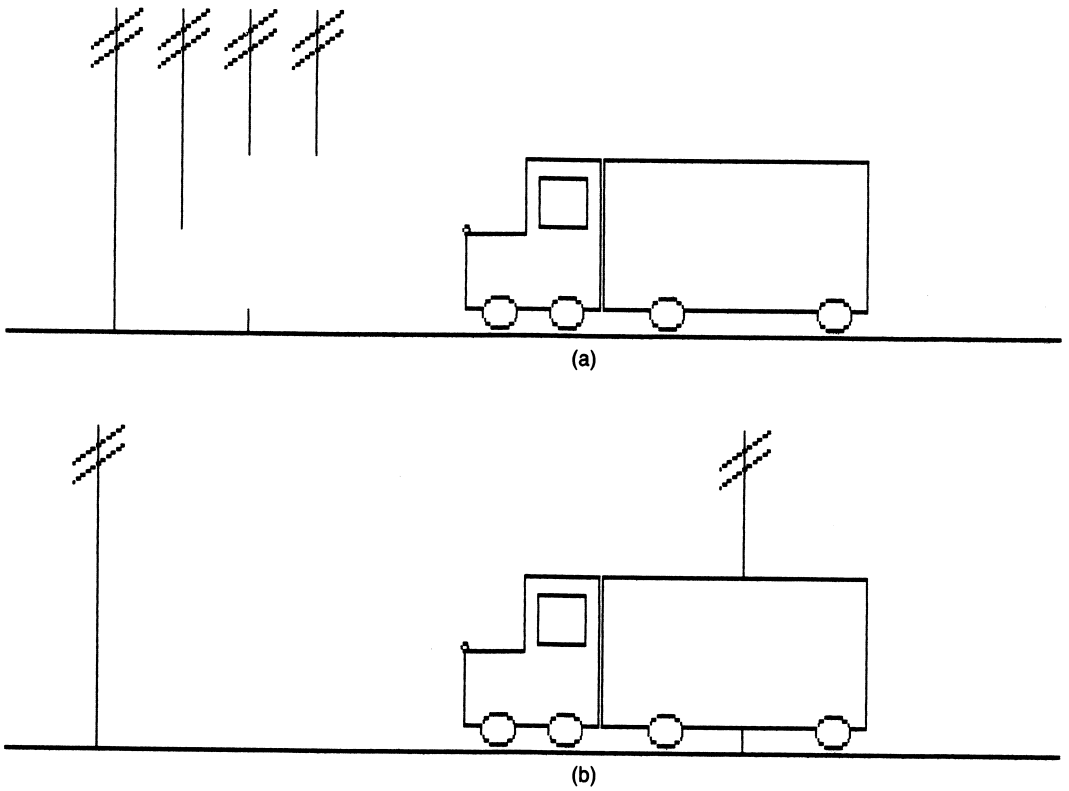


Figure 8–20 (a) Four views of the telephone poles used in Prog. 8–23 to produce background motion. (b) Simulation of a truck traveling to the left is accomplished by moving the telephone poles to the right.

Program 8–23 Background motion: moving telephone poles past an object on a road.

```
10 'PROGRAM 8-23. GET & PUT BACKGROUND MOVEMENT OF TELEPHONE POLES
20  DIM POLE1%(374),POLE2%(374),POLE3%(374),POLE4%(374)
30  CLS:SCREEN 2
40  GOSUB 280              'DRAW SCENE
50  TIMER = 100
60  GET (55,65)-(85,150),POLE1%          'GET FOUR DIFFERENT POLES
70  GET (95,65)-(125,150),POLE2%
80  GET (135,65)-(165,150),POLE3%
90  GET (175,65)-(205,150),POLE4%
100 PUT (55,65),POLE1%                   'ERASE THE POLES THAT ARE ON THE SCREEN
120 PUT (95,65),POLE2%
110 PUT (95,65),POLE2%
120 PUT (135,65),POLE3%
130 PUT (175,65),POLE4%
140 X1 = 50: X2 = 436            'SET INITIAL PUT POSITIONS
150 A$ = ""
160 WHILE A$ <> "Q" AND A$ <> "q"
170     X = X1: GOSUB 460
```

Program 8–23 (cont.)

```
180      X = X2: GOSUB 460
190      FOR DELAY = 1 TO TIMER: NEXT
200      X = X1: GOSUB 460          'PUT AGAIN TO ERASE POLES
210      X = X2: GOSUB 460
220      X1 = (X1 + 80) MOD 805
230      X2 = (X2 + 80) MOD 805
240      A$ = INKEY$
250      IF A$ = " " THEN TIMER = TIMER - 25      'PRESS SPACE BAR TO SPEED UP
260 WEND
270 GOTO 620
280      'DRAWS SCENE
290 LINE (5,151)-(635,151)
300 CIRCLE (300,145),10             'DRAW WHEELS
310 CIRCLE (340,145),10
320 CIRCLE (400,145),10
330 CIRCLE (500,145),10
340 FOR TIME = 1 TO 20
350      READ X1,Y1,X2,Y2: LINE (X1,Y1) - (X2,Y2)
360 NEXT
370 FOR TIME = 1 TO 4
380      READ X2,Y2: LINE - (X2,Y2)
390 NEXT
400 READ X1,Y1,X2,Y2: LINE (X1,Y1) - (X2,Y2)
410 READ X2,Y2: LINE - (X2,Y2)
420 READ X2,Y2: LINE - (X2,Y2)
430 LINE (323,109)-(352,122),,B
440 CIRCLE (280,123),2
450 RETURN
460      '*************** PUTS A POLE ******************
470 XC = X + 15
480 IF XC > 609 THEN 530
490 IF XC < 280 OR XC > 520 THEN PUT (X,65),POLE1%: GOTO 530
500 IF (XC>330 AND XC<350) OR (XC>390 AND XC<410) OR (XC>490 AND XC<510) THEN
      PUT (X,65),POLE4%: GOTO 530
510 IF XC >= 315 AND XC < 520 THEN PUT (X,65),POLE3%: GOTO 530
520 IF XC >= 280 AND XC < 315 THEN PUT (X,65),POLE2%
530 RETURN
540 DATA 280,144,290,144,310,144,330,144,350,144,360,144,362,144,390,144
550 DATA 410,144,490,144,510,144,520,144
560 DATA 70,65,70,150,55,75,85,65,55,80,85,70
570 DATA 110,65,110,123,95,75,125,65,95,80,125,70
580 DATA 150,65,150,103,150,145,150,150,135,75,165,65,135,80,165,70
590 DATA 190,65,190,103,175,75,205,65,175,80,205, 70
600 DATA 280,144,280,124,315,124,315,104,360,104,360,144
610 DATA 362,144,362,104,520,104,520,144
620 END
```

Program 8–24 Simulating movement with background motion: train with moving rod and moving tracks.

```
10 'PROGRAM 8-24. TRAIN WITH MOVING WHEEL BAR AND TRACKS.
20      'DRAWS A TRAIN AND ROTATES THE HORIZONTAL BAR CONNECTING
30      'THE TWO REAR WHEELS. ALSO MOVES POINTS ACROSS THE BOTTOM
40      'OF THE TRAIN, TO SIMULATE TRAIN TRACKS.
50      '************************************************************
60 SCREEN 1: COLOR 1,1: CLS
70 YADJUST = 5/6                        'YA IS RESOLUTION ADJUSTMENT
80      'DRAW TRAIN
```

```
90  READ X1,Y1: PSET (X1,Y1)
100 FOR K = 1 TO 25
110     READ X2,Y2: LINE - (X2,Y2)
120 NEXT
130 READ XL, XR, YT, YB: LINE (XL,YT) - (XR,YB),,BF      'WINDOW
140 FOR K = 1 TO 6                                       'ADD DETAIL LINES
150     READ X1, Y1, X2, Y2: LINE (X1,Y1) - (X2,Y2)
160 NEXT
170 FOR K = 1 TO 4                                       'ADD WHEELS
180     READ XC, YC, R
190     CIRCLE (XC,YC),R        'FINAL XC AND YC VALUES ARE FOR REAR WHEEL
200 NEXT
210     '****************** ROTATE WHEEL BAR, MOVE TRACKS ******************
220 RADIUS = 15                          'RADIUS OF WHEEL BAR'S ROTATION
230 DA = 50  * 3.14159 / 180
240 A$ = ""
250 WHILE A$ = ""
260     FOR ANGLE = DA TO 6.28318 STEP DA
270         XBAR = XC + RADIUS * SIN(ANGLE)
280         YBAR = YC + RADIUS * COS(ANGLE) * YADJUST
290         LINE (XBAR,YBAR) - (XBAR-90,YBAR)    'DRAW NEW BAR POSITION
300         FOR X = XSTART TO 319 STEP 35        'PLOT TRACKS 35 UNITS APART
310             PSET (X,152)
320         NEXT
330         FOR X = XSTART TO 319 STEP 35        'ERASE TRACKS
340             PRESET (X,152)
350         NEXT
360         XSTART = XSTART + 7          'NEXT POINT SET WILL BE 7 PIXELS OVER
370         IF XSTART >= 30 THEN XSTART = 0      'POINTS KEEP COMING FROM LEFT
380         LINE (XBAR,YBAR) - (XBAR-90,YBAR),0          'ERASE CURRENT POSITION
390     NEXT
400     A$ = INKEY$
410 WEND
420 '###########################################################################
430     'OUTLINE
440 DATA 270,130,290,130,290,50,300,50,300,30,220,30,220,70,200,70
450 DATA 200,50,170,50,170,70,90,70,90,50,100,40,60,40,70,50,70,70
460 DATA 60,70,50,80,50,110,60,120,40,120,20,150,50,150,50,130,90,130
470     'WINDOW
480 DATA 230,280,40,70
490     'DETAIL LINES
500 DATA 220,70,220,100,220,100,65,100
510 DATA 65,100,45,125,130,130,140,130
520 DATA 0,154,319,154,180,130,230,130
530     'WHEELS
540 DATA 65,140,10,110,130,20,160,130,20,250,130,20
550 END
```

background motion with simple object motion to simulate animation of a more complex object. The output is given in Fig. 8–21. A rotating bar connecting the wheels of a train and a series of constantly moving railroad tracks, made up of single pixels, are used to simulate the motion of a train. We can produce slower or faster motion of the background by taking smaller or larger displacements in the position of background objects. We can also slow the motion as much as we want with time delays between frames.

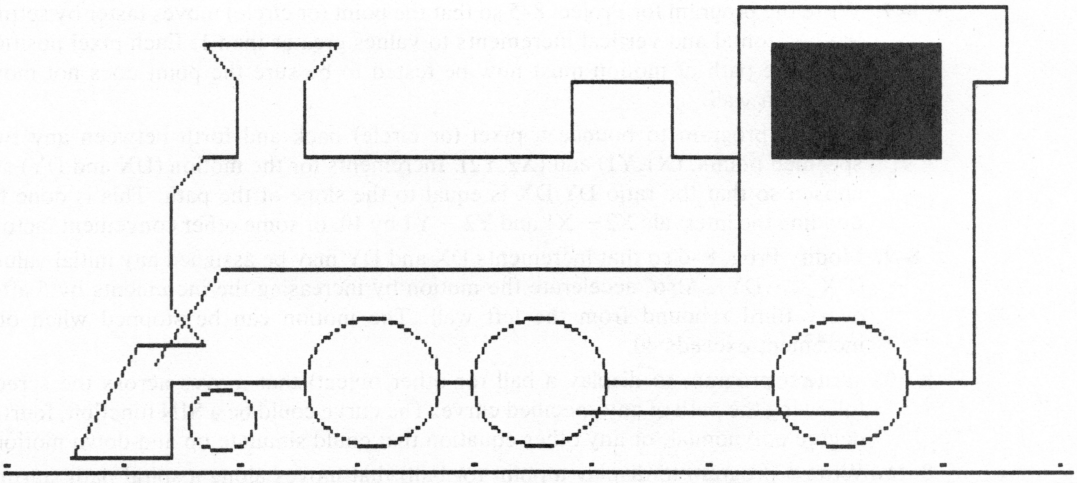


Figure 8–21 Simulation of a train chugging to the left is produced by Prog. 8–24 with a moving wheel rod and moving tracks.

PROGRAMMING PROJECTS

8–1. Animate a character by bouncing it off several randomly placed horizontal and vertical boundaries. Start the character moving along a diagonal line, with both R and C incremented by 1 unit at each step. When the character encounters a vertical wall (or the top or bottom of the screen), reverse the row increment. When the character encounters a horizontal wall (or a side of the screen), reverse the column increment. Use the SCREEN function to test for wall collisions.

8–2. Modify the program in Project 8–1 to display the character in various colors as it bounces around the screen. Other possible modifications include changing colors only when it rebounds, not erasing the character (so that a pattern of colors is displayed), or producing a sound whenever the character rebounds from a boundary.

8–3. Modify the program in Project 8–1 to move the character faster by setting the row and column increments to some value greater than 1 (keeping both increments the same). Each character position along the path, from the current position to the possible final position, must now be tested to determine whether there is a wall along the line of motion.

8–4. Revise Prog. 8–3 to display the airplane as it flies across the screen, using the SCREEN statement and the pages available in WIDTH 40. When the space bar is pressed, the block must be added to each page before the page is displayed and a test must be made for a collision between the block and the plane.

8–5. Write the program of Project 8–1 for pixel motion (a point or circle). Use the POINT function to test for wall collisions.

8–6. Write the program for Project 8–5 with one or more of the following options:

(a) Display the point (or circle) in random colors and do not erase it as it moves so that color patterns are produced.

(b) Change colors or make a sound when the point rebounds.

8–7. Write the program for Project 8–5 so that the point (or circle) moves faster by setting the horizontal and vertical increments to values greater than 1. Each pixel position along the path of motion must now be tested to be sure the point does not move through a wall.

8–8. Write a program to bounce a pixel (or circle) back and forth between any two specified points, (X1,Y1) and (X2,Y2). Increments for the motion (DX and DY) are chosen so that the ratio DY/DX is equal to the slope of the path. This is done by dividing the intervals X2 − X1 and Y2 − Y1 by 10, or some other convenient factor.

8–9. Modify Prog. 8–6 so that increments DX and DY may be assigned any initial values (DX <> DY). Also, accelerate the motion by increasing the increments by 5 after every third rebound from the left wall. The motion can be stopped when one increment exceeds 40.

8–10. Write a program to display a ball (or other object) that moves across the screen following the path of any specified curve. The curve could be a SIN function, fourth-degree polynomial, or any other equation that could simulate up-and-down motion.

8–11. Write a program to display a point (or ball) that moves along a spiral path starting from the center of the screen. By alternating the color and not erasing the point, the screen can be filled with a color spiral. By reversing the motion when the point reaches the edge of the screen, the point can be made to spiral in and out repeatedly.

8–12. Write a program to animate a spaceship using only the PUT statement. The array for the PUT command is to be defined in DATA statements, as discussed with Prog. 8–17, and the picture definition and display positions are to be set on byte boundaries. Move the spaceship along a straight-line path.

8–13. Set up the program for Project 8–12 so that the path of motion is controlled by a joystick.

8–14. Modify Prog. 8–19 so that the runner recedes into the distance while moving across the screen. This can be done by successively scaling each frame of the motion.

8–15. Write a program to animate an object along the path of a sine curve or a third-degree polynomial by keeping a line in the object tangent to the curve as it moves.

8–16. Write a program that will display an airplane or spacecraft flying loops around a circle.

8–17. Revise Prog. 8–21 so that the three airplanes start across the screen at different, randomly chosen times.

8–18. Modify Project 8–17 so that the airplanes disintegrate if they collide. This can be accomplished by replacing each colliding airplane with a random dot pattern that spreads out from the center of each plane, with the number of dots decreasing rapidly as the pattern expands. If any of this debris hits the third plane, it also disintegrates and the program ends.

8–19. Write a program to display a clock face in the center of the screen. Draw the hands as arrows (for hours and minutes) and rotate the hands so that the smaller hand moves through 30 degrees each time the larger hand makes one complete revolution (360 degrees). A second hand can be added that sweeps through 360 degrees for each 6 degrees of rotation of the minute hand.

Chapter 9

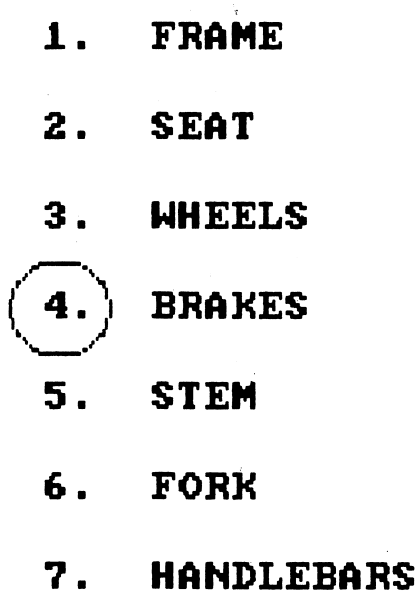
Windows and Spotlights

Graphics displays can be manipulated in many ways. We have explored methods for changing display size and for moving the figures from one screen location or orientation to another. These transformations can be applied to a total display or to designated areas within a display. We can also select areas of a display for other types of modification. The areas could be "spotlighted" for special emphasis, they could be deleted from the display, or they could be retained and transformed while the rest of the display is deleted.

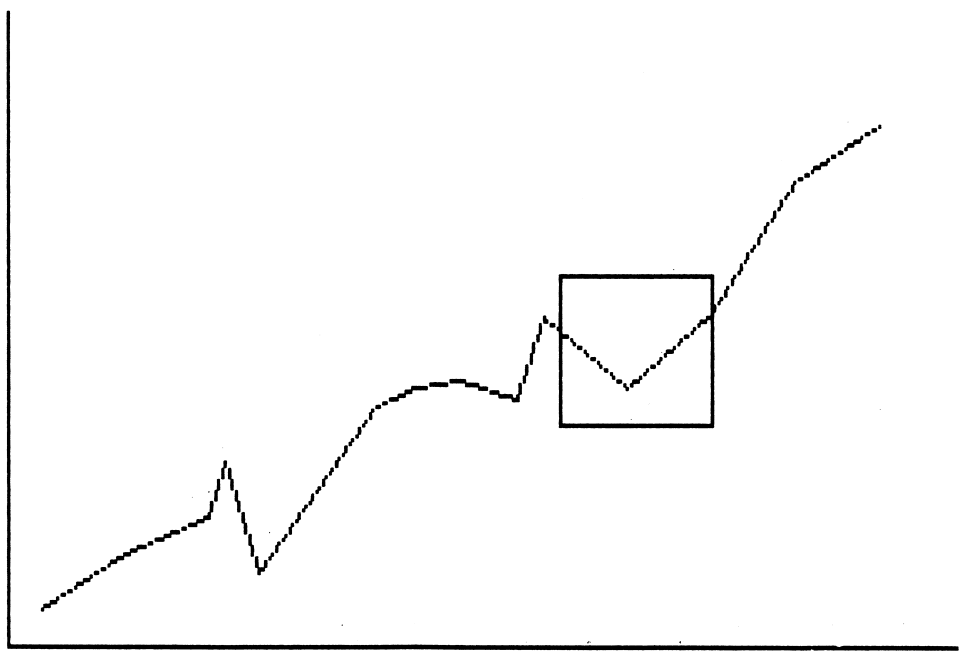
9–1 SPOTLIGHTING

We would often like to add special emphasis to some area of a display in order to focus attention there. The area to be spotlighted could be redrawn in a bright color or highlighted with added intensity. With the PC, a high-intensity white option is available that has the effect of making displayed text brighter than other parts of the screen. A similar effect can be produced in graphics mode by drawing lines in the spotlight area with double thickness: drawing two lines 1 unit apart instead of a single line. Blinking a section of a display (in text mode) is another technique we can use to emphasize a screen area. We can also draw attention to a selected area by putting a circle or box around it, as shown in Fig. 9–1.

To superimpose a circle onto a display, we simply choose the center coordinates (XC,YC) and radius R for the circle. A circle-drawing routine or the CIRCLE command is then added to our program and referenced each time we wish to spotlight an area. We can write programs that allow us to experiment with different circle locations and sizes to achieve the exact spotlight effect desired, as in Prog. 9–1.



(a)



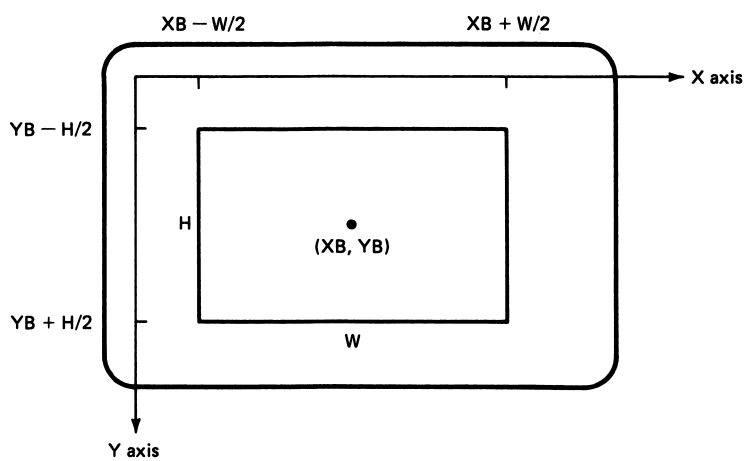
(b)

Figure 9–1 Spotlighting displays with (a) a circle and (b) a box.

Program 9–1 Spotlighting with circles.

```
10 'PROGRAM 9-1. CIRCLE SPOTLIGHTS.
20  SCREEN 1: CLS
30      '***************** READ DATA POINTS FOR DISPLAY *********
40      '
50      '
60      '
70 GOSUB 100              'DRAW DISPLAY
80 GOSUB 150              'CREATE CIRCLE SPOTLIGHT
90 GOTO 340
100     '********************** DRAW ROUTINE **********************
110     '
120     '
130     '
140 RETURN
150     '****************** ADD CIRCLE SPOTLIGHT *****************
160 LOCATE 2,1: INPUT "CIRCLE CENTER AND RADIUS"; XCENTER,YCENTER,RADIUS
170 IF XCENTER+RADIUS <= 319 AND XCENTER-RADIUS >= 0 AND YCENTER+RADIUS <= 199
      AND YCENTER-RADIUS >= 0 THEN 210
180 LOCATE 1,1: PRINT "CIRCLE OFF SCREEN"
190 PRINT STRING$(80," ");                'ERASE INSTRUCTION AND PREVIOUS INPUT
200 GOTO 160
210 CIRCLE (XCENTER,YCENTER),RADIUS
220 LOCATE 1,1: PRINT STRING$(160," ");          'ERASE ANY INSTRUCTIONS
230 LOCATE 2,1: INPUT "TYPE E TO END, C TO CHANGE CIRCLE"; M$
240 LOCATE 2,1: PRINT STRING$(80," ");           'ERASE INSTRUCTIONS
250 IF M$ = "E" THEN 330
260 CIRCLE (XCENTER,YCENTER),RADIUS,0            'ERASE CURRENT POSITION
270    'DID ERASING DESTROY DISPLAY?
280 LOCATE 2,1: INPUT "DO YOU WANT TO RE-DRAW DISPLAY (Y/N)"; D$
290 LOCATE 2,1: PRINT STRING$(80," ");
300 IF D$ = "N" THEN 150
310 GOSUB 100                                    'REDRAW DISPLAY
320 GOTO 150
330 RETURN
340 END
```

Figure 9–2 Coordinates for the corners of a rectangle can be determined from values for the width W, height H, and center position (XB,YB).



For a box spotlight, we could use the LINE command and input two opposite box corners. We could also devise a routine that simply asks us where to put the center of the rectangle and how big it is to be (Fig. 9–2). This type of routine is illustrated in Prog. 9–2, which draws a box centered at coordinates (XB,YB) with a width W and a height H.

Spotlights with circles or boxes are useful for emphasizing parts of displays that accompany reports or presentations. A screen display with the spotlight in different locations can be output to a printer each time the spotlight location is changed. The printed displays can then be included in a report. We can make a moving spotlight to accompany a presentation, as well. The spotlight would change locations to coincide with the display area under discussion in the

Program 9–2 Spotlighting with rectangles.

```
10 'PROGRAM 9-2. BOX SPOTLIGHTS.
20  SCREEN 1: CLS
30      '**************** READ DATA POINTS FOR DISPLAY *********
40      '
50      '
60      '
70  GOSUB 100              'DRAW DISPLAY
80  GOSUB 150              'CREATE CIRCLE SPOTLIGHT
90  GOTO 420
100     '****************** DRAW ROUTINE **********************
110     '
120     '
130     '
140 RETURN
150     '*************** ADD BOX SPOTLIGHT ********************
160 LOCATE 2,1: INPUT "BOX CENTER (X AND Y)"; XBOX,YBOX
170 LOCATE 2,1: PRINT STRING$(80," ");          'ERASE INSTRUCTION
180 LOCATE 2,1: INPUT "BOX WIDTH AND HEIGHT"; BWIDTH, BHEIGHT
190     'DETERMINE BOX SIDES (LEFT, RIGHT, TOP AND BOTTOM)
200 LEFT = XBOX - BWIDTH / 2
210 RIGHT = XBOX + BWIDTH / 2
220 TOP = YBOX - BHEIGHT / 2
230 BOTTOM = YBOX + BHEIGHT / 2
240 IF LEFT < RIGHT AND LEFT >= 0 AND RIGHT <= 319 AND TOP < BOTTOM AND
    TOP >= 0 AND BOTTOM <= 199 THEN 280
250 LOCATE 1,1: PRINT "BOX OFF SCREEN"
260 PRINT STRING$(80," ");              'ERASE INSTRUCTION AND PREVIOUS INPUT
270 GOTO 150
280 LINE (LEFT,TOP) - (RIGHT,BOTTOM),,B         'DRAW BOX
290 LOCATE 1,1: PRINT STRING$(160," ");
300 LOCATE 2,1: INPUT "TYPE E TO END, C TO CHANGE BOX"; M$
310 LOCATE 2,1: PRINT STRING$(80," ");
320 IF M$ = "E" THEN 400
330 LINE (LEFT,TOP) - (RIGHT,BOTTOM)            'ERASE CURRENT POSITION
340     'DID ERASING DESTROY DISPLAY?
350 LOCATE 2,1: INPUT "DO YOU WANT TO RE-DRAW DISPLAY (Y/N)"; D$
360 LOCATE 2,1: PRINT STRING$(80," ");
370 IF D$ = "N" THEN 150
380 GOSUB 100                                   'REDRAW DISPLAY
390 GOTO 150
400 RETURN
410 '##########################################################################
420 END
```

presentation. We could use some type of time delay to hold the spotlight in one position until we were ready to discuss the next area. The areas spotlighted in this way might be some type of list, parts of a graph, or the components of a diagram.

9–2 ERASING AND CLIPPING

Instead of emphasizing areas, we might want to eliminate parts of a developed display. We can use spotlight methods to identify areas for erasing or clipping. In the first case we delete the selected area; in the second, we keep the area and delete the rest of the display.

ERASING

The box or circle method for spotlighting can be used to select areas for erasure. There are many reasons why we might decide to take out certain parts of a developed display. We may need to simplify a complicated display or to make room for additions to the display. We might be trying out various design layouts or experimenting with visual effects. Or we may need to provide space for enlarging or some other display manipulations.

Parts of a display to be eliminated could, of course, be taken out by changing the program to redraw the display without these parts. This might mean significantly altering the graphics display program. If we later wanted the parts back in, we would have to change the program again. If frequent display deletions are necessary, as in a design application, a general erasing program can be set up to erase everything within a designated area.

Identifying the area of a display to be deleted with a box provides a simple method for erasing characters, pixels, and line sections within the box. We just state the command LINE(X1,Y1)−(X2,Y2),0,BF and the specified area is filled with the background color. The program to accomplish this would need to have as input the size and location of the box, as in Prog. 9–2.

Erasing can also be done within a circular boundary. With advanced BASIC, we can use the CIRCLE and PAINT commands to erase within a designated circular area by filling in with the background color. If the designated circular area overlaps with the background, we must be careful to choose a point for PAINT that is on the object to be erased and not on the background. Otherwise, the command PAINT(X,Y),0,0 will have no effect. Without the PAINT command, erasing inside a circular boundary is a slower process than using a box since the program would have to calculate endpoints for each paint line within the circle boundary.

CLIPPING

In some situations we would like to save an area in a display and erase everything else. This would be the case if we wanted to expand some small area in a picture or graph to the size of the screen, or if we wanted to have figures showing the

parts of a display separately for a report or presentation. For this purpose, we can select the region to be saved within a rectangular area, or **window**. Erasing all parts of a display outside of this window is called **clipping**.

A program to perform general clipping must identify and save all text, unconnected pixels, and line sections of figures within the window area. Figure 9–3 illustrates a framed area or window selected on a diagram. Everything inside the window is to be saved; everything outside is to be clipped. The clipping program would check coordinate endpoints of the lines and text to determine what to save.

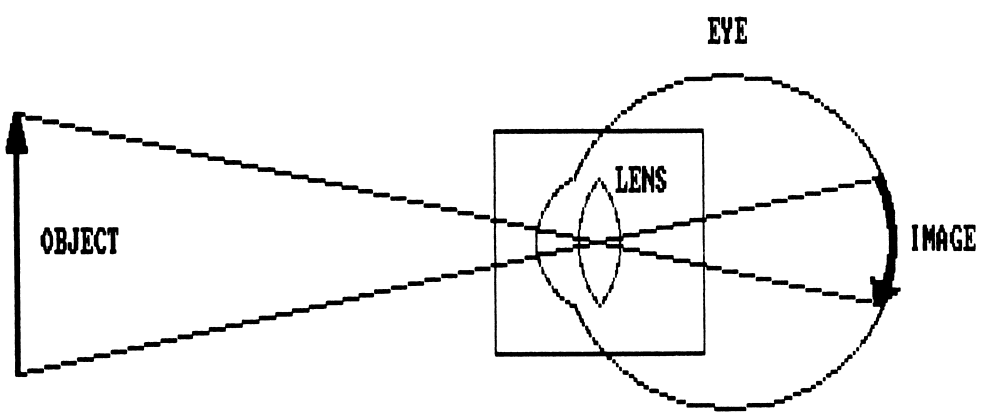
We have a simple method for clipping with the GET and PUT statements. The rectangular area to be saved is first stored in an array with GET. Then the screen is cleared and the saved area is PUT back on the screen. This is demonstrated with Prog. 9–3. Figure 9–4 shows the screen before and after clipping with a selected window.

Although the GET and PUT method of clipping is effective for both pixels and characters, it is a difficult method to use when we want to scale or rotate the saved window area. We need coordinate information to perform these transformations, and this information is not readily available from the pixel color information stored by GET. However, we can perform clipping without using GET and PUT so that coordinate information from the window is saved for later use.

We first consider the design of a clipping program for points and lines only. Our program will input the coordinates for the top left corner of the window as (XW,YW). The width and height of the window will be input as WW and WH. We will store display coordinates in arrays X and Y. Any coordinate information saved by the clipping program will be stored in new arrays.

An isolated point (one not part of a line) will be saved by the clipping program if it is inside the window. That is, its X-coordinate value is between XW and XW + WW and its Y-coordinate value is between YW and YW + WH. A line will be saved by our clipping program if both its endpoints are within the window

Figure 9–3 A window drawn around a section of a display identifies the text and figure parts to be saved by the clipping program.



Program 9–3 Clipping with GET and PUT statements.

```
10 'PROGRAM 9-3. CLIPPING WITH GET AND PUT
20  DIM X(8,20),Y(8,20),LABEL$(5),ROW(5),COLUMN(5),WINDOW%(8002)
30  SCREEN 1: COLOR 1,1: CLS
40  GOSUB 110              'READ PICTURE POINTS
50  GOSUB 250              'DRAW PICTURE
60  GOSUB 370              'ESTABLISH WINDOW
70  GET (LEFT,TOP) - (RIGHT,BOTTOM),WINDOW%
80  CLS
90  PUT (LEFT,TOP),WINDOW%,PSET
100 GOTO 780
110     '*************** READ PICTURE PARTS ******************
120 READ TOTAL                  'TOTAL IS NUMBER OF PICTURE PARTS
130 FOR PART = 1 TO TOTAL
140     READ POINTCOUNT(PART)         'NUMBER OF ELEMENTS IN PART P
150     FOR VERTEX = 1 TO POINTCOUNT(PART)
160         READ X(PART,VERTEX), Y(PART,VERTEX)
170     NEXT
180 NEXT
190     'READ IN TEXT ITEMS, ROW, AND COLUMN PLACEMENT
200 READ LABELTOTAL
210 FOR K = 1 TO LABELTOTAL
220     READ LABEL$(K),ROW(K),COLUMN(K)
230 NEXT
240 RETURN
250     '****************** DRAW ROUTINE **********************
260 FOR PART = 1 TO TOTAL
270     IF POINTCOUNT(PART) = 1 THEN PSET (X(PART,1),Y(PART,1)): GOTO 320
280     PSET (X(PART,1),Y(PART,1))
290     FOR VERTEX = 2 TO POINTCOUNT(PART)
300         LINE - (X(PART,VERTEX),Y(PART,VERTEX))
310     NEXT
320 NEXT
330 FOR K = 1 TO LABELTOTAL             'PLACE TEXT ITEMS
340     LOCATE ROW(K), COLUMN(K): PRINT LABEL$(K);
350 NEXT
360 RETURN
370     '*************** ESTABLISH WINDOW ********************
380 LOCATE 1,1: INPUT "TOP LEFT CORNER OF WINDOW"; XWINDOW,YWINDOW
390 LOCATE 1,1: PRINT STRING$(80," ");
400 LOCATE 1,1: INPUT "WIDTH AND HEIGHT OF WINDOW";WINDOWWIDTH,WINDOWHEIGHT
410 LOCATE 1,1: PRINT STRING$(80,32);
420 LEFT = XWINDOW
430 RIGHT = XWINDOW + WINDOWWIDTH
440 TOP = YWINDOW
450 BOTTOM = YWINDOW + WINDOWHEIGHT
460 IF LEFT < RIGHT AND LEFT >= 0 AND RIGHT <= 319 AND TOP < BOTTOM AND
       TOP >= 0 AND BOTTOM <= 199 THEN 500
470 LOCATE 1,1: PRINT "WINDOW OFF SCREEN. TRY AGAIN"
480 LOCATE 1,1: PRINT STRING$(80,32);
490 GOTO 380
500 LINE (LEFT,TOP) - (RIGHT,BOTTOM),,B
510 LOCATE 1,1: INPUT "TYPE G TO GO ON, C TO CHANGE WINDOW"; M$
520 IF M$ = "G" THEN 570
530     'ERASE CURRENT POSITION
540 LOCATE 1,1: PRINT STRING$(80,32);
550 LINE (LEFT,TOP) - (RIGHT,BOTTOM),0,B
560 GOTO 370
570 RETURN
580     '*********************************************************
590 DATA 7
600     'OUTLINE
610 DATA 12,195,127,210,130,232,105,232,100,213,90,200,75,165,77
```

Program 9–3 (cont.)

```
620 DATA 145,90,85,95,70,70,60,70,60,97
630 DATA 3,69,106,85,120,152,125
640     'WINGS
650 DATA 6,160,110,138,168,145,175,170,175,178,168,200,115
660 DATA 6,203,80,198,45,195,40,170,40,165,45,167,76
670     'TAIL
680 DATA 3,73,75,78,75,80,85
690 DATA 4,65,90,53,110,65,110,75,100
700     'PROPELLOR
710 DATA 7,230,103,235,103,230,65,240,65,230,140,240,140,235,103
720     'TEXT ITEMS
730 DATA 4
740 DATA FUSELAGE,14,12
750 DATA EMPENNAGE,8,2
760 DATA WING,20,20
770 DATA PROPELLOR,7,28
780 END
```

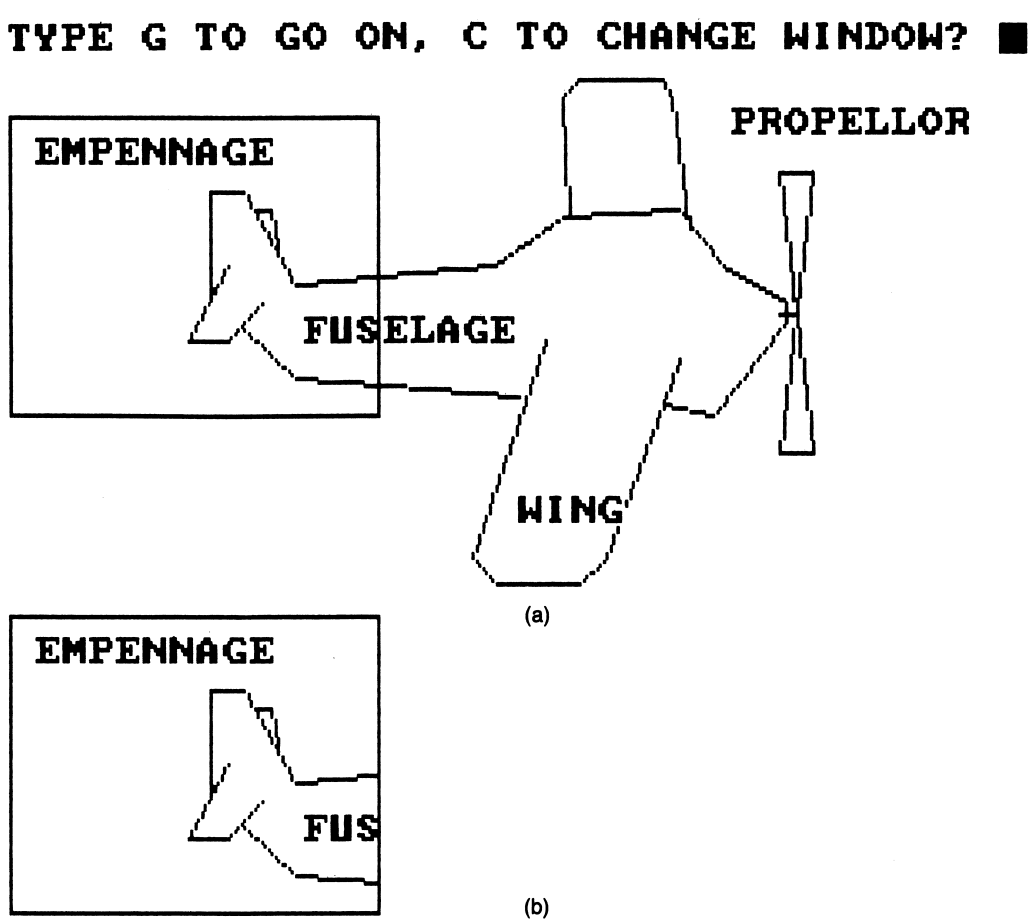


Figure 9–4 Picture displayed (a) before and (b) after clipping by Prog. 9–3, using GET and PUT statements.

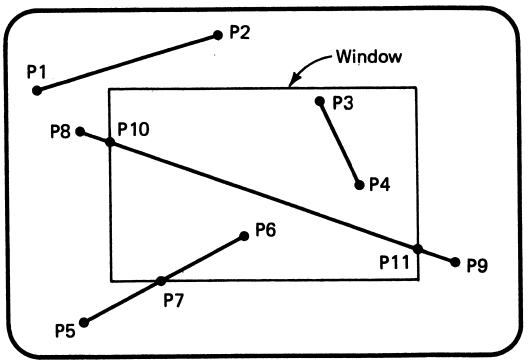
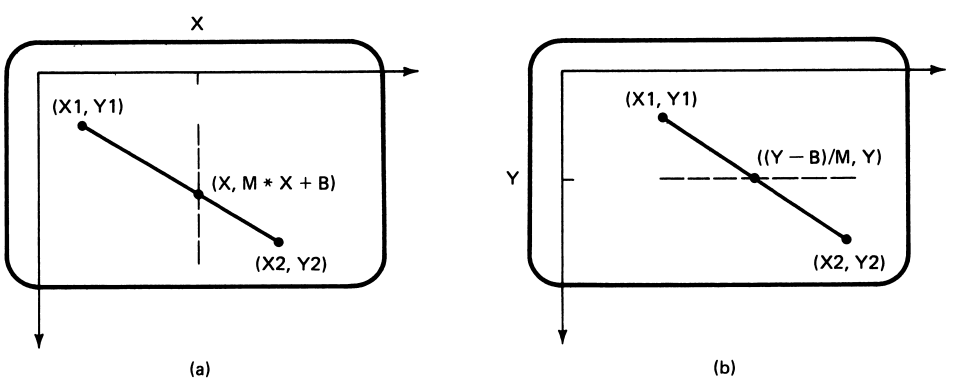


Figure 9–5 A line with endpoint positions P1 and P2, both outside a window, will be erased by the clipping program. A line with endpoints at P3 and P4, both inside the window, will be saved. The clipping routine will also save line segments from P7 to P6 and from P10 to P11.

area. It will be completely erased if all of the line lies outside the window. If parts of the line are outside the window, we will clip off the parts outside and save the segment that falls inside the window. Figure 9–5 illustrates the possible relationships between a line and a window. Intersection points with the vertical and horizontal boundaries of the window are calculated from the window coordinates and line equations (slope M and Y-intercept B), as shown in Fig. 9–6.

In Prog. 9–4, we begin by testing coordinates against the left edge of the window. Points with an X coordinate greater than XW (the left side of the window) are stored in arrays X1 and Y1. Intersection points for the lines that cross the left window boundary are also stored in X1 and Y1. Next, the points stored in X1 and Y1 are clipped against the top edge of the window. All intersection points and any points with Y coordinate greater than this boundary (YW) are stored in arrays X2 and Y2. These remaining points are then clipped against the right edge. Arrays X1 and Y1 are reused to store intersections and points whose X coordinate is less than XW + WW (the right edge). Finally, we

Figure 9–6 Line with slope M and Y-intercept B crossing vertical and horizontal boundaries: (a) intersection with the vertical boundary is at coordinates (X,M * X + B); (b) intersection with the horizontal boundary is at coordinates ((Y−B)/M,Y).



Program 9–4 Point and line clipping (airplane).

```
10 'PROGRAM 9-4. CLIPPING OUTISDE A WINDOW.
20      'DRAWS A FIGURE FROM DATA POINTS STORED IN ARRAYS X AND Y.
30      'A WINDOW AREA IS SELECTED, AND THEN ALL LINES OUTSIDE OF
40      'THE WINDOW AREA ARE CLIPPED. CLIPPING OCCURS AGAINST EACH
50      'BOUNDARY IN THE ORDER LEFT, TOP, RIGHT, BOTTOM. DURING
60      'CLIPPING, POINTS STORED IN X AND Y ARE CLIPPED AGAINST THE
70      'LEFT EDGE; POINTS WITHIN THE WINDOW ARE STORED IN X1, Y1.
80      'THESE POINTS ARE THEN CLIPPED AGAINST THE TOP EDGE, WITH
90      'INCLUDED POINTS SAVED IN X2,Y2. POINTS IN X2,Y2 ARE THEN
100     'CLIPPED AGAINST THE RIGHT EDGE, AND X1,Y1 ARE RE-USED TO
110     'STORE POINTS STILL WITHIN THE WINDOW. THESE POINTS ARE
120     'FINALLY CLIPPED AGAINST THE BOTTOM EDGE, WITH INSIDE
130     'POINTS STORED IN X2,Y2. THE SCREEN IS CLEARED AND THE
140     'WINDOW AND THE PICTURE PART WITHIN THE WINDOW ARE DRAWN.
150     '*************************************************************
160 DIM X(8,20),Y(8,20),X1(8,20),Y1(8,20),X2(8,20),Y2(8,20)
170 SCREEN 1: COLOR 1,1: CLS
180 GOSUB 240           'READ PICTURE POINTS
190 GOSUB 330           'DRAW PICTURE
200 GOSUB 420           'ESTABLISH WINDOW
210 GOSUB 630           'CLIP
220 GOSUB 2010          'DRAW CLIPPED POINTS
230 GOTO 2240
240     '*************** READ PICTURE PARTS ******************
250 READ TOTAL                  'TOTAL IS NUMBER OF PICTURE PARTS
260 FOR PART = 1 TO TOTAL
270     READ POINTCOUNT(PART)       'NUMBER OF ELEMENTS IN PART P
280     FOR VERTEX = 1 TO POINTCOUNT(PART)
290         READ X(PART,VERTEX), Y(PART,VERTEX)
300     NEXT
310 NEXT
320 RETURN
330     '****************** DRAW ROUTINE **********************
340 FOR PART = 1 TO TOTAL
350     IF POINTCOUNT(PART) = 1 THEN PSET (X(PART,1),Y(PART,1)): GOTO 400
360     PSET (X(PART,1),Y(PART,1))
370     FOR VERTEX = 2 TO POINTCOUNT(PART)
380         LINE - (X(PART,VERTEX),Y(PART,VERTEX))
390     NEXT
400 NEXT
410 RETURN
420     '***************** ESTABLISH WINDOW *******************
430 LOCATE 1,1: INPUT "TOP LEFT CORNER OF WINDOW"; XWINDOW,YWINDOW
440 LOCATE 1,1: PRINT STRING$(80," ");
450 LOCATE 1,1: INPUT "WIDTH AND HEIGHT OF WINDOW";WINDOWWIDTH,WINDOWHEIGHT
460 LOCATE 1,1: PRINT STRING$(80,32);
470 LEFT = XWINDOW
480 RIGHT = XWINDOW + WINDOWWIDTH
490 TOP = YWINDOW
500 BOTTOM = YWINDOW + WINDOWHEIGHT
510 IF LEFT < RIGHT AND LEFT >= 0 AND RIGHT <= 319 AND TOP < BOTTOM AND
      TOP >= 0 AND BOTTOM <= 199 THEN 550
520 LOCATE 1,1: PRINT "WINDOW OFF SCREEN. TRY AGAIN"
530 LOCATE 1,1: PRINT STRING$(80,32);
540 GOTO 430
550 LINE (LEFT,TOP) - (RIGHT,BOTTOM),,B
560 LOCATE 1,1: INPUT "TYPE G TO GO ON, C TO CHANGE WINDOW"; M$
570 IF M$ = "G" THEN 620
580     'ERASE CURRENT POSITION
590 LOCATE 1,1: PRINT STRING$(80,32);
600 LINE (LEFT,TOP) - (RIGHT,BOTTOM),0,B
```

Program 9–4 (cont.)

```
610 GOTO 420
620 RETURN
630 '############################ CLIPPING ROUTINE ##############################
640    'CLIP POINTS IN X AND Y AGAINST LEFT EDGE. STORE IN X1 AND Y1.
650 PARTSIN = 1
660 FOR PART = 1 TO TOTAL
670     VERTIN = 0
680     FOR VERT = 1 TO POINTCOUNT(PART) - 1
690         IF X(PART,VERT) < LEFT THEN 770
700         VERTIN = VERTIN + 1                    'FIRST PART OF LINE IS IN
710         X1(PARTSIN,VERTIN) = X(PART,VERT)
720         Y1(PARTSIN,VERTIN) = Y(PART,VERT)
730             'WHAT ABOUT SECOND POINT?
740         IF X(PART,VERT+1) < LEFT THEN GOSUB 900    'FIND INTERSECTION
750         GOTO 790              'ELSE IT'S IN, SO JUST CONTINUE
760             '-------------------------------------------------
770            'FIRST POINT IS OUT. WHAT ABOUT SECOND POINT?
780         IF X(PART,VERT+1) >= LEFT THEN GOSUB 900   'FIND INTERSECTION
790     NEXT
800     IF X(PART,POINTCOUNT(PART)) < LEFT THEN 840            'DO FINAL POINT
810     VERTIN = VERTIN + 1
820     X1(PARTSIN,VERTIN) = X(PART,POINTCOUNT(PART))
830     Y1(PARTSIN,VERTIN) = Y(PART,POINTCOUNT(PART))
840     IF VERTIN = 0 THEN 870                  'NO ELEMENTS IN WINDOW
850     POINTCOUNTIN(PARTSIN) = VERTIN          'ELSE, SAVE COUNT OF IN POINTS
860     PARTSIN = PARTSIN + 1                   'GO ON TO NEXT PART
870 NEXT
880 TOTALIN = PARTSIN - 1       'TOTALIN IS NUMBER OF PARTS WITH POINTS INSIDE
890 GOTO 970                   'GO ON TO NEXT EDGE
900             '^^^^^^^^^ FIND INTERSECTION ROUTINE ^^^^^^^^^^^
910 VERTIN = VERTIN + 1
920 SLOPE = (Y(PART,VERT+1) - Y(PART,VERT)) / (X(PART,VERT+1) - X(PART,VERT))
930 Y1(PARTSIN,VERTIN) = SLOPE * (LEFT - X(PART,VERT)) + Y(PART,VERT)
940 X1(PARTSIN,VERTIN) = LEFT
950 RETURN
960                     '*****************************
970    'CLIP POINTS IN X1, Y1 AGAINST TOP. STORE INSIDE POINTS IN X2, Y2
980 PARTSIN = 1
990 FOR PART = 1 TO TOTALIN
1000     VERTIN = 0
1010     FOR VERT = 1 TO POINTCOUNTIN(PART) - 1
1020         IF Y1(PART,VERT) <  TOP THEN 1100
1030         VERTIN = VERTIN + 1                'FIRST PART OF LINE IS IN
1040         Y2(PARTSIN,VERTIN) = Y1(PART,VERT)
1050         X2(PARTSIN,VERTIN) = X1(PART,VERT)
1060            'WHAT ABOUT SECOND POINT?
1070         IF Y1(PART,VERT+1) < TOP THEN GOSUB 1230   'FIND INTERSECTION
1080         GOTO 1120           'ELSE IT'S IN SO JUST CONTINUE
1090            '-------------------------------------------------
1100            'FIRST POINT IS OUT. WHAT ABOUT SECOND POINT?
1110         IF Y1(PART,VERT+1) >= TOP THEN GOSUB 1230  'FIND INTERSECTION
1120     NEXT
1130     IF Y1(PART,POINTCOUNTIN(PART)) < TOP THEN 1170          'DO FINAL POINT
1140     VERTIN = VERTIN + 1
1150     Y2(PARTSIN,VERTIN) = Y1(PART,POINTCOUNTIN(PART))
1160     X2(PARTSIN,VERTIN) = X1(PART,POINTCOUNTIN(PART))
1170     IF VERTIN = 0 THEN 1200                'NO ELEMENTS IN WINDOW
1180     POINTCOUNTIN(PARTSIN) = VERTIN         'ELSE SAVE COUNT OF IN POINTS
1190     PARTSIN = PARTSIN + 1                  'GO ON TO THE NEXT PART
1200 NEXT
1210 TOTALIN = PARTSIN - 1       'TOTALIN IS # OF PARTS WITH POINTS INSIDE
```

Program 9–4 (cont.)

```
1220 GOTO 1330                    'GO ON TO NEXT EDGE
1230    '       '^^^^^^^^^^^^^^ FIND INTERSECTION ROUTINE ^^^^^^^^^^^^^^^^
1240 VERTIN = VERTIN + 1
1250 IF X1(PART,VERT+1) <> X1(PART,VERT) THEN 1280
1260 X2(PARTSIN,VERTIN) = X1(PART,VERT)                    'VERTICAL LINE
1270 GOTO 1300
1280 SLOPE = (Y1(PART,VERT+1)-Y1(PART,VERT)) / (X1(PART,VERT+1)-X1(PART,VERT))
1290 X2(PARTSIN,VERTIN) = (TOP - Y1(PART,VERT)) / SLOPE + X1(PART,VERT)
1300 Y2(PARTSIN,VERTIN) = TOP
1310 RETURN
1320                    '***************************
1330    'CLIP POINTS IN X2, Y2 AGAINST RIGHT. STORE INSIDE POINTS IN X1, Y1.
1340 PARTSIN = 1
1350 FOR PART = 1 TO TOTALIN
1360     VERTIN = 0
1370     FOR VERT = 1 TO POINTCOUNTIN(PART) - 1
1380         IF X2(PART,VERT) >  RIGHT THEN 1460
1390         VERTIN = VERTIN + 1                   'FIRST POINT IS IN
1400         X1(PARTSIN,VERTIN) = X2(PART,VERT)
1410         Y1(PARTSIN,VERTIN) = Y2(PART,VERT)
1420            'WHAT ABOUT SECOND POINT
1430         IF X2(PART,VERT+1) > RIGHT THEN GOSUB 1590   'FIND INTERSECTION
1440         GOTO 1480           'ELSE IT'S IN SO JUST CONTINUE
1450            '----------------------------------------
1460            'FIRST POINT IS OUT. WHAT ABOUT SECOND POINT?
1470         IF X2(PART,VERT+1)< RIGHT THEN GOSUB 1590  'FIND INTERSECTION
1480     NEXT
1490     IF X2(PART,POINTCOUNTIN(PART)) > RIGHT THEN 1530       'DO FINAL POINT
1500     VERTIN = VERTIN + 1
1510     X1(PARTSIN,VERTIN) = X2(PART,POINTCOUNTIN(PART))
1520     Y1(PARTSIN,VERTIN) = Y2(PART,POINTCOUNTIN(PART))
1530     IF VERTIN = 0 THEN 1560                   'NO ELEMENTS IN WINDOW
1540     POINTCOUNTIN(PARTSIN) = VERTIN            'ELSE SAVE COUNT OF IN POINTS
1550     PARTSIN = PARTSIN + 1                     'GO ON TO NEXT PART
1560 NEXT
1570 TOTALIN = PARTSIN - 1       'TOTALIN IS NUMBER OF PARTS WITH POINTS INSIDE
1580 GOTO 1660                   'GO ON TO NEXT EDGE
1590          '^^^^^^^^^^^^^^ FIND INTERSECTION ROUTINE ^^^^^^^^^^
1600 VERTIN = VERTIN + 1
1610 SLOPE = (Y2(PART,VERT+1)-Y2(PART,VERT)) / (X2(PART,VERT+1)-X2(PART,VERT))
1620 Y1(PARTSIN,VERTIN) = SLOPE * (RIGHT - X2(PART,VERT)) + Y2(PART,VERT)
1630 X1(PARTSIN,VERTIN) = RIGHT
1640 RETURN
1650                    '***************************
1660    'CLIP POINTS IN X1, Y1 AGAINST BOTTOM. STORE INSIDE POINTS IN X2, Y2
1670 PARTSIN = 1
1680 FOR PART = 1 TO TOTALIN
1690     VERTIN = 0
1700     FOR VERT = 1 TO POINTCOUNTIN(PART) - 1
1710         IF Y1(PART,VERT) >  BOTTOM THEN 1790
1720         VERTIN = VERTIN + 1                   'FIRST POINT IS IN
1730         Y2(PARTSIN,VERTIN) = Y1(PART,VERT)
1740         X2(PARTSIN,VERTIN) = X1(PART,VERT)
1750            'WHAT ABOUT SECOND POINT?
1760         IF Y1(PART,VERT+1)>BOTTOM THEN GOSUB 1920  'FIND INTERSECTION
1770         GOTO 1810           'ELSE IT'S IN SO JUST CONTINUE
1780            '----------------------------------------
1790            'FIRST POINT IS OUT. WHAT ABOUT SECOND POINT?
1800         IF Y1(PART,VERT+1)<=BOTTOM THEN GOSUB 1920 'FIND INTERSECTION
1810     NEXT
1820     IF Y1(PART,POINTCOUNTIN(PART)) > BOTTOM THEN 1860      'DO FINAL POINT
```

Program 9–4 (cont.)

```
1830      VERTIN = VERTIN + 1
1840      Y2(PARTSIN,VERTIN) = Y1(PART,POINTCOUNTIN(PART))
1850      X2(PARTSIN,VERTIN) = X1(PART,POINTCOUNTIN(PART))
1860      IF VERTIN = 0 THEN 1890                    'NO ELEMENTS IN WINDOW
1870      POINTCOUNTIN(PARTSIN) = VERTIN             'ELSE SAVE COUNT OF IN POINTS
1880      PARTSIN = PARTSIN + 1                      'GO ON TO NEXT PART
1890 NEXT
1900 TOTALIN = PARTSIN - 1        'TOTALIN IS # OF PARTS STILL INSIDE
1910 RETURN                       'END OF CLIPPING ROUTINE
1920             '^^^^^^^^^^^^ FIND INTERSECTION ROUTINE ^^^^^^^^^^^
1930 VERTIN = VERTIN + 1
1940 IF X1(PART,VERT+1) <> X1(PART,VERT) THEN 1970
1950 X2(PARTSIN,VERTIN) = X1(PART,VERT)           'VERTICAL LINE
1960 GOTO 1990
1970 SLOPE = (Y1(PART,VERT+1)-Y1(PART,VERT)) / (X1(PART,VERT+1)-X1(PART,VERT))
1980 X2(PARTSIN,VERTIN) = (BOTTOM - Y1(PART,VERT)) / SLOPE + X1(PART,VERT)
1990 Y2(PARTSIN,VERTIN) = BOTTOM
2000 RETURN
2010 '#################### DRAW CLIPPED POINTS ###################
2020 CLS
2030 LINE (LEFT,TOP) - (RIGHT,BOTTOM),,B
2040 FOR PART = 1 TO TOTALIN
2050      FOR VERT = 1 TO POINTCOUNTIN(PART) - 1
2060           LINE (X2(PART,VERT),Y2(PART,VERT)) -
                    (X2(PART,VERT+1),Y2(PART,VERT+1))
2070      NEXT
2080 NEXT
2090 RETURN
2100 '#########################################################################
2110 DATA 7
2120    'OUTLINE
2130 DATA 12,195,127,210,130,232,105,232,100,213,90,200,75,165,77
2140 DATA 145,90,85,95,70,70,60,70,60,97
2150 DATA 3,69,106,85,120,152,125
2160    'WINGS
2170 DATA 6,160,110,138,168,145,175,170,175,178,168,200,115
2180 DATA 6,203,80,198,45,195,40,170,40,165,45,167,76
2190    'TAIL
2200 DATA 3,73,75,78,75,80,85
2210 DATA 4,65,90,53,110,65,110,75,100
2220    'PROPELLOR
2230 DATA 7,230,103,235,103,230,65,240,65,230,140,240,140,235,103
2240 IF INKEY$ = "" THEN 2240
2250 END
```

clip the points now held in arrays X1 and Y1 against the bottom border of the window (YW + WH). Arrays X2 and Y2 are used to store the final coordinates for all points and lines to be redrawn inside the window. Figure 9–7 shows a picture before and after clipping by Prog. 9–4.

Now let us expand Prog. 9–4 to clip any labels that might be in the display. We could treat labels like lines and save any part inside the window, but we would need to treat the labels in terms of individual characters or even individual pixels. To simplify our program we will save a character string label only if it is entirely within the window boundaries. In Fig. 9–8, the string "LABEL 3" is the only text information we would save for the window indicated. Our test for saving a string

TYPE G TO GO ON, C TO CHANGE WINDOW? ■

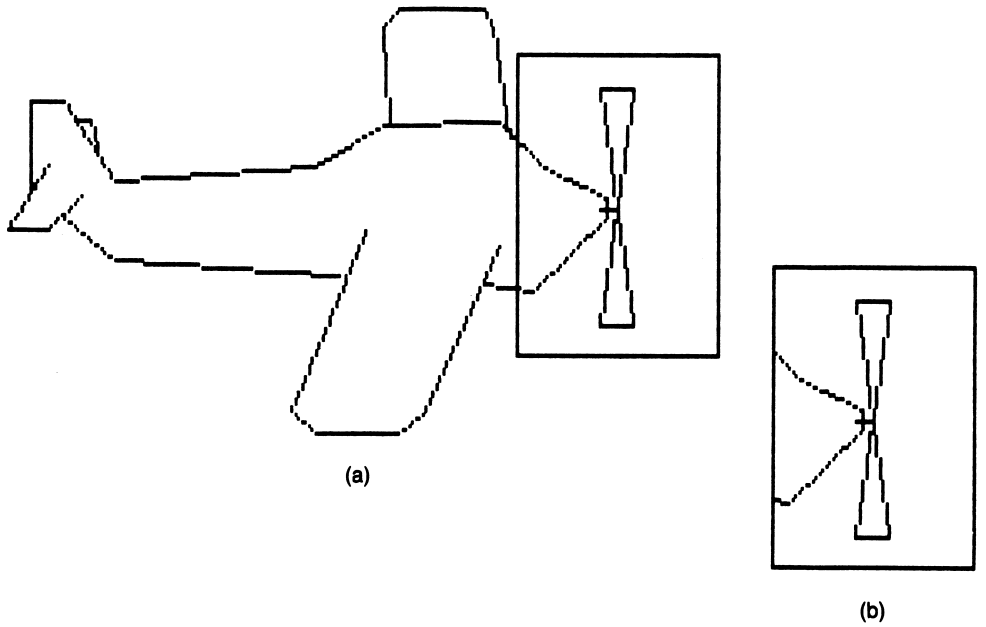
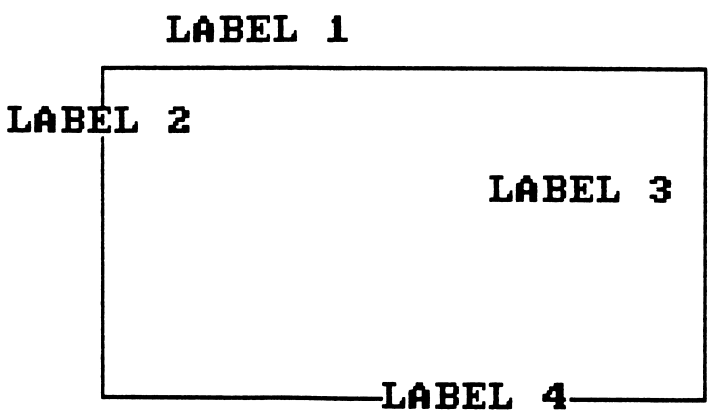


Figure 9–7 Picture displayed (a) before and (b) after clipping by Prog. 9–4.

will be to determine whether the string starts after the left window boundary and ends before the right window boundary. Also, the string must be on a print line that lies within the top and bottom window limits.

Our program will use the array TEXT$ to store the text character strings.

Figure 9–8 Clipping any text strings not completely inside the window will erase the strings "LABEL 1", "LABEL 2", and "LABEL 4".

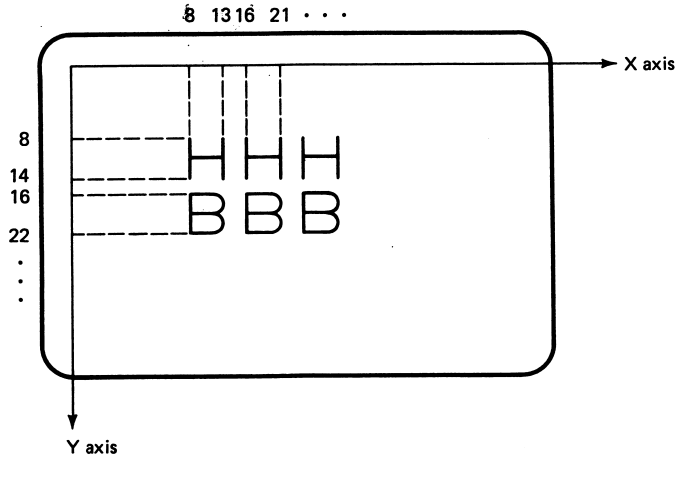


Arrays XLEFT and XRIGHT will be used to store the X locations for the beginning and ending positions of each string. Arrays YTOP and YBOTTOM will be used to store Y coordinates for the top and bottom of the strings. Values for these coordinate arrays are obtained from the dimensions of the character pixel grid.

With the Color/Graphics option, our PC uses an 8 by 8 pixel area to display characters. Each character is actually drawn on the screen with a 6 by 7 pixel grid within this area. That leaves a two-pixel horizontal separation between characters and a one-pixel separation between the print lines (Fig. 9–9). Characters printed across a line begin at pixel locations 0, 8, 16, 24, ... and end at pixel locations 5, 13, 21, and so forth. The tops of characters printed down the screen are at pixel locations 0, 8, 16, and so forth. To determine the beginning X coordinate for a character, we subtract 1 from the character print position and multiply by 8. To get the ending position, we add 5 to the beginning position. For a character printed at a position specified by LOCATE 6,21, the beginning X coordinate is 160 and the ending X coordinate is 165. To determine the Y coordinate for the top of any character, we subtract 1 from the print line and multiply by 8. Adding 6 to the top value gives the Y coordinate for the bottom of the character. For the character position specified as LOCATE 6,21, the Y coordinate of the top is 40 and the Y coordinate of the bottom is 46. These text clipping modifications to Prog. 9–4 are included in Prog. 9–5. Output of this program is given in Fig. 9–10.

Improvements could be made in the line-clipping routines to speed up the calculations, especially if very many lines are to be clipped. For instance, we could process each point in the picture once, instead of line by line. A code could be set up for each point, which tells us the position of the point relative to the

Figure 9–9 Coordinate positions for characters with an 8 by 8 pixel grid. Horizontally, characters start at pixel locations 0, 8, 16, . . . and end at pixel locations 5, 13, 21, and so on across the screen. Tops of the print lines are at pixel positions 0, 8, 16, . . . , and bottoms of the print lines are at pixel positions 6, 14, 22, and so on down the screen.



Program 9–5 Point, line, and text clipping (airplane).

```
10 'PROGRAM 9-5. CLIPPING OUTISDE A WINDOW WITH TEXT. (ADD TO PROG. 9-4)

165 DIM TEXT$(10), TLEFT(10), TRIGHT(10), TTOP(10), TBOTTOPOM(10), ROW(10),
      COLUMN(10)

215 GOSUB 2001          'CLIP TEXT

311     'READ IN TEXT ITEMS, ROW, AND COLUMN PLACEMENT
312 READ TEXTOTAL
313 FOR K = 1 TO TEXTOTAL
314     READ TEXT$(K),ROW(K),COLUMN(K)
315     TLEFT(K) = (COLUMN(K) - 1) * 8          'CONVERT TO PIXEL POSITIONS
316     TRIGHT(K) = TLEFT(K) - 1 + LEN(TEXT$(K)) * 8
317     TTOP(K) = (ROW(K) - 1) * 8
318     TBOTTOPOM(K) = TTOP(K) + 7
319 NEXT

401     'PLACE TEXT ITEMS
402 FOR K = 1 TO TEXTOTAL
403     LOCATE ROW(K), COLUMN(K): PRINT TEXT$(K);
404 NEXT

2001 COUNTIN = 0      'COUNT IS TEXT ITEMS THAT ARE IN WINDOW
2002 FOR K = 1 TO TEXTOTAL
2003     IF TLEFT(K) < LEFT OR TRIGHT(K) > RIGHT OR TTOP(K) < TOP OR
         TBOTTOPOM(K) > BOTTOM THEN 2008
2004     COUNTIN = COUNTIN + 1
2005     TEXT$(COUNTIN) = TEXT$(K)
2006     ROW(COUNTIN) = ROW(K)
2007     COLUMN(COUNTIN) = COLUMN(K)
2008 NEXT
2009 RETURN

2231    'TEXT ITEMS
2232 DATA 4
2233 DATA FUSELAGE,14,12
2234 DATA EMPENNAGE,8,2
2235 DATA WING,20,20
2236 DATA PROPELLOR,6,28
```

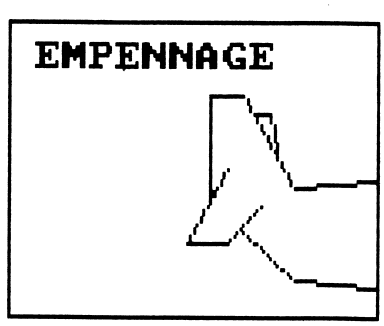


Figure 9–10 Picture with text displayed after clipping by Prog 9–5.

window boundary. That is, the code could tell us whether a point is above, below, left, right, or inside the window. This code can then be used to test the location of lines relative to the window. A line with both endpoints to the left of a window, for example, is completely outside the window.

After a picture is clipped, we could transform the window area in some way. We might display the window in another location, or we might scale or rotate the windowed part of the picture. We could also keep the original display and superimpose the window—perhaps enlarged—in one corner of the screen. The next section considers a method for transforming windows.

9–3 VIEWPORTS

Having established a window in a display, we can translate and scale the window to any size by moving it to a specified rectangular area on the screen. This area is called the **viewing area**, or **viewport**. We can establish both a window and a viewport on the screen by giving the coordinates for the top left corners and the size of each rectangular area, as shown in Fig. 9–11.

The window defines "what" we want to see in the display; the viewport establishes "where" we would like to see it on the screen. We can make the viewport larger or smaller than the window, or we can make it the same size. The viewport can be made to fill the screen or it can be set as a small insert in the display. We can make the window and viewport in separate screen areas, or we could make them overlap. If a viewport is used to enlarge an area of a picture, this enlargement can magnify details that are too small to be visible in the original display.

A program to transform a window area to a viewport area must transform the coordinates of each point in the window, such as (X,Y) in Fig. 9–11, to the

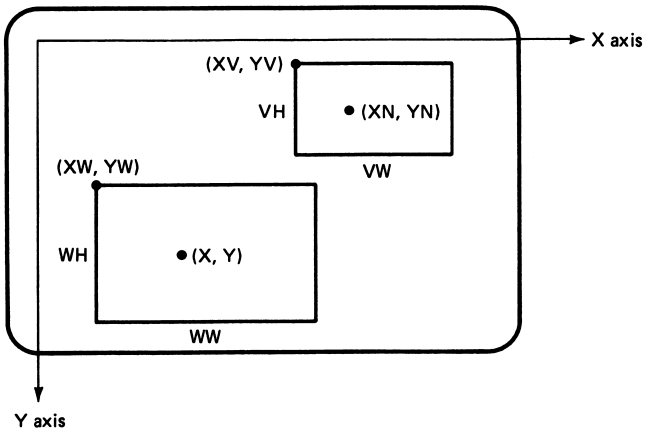


Figure 9–11 Window and viewport specifications. Upper left corner of the window is at position (XW, YW); upper left corner of the viewport is at position (XV,YV). Size of each area is specified by width and height (WW, WH and VW, VH). A point with coordinates (X,Y) in the window will be transformed to a new position (XN,YN) in the viewport.

corresponding new point (XN,YN) in the viewport. This transformation is accomplished in much the same way that we set up graphs on the screen within specified areas in Chapter 4. New coordinates in the viewport are related to the original coordinates of the point in the window by the relations

$$XN = (X - XW) * (VW/WW) + XV$$
$$YN = (Y - YW) * (VH/WH) + YV \qquad (9\text{–}1)$$

The factors (VW/WW) and (VH/WH) in the equations of (9–1) represent the scaling transformation. We have a different size for the area in the viewport if these factors are not equal to 1. A value greater than 1 enlarges the window; a value smaller than 1 reduces the size. If the ratio VW/VH is not equal to the ratio WW/WH, we will distort the window area. This is equivalent to a different scaling for the X and Y directions, as discussed in Chapter 7. Terms XV and YV represent the translation. If these coordinates are different from XW and YW, the area has been moved.

A mapping from a window area to a specified viewport is accomplished by Prog. 9–6, an extension of Prog. 9–4. Figure 9–12 illustrates the output from this

Program 9–6 Displaying viewports (airplane).

```
2240 '##### PROGRAM 9-6. DISPLAY WINDOW AREA IN VIEWPORT (add to Prog. 9-4)
2250      'ESTABLISHES A VIEWPORT ON THE SCREEN. TAKES THE PICTURE
2260      'PARTS FOUND TO BE INSIDE THE WINDOW AND MOVES THEM TO
2270      'THE CHOSEN VIEWPORT.
2280      '*****************************************************
2290 GOSUB 2330          'ESTABLISH VIEWPORT
2300 GOSUB 2460          'CONVERT WINDOW AREA TO VIEWPORT AREA
2310 GOSUB 2010          'DRAW
2320 GOTO 2540
2330 '###################### ESTABLISH VIEWPORT ######################
2340 LOCATE 1,1: INPUT "TOP LEFT CORNER OF VIEWPORT"; XVIEW,YVIEW
2350 LOCATE 1,1: PRINT STRING$(80,32);
2360 LOCATE 1,1: INPUT "WIDTH AND HEIGHT OF VIEWPORT"; VIEWWIDTH,VIEWHEIGHT
2370 LOCATE 1,1: PRINT STRING$(80,32);
2380 LEFT   = XVIEW
2390 RIGHT  = XVIEW + VIEWWIDTH
2400 TOP    = YVIEW
2410 BOTTOM = YVIEW + VIEWHEIGHT
2420 IF LEFT < RIGHT AND LEFT >= 0 AND RIGHT <= 319 AND TOP < BOTTOM AND
     TOP >= 0 AND BOTTOM <= 199 THEN 2450
2430 LOCATE 1,1: PRINT "VIEWPORT OFF SCREEN. TRY AGAIN"
2440 LOCATE 1,1: PRINT STRING$(80,32);: GOTO 2340
2450 RETURN
2460 '############# CONVERT WINDOW AREA TO VIEWPORT AREA #############
2470 FOR PART = 1 TO TOTALIN
2480     FOR VERT = 1 TO POINTCOUNTIN(PART)
2490         X2(PART,VERT) = (X2(PART,VERT) - XWINDOW) *
                             (VIEWWIDTH / WINDOWWIDTH) + XVIEW
2500         Y2(PART,VERT) = (Y2(PART,VERT) - YWINDOW) *
                             (VIEWHEIGHT / WINDOWHEIGHT) + YVIEW
2510     NEXT
2520 NEXT
2530 RETURN
2540 END
```

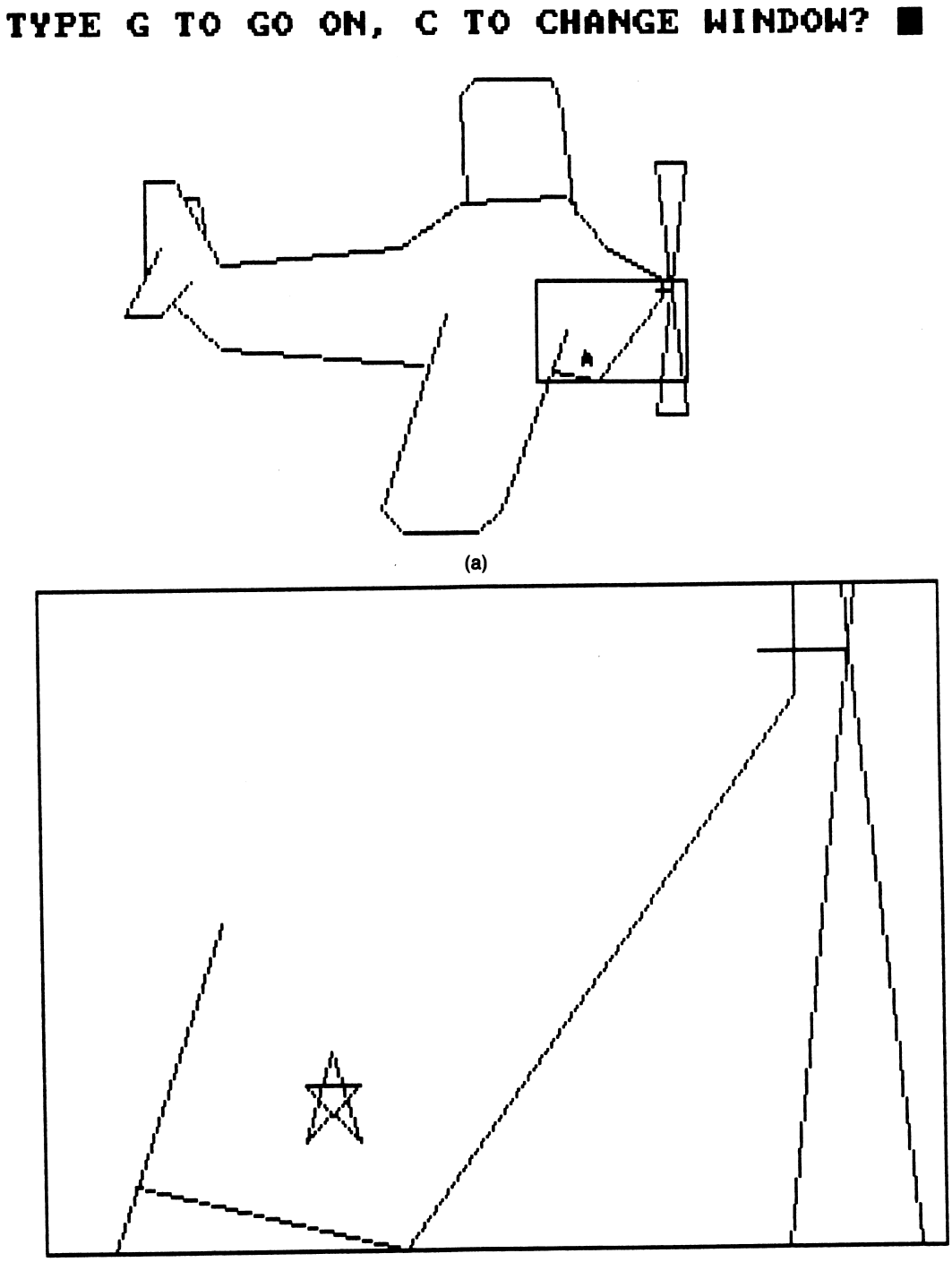


Figure 9–12 A window area of a picture (a) is magnified into a viewport area (b) by Prog. 9–6.

program. In this example, we demonstrate how viewports can be used to magnify details of a picture. The star on the airplane fuselage, seen as a cluster of points when displayed in relative scale, is clearly identified in the viewport display. Such viewport magnifications can be useful with maps or complex diagrams to show levels of detail.

For some applications, we might want to display both the original scene and the viewport together on the screen. We could also window more than one area, producing more than one viewport. As a final transformation on a window, a viewport could be rotated to present the area in a different orientation.

PROGRAMMING PROJECTS

9–1. Revise Prog. 9–1 (or 9–2) to create spotlights with either light pen or joystick input.

9–2. Write a spotlight program that will accent lines within the spotlight by drawing them in a brighter (or different) color, using either a rectangular or circular spotlight. As an additional feature, change the background color of the display area within the spotlight.

9–3. Revise Prog. 9–1 to spotlight areas of a display using an ellipse of any specified dimensions.

9–4. Modify Project 9–3 to erase the elliptical spotlight area instead of spotlighting.

9–5. Write a general erasing program that will erase character strings, straight lines, rectangles, or circles. The type of area to be erased will be specified by input along with the area or line location and dimensions.

9–6. Modify Prog. 9–4 so that each point of a picture is compared to the window boundary only once. The position of each point relative to the window can be specified in an array that states whether the point is "IN" or "OUT," "ABOVE" or "BELOW," and "LEFT" or "RIGHT." This information could be coded in a character string of length 3 (for example, "OBR" would state that a point is outside, below and to the right). After processing all points to set the position string for each point, the lines in the picture can be clipped by examining the corresponding position string for the endpoints. A line is saved if both points are "IN." It is eliminated if both points are "LEFT" or both points are "ABOVE," and so forth. Intersection points are then found for the overlapping lines. The position string can be reduced to two characters by setting the first character in the string to be "X" (inside), "A" (above), or "B" (below), and by setting the second character to "L" (left) or "R" (right).

9–7. Modify Prog. 9–5 to clip character strings against a rectangular boundary by erasing only that part of the string within the window.

9–8. Write a clipping program that will erase all parts of a display outside of a specified circular area.

9–9. Write a clipping program that uses a viewport to enlarge some area of a picture, then

superimposes the enlarged portion onto the original picture in one corner of the screen. Instead of erasing the entire screen, erase only the area that is to contain the viewport.

9–10. Write a program that will display any number of viewports of picture areas on the screen.

9–11. Write a program that will rotate a viewport to any specified orientation.

Part IV

THREE DIMENSIONS

To display solid objects, we need to use a third dimension. We now consider techniques for displaying and manipulating three-dimensional objects.

Chapter 10

Displaying Solid Objects

Real objects are perceived in three dimensions. They have depth as well as breadth and height. When we represent objects on a flat (two-dimensional) screen, we can either ignore the depth or project the objects onto the screen in such a way that depth is represented. By including depth, we can add greatly to the realism of a picture or to the information content of a graph. We now consider some of the ways to include this third dimension in our displays.

10–1 GRAPH PAPER LAYOUTS

A three-dimensional object can be projected onto a screen by first producing a graph paper layout, as in Fig. 10–1. We then determine X and Y coordinates for line endpoints from the layout. Usually, we want to plot only the visible lines. As drawn in Fig. 10–1, it is not clear which three sides are presented to us. We could be viewing the box from above and to the right or from below and to the left. But we can identify the hidden lines in the layout for the view we want to see and only display the lines that should be visible. If we decide to view the box from above, we draw it without the lines connecting points 3 and 7, points 6 and 7, and points 7 and 8. This manual method can be used to display three-dimensional scenes: we draw all sides of an object or group of objects on graph paper, erase lines that are hidden for the view we want to display, and plot the remaining lines on the screen.

For some situations we would like to use a more general method for obtaining a three-dimensional view that would let the graphics display program distinguish the "front" and "back" of objects. A rotating figure that continuously brings different sides into view is constantly changing front faces into back faces. Various techniques can be used in display programs to distinguish the front from

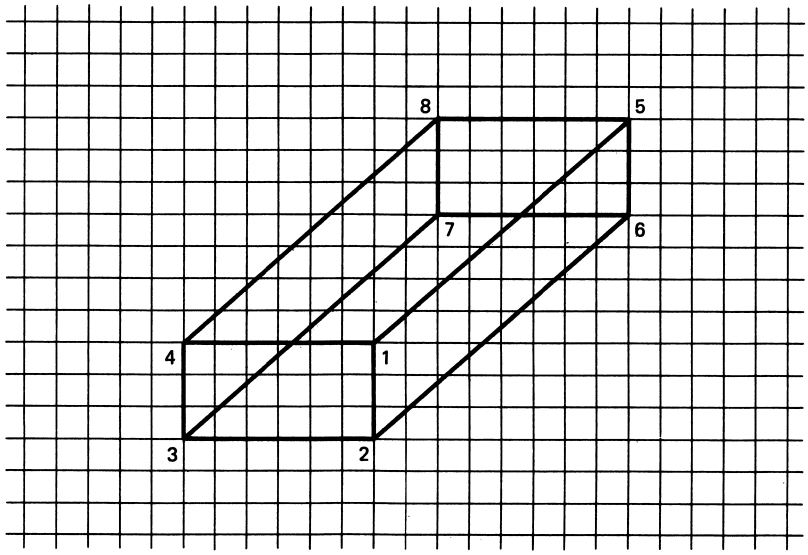


Figure 10–1 Graph paper layout for a three-dimensional box.

the back of three-dimensional objects, in order to give us depth information. One method is to erase all hidden parts of objects, as we did in the graph paper layouts. We can also project objects on the screen so that a perspective view provides depth information. A perspective view makes closer parts larger than the objects and surfaces farther away from us. Another technique is to highlight nearer lines, so that front parts of objects are brighter than the back lines. More elaborate shading schemes can be used that produce gradual light to dark patterns across surfaces. Each of these methods requires that the depth of points in the picture be specified in the display program.

10–2 THREE-DIMENSIONAL COORDINATES

Figure 10–2 illustrates a coordinate representation for specifying the **depth**, or Z coordinate, in addition to specifying the X and Y coordinates. The upper left corner of the screen is chosen to be the coordinate origin for the three axes. As before, positive X values are measured from left to right across the screen, and positive Y values are measured from top to bottom. Coordinate Z values are measured from zero at the screen face, with positive values in back of the screen and negative values in front of the screen. Each point of a picture is then assigned three coordinate values, (X,Y,Z). Position on the screen is determined by the X and Y values, and Z denotes the depth of the point relative to the display screen. Points farther away have larger Z values; nearer points have smaller Z values. We use the Z coordinates to obtain different views of objects, to identify hidden lines or surfaces of objects, and to obtain perspective views.

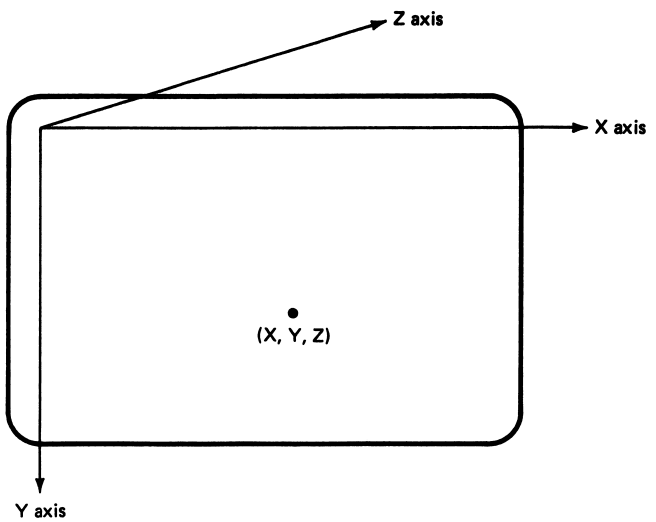
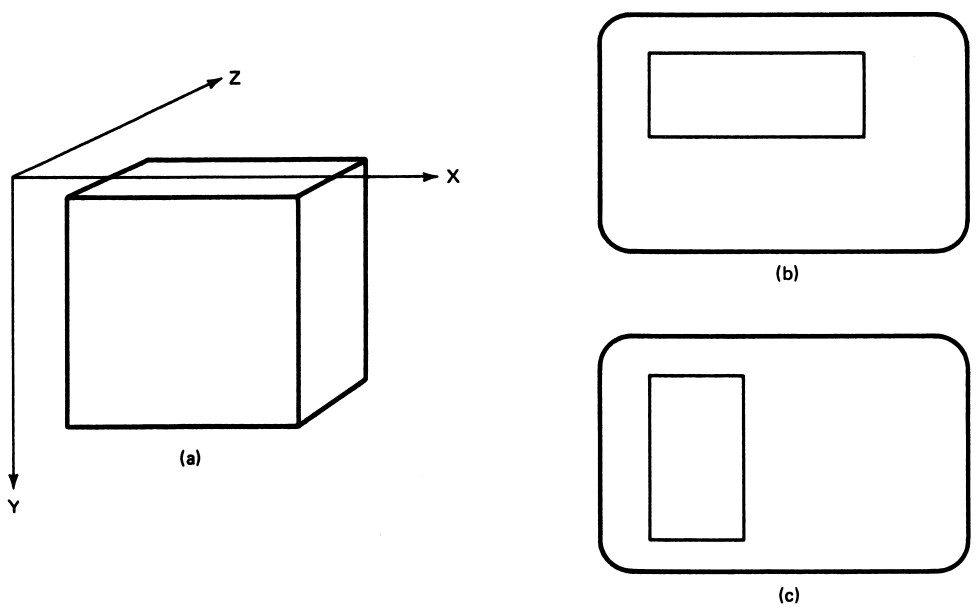


Figure 10–2 Three-dimensional coordinate system for representing positions in terms of the three values (X,Y,Z). Positive Z values indicate distances in back of the display screen.

Different views of objects can be displayed by substituting the Z coordinate of each point for either the X or Y values when we plot screen positions. Thus, plotting (X,Z) for all points gives a top or bottom view and plotting (Z,Y) gives side views, as shown in Fig. 10–3. In the first case, we assume that Y values are positive in back of the screen. Otherwise, we have a bottom view. In the second

Figure 10–3 Orthographic projections. A three-dimensional object (a) can be viewed from the top or bottom by plotting (X,Z) values on the screen (b). Side views (c) are obtained by plotting (Z,Y) values.



case, we obtain a left-side view if X is assumed positive in back of the screen, and we obtain a right-side view if X is assumed positive in front of the screen. Such views are called **orthographic projections**. To obtain these top, bottom, or side views of an object, we input the (X,Y,Z) coordinates for each vertex and plot the appropriate coordinate pair, eliminating the "back" faces.

10–3 ERASING HIDDEN LINES AND SURFACES

There are two general approaches to erasing hidden parts of objects. One approach is to think of an object in terms of its various surfaces. We can then identify and eliminate those surfaces that are in back of, or hidden by, other surfaces. The second method treats objects in terms of component lines, identifying and erasing individual hidden lines instead of surfaces.

HIDDEN SURFACES

A technique for eliminating hidden surfaces in a display is to paint each surface onto the screen from back to front. Surfaces with larger Z values are painted first, so that closer surfaces obscure back surfaces if they overlap. Each surface can be painted in a different color, or they can all be painted with the background color.

Program 10–1 illustrates this method of erasing hidden lines. A maximum of 20 polygon faces may be input, together with a color code from 0 to 3 and an interior point. Each plane face is assumed to be parallel to the display screen, so that only one Z value is specified for each plane. We draw each polygon outline first in color 3. Then the polygon interior is filled with the specified color up to the boundary color 3. Finally, the figure outline is redrawn in the interior color. Overlapping planes will then color over the faces farther away (Fig. 10–4). Color 3 can be used as the filler color in this program only for those shapes that will be entirely visible; that is, no other shape is to be placed over this one. If we were to try to place a figure in front of one filled with color 3, the PAINT command could not distinguish between the interior of the filled-in shape and the boundary of the new shape to be displayed.

Without advanced BASIC, we could fill the interiors of the polygons with individual horizontal lines. The endpoints of each of these interior "erase" lines would be determined from the line equations that specify the boundary of each surface. We could also specify the Z coordinate for each vertex point of a surface, rather than for the entire surface. This allows a plane to be tilted, with parts of the plane farther from us than other parts, as are the sides of solid objects. Sorting of the surfaces would then be accomplished according to the smallest Z value for each surface. For an arbitrary pattern of tilted surfaces, we might want to set up the planes so that they do not run into each other if their range of Z values overlap. Otherwise two surfaces could alternately obscure one another, as shown in Fig. 10–5. We could allow for this possibility by finding the intersection line and dividing one or both planes into two parts, or we could use other geometric

Program 10–1 Erasing hidden lines by painting surfaces on the screen from back to front.

```
10 'PROGRAM 10-1. ELIMINATING HIDDEN PARTS OF A 3-DIMENSIONAL SCENE
20      'PROGRAM READS SERIES OF DRAW STRINGS FROM DATA STATEMENTS, ALONG
30      'WITH Z VALUES AND COLOR CODES FOR EACH STRING. SHAPES ARE STORED
40      'ON THE BASIS OF THEIR Z VALUES, AND THEN DRAWN IN ORDER FROM
50      'THE BACK OF THE SCENE (BIGGER Z VALUES) TO THE FRONT. DRAWING
60      'OF EACH SHAPE ON TOP OF "BEHIND" SHAPES ERASES THE HIDDEN PARTS.
70  DIM DRAWSTRING$(20), XDRAW(20), YDRAW(20), ZDRAW(20), XINT(20),
      YINT(20), DRAWCOLOR(20)
80  SCREEN 1: COLOR 1,0: CLS
90  READ TOTAL                                      'READ DRAWSTRINGS
100 FOR EACH = 1 TO TOTAL
110     READ DRAWSTRING$(EACH), XDRAW(EACH), YDRAW(EACH), ZDRAW(EACH),
              XINT(EACH), YINT(EACH), DRAWCOLOR(EACH)
120 NEXT
130     'SORT THE SHAPES ON Z VALUES
140 FOR PLACE = 1 TO TOTAL - 1
150     BIGGEST = PLACE
160     FOR REST = PLACE + 1 TO TOTAL     'LOOK THROUGH REST OF Z VALUES
170         IF ZDRAW(BIGGEST) < ZDRAW(REST) THEN BIGGEST = REST
180     NEXT
190     SWAP XDRAW(PLACE),XDRAW(BIGGEST)          'PUT VALUES FOR ENTRY WITH
200     SWAP YDRAW(PLACE),YDRAW(BIGGEST)          'BIGGEST Z VALUE IN THIS
210     SWAP ZDRAW(PLACE),ZDRAW(BIGGEST)          'PLACE. PUT VALUES FOR ENTRY
220     SWAP XINT(PLACE),XINT(BIGGEST)            'CURRENTLY IN PLACE IN THE
230     SWAP YINT(PLACE),YINT(BIGGEST)            'ENTRY VACATED BY BIGGEST
240     SWAP DRAWSTRING$(PLACE),DRAWSTRING$(BIGGEST)
250     SWAP DRAWCOLOR(PLACE),DRAWCOLOR(BIGGEST)
260 NEXT
270 FOR EACH = 1 TO TOTAL
280     PSET (XDRAW(EACH),YDRAW(EACH)),3          'GO TO PLACE TO DRAW
290     DRAW "C3;XDRAWSTRING$(EACH);"             'DRAW SHAPE IN BACKGROUND
300     PAINT (XINT(EACH),YINT(EACH)),DRAWCOLOR(EACH),3          'PAINT
310     PSET (XDRAW(EACH),YDRAW(EACH)),DRAWCOLOR(EACH)
320     DRAW "C=DRAWCOLOR(EACH);XDRAWSTRING$(EACH);"     'REDRAW IN RIGHT COLOR
330 NEXT
340 DATA 16
350 DATA "S4L80E25L20E25L15E20L10E15F15L10F20L15F25L20F25",
      315,135,50,295,125,1
360 DATA "S3L160;E50;F50;L50;U50;R60;F50",300,145,40,280,144,2
370 DATA "S3;L46U46F46",252,144,39,220,140,3
380 DATA "S3;L46E46D46",216,144,38,210,135,3
390 DATA "S3L80E25L20E25L15E20L10E15F15L10F20L15F25L20F25",
      115,165,30,95,153,1
400 DATA "S1L80E25L20E25L15E20L10E15F15L10F20L15F25L20F25",
      40,50,100,30,45,1
410 DATA "S1L80E25L20E25L15E20L10E15F15L10F20L15F25L20F25",
      55,50,95,45,45,1
420 DATA "S2L80E25L20E25L15E20L10E15F15L10F20L15F25L20F25",
      85,65,80,75,60,1
430 DATA "S3F3G4H3D10F20G3H20G20H3E20L42F20G3H20D20L3U35L5D1L5D1L5D1L5
      U5E15R5F15R37F3",65,160,25,55,160,3
440 DATA "S4U9H5F5U7D7E5G5",25,144,24,25,142,3
450 DATA "S3H10G5E5H5F5U5D5",25,144,23,25,144,3
460 DATA "S3E10U5D5E5G5R5",25,146,23,25,146,3
470 DATA "S4L5H5U5R2U2R1U2R2U2R1U2R2U2D2R1D2R2D2R2D2R2D2R1D5G5",
       160,188,21,160,183,2
480 DATA "S3L5H5U5R2U2R1U2R2U2R1U2R2U2D2R1D2R2D2R2D2R2D2R1D5G5",
       171,182,20,168,180,2
490 DATA "S3L5H5U5R2U2R1U2R2U2R1U2R2U2D2R1D2R2D2R2D2R2D2R1D5G5",
       167,187,19,165,185,3
500 DATA "S3G13F3E13D15R5U15F13E3H13L15",158,185,22,158,187,1
510 IF INKEY$ = "" THEN 510
520 END
```

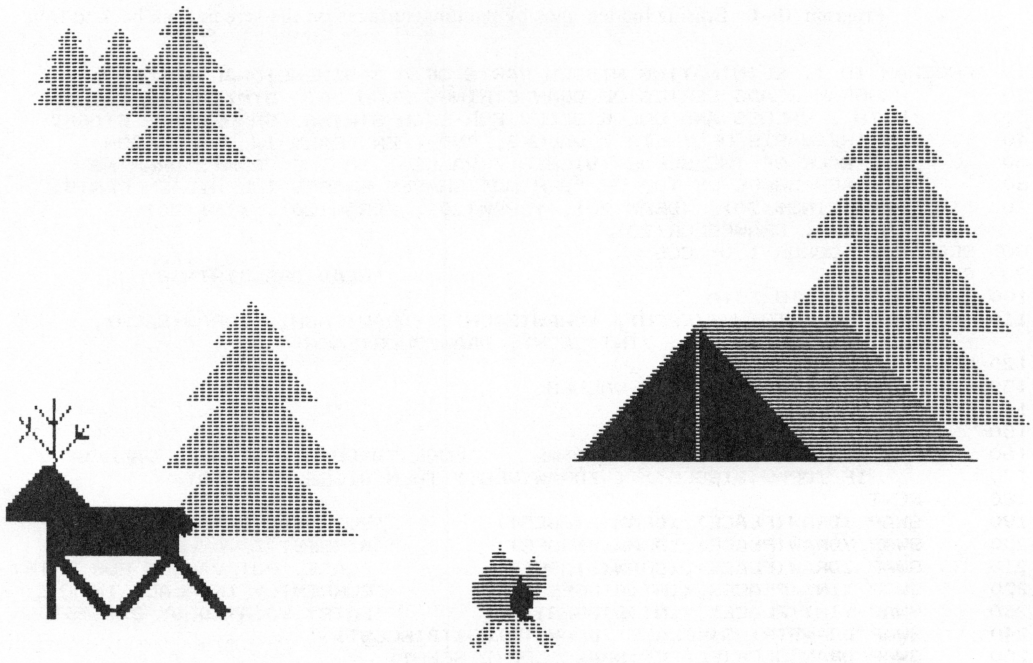


Figure 10–4 Overlapping polygon planes displayed by Prog. 10–1.

methods to determine which part of each surface was visible. Various curved surfaces, rather than planes, could be overlapped if we treated the curved surfaces as plane surfaces. We could blank out the interiors of the curved surfaces, then draw lines to indicate the curvature (Fig. 10–6).

As an alternative to erasing the entire interior of all surfaces in a picture, we can employ methods to identify and eliminate only the surfaces that are actually hidden. For objects with symmetry, we can usually set up methods that decide visibility between two opposite faces. The box of Fig. 10–1 has three pairs of opposite faces. We can see only one face from each of these pairs at any one time. If we see the face with vertices 1, 2, 3, and 4, then we cannot see face 5, 6, 7, 8. A program to eliminate hidden lines for this box need only display the side from each pair with the smaller Z value. This is accomplished with Prog. 10–2, which inputs three-dimensional coordinates for each vertex of a box. All lines for the box are displayed, then the screen is cleared and the visible sides only are redrawn. If the box were rotating, Prog. 10–2 would recalculate the visible faces from the new Z values after each rotation. Similar methods can be used for other symmetrical objects.

A method for eliminating hidden surfaces that does not depend on object symmetry is given in Prog. 10–3. A rectangular bounding area is established for each face of the input object. Each vertex point of each face is tested against all

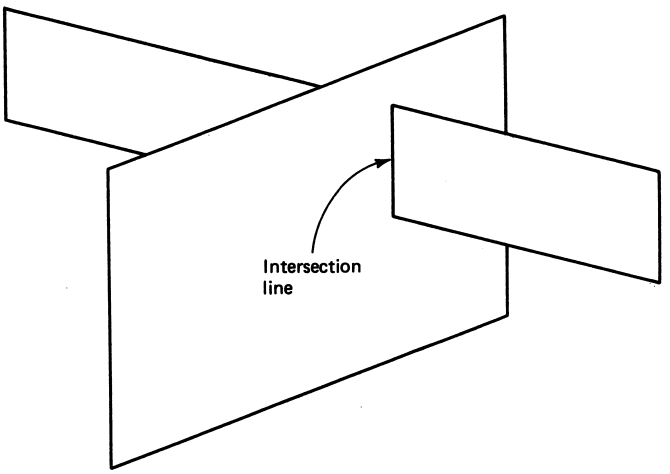
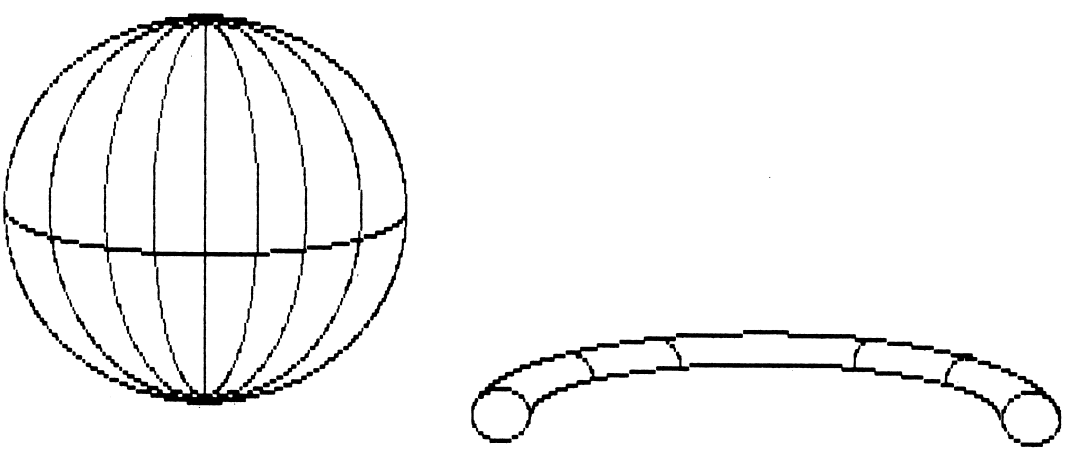


Figure 10–5 Planes with variable depth may overlap and intersect so that part of each plane is obscured by the other.

other faces to determine whether that point is within the rectangular boundary of the face. If the point is within the boundary and has greater depth, the visibility flag for the face containing that point is set to "OFF." After all faces have been tested, the object is redrawn with visible faces only. Figure 10–7 shows the two views of an object output by this program.

Objects with complex geometric shapes may be inaccurately drawn with the techniques used in Prog. 10–3, since the visibility test is highly simplified. Each face of the object is determined to be either completely visible or completely

Figure 10–6 Representation of three-dimensional curved surfaces with lines drawn to indicate curvature.



Program 10–2 Eliminating hidden lines by displaying only the one visible surface from each pair of symmetrical faces of an object (box).

```
10 'PROGRAM 10-2. ERASING HIDDEN LINES BY SYMMETRY.
20      'READS AND STORES THE POINTS OF A 3-DIMENSIONAL BOX.
30      'COMPARES THE Z VALUES FOR EACH PAIR OF SYMMETRIC
40      'SURFACES AND DRAWS ONLY THE NEARER SURFACE.
50  DIM X(8), Y(8), Z(8)
60  SCREEN 1: CLS
70      '****************** READ VERTICES FOR CUBE ******************
80  FOR K = 1 TO 8
90        READ X(K), Y(K), Z(K)
100       IF X(K)<0 OR X(K)>319 OR Y(K)<0 OR Y(K)>199 THEN 590
110 NEXT
120     '********************** DRAW ALL FACES **********************
130 FOR K = 1 TO 3
140     LINE (X(K),Y(K)) - (X(K+1),Y(K+1))
150     LINE (X(K),Y(K)) - (X(K+4),Y(K+4))
160 NEXT
170 LINE (X(4),Y(4)) - (X(1),Y(1))
180 LINE (X(4),Y(4)) - (X(8),Y(8))
190 FOR K = 5 TO 7
200     LINE (X(K),Y(K)) - (X(K+1),Y(K+1))
210 NEXT
220 LINE (X(8),Y(8)) - (X(5),Y(5))
230 FOR DELAY = 1 TO 1000: NEXT
240     '****************** DRAW ONLY VISIBLE FACES ******************
250 CLS
260 IF Z(1) > Z(5) THEN 320
270 FOR K = 1 TO 3                          'DRAW FACE CONTAINING POINT 1
280     LINE (X(K),Y(K)) - (X(K+1),Y(K+1))
290 NEXT
300 LINE (X(4),Y(4)) - (X(1),Y(1))
310 GOTO 360
320 FOR K = 5 TO 7                          'DRAW FACE CONTAINING POINT 5
330     LINE (X(K),Y(K)) - (X(K+1),Y(K+1))
340 NEXT
350 LINE (X(8),Y(8)) - (X(5),Y(5))
360 IF Z(1) > Z(4) THEN 420
370 LINE (X(1),Y(1)) - (X(2),Y(2))      'DRAW FACE CONTAINING POINT 1
380 LINE (X(2),Y(2)) - (X(6),Y(6))
390 LINE (X(6),Y(6)) - (X(5),Y(5))
400 LINE (X(5),Y(5)) - (X(1),Y(1))
410 GOTO 460
420 LINE (X(4),Y(4)) - (X(3),Y(3))      'DRAW FACE CONTAINING POINT 4
430 LINE (X(3),Y(3)) - (X(7),Y(7))
440 LINE (X(7),Y(7)) - (X(8),Y(8))
450 LINE (X(8),Y(8)) - (X(4),Y(4))
460 IF Z(1) > Z(2) THEN 520
470 LINE (X(1),Y(1)) - (X(4),Y(4))      'DRAW FACE CONTAINING POINT 1
480 LINE (X(4),Y(4)) - (X(8),Y(8))
490 LINE (X(8),Y(8)) - (X(5),Y(5))
500 LINE (X(5),Y(5)) - (X(1),Y(1))
510 GOTO 590
520 LINE (X(2),Y(2)) - (X(3),Y(3))      'DRAW FACE CONTAINING POINT 2
530 LINE (X(3),Y(3)) - (X(7),Y(7))
540 LINE (X(7),Y(7)) - (X(6),Y(6))
550 LINE (X(6),Y(6)) - (X(2),Y(2))
560     '************************************************************
570 DATA 150,110,160,150,140,110,90,140,100,90,110,150
580 DATA 230,50,60,230,80,10,170,80,0,170,50,50
590 END
```

hidden, so that partially hidden areas cannot be displayed. Also, the bounding face area used to test for hidden points will become more inaccurate as the face area differs from a rectangle. A long, thin, diagonal surface will have a large rectangular boundary which could lead to an erroneous determination of visibility.

Program 10–3 Erasing hidden surfaces by locating hidden vertices.

```
10 'PROGRAM 10-3. ERASING HIDDEN LINES BY RECTANGULAR BOUNDARIES.
20      'ALL FACES ARE INITIALLY SET TO "ON" AND ARE DRAWN.
30      'SMALLEST & LARGEST X AND Y AND Z VALUES ARE FOUND
40      'FOR EACH FACE AND STORED. THESE X AND Y VALUES ARE
50      'CONSIDERED TO THE "BOUNDING RECTANGLE" OF THE FACE.
60      'TAKING EACH FACE, ALL OTHER FACES ARE TESTED AGAINST
70      'IT. IF ANY VERTEX OF THE TEST FACE FALLS WITHIN THE
80      'BOUNDING RECTANGLE, Z VALUES ARE COMPARED. IF THE
90      'Z VALUE OF THE TEST FACE VERTEX IS GREATER THAN THE
100     'LARGEST Z VALUE FOR THE FACE, THEN THE TEST FACE IS
110     'TURNED OFF. WHEN ALL FACES HAVE BEEN TESTED AGAINST
120     'ALL OTHER FACES, WE RE-DRAW THE FIGURE, USING ONLY
130     'THOSE FACES THAT ARE STILL "ON".
140     '*************************************************************************
150 DIM X(9,6), Y(9,6), Z(9,6), C$(9)
160 DIM XS(9), YS(9), ZS(9), XL(9), YL(9), ZL(9)
170 SCREEN 1: CLS
180     '************************** READ POINTS **************************
190 READ N          'N IS HOW MANY SURFACES
200 FOR S = 1 TO N
210     C$(S) = "ON"
220     READ NV(S)                          'NV IS # OF VERTICES IN SURFACE
230     FOR V = 1 TO NV(S)
240         READ X(S,V), Y(S,V), Z(S,V)
250         IF X(S,V)<0 OR X(S,V)>319 OR Y(S,V)<0 OR Y(S,V)>199 THEN 820
260     NEXT
270     X(S,NV(S)+1) = X(S,1)
280     Y(S,NV(S)+1) = Y(S,1)
290 NEXT
300 GOSUB 620                                   'DRAW FACES THAT ARE ON
310     '************* FIND OUTER BOUNDARIES FOR EACH FACE ****************
320 FOR S = 1 TO N
330     XS(S) = X(S,1)          'XS & XL ARE SMALLEST & LARGEST X VALUES
340     YS(S) = Y(S,1)
350     ZS(S) = Z(S,1)
360     FOR V = 2 TO NV(S)
370         IF X(S,V) < XS(S) THEN XS(S) = X(S,V)
380         IF Y(S,V) < YS(S) THEN YS(S) = Y(S,V)
390         IF Z(S,V) < ZS(S) THEN ZS(S) = Z(S,V)
400         IF X(S,V) > XL(S) THEN XL(S) = X(S,V)
410         IF Y(S,V) > YL(S) THEN YL(S) = Y(S,V)
420         IF Z(S,V) > ZL(S) THEN ZL(S) = Z(S,V)
430     NEXT
440 NEXT
450     '******************** TURN OFF HIDDEN FACES *************************
460 FOR S = 1 TO N
470     IF C$(S) = "OFF" THEN 590                       'FACE IS ALREADY OFF
480     FOR R = 1 TO N
490         IF C$(R) = "OFF" OR R = S THEN 580          'NO NEED TO COMPARE
500         FOR V = 1 TO NV(R)
510             IF X(R,V) <= XS(S) OR X(R,V) >= XL(S) OR
                    Y(R,V) <= YS(S) OR Y(R,V) >= YL(S) THEN 570
520             'OTHERWISE POINT (R,V) IS WITHIN THE BOUNDING RECTANGLE
```

Program 10–3 (cont.)

```
530             'IS IT IN FRONT OR IN BACK?
540             IF Z(R,V) <= ZL(S) THEN 580      'FACE R IS NOT IN BACK OF S
550             C$(R) = "OFF"                    'FACE R IS IN BACK OF FACE S
560             GOTO 580                         'GO ON TO TEST THE NEXT FACE
570         NEXT V
580     NEXT R
590 NEXT S
600 GOSUB 620                                    'DRAW FACES THAT ARE "ON"
610 GOTO 820
620 '########################## DRAW ROUTINE ##############################
630 CLS
640 FOR S = 1 TO N
650     IF C$(S) = "OFF" THEN 700                'SKIP THIS ONE - DON'T DRAW
660     PSET(X(S,1),Y(S,1))
670     FOR V = 2 TO NV(S) + 1
680         LINE - (X(S,V),Y(S,V))
690     NEXT
700 NEXT
710 RETURN
720 '########################################################################
730 DATA 7
740 DATA 5,90,140,60,150,140,10,150,110,0,120,70,20,90,110,50
750 DATA 4,150,110,10,150,140,10,230,80,110,230,50,100
760 DATA 5,230,50,100,230,80,110,170,80,160,170,50,150,200,10,120
770 DATA 4,170,50,150,170,80,160,90,140,60,90,110,50
780 DATA 4,90,140,50,150,140,10,230,80,110,170,80,160
790 DATA 4,150,110,0,230,50,100,200,10,120,120,70,20
800 DATA 4,90,110,50,120,70,20,200,10,120,170,50,150
810 '########################################################################
820 IF INKEY$ = "" THEN 820
830 END
```

HIDDEN LINES

The preceding two programs tested object surfaces to determine visibility. Each surface tested was deemed either entirely visible or entirely invisible. We now consider a method for determining partial visibility by testing the visibility of individual lines instead of complete surfaces. Our program will test a line to determine if any part of the line is hidden by a surface.

A line and a surface can be related in several ways. In Fig. 10–8, the various relationships between a displayed line and a surface are illustrated. For this example, we assume that the surface is nearer to us than any of the lines. Then lines B and E are completely visible, line D is completely hidden, the top overlapping segment of line C is also hidden, and the middle overlapping section of line A is invisible. To simplify our hidden line programs, we will assume that these are the only possible relationships. That is, a line can intersect a surface at no more than two points. This restricts surface shapes to be either circles, ellipses, or convex polygons (Fig. 10–9(a)), and eliminates from consideration all concave polygons (Fig. 10–9(b)) which could have more than two intersection points. We can treat objects with concave surfaces by redefining surface

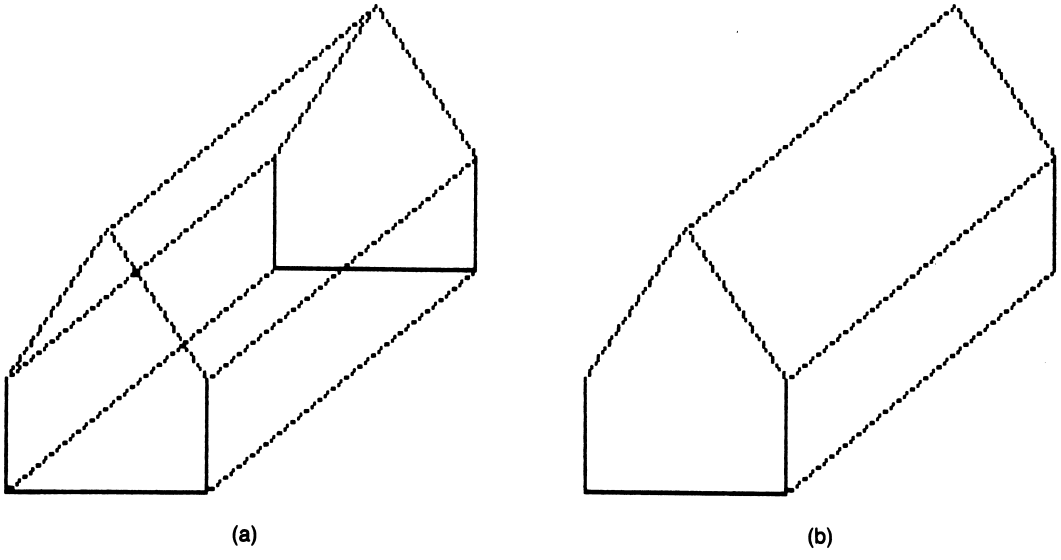


Figure 10–7 Three-dimensional object displayed (a) before and (b) after erasing hidden surfaces by Prog. 10–3.

boundaries. Any concave surface can be reorganized into two or more convex polygons. But this technique will add additional lines to objects, which may be undesirable.

Program 10–4 demonstrates a method for detecting and erasing hidden segments of lines. Any number of surfaces may be input by specifying the coordinates for all vertices in each surface. For the demonstration example, we

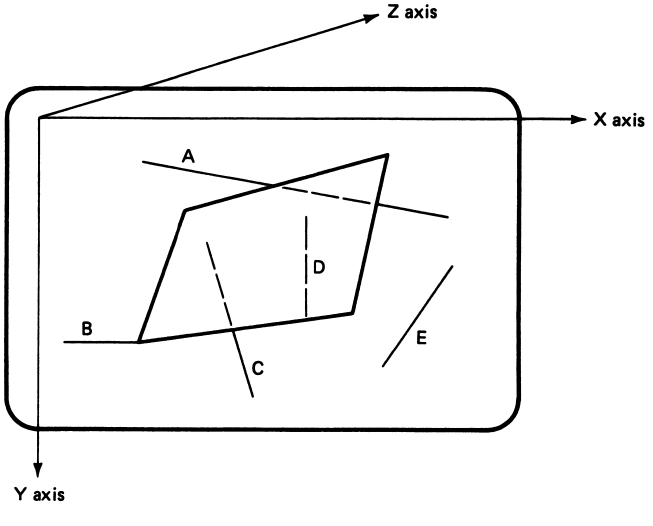


Figure 10–8 Possible line and surface relationships. Line A intersects the face boundary at two points; lines B and C intersect at one point; line D is completely hidden; and line E is completely visible. (All lines are assumed to have greater Z coordinates than the surface.)

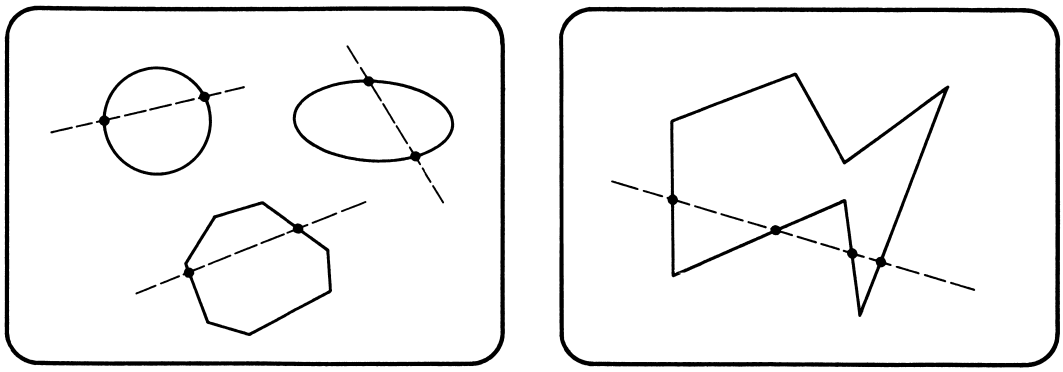


Figure 10–9 (a) Circles, ellipses, and convex polygons (interior angles less than 180 degrees) have no more than two intersection points with a straight line. (b) Concave polygons (interior angles greater than 180 degrees) can have more than two intersection points with a straight line.

have 14 surfaces with four vertices each. The surfaces could all be part of one object, as in the example, or represent several different objects. Isolated straight lines can be input as surfaces with two vertices. After all surfaces have been defined, the program draws the input lines (Fig. 10–10(a)) then finds and erases

Program 10–4 Erasing hidden line segments for partially visible lines and surfaces.

```
10 'PROGRAM 10-4. ERASING LINES ON PARTIALLY VISIBLE SURFACES.
20      'ENDPOINTS OF LINES FOR EACH SURFACE (S) ARE STORED IN ARRAYS
30      'X1,Y1,Z1 AND X2,Y2,Z2. PROGRAM DRAWS FIGURE AND THEN BEGINS
40      'TO ERASE HIDDEN LINES. EACH FACE IS TAKEN ONE AT A TIME. THE
50      'MINIMUM AND MAXIMUM X AND Y VALUES AND THE MINIMUM Z ARE FOUND
60      'TO ESTABLISH BOUNDARIES FOR THIS SURFACE. THE SLOPES AND Y
70      'INTERCEPTS OF EACH LINE OF THIS SURFACE ARE FOUND. THE PROGRAM
80      'THEN TAKES EVERY OTHER LINE OF EVERY OTHER FACE TO USE AS A
90      'TEST LINE AND TESTS IT AGAINST THE SURFACE (S). IF THE TEST LINE
100     'IS OUTSIDE THE BOUNDARIES OF THE SURFACE, THE LINE IS VISIBLE SO
110     'WE GO ON TO A NEW TEST LINE. OTHERWISE WE ATTEMPT TO FIND
120     'INTERSECTION POINTS OF THE TEST LINE AND THE VARIOUS LINES OF
130     'THE SURFACE (S). CHECKS MUST BE MADE TO DETERMINE IF THE
140     'CALCULATED INTERSECTION POINT IS ACTUALLY PART OF THE TWO LINES
150     '(PERHAPS THE TEST LINE AND/OR LINE OF THE SURFACE END BEFORE
160     'ACTUALLY INTERSECTING). Z VALUES ARE COMPUTED AND USED TO DETERMINE
170     'IF THE TEST LINE IS IN FRONT OR IN BACK OF THE SURFACE AT THE
180     'CALCULATED INTERSECTION POINT. IF WE FIND TWO INTERSECTION POINTS
190     'TO USE AS ENDPOINTS FOR A HIDDEN SEGMENT, WE ERASE THE SEGMENT.
200     'ONCE WE HAVE TESTED EVERY OTHER LINE OF THE FIGURE AGAINST
210     'THIS SURFACE AND ERASED HIDDEN SEGMENTS, WE GO ON TO THE
220     'NEXT SURFACE.
230 DIM X1(14,4), X2(14,4), Y1(14,4), Y2(14,4), Z1(14,4), Z2(14,4)
240 DIM NV(14), M(4), B(4)
250 SCREEN 1: CLS
260       '************** READ DATA POINTS ******************
270 READ NS                 'NS IS NUMBER OF SURFACES
280 FOR S = 1 TO NS
290     READ NV(S)              'NV(S) IS NUMBER OF VERTICES FOR THIS SURFACE
300     READ X1(S,1), Y1(S,1), Z1(S,1)
310     FOR V = 2 TO NV(S)
```

Program 10–4 (cont.)

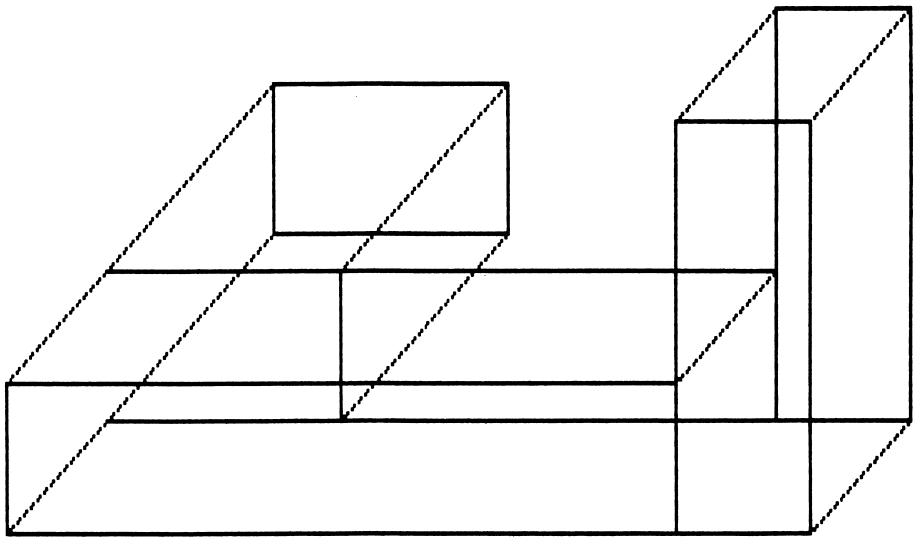
```
320         READ X1(S,V), Y1(S,V), Z1(S,V)
330         IF X1(S,V)<0 OR X1(S,V)>319 THEN 2150       'OFF SCREEN?
340         IF Y1(S,V)<0 OR Y1(S,V)>199 THEN 2150
350         X2(S,V-1) = X1(S,V)
360         Y2(S,V-1) = Y1(S,V)
370         Z2(S,V-1) = Z1(S,V)
380     NEXT
390     X2(S,NV(S)) = X1(S,1)
400     Y2(S,NV(S)) = Y1(S,1)
410     Z2(S,NV(S)) = Z1(S,1)
420 NEXT S
430     'DRAW POLYGON SURFACE
440 FOR S = 1 TO NS
450     FOR V = 1 TO NV(S)
460         LINE (X1(S,V),Y1(S,V)) - (X2(S,V),Y2(S,V))
470     NEXT V
480 NEXT S
490 FOR S = 1 TO NS
500     GOSUB 600       'FIND BOUNDARIES & EQUATIONS FOR THIS SURFACE
510     FOR SR = 1 TO NS
520         IF S = SR THEN 560
530         FOR L = 1 TO NV(SR)
540             GOSUB 900       'TEST FOR VISIBILITY. ERASE ANY HIDDEN PARTS
550         NEXT L
560     NEXT SR
570 NEXT S
580 GOTO 2150
590     '
600     '######### FIND BOUNDARY FOR SURFACE & EQUATIONS OF EACH LINE ##########
610     'FIND X, Y, AND Z BOUNDARIES FOR SURFACE
620 XL = X1(S,1)            'XL IS LOWEST X VALUE. WANT GREATEST X VALUE IN
630 XR = X1(S,1)            'XR, LOWEST Z VALUE IN ZF, LOWEST Y VALUE IN
640 YT = Y1(S,1)            'YT, AND GREATEST Y VALUE IN YB
650 YB = Y1(S,1)
660 ZF = Z1(S,1)
670 FOR K = 1 TO NV(S)
680     IF Z1(S,K) < ZF THEN ZF = Z1(S,K)
690     IF Z2(S,K) < ZF THEN ZF = Z2(S,K)
700     IF Y1(S,K) < YT THEN YT = Y1(S,K)
710     IF Y2(S,K) < YT THEN YT = Y2(S,K)
720     IF Y1(S,K) > YB THEN YB = Y1(S,K)
730     IF Y2(S,K) > YB THEN YB = Y2(S,K)
740     IF X1(S,K) > XR THEN XR = X1(S,K)
750     IF X2(S,K) > XR THEN XR = X2(S,K)
760     IF X1(S,K) < XL THEN XL = X1(S,K)
770     IF X2(S,K) < XL THEN XL = X1(S,K)
780 NEXT K
790     '################ DETERMINE LINE EQUATIONS ###################
800     'FOR EACH LINE OF THE SURFACE, THE SLOPE AND Y INTERCEPT ARE
810     'FOUND AND STORED IN M AND B
820 FOR P = 1 TO NV(S)
830     IF Y1(S,P) = Y2(S,P) THEN M(P)=0: B(P)=Y1(S,P): GOTO 870   'HORIZONTAL
840     IF X1(S,P) = X2(S,P) THEN M(P) = 9999: GOTO 870         'VERTICAL LINE
850     M(P) = (Y2(S,P) - Y1(S,P)) / (X2(S,P) - X1(S,P))
860     B(P) = Y1(S,P) - M(P) * X1(S,P)
870 NEXT P
880 RETURN
890     '
900     '###################### TEST LINE FOR VISIBILITY ######################
910     'IF THE POINTS OF THE TEST LINE (SR,L) ARE OUTSIDE THE BOUNDARIES
920    'OF THE SURFACE (S), THEN THE LINE IS VISIBLE IN RELATION TO THIS
930    'SURFACE
```

Program 10–4 (cont.)

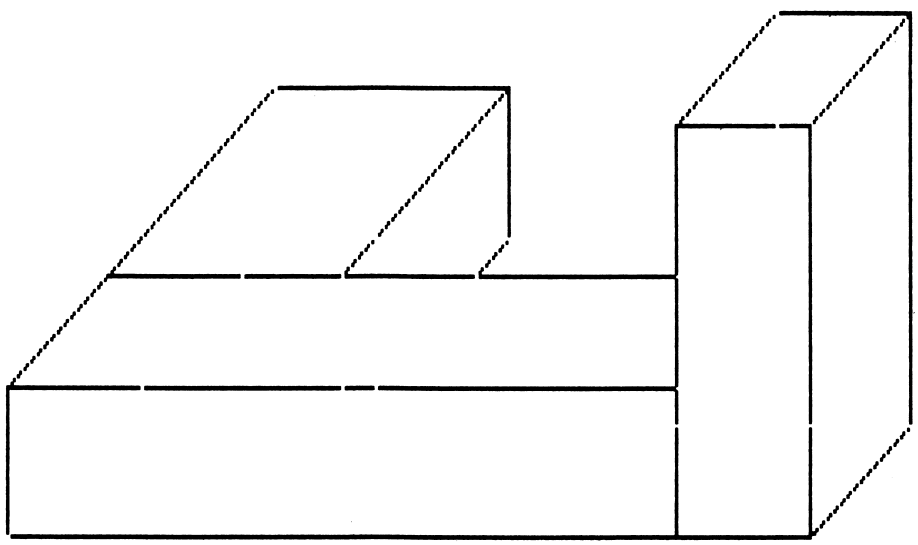
```
940 IF Z1(SR,L) <= ZF AND Z2(SR,L) <= ZF THEN 1980          'LINE IS VISIBLE
950 IF Y1(SR,L) <= YT AND Y2(SR,L) <= YT THEN 1980          'LINE IS VISIBLE
960 IF Y1(SR,L) >= YB AND Y2(SR,L) >= YB THEN 1980          'LINE IS VISIBLE
970 IF X1(SR,L) <= XL AND X2(SR,L) <= XL THEN 1980          'LINE IS VISIBLE
980 IF X1(SR,L) >= XR AND X2(SR,L) >= XR THEN 1980          'LINE IS VISI
990     'OTHERWISE AT LEAST ONE POINT IS WITHIN THE RECTANGLE THAT
1000    'BOUNDS THIS SURFACE
1010 'FIND SLOPE AND INTERCEPT OF TEST LINE. SAVE IN MT AND BT
1020 IF X1(SR,L) = X2(SR,L) THEN MT=9999: GOTO 1060          'VERTICAL LINE
1030 IF Y1(SR,L) = Y2(SR,L) THEN MT=0: BT=Y1(SR,L): GOTO 1060     'HORIZONTAL
1040 MT = (Y1(SR,L) - Y2(SR,L)) / (X1(SR,L) - X2(SR,L))
1050 BT = Y1(SR,L) - MT * X1(SR,L)
1060 C = 0                  'C IS COUNT OF ENDPOINTS OF HIDDEN SEGMENT
1070    'TEST AGAINST ALL LINES OF THE SURFACE
1080 FOR K = 1 TO NV(S)
1090    'IF THIS IS A SHARED EDGE GO ON TO NEXT LINE OF SR
1100    IF X1(S,K)=X1(SR,L) AND Y1(S,K)=Y1(SR,L) AND X2(S,K)=X2(SR,L)
            AND Y2(S,K)=Y2(SR,L) THEN 1980
1110    IF X1(S,K)=X2(SR,L) AND Y1(S,K)=Y2(SR,L) AND X2(S,K)=X1(SR,L)
            AND Y2(S,K)=Y1(SR,L) THEN 1980
1120    IF M(K)=MT THEN 1940            'LINES ARE PARALLEL
1130    'OTHERWISE FIND INTERSECTION POINT XP,YP OF TEST LINE
1140    '(L OF SR) AND SURFACE LINE (K OF S)
1150    IF X1(S,K) <> X2(S,K) THEN 1190
1160    XP = X1(S,K)                                 'FACE LINE IS VERTICAL
1170    YP = MT * XP + BT
1180    GOTO 1340
1190    IF X1(SR,L) <> X2(SR,L) THEN 1230
1200    XP = X1(SR,L)                                'TEST LINE IS VERTICAL
1210    YP = M(K) * XP + B(K)
1220    GOTO 1340
1230    IF Y1(S,K) <> Y2(S,K) THEN 1270
1240    YP = Y1(S,K)                                 'FACE LINE IS HORIZONTAL
1250    XP = (YP - BT) / MT
1260    GOTO 1340
1270    IF Y1(SR,L) <> Y2(SR,L) THEN 1310
1280    YP = Y1(SR,L)                                'TEST LINE IS HORIZONTAL
1290    XP = (YP - B(K)) / M(K)
1300    GOTO 1340
1310    XP = (BT - B(K)) / (M(K) - MT)
1320    YP = (MT * XP + BT)
1330    '
1340    'LINES INTERSECT AT XP,YP. IF XP.YP IS NOT BETWEEN THE ENDPOINTS
1350    'OF LINE K OR S, GO ON TO NEXT LINE (L OF SR)
1360     IF XP<X1(S,K) AND XP<X2(S,K) THEN 1940
1370     IF XP>X1(S,K) AND XP>X2(S,K) THEN 1940
1380     IF YP<Y1(S,K) AND YP<Y2(S,K) THEN 1940
1390     IF YP>Y1(S,K) AND YP>Y2(S,K) THEN 1940
1400    '
1410    'OTHERWISE FIND Z VALUE FOR XP,YP ON TEST LINE AND ON
1420    'SURFACE. IF Z VALUE FOR XP,YP ON TEST LINE (ZL) IS
1430    'LESS THEN Z VALUE FOR XP,YP ON SURFACE LINE (ZS) THEN
1440    'THE TEST LINE IS IN FRONT OF SURFACE, AND SO IS VISIBLE
1450     IF X1(SR,L) = X2(SR,L) THEN 1480
1460     ZL = (XP-X1(SR,L))/(X2(SR,L)-X1(SR,L))*(Z2(SR,L)-Z1(SR,L))+Z1(SR,L)
1470     GOTO 1490
1480     ZL = (YP-Y1(SR,L))/(Y2(SR,L)-Y1(SR,L))*(Z2(SR,L)-Z1(SR,L))+Z1(SR,L)
1490    'FIND Z VALUE OF INTERSECTION ON SURFACE LINE
1500     IF X1(S,K) = X2(S,K) THEN 1530
1510     ZS = (XP-X1(S,K))/(X2(S,K)-X1(S,K))*(Z2(S,K)-Z1(S,K))+Z1(S,K)
1520     GOTO 1540
1530     ZS = (YP-Y1(S,K))/(Y2(S,K)-Y1(S,K))*(Z2(S,K)-Z1(S,K))+Z1(S,K)
```

Program 10–4 (cont.)

```
1540     IF ZL < ZS THEN 1940
1550     '
1560     'IF THE POINT XP,YP IS NOT BETWEEN THE ENDPOINTS OF LINE L OF
1570     'SR, THEN XP OR YP NEEDS TO BE ADJUSTED BY BEING SET EQUAL
1580     'TO THE APPROPRIATE ENDPOINT OF L OF SR
1590     IF XP < X1(SR,L) AND XP < X2(SR,L) THEN 1620
1600     IF XP > X1(SR,L) AND XP > X2(SR,L) THEN 1620
1610     GOTO 1690
1620     IF ABS(XP - X1(SR,L)) < ABS(XP - X2(SR,L)) THEN 1660
1630     XP = X2(SR,L)
1640     YP = Y2(SR,L)
1650     GOTO 1820
1660     XP = X1(SR,L)
1670     YP = Y1(SR,L)
1680     GOTO 1820
1690     IF YP < Y1(SR,L) AND YP < Y2(SR,L) THEN 1720
1700     IF YP > Y1(SR,L) AND YP > Y2(SR,L) THEN 1720
1710     GOTO 1820
1720     IF ABS(YP - Y1(SR,L)) < ABS(YP - Y2(SR,L)) THEN 1760
1730     YP = Y2(SR,L)
1740     XP = X2(SR,L)
1750     GOTO 1820
1760     YP = Y1(SR,L)
1770     XP = X1(SR,L)
1780     '
1790     'HAVE WE INADVERTENTLY GENERATED AN EDGE LINE? IF THE TEST
1800     'LINE HAS AN EQUAL SLOPE AND SHARES A POINT WITH ANY LINE
1810     'OF THIS SURFACE, THEN DON'T ERASE
1820     FOR J= 1 TO NV(S)
1830          IF MT = M(J) AND YP = M(J) * XP + B(J) THEN 1980
1840     NEXT J
1850     IF C <> 0 THEN 1900
1860     XA = XP          'THIS IS THE FIRST POINT OF THE HIDDEN SEGMENT
1870     YA = YP          'STORE IN XA, YA
1880     C = C + 1
1890     GOTO 1940
1900     IF XP = XA AND YP = YA THEN 1940          'SAME POINT (VERTEX)
1910     XD = XP          'STORE SECOND POINT IN XD, YD
1920     YD = YP
1930     C = C + 1
1940 NEXT K
1950     '
1960 IF C < 2 THEN 1980
1970 LINE (XA,YA) - (XD,YD),0              'ERASE HIDDEN LINE SEGMENT
1980 RETURN
1990 '##############################################################################
2000 DATA 14
2010 DATA 4,30,130,67,30,170,71,230,170,21,230,130,17
2020 DATA 4,230,60,10,230,170,21,270,170,11,270,60,0
2030 DATA 4,270,60,0,270,170,11,300,140,61,300,30,50
2040 DATA 4,260,30,60,260,140,71,300,140,61,300,30,50
2050 DATA 4,130,100,99.5,130,140,103.5,260,140,71,260,100,67
2060 DATA 4,130,100,99.5,130,140,103.5,180,90,186.8,180,50,182.8
2070 DATA 4,110,50,200.3,110,90,204.3,180,90,186.8,180,50,182.8
2080 DATA 4,30,130,67,30,170,71,110,90,204.3,110,50,200.3
2090 DATA 4,230,60,10,230,130,17,260,100,67,260,30,60
2100 DATA 4,260,30,60,230,60,10,270,60,0,300,30,50
2110 DATA 4,110,50,200.3,60,100,117,130,100,99.5,180,50,182.8
2120 DATA 4,60,100,117,30,130,67,230,130,17,260,100,67
2130 DATA 4,110,90,204.3,60,140,121,130,140,103.5,180,90,186.8
2140 DATA 4,60,140,121,30,170,71,270,170,11,300,140,61
2150 END
```



(a)



(b)

Figure 10–10 A three-dimensional object (a) input to Prog. 10–4 is displayed (b) with hidden lines erased.

each line segment hidden by a surface (Fig. 10–10(b)). In order to avoid concave polygons, some surfaces of the object were reorganized to form two convex polygons.

Each face input to Prog. 10–4 is selected in turn and all remaining lines in the display are tested for visibility with respect to the selected surface. If a line is outside the rectangular bounding area of the surface, it is visible. Otherwise, we have to test it against the actual polygon boundaries. This is done by calculating an intersection point for each boundary line of the surface. The test line and the boundary line intersect if this intersection point is between the endpoints of both lines. In this case, we check to determine whether the Z value of the test line is greater than the Z value of the boundary line at that point. If so, the point is stored as one end of the hidden line segment. If the point is between the endpoints of the surface line and not between the endpoints of the test line, we store the nearest test line endpoint as one end of the hidden line segment. For all other cases, no point is stored. When two separate points on the test line have been found in this way, we erase the line segment between the two points. If two points cannot be found, the line is completely visible with respect to this surface.

This program could be expanded to allow concave surfaces and more complex intersection possibilities. A line intersecting a concave surface can have more than two intersection points. We could treat this situation by determining whether each line endpoint was inside or outside the polygon boundary and identifying the hidden segments from one end of the line. If one endpoint is inside, we erase the line segment from this endpoint to the first intersection point with the boundary along the line from that end. We also erase the segment from the second intersection point to the third, and so on. If an endpoint is outside, erasing is done between the first and second intersection points, between the third and fourth points, and so on. It is also possible for a line to be partially visible if it intersects a surface at an interior point. This can occur if either the line or the surface have varying values for Z coordinates, and one end of the line is in front of the plane and the other end behind the plane. We could locate the intersection point in this case by using the defining equations for three-dimensional lines and planes, but then more computation is required in the program.

We have discussed a few of the many possible methods for erasing hidden lines and surfaces. Some methods work only for restricted shapes. Some methods require more memory or more computation time than others. As we allow more complicated shapes in our displays, we can expect the erasing techniques to become more complicated. Usually, we want to select a method that will erase hidden sections in the shortest time. This is especially important for applications involving animated displays.

10–4 PERSPECTIVE VIEWS

Erasing hidden lines in a three-dimensional scene provides depth information and adds realism to our flat representation. An additional means for achieving realism

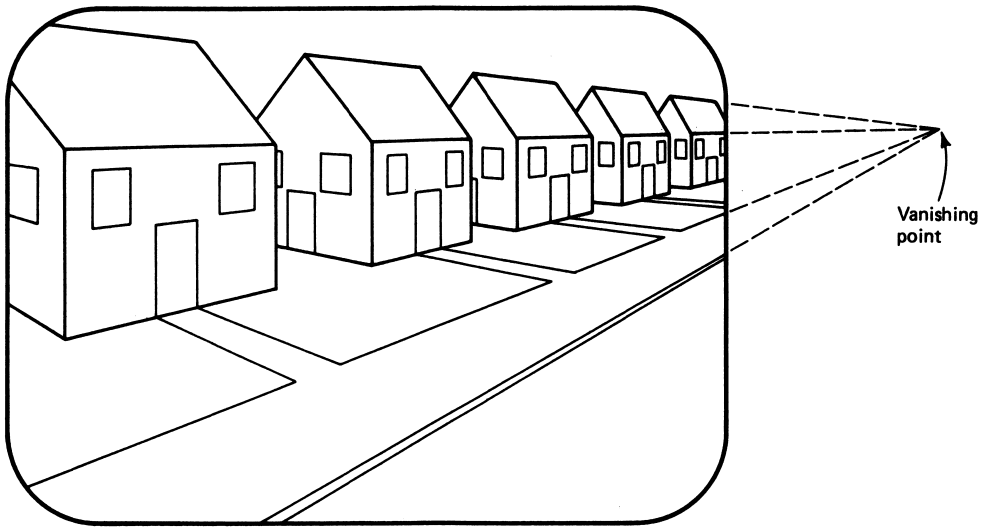


Figure 10–11 In a perspective view, parallel lines are redrawn to converge to a vanishing point so that distant objects are displayed smaller than nearer objects.

and depth in our displays is to project objects onto the screen in **perspective**. When we view natural objects, they appear smaller when they are farther away from us. A row of buildings, as in Fig. 10–11, appears as though the closer buildings are larger than the more distant buildings. In this perspective view, parallel lines appear nearer to each other in the distance than they do when closer.

To get a perspective view, we can lay out on graph paper a scene such as that shown in Fig. 10–11 and project all parallel lines so that they meet at a distant point, called the **vanishing point**. The X and Y coordinates for the objects are then determined and plotted. In this way, we can display a perspective view of any scene drawn on graph paper. But each change in the scene and each different scene requires that we repeat the entire process from the graph paper layout.

Another way to achieve perspective is to use transformation equations for perspective in the display program. This provides us with greater flexibility. Animated scenes can then be projected in perspective, and we can put repeated patterns in perspective without having to manually determine the view for each occurrence of the pattern. The coordinates of the general house design in Fig. 10–11 can be defined once, then repeatedly plotted in relative size according to the depth position.

For any coordinate point (X,Y,Z) of a three-dimensional scene, the transformed position (XP,YP) on the screen to provide a perspective view is calculated as

$$XP = XV + (XV - X) * ZV/(Z - ZV)$$
$$YP = YV + (YV - Y) * ZV/(Z - ZV) \qquad (10\text{–}1)$$

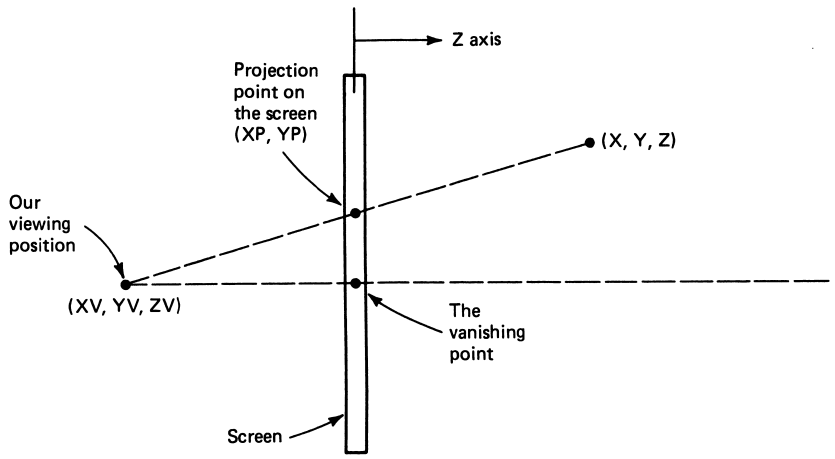


Figure 10–12 A point with coordinates (X,Y,Z) is projected onto the screen at position (XP,YP) by a perspective transformation when our viewing position is at (XV,YV,ZV).

where the point (XV,YV,ZV) is our **viewing position** in front of the screen. The point (XV,YV) on the screen is the vanishing point for the perspective view. Figure 10–12 illustrates the relationship between these various coordinate values with a side view of the video screen. For points with Z = 0 there is no change in coordinates: (XP,YP) = (X,Y). For points in back of the screen (Z > 0), the projection onto the screen gets closer to the vanishing point as Z increases. The coordinate ZV must always be a negative number, since we are viewing from in front of the screen. Larger magnitudes of ZV will produce less perspective (less converging of parallel lines). As we move the viewing point closer to the screen, we increase the perspective view of the objects by displaying greater convergence of parallel lines.

In Prog. 10–5, we calculate and display a perspective view of a road lined with telephone poles. The resulting output is shown in Fig. 10–13. This program illustrates the technique of repeatedly plotting a defined object (in this case, a telephone pole) by varying its depth and calculating the transformed positions from the perspective equations (10–1). We can vary the position of the vanishing point and the value of ZV, as long as objects are not projected off the screen.

A perspective view of a single object is displayed by Prog. 10–6. The figure is defined as an orthographic projection in the center of the screen. By selecting various viewing positions, we can display different perspective views of the object (Fig. 10–14). This program illustrates a general method for defining a three-dimensional object in the screen coordinate system and projecting a particular view using the perspective transformation equations.

Hidden surfaces are identified in Prog. 10–6 by calculating distances from the viewing position. Since the box used for this example is symmetrical, we check the distances of symmetrically opposite faces. The face from each pair that

Program 10–5 Drawing a three-dimensional scene with repeated perspective views of an object (road lined with telephone poles).

```
10 'PROGRAM 10-5. DRAWING TELEPHONE POLES IN PERSPECTIVE.
20      'TELEPHONE POLES ARE DEFINED ONCE IN DATA STATEMENTS.
30      'A VIEWING POINT (XV,YV,ZV) IS ESTABLISHED THROUGH INPUT.
40      'PERSPECTIVE EQUATIONS ARE USED TO CALCULATE NEW X AND Y
50      'COORDINATES FOR VARYING VALUES OF Z.
60  DIM X1(8), Y1(8), X2(8), Y2(8)
70  SCREEN 0: WIDTH 80: CLS
80      'ESTABLISH VIEWING POINT
90  PRINT "X AND Y MUST BE ON SCREEN, Z MUST BE NEGATIVE"
100 INPUT "X,Y,Z OF VIEWING POINT"; XV,YV,ZV
110 IF XV<0 OR XV>319 OR YV<0 OR YV>199 OR ZV>=0 THEN 190
120 SCREEN 1: CLS
130 FOR K = 1 TO 6
140     READ X1, Y1, X2, Y2
150     FOR Z = 0 TO 5000 STEP 500
160         'CALCULATE CONSTANT PART OF EQUATION
170         P = -ZV / (Z - ZV)
180         'CALCULATE POINTS FOR LINE AT THIS Z VALUE
190         XA = XV + (X1 - XV) * P
200         YA = YV + (Y1 - YV) * P
210         XB = XV + (X2 - XV) * P
220         YB = YV + (Y2 - YV) * P
230         LINE (XA,YA) - (XB,YB)
240     NEXT
250 NEXT
260     'DRAW IN ROAD EDGES AND CENTER
270 FOR K = 1 TO 3
280     READ XA,YA
290     XB = XV + (XA - XV) * P
300     YB = YV + (YA - YV) * P
310     LINE (XA,YA) - (XB,YB)
320 NEXT
330     '**************************************
340 DATA 50,45,50,155
350 DATA 40,60,60,60
360 DATA 40,50,60,50
370 DATA 260,45,260,155
380 DATA 250,60,270,60
390 DATA 250,50,270,50
400 DATA 70,155,160,155,240,155
410 IF INKEY$ = "" THEN 410
420 END
```

is farther from the viewing position is not visible. Distance of a point (X,Y,Z) from (XV,YV,ZV) is computed as

$$D = SQR((X - XV) \wedge 2 + (Y - YV) \wedge 2 + (Z - ZV) \wedge 2) \quad (10\text{–}2)$$

We next need to check the other face for visibility. It is possible that neither side is visible. For example, viewing the cube from directly in front hides all other sides. The sides, top, or bottom will be visible only if the viewing position is displaced from a direct front view. For nonsymmetrical objects, we can use the method of Prog. 10–4 and the distance D, calculated from (10–2), for depth checks.

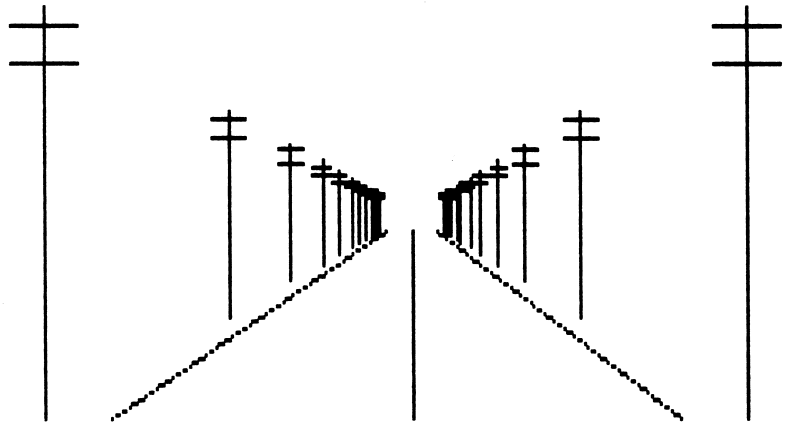


Figure 10–13 Output of Prog. 10–5, showing a perspective view of a road lined with telephone poles.

Program 10–6 Three-dimensional perspective views of a single object (box).

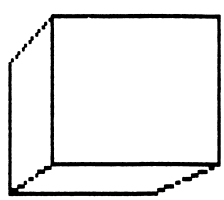
```
10 'PROGRAM 10-6. DEFINING AN OBJECT IN 3 DIMENSIONS USING PERSPECTIVE.
20      'DEFINES AN OBJECT IN 3 DIMENSIONS, ESTABLISHES A
30      'POINT FROM WHICH TO VIEW OBJECT, APPLIES PERSPECTIVE
40      'TO OBJECT'S POINTS AND DRAWS VISIBLE PORTIONS OF
50      'OBJECT. SYMMETRY OF OBJECT IS USED TO DETERMINE
60      'WHICH SIDES ARE VISIBLE.
70  DIM X(8), Y(8), Z(8), XP(8), YP(8), D(8)
80  SCREEN 1: CLS
90      '********************** READ DATA POINTS **********************
100 READ N
110 FOR K = 1 TO N
120       READ X(K), Y(K), Z(K)
130     IF X(K)<0 OR X(K)>319 OR Y(K)<0 OR Y(K)>199 THEN 760
140 NEXT
150 FOR K = 1 TO N/2 - 1
160     LINE (X(K),Y(K)) - (X(K+1),Y(K+1))
170 NEXT
180 LINE (X(N/2),Y(N/2)) - (X(1),Y(1))
190     '****************** ESTABLISH VIEWPOINT & DRAW ***************
200 LOCATE 1,1: PRINT "ENTER 0,0,0 TO QUIT"
210 PRINT "Z COORDINATE MUST BE NEGATIVE"
220 INPUT "COORDINATES OF VIEW POSITION"; XV,YV,ZV
230 IF XV = 0 AND YV = 0 AND ZV = 0 THEN 760
240 IF ZV >= 0 THEN 200
250     'PUT POINTS IN PERSPECTIVE
260 FOR K = 1 TO N
270     XP(K) = XV + (X(K) - XV) * -ZV / (Z(K) - ZV)
280     YP(K) = YV + (Y(K) - YV) * -ZV / (Z(K) - ZV)
290 NEXT
300     'FIND DISTANCES OF POINTS FROM VIEWPOINT
310 FOR K = 1 TO N
320     D(K) = SQR((X(K)-XV)^2 + (Y(K)-YV)^2 + (Z(K)-ZV)^2)
330 NEXT
340 GOSUB 360           'DRAW
350 GOTO 190
360 '################################## DRAW ROUTINE ##################################
```

Program 10–6 (cont.)

```
370 CLS
380 IF D(1) = D(5) THEN 490
390 IF D(1) > D(5) THEN 450
400 FOR K = 1 TO 3                                  'DRAW FACE CONTAINING POINT 1
410     LINE (XP(K),YP(K)) - (XP(K+1),YP(K+1))
420 NEXT
430 LINE (XP(4),YP(4)) - (XP(1),YP(1))
440 GOTO 490
450 FOR K = 5 TO 7                                  'DRAW FACE CONTAINING POINT 5
460     LINE (XP(K),YP(K)) - (XP(K+1),YP(K+1))
470 NEXT
480 LINE (XP(8),YP(8)) - (XP(5),YP(5))
490 IF D(1) = D(4) THEN 600
500 IF D(1) > D(4) THEN 560
510 LINE (XP(1),YP(1)) - (XP(2),YP(2))              'DRAW FACE CONTAINING POINT 1
520 LINE (XP(2),YP(2)) - (XP(6),YP(6))
530 LINE (XP(6),YP(6)) - (XP(5),YP(5))
540 LINE (XP(5),YP(5)) - (XP(1),YP(1))
550 GOTO 600
560 LINE (XP(4),YP(4)) - (XP(3),YP(3))              'DRAW FACE CONTAINING POINT 4
570 LINE (XP(3),YP(3)) - (XP(7),YP(7))
580 LINE (XP(7),YP(7)) - (XP(8),YP(8))
590 LINE (XP(8),YP(8)) - (XP(4),YP(4))
600 IF D(1) = D(2) THEN 710
610 IF D(1) > D(2) THEN 670
620 LINE (XP(1),YP(1)) - (XP(4),YP(4))              'DRAW FACE CONTAINING POINT 1
630 LINE (XP(4),YP(4)) - (XP(8),YP(8))
640 LINE (XP(8),YP(8)) - (XP(5),YP(5))
650 LINE (XP(5),YP(5)) - (XP(1),YP(1))
660 GOTO 710
670 LINE (XP(2),YP(2)) - (XP(3),YP(3))              'DRAW FACE CONTAINING POINT 2
680 LINE (XP(3),YP(3)) - (XP(7),YP(7))
690 LINE (XP(7),YP(7)) - (XP(6),YP(6))
700 LINE (XP(6),YP(6)) - (XP(2),YP(2))
710 RETURN
720     '*****************************************************************
730 DATA 8
740 DATA 150,80,0,150,120,0,100,120,0,100,80,0
750 DATA 150,80,30,150,120,30,100,120,30,100,80,30
760 END
```

Figure 10–14 Perspective view of a three-dimensional object, displayed by Prog. 10–6 for a viewing position below and to the left of the object.

```
ENTER COORDINATES OF VIEWING POSITION
Z COORDINATE MUST BE NEGATIVE
ENTER 0,0,0 TO QUIT
```



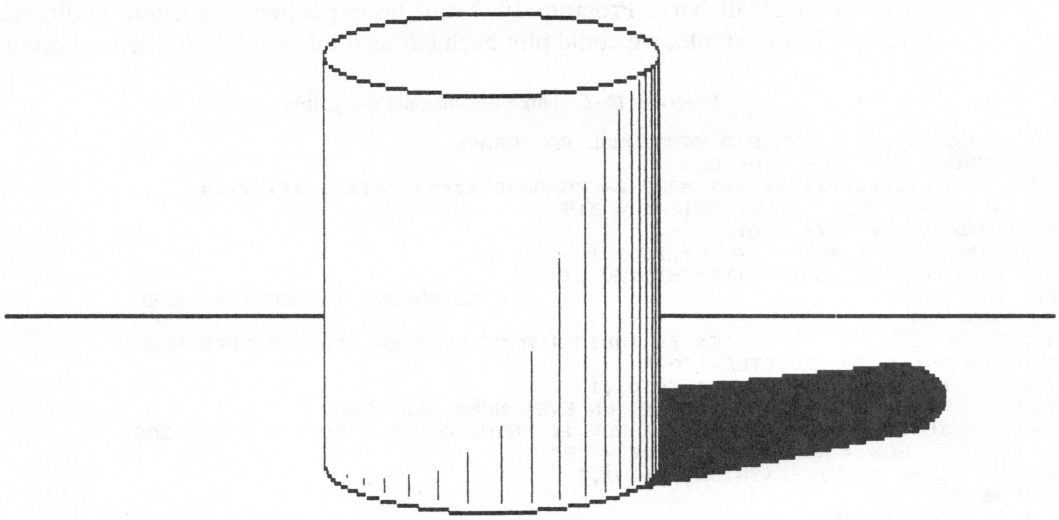
10–5 SHADING AND HIGHLIGHTING

Depth information and realism can be enhanced in our three-dimensional scenes through the use of shading. Figure 10–15 illustrates a possible shading pattern for curved surfaces. Shading patterns can help to establish the contour or curvature of surfaces. Darker areas and shadows also help to identify the back parts of objects. To add shadows, we choose some position for the light source, such as to the left and in front of the screen, and apply the shading and shadows on the sides opposite from the light source. In general, subtle shading patterns and shadows are difficult to achieve, but simple shading of the sides opposite a selected light source position can be effective.

Some of the shading patterns discussed in Chapter 3 could be defined as routines to be applied to the interior areas of objects. The boundary of an area to be shaded could be specified and the shading pattern applied to that area. Such routines are called shading masks and can be set up to shade various polygon, circular, or elliptic areas.

Highlighting can be used as an alternative to erasing hidden lines in a three-dimensional display. With this technique, we add depth information by simply identifying the front lines of objects. This can be accomplished by sorting the lines according to depth, and emphasizing closer lines with bright colors or with double-wide lines. Figure 10–16 shows an example of highlighting the front lines by making them bigger than the back, or hidden, lines. For objects drawn in outline form, highlighting can provide a quick method for identifying the front from the back of objects.

Figure 10–15 Shading patterns and shadows can help to add realism to three-dimensional scenes.



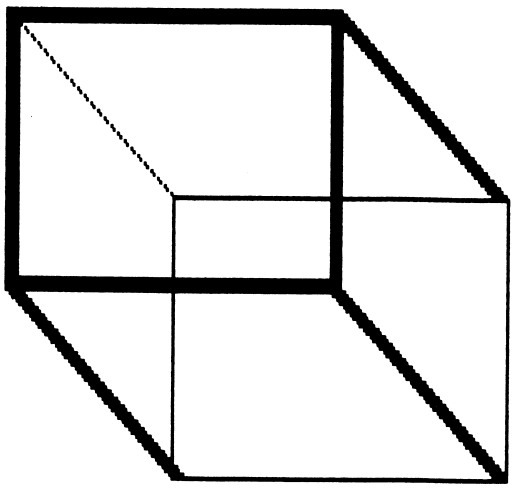


Figure 10–16 Highlighted lines of a three-dimensional object can be used to identify nearer sides.

10–6 GRAPHS

Three-dimensional charts and graphs can be effective methods for presenting multiple relationships. The three-dimensional bar chart produced by Prog. 10–7 plots relationships between three variables: population figures for two cities over three decades. Figure 10–17 shows the output for the data chosen. This program uses the techniques discussed with Prog. 10–1. More distant surfaces are drawn first, so that the nearer surfaces overlap and erase the back surfaces. The height of the bars is scaled and plotted on the screen by the same methods discussed in Chapter 4. Spacing between bars is increased to display more clearly the height and position of all bars. Program 10–7 can be expanded to include additional variables. For example, we could plot each bar as total population, then subdivide

Program 10–7 Three-dimensional bar graph.

```
10 'PROGRAM 10-7. THREE DIMENSIONAL BAR GRAPH
20  SCREEN 1: COLOR 0,0: CLS
30      '****************** MAKE BACKGROUND **********************
40  X = 230: XLEFT = 20: XRIGHT = 319
50  LINE (X,0) - (X,120),1
60  LINE (XLEFT,30) - (XLEFT,150),1
70  LINE (XRIGHT,30) - (XRIGHT,150),1
80  A = 600                                  'A IS AMOUNT TO USE FOR LABEL
90  ROW = 4
100 C$ = "E"              'C$ IS INDICATOR OF EVEN OR ODD NUMBERED ROW
110 FOR Y = 0 TO 120 STEP 120/6
120     LINE (X,Y) - (XLEFT,Y+30),1
130     'ONLY PRINT LABEL AMOUNT ON EVEN NUMBERED ROWS
140     IF C$ = "E" THEN LOCATE ROW,1: PRINT A: C$ = "O": A = A - 200:
           ROW = ROW + 5 ELSE C$ = "E"
150     LINE (X,Y) - (XRIGHT,Y+30),1
160 NEXT
170     'MAKE BASE
```

Program 10–7 (cont.)

```
180 LINE (20,150) - (109,180): LINE - (319,150),1
190 LOCATE 1,1: PRINT "Population in";
200 LOCATE 2,1: PRINT " thousands";
210 LOCATE 23,17:PRINT "1950";
220 LOCATE 22,26:PRINT "1960";
230 LOCATE 21,35:PRINT "1970";
240 LOCATE 25,1
250 PRINT "  Buffalo                    Atlanta";
260 SLOPELEFT = -30/210                  'SLOPE OF LEFT GRID MARKINGS
270 SLOPERIGHT = 30/89                   'SLOPE OF RIGHT GRID MARKINGS
280 H = 40                               'H IS HORIZONTAL MEASUREMENT OF EACH BA
290 C = 0                'C IS COUNT OF HOW MANY CITIES HAVE BEEN GRAPHED
300 XLEFT = 40                           'LOWER LEFT POINT OF A BAR IS XL,YL
310 YLEFT = 150
320 FILL = 2                     'F IS COLOR TO USE FOR FILLED IN AREA OF BAR
330 OUTLINE = 1
340 XSTART = 78
350     'MAKE CITY-COLOR CODE BLOCK
360 LINE (XSTART,191) - (XSTART+20,199),FILL,BF
370     '***************** MAKE BARS FOR CITIES ********************
380 FOR K = 1 TO 3
390     READ V
400     'SCALE POPULATION VALUE TO PIXEL RANGE
410     HT = (120 * V) / 600
420     GOSUB 570                                    'FILL IN BOX AREA
430     GOSUB 700                                    'MAKE BOX OUTLINE
440     GOSUB 760
450     XLEFT = XLEFT + 73                           'MOVE OVER AND UP FOR NEXT BAR
460     YLEFT = YLEFT - 10
470 NEXT
480 COUNT = COUNT + 1
490 IF COUNT = 2 THEN 870                            'ARE WE DONE YET?
500     'RESET VARIABLES FOR NEXT CITY'S VALUES
510 XLEFT = 91
520 YLEFT = 165
530 FILL = 3
540 OUTLINE = 1
550 XSTART = 272
560 GOTO 350
570 '###################### FILL IN BOX AREA ########################
580 BLEFT = (YLEFT-HT) - SLOPELEFT * XLEFT  'Y INTERCEPT-UPPER LEFT EDGE OF BAR
590 BRIGHT=(YLEFT)-SLOPERIGHT*XLEFT   'Y INTERCEPT - LOWER LEFT EDGE OF BAR
600 YUL = SLOPELEFT * XLEFT + BLEFT              'Y OF UPPER LEFT OF BAR
610 YLL = SLOPERIGHT * XLEFT + BRIGHT            'Y OF LOWER LEFT OF BAR
620 YUM = SLOPELEFT * (XLEFT+H/2) + BLEFT        'Y OF UPPER MIDDLE OF BAR
630 YLM = SLOPERIGHT * (XLEFT+H/2) + BRIGHT      'Y OF LOWER MIDDLE OF BAR
640 BLEFT = YLM - SLOPELEFT * (XLEFT + H/2)
650 BRIGHT = YUM - SLOPERIGHT * (XLEFT + H/2)
660 YUR = SLOPERIGHT * (XLEFT+H) + BRIGHT        'Y OF LOWER RIGHT OF BAR
670 YLR = SLOPELEFT * (XLEFT+H) + BLEFT          'Y OF UPPER RIGHT OF BAR
680 YRIGHT = YLR
690 RETURN
700 '######################## MAKE BOX OUTLINE ######################
710 LINE (XLEFT,YUL) - (XLEFT+H/2,YUM),FILL
720 LINE - (XLEFT+H,YUR),FILL: LINE - (XLEFT+H,YLR),FILL
730 LINE - (XLEFT+H/2,YLM),FILL: LINE - (XLEFT,YLL),FILL
740 LINE - (XLEFT,YUL),FILL
750 PAINT (XLEFT+H/2,YUM+10),FILL,FILL
760 '######################## MAKE BOX OUTLINE ######################
770 LINE (XLEFT,YUL) - (XLEFT+H/2,YUM),OUTLINE
780 LINE - (XLEFT+H,YUR),OUTLINE: LINE - (XLEFT+H,YLR),OUTLINE
```

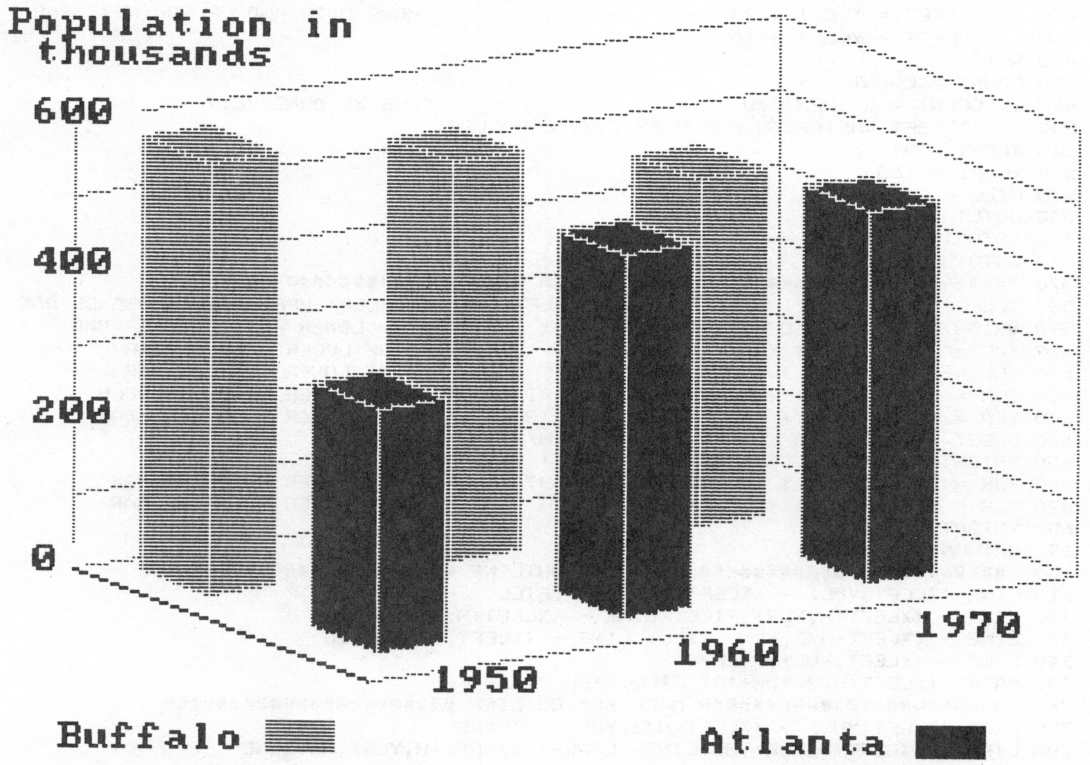
Program 10–7 (cont.)

```
790 LINE - (XLEFT+H/2,YLM),OUTLINE: LINE - (XLEFT,YLL),OUTLINE
800 LINE - (XLEFT,YUL),OUTLINE: LINE - (XLEFT+H/2,YLM-HT),OUTLINE
810 LINE - (XLEFT+H,YUR),OUTLINE
820 LINE (XLEFT+H/2,YLM) - (XLEFT+H/2,YLM-HT),OUTLINE
830 RETURN
840 '############################################################
850 DATA 580,532,462
860 DATA 331,487,498
870 IF INKEY$ = "" THEN 870
```

each bar according to the percent of the population in various age ranges. The bar sections could then be color coded to indicate age range. There are many possibilities for the labeling. We could devise identifying labels that are slanted to align with the directions of the bars, or we could simply code the bars with various colors or shading and use a key, as in Fig. 10–17.

We can also plot curves in three dimensions to show relationships among several variables. As an illustration, Prog. 10–8 outputs a plot of the function HEIGHT * SIN(FREQUENCY * SQR(X * X + Z * Z)) for a chosen range of X

Figure 10–17 Three-dimensional bar chart showing population figures (in thousands) for two cities in the years 1950, 1960, 1970, produced by Prog. 10–7.

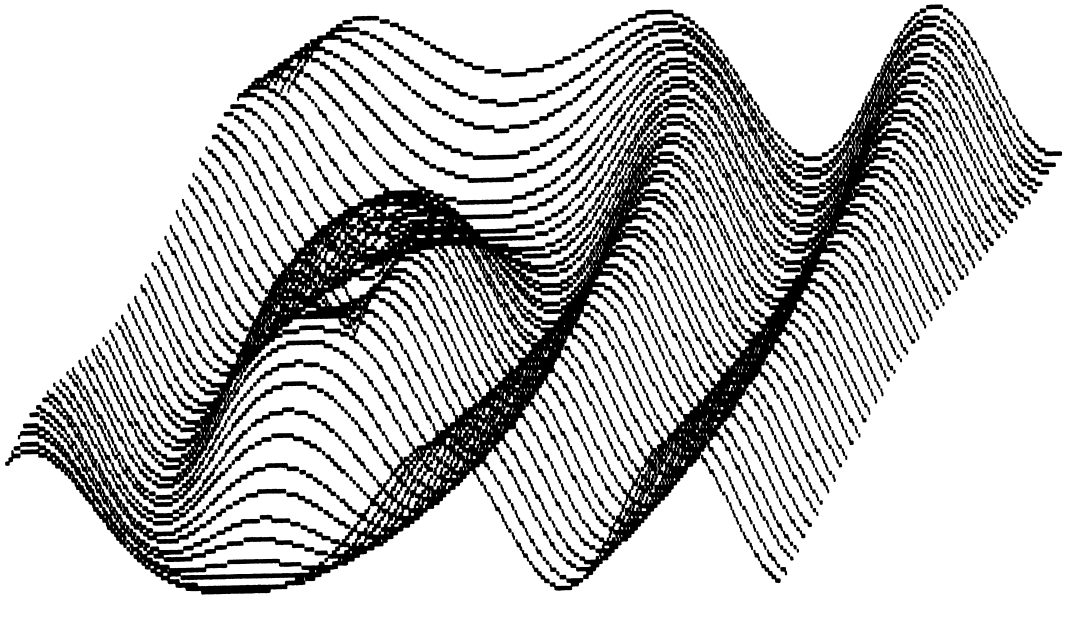


Program 10–8 Three-dimensional curve plotting.

```
10 'PROGRAM 10-8. THREE-DIMENSIONAL PLOT.
20      'PLOTS A SIN FUNCTION IN 3 DIMENSIONS. Z IS VARIED
30      'FROM -100 TO 130. FOR EACH VALUE OF Z AND THE X VALUES
40      '-135 TO 325, A Y VALUE IS CALCULATED USING A MODIFIED
50      'SIN EQUATION. X AND Y ARE ADJUSTED AND THEN PLOTTED.
60      '*****************************************************************
70  SCREEN 2: CLS
80  XCENTER = 214: YCENTER = 100
90  HEIGHT = 20
100 FREQUENCY = .05
110 FIRSTX = -135
120 FOR Z = -100 TO 130 STEP 5
130     ZSQUARED = Z * Z
140     XSCREENADJUST = XCENTER + .75 * Z
150     YSCREENADJUST = 199 - (YCENTER + .5 * Z)
160     FOR X = FIRSTX TO 325
170         Y = INT(HEIGHT * SIN(FREQUENCY * SQR(X * X + ZSQUARED)) + .5)
180         XSCREEN = XSCREENADJUST + X
190         YSCREEN = YSCREENADJUST - Y
200         IF X > FIRSTX THEN LINE - (XSCREEN,YSCREEN) ELSE
               PSET (XSCREEN,YSCREEN)
210     NEXT
220 NEXT
230 END
```

and Z coordinates and for selected constant values HEIGHT and FREQUENCY. We have chosen the range of X values to be between −135 and 325, which selects the section of the surface to be displayed across the screen. The resulting surface appearance is shown in Fig. 10–18. Other ranges for X produce different sections

Figure 10–18 Three-dimensional curves plotted by Prog. 10–8.



of the surface, and we could adjust the edges of the surface by calculating the X range as a function of Z (such as a linear or elliptical function). The Z range and constants XCENTER, YCENTER, HEIGHT, and FREQUENCY are chosen to fit the curve onto the screen. We have selected high resolution in Prog. 10–8 to get

Program 10–9 Three-dimensional curve plotting—displaying only visible line segments to give a surface appearance.

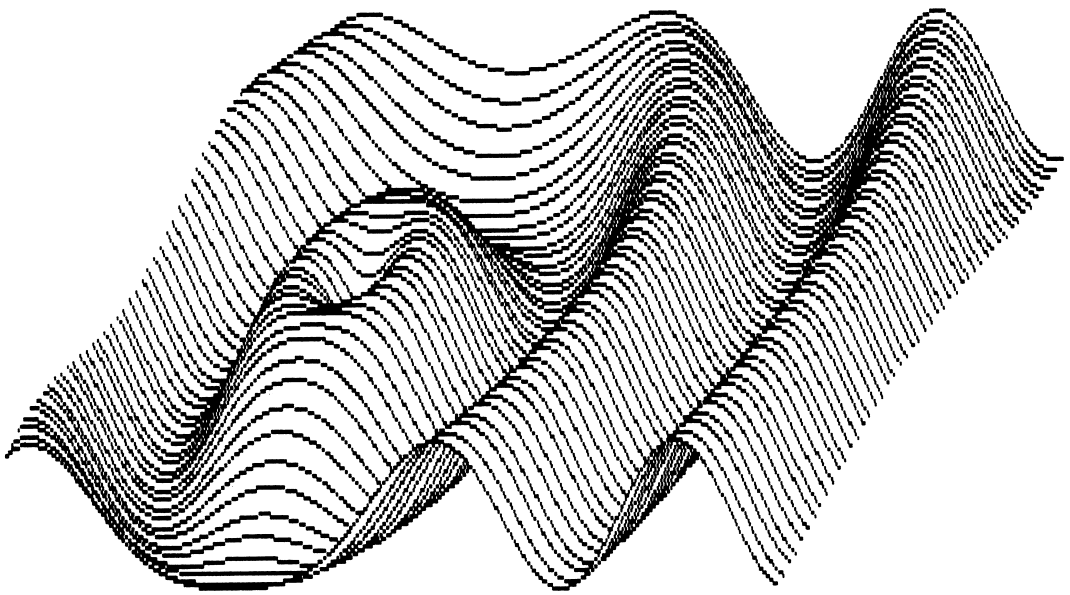
```
10 'PROGRAM 10-9. VISIBLE LINE PLOT OF A THREE-DIMENSIONAL GRAPH.
20      'PLOTS A SIN FUNCTION IN 3 DIMENSIONS. Z IS VARIED
30      'FROM -100 TO 130. FOR EACH VALUE OF Z AND THE X VALUES
40      '-135 TO 325, A Y VALUE IS CALCULATED USING A MODIFIED
50      'SIN FUNCTION. X AND Y ARE ADJUSTED AND PLOTTED. FOR
60      'EVERY X VALUE ON THE SCREEN (0 - 639) A BIGGEST Y VALUE AND
70      'A SMALLEST Y VALUE IS MAINTAINED, REFLECTING THE RANGE OF Y
80      'VALUES THAT HAVE ALREADY BEEN PLOTTED ON THE SCREEN AT
90      'THIS X. EACH Y (AFTER ADJUSTMENT TO FIT ON SCREEN) IS
100     'CHECKED AGAINST THIS RANGE -- IF IT IS INSIDE THE RANGE
110     'NO LINE IS DRAWN. IF OUTSIDE THE RANGE WE DRAW THE LINE
120     'AND UPDATE THE SMALLEST Y OR BIGGEST Y VALUE. THE FLAGS
130     'LASTPOINT AND CHANGE ARE USED TO INDICATE WHETHER WE
140     'SHOULD JUST PLOT THE POINT INDICATED BY X AND Y (AT THE
150     'VERY START OF A VISIBLE PORTION OF THE CURVE) OR DRAW
160     'A LINE FROM THE LAST POINT TO THIS POINT (WHEN WE'RE
170     'ALREADY IN A VISIBLE PORTION OF THE CURVE).
180     '*****************************************************************
190 SCREEN 2: CLS
200 DIM SMALLEST(639), BIGGEST(639)
210 XCENTER = 214: YCENTER = 100
220 HEIGHT = 20
230 FREQUENCY = .05
240 FIRSTX = -135
250 LASTPOINT = 1
260      'INITIALIZE ARRAYS
270 FOR ELEMENT = 1 TO 639
280     BIGGEST(ELEMENT) = 0: SMALLEST(ELEMENT) = 1000
290 NEXT
300 FOR Z = -100 TO 130 STEP 5
310     ZSQUARED = Z * Z
320     XSCREENADJUST = XCENTER + .75 * Z
330     YSCREENADJUST = 199 - (YCENTER + .5 * Z)
340     FOR X = FIRSTX TO 325 STEP 1
350         Y = INT(HEIGHT * SIN(FREQUENCY * SQR(X * X + ZSQUARED)) + .5)
360         XSCREEN = XSCREENADJUST + X
370         YSCREEN = YSCREENADJUST - Y
380         'IS THIS POINT WITHIN THE BOUNDS OF WHAT'S ALREADY DRAWN?
390         IF YSCREEN >= SMALLEST(XSCREEN) AND YSCREEN <= BIGGEST(XSCREEN)
                THEN LASTPOINT = 0: GOTO 450
400         IF LASTPOINT = 0 THEN CHANGE = 1
410         LASTPOINT = 1
420         IF YSCREEN < SMALLEST(XSCREEN) THEN SMALLEST(XSCREEN) = YSCREEN:
                IF BIGGEST(XSCREEN)=0 THEN BIGGEST(XSCREEN) = YSCREEN: GOTO 440
430         IF YSCREEN > BIGGEST(XSCREEN) THEN BIGGEST(XSCREEN) = YSCREEN
440         IF X = FIRSTX OR CHANGE = 1 THEN PSET(XSCREEN,YSCREEN): CHANGE = 0
                ELSE LINE - (XSCREEN,YSCREEN)
450     NEXT
460 NEXT
470 END
```

smoother curves. The plotted coordinate points (XSCREEN,YSCREEN) are calculated as offsets from XCENTER and YCENTER by the functional values X + 0.75 * Z and Y + Z/2. We do this in order to shift each curve a little to the right and up the screen. This spreads the curves out and gives a three-dimensional surface appearance.

Program 10–9 is a modification of Prog. 10–8 that eliminates overlapping lines, as shown in Fig. 10–19. This is accomplished by drawing the curves from "front" to "back" and by not drawing the hidden segments; that is, the curve segments that would overlap with previously drawn lines. For each value of X, an upper screen bound and lower screen bound of Y values is stored in arrays SMALLEST and BIGGEST for the curves drawn previously. Any curve sections calculated subsequently that would fall within these bounds are not drawn. Subsequent curve sections that are outside the bounds are drawn and also cause the bounds to be reset.

Three-dimensional curves can be plotted for any functional relationship involving the X, Y, and Z coordinates using techniques similar to those of Prog. 10–9. Coordinate points can be calculated from equations or supplied as data tables, such as topographic data on elevations or population density. Axes and labeling for the three coordinate directions can be added, as in the graph of Fig. 10–17.

Figure 10–19 Three-dimensional "surface" formed from the curves in Fig. 10–18 with hidden lines removed (Prog. 10–9).



PROGRAMMING PROJECTS

10–1. Lay out any three-dimensional object on graph paper, specifying (X,Y,Z) for each point in the object. Write a program to display any selected orthographic projection for this object.

10–2. Modify Prog. 10–1 to display three-dimensional objects with polygon faces. Input to the program will include the X, Y, and Z coordinates for each vertex of a face. The interior of each face may be filled using the PAINT command or using a series of horizontal lines. The boundary of each face is formed from straight line segments. The slope and Y-intercept of each line segment are determined from the coordinates of the pair of vertex endpoints for that line segment. These line equations can be used to determine coordinates for the ends of the horizontal erase lines.

10–3. Using the method of Prog. 10–1, display a series of overlapping spheres (or other curved surfaces), as shown in Fig. 10–6.

10–4. Use the method of Prog. 10–2 to erase the hidden lines in any display of a solid bar whose ends are six-sided polygons (Fig. 10–20). Display all lines in the object, then determine which face from each of the four pairs of opposite faces is invisible and erase the hidden faces.

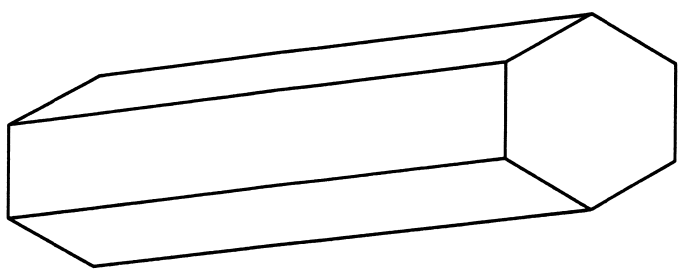


Figure 10–20 Hexagonal solid.

10–5. Using the method of Prog. 10–4, erase hidden lines behind circles. The program is to display any circle, then display a specified line and erase any part of the line hidden by the circle.

10–6. Lay out any three-dimensional object whose surfaces are convex polygons on graph paper. Write a program to display the object and use the method of Prog. 10–4 to erase any hidden lines. Finally, apply perspective equations (10–1) from any specified viewing position (XV,YV,ZV).

10–7. Revise Prog. 10–6 to display visible surfaces of a three-dimensional object using the method of Prog. 10–4 and equation (10–2).

10–8. Modify Prog. 10–4 to input concave polygon surfaces and straight lines. Find all intersection points (there may be more than two) of a line with a surface and erase the hidden segments.

10–9. Define a shading pattern to be used in a program displaying three-dimensional objects. For any displayed object, shade the faces opposite from a specified light source position.

10–10. Expand Prog. 10–7 to produce a three-dimensional bar chart with more than one section to each bar. Draw the different sections of each bar in a different color.

10–11. Revise Prog. 10–8 to plot the three-dimensional surface with an X range that is calculated as a linear function of Z: A * Z + B, with constants A and B chosen to fit the surface onto the screen.

10–12. Revise Prog. 10–8 to plot the three-dimensional surface with an X range that is calculated as the elliptical function XR = 175 * SQR(1 − Z * Z/(120 * 120)). This function is to be used to calculate the upper and lower bounds (−XR to +XR) of the X range for each value of Z. Values of Z will range from −120 to +120. Other constants will give different sections of the surface.

10–13. Using the technique of Prog. 10–9, plot a three-dimensional surface formed from the function

$$Y = A * SIN(X) + B * SIN(3 * X)$$

Parameters A and B are chosen so that the surface fits on the screen for the range of X and Z plotted. Plot the surface with different determinations for the X range (as in Projects 10–11 and 10–12).

Chapter 11

Three-Dimensional Transformations

We now extend two-dimensional transformation methods (translation, scaling, rotation) to include depth. These added transformation techniques will give us considerable flexibility in manipulating both three-dimensional and two-dimensional displays.

11–1 TRANSLATION

Equations for two-dimensional translation are extended to three dimensions by allowing for changes in Z coordinates. If we let H represent the amount of horizontal displacement of a point, V the amount of vertical displacement, and D the amount of depth displacement, then the translated position (XT,YT,ZT) of a point originally at position (X,Y,Z) is calculated as

$$\begin{aligned} XT &= X + H \\ YT &= Y + V \\ ZT &= Z + D \end{aligned} \tag{11–1}$$

Positive values for H move a point to the right on the screen. Positive values for V move a point down the screen. Positive values for D move a point away from us, and negative values for D move a point closer to us. Values for the translation distances, H, V, and D, should not be chosen so large that we move points off the screen or so small that we simply replot the same points.

Pictures and graphs can be translated or animated in space by applying the same values for the translation distances, H, V, D, to all points in the display. We can use different values of H, V, or D for different points in the display, but then we distort the original shapes of the display.

An example of three-dimensional translation is given in Prog. 11–1. A block is translated and projected onto the screen. This program inputs the translation distances, calculates the new position, and applies the perspective equations discussed in Chapter 10 to display the translated object. For larger values of D, the size of the block is diminished and it moves closer to the vanishing point. Figure 11–1 shows the original and translated positions for the values H = 60, V = 60, and D = 20.

Program 11–1 Three-dimensional translation and perspective views (box).

```
10 'PROGRAM 11-1. TRANSLATION IN 3 DIMENSIONS.
20      'DEFINES AN OBJECT IN THREE DIMENSIONS AND TRANSLATES
30      'IT TO VARIOUS LOCATIONS. OBJECT IS DISPLAYED IN
40      'RELATION TO A VIEWING POINT IN FRONT OF THE CENTER
50      'POINT OF THE SCREEN. SYMMETRY OF OBJECT IS USED
60      'TO DETERMINE WHICH SIDES ARE VISIBLE.
70  SCREEN 1: CLS
80  DIM X(8), Y(8), Z(8), XP(8), YP(8), DISTANCE(8)
90      '********************** READ DATA POINTS **********************
100 READ N
110 FOR K = 1 TO N
120     READ X(K), Y(K), Z(K)
130     IF X(K)<0 OR X(K)>319 OR Y(K)<0 OR Y(K)>199 THEN 840
140 NEXT
150 FOR K = 1 TO N/2 - 1
160     LINE (X(K),Y(K)) - (X(K+1),Y(K+1))
170 NEXT
180 LINE (X(N/2),Y(N/2)) - (X(1),Y(1))
190 XV = 160                        'FIX VIEWING POINT AT SCREEN CENTER
200 YV = 100
210 ZV = -100
220     '********************** TRANSLATE OBJECT ******************************
230 PRINT "ENTER 1000,1000,1000 TO END"
240 INPUT "H, V, & D TRANSLATION AMOUNT"; HORIZONTAL,VERTICAL,DEPTH
250 IF HORIZONTAL = 1000 AND VERTICAL = 1000 AND DEPTH = 1000 THEN 840
260     'CALCULATE NEW POINTS
270 FOR K = 1 TO N
280     XT(K) = X(K) + HORIZONTAL
290     YT(K) = Y(K) + VERTICAL
300     ZT(K) = Z(K) + DEPTH
310 NEXT
320     'FIND DISTANCES AND PUT POINTS IN PERSPECTIVE
330 FOR K = 1 TO N
340     DISTANCE(K) = SQR((XT(K)-XV)^2 + (YT(K)-YV)^2 + (ZT(K)-ZV)^2)
350     XP(K) = XV + (XT(K) - XV) * -ZV / (ZT(K) - ZV)
360     YP(K) = YV + (YT(K) - YV) * -ZV / (ZT(K) - ZV)
370     IF XP(K)>=0 AND XP(K)<=319 AND YP(K)>=0 AND YP(K)<=199 THEN 400
380     PRINT "COORDINATE OFF SCREEN. TRY AGAIN"
390     GOTO 230
400 NEXT
410 GOSUB 430           'DRAW
420 GOTO 230
430 '################################ DRAW ROUTINE ################################
440 CLS
450     '********* FRONT AND BACK FACES *************
460 IF DISTANCE(1) = DISTANCE(5) THEN 600        'NEITHER SIDE IS VISIBLE
470 IF DISTANCE(1) > DISTANCE(5) THEN 540
480 IF ZT(1) <= ZV THEN 600                'IS FACE SEEN FROM THIS VIEWING POINT?
490 FOR K = 1 TO 3                                  'DRAW FACE CONTAINING POINT 1
```

Program 11–1 (cont.)

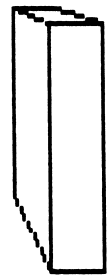
```
500     LINE (XP(K),YP(K)) - (XP(K+1),YP(K+1))
510 NEXT
520 LINE (XP(4),YP(4)) - (XP(1),YP(1))
530 GOTO 600
540 IF ZT(5) >= ZV THEN 600                'IS FACE SEEN FROM THIS VIEWING POINT?
550 FOR K = 5 TO 7                                 'DRAW FACE CONTAINING POINT 5
560     LINE (XP(K),YP(K)) - (XP(K+1),YP(K+1))
570 NEXT
580 LINE (XP(8),YP(8)) - (XP(5),YP(5))
590     '************** SIDE FACES ******************
600 IF DISTANCE(1) = DISTANCE(4) THEN 700        'NEITHER SIDE IS VISIBLE
610 IF DISTANCE(1) > DISTANCE(4) THEN 660
620 IF XT(1) >= XV THEN 700                'IS FACE SEEN FROM THIS VIEWING POINT?
630 LINE (XP(1),YP(1)) - (XP(2),YP(2))             'DRAW FACE CONTAINING POINT 1
640 LINE - (XP(6),YP(6)): LINE - (XP(5),YP(5)): LINE - (XP(1),YP(1))
650 GOTO 700
660 IF XT(1) <= XV THEN 700                'IS FACE SEEN FROM THIS VIEWING POINT?
670 LINE (XP(4),YP(4)) - (XP(3),YP(3))             'DRAW FACE CONTAINING POINT 4
680 LINE - (XP(7),YP(7)): LINE - (XP(8),YP(8)): LINE - (XP(4),YP(4))
690     '*********** TOP AND BOTTOM FACES **********
700 IF DISTANCE(1) = DISTANCE(2) THEN 790        'NEITHER SIDE IS VISIBLE
710 IF DISTANCE(1) > DISTANCE(2) THEN 760
720 IF YT(1) <= YV THEN 790                'IS FACE SEEN FROM THIS VIEWING POINT?
730 LINE (XP(1),YP(1)) - (XP(4),YP(4))             'DRAW FACE CONTAINING POINT 1
740 LINE - (XP(8),YP(8)): LINE - (XP(5),YP(5)): LINE - (XP(1),YP(1))
750 GOTO 790
760 IF YT(2) >= YV THEN 790
770 LINE (XP(2),YP(2)) - (XP(3),YP(3))             'DRAW FACE CONTAINING POINT 2
780 LINE - (XP(7),YP(7)): LINE - (XP(6),YP(6)): LINE - (XP(2),YP(2))
790 RETURN
800     '************************************************************
810 DATA 8
820 DATA 138,56,0,138,136,0,118,136,0,118,56,0
830 DATA 138,56,40,138,136,40,118,136,40,118,56,40
840 END
```

Figure 11–1 Output of Prog. 11–1, showing (a) the original and (b) translated perspective views of an object from a viewing position of (160,100,-100).

```
ENTER H, V, & D TRANSLATION AMOUNT
ENTER 1000,1000,1000 TO END
?
```

(a)

```
ENTER H, V, & D TRANSLATION AMOUNT
ENTER 1000,1000,1000 TO END
? 
```



(b)

Figure 11–1 (cont.)

11–2 SCALING

Generalizing the two-dimensional scaling equations to three dimensions is accomplished by allowing for a reduction or enlargement in the Z direction, as well as the X and Y directions. As with two-dimensional scaling, we choose a fixed position and calculate the new coordinate positions of each point of a display relative to this fixed position using specified scaling factors. Taking the fixed position at coordinates (XF,YF,ZF), we can calculate scaled coordinates (XS,YS,ZS) of a display point from the original coordinates (X,Y,Z) as

$$\begin{aligned} XS &= X * HS + XF * (1 - HS) \\ YS &= Y * VS + YF * (1 - VS) \\ ZS &= Z * DS + ZF * (1 - DS) \end{aligned} \tag{11–2}$$

where the scaling factors for each coordinate direction are denoted as HS, VS, and DS. Each of these scaling factors may be assigned any value greater than zero. Values greater than 1 produce an enlargement, while values smaller than 1 reduce the size of objects.

The scaling factors HS, VS, and DS are usually all assigned the same value. This uniformly enlarges or reduces the object by the same amount in all directions. If we want to stretch or shrink the object by different amounts in different directions, we could assign different values to the different coordinate scaling factors. Large values for the scaling factors can scale objects out of range of the screen size. Small values for the scaling factors can reduce objects to single points.

Scaling a picture using the calculations in (11–2) is illustrated by Prog. 11–2.

Program 11–2 Three-dimensional scaling and perspective views (robot).

```
10 'PROGRAM 11-2. SCALING IN 3 DIMENSIONS.
20      'DEFINES A ROBOT IN THREE DIMENSIONS AND SCALES
30      'IT IN RELATION TO XF,YF,ZF. A VIEWING POINT IS
40      'ESTABLSIHED AT XV,YV,ZV. PARTIAL SYMMETRY OF THE
50      'ROBOT IS USED TO DETERMINE WHICH SIDES ARE
60      'VISIBLE. THE INTERIOR OF EACH VISIBLE SIDE IS
70      'BLANKED OUT, ERASING ANY HIDDEN LINES.
80  SCREEN 1: COLOR 1,0: CLS
90  DIM X(10,4), Y(10,4), Z(10,4), XS(10,4), YS(10,4), ZS(10,4)
100 DIM XP(10,4), YP(10,4), D(10,4)
110 XF = 80      'FIGURE IS SCALED IN RELATION TO XF,YF,ZF
120 YF = 110
130 ZF = 7
140 XV = 160     'XV,YV,ZV IS VIEWING POINT
150 YV = 10
160 ZV = -100
170      '********************** READ DATA POINTS **********************
180 READ NS      'NS IS NUMBER OF SURFACES
190 FOR S = 1 TO NS
200     READ NV(S)       'NV IS NUMBER OF VERTICES FOR THIS SURFACE
210     FOR P = 1 TO NV(S)
220         READ X(S,P), Y(S,P), Z(S,P)
230         IF X(S,P)<0 OR X(S,P)>319 OR Y(S,P)<0 OR Y(S,P)>199 THEN 1230
240     NEXT
250 NEXT
260 HS = 1: VS = 1: DS =1        'USE SCALING VALUES OF 1 FOR INITIAL DISPLAY
270 GOTO 310
280 LOCATE 1,1: PRINT "ENTER 0,0,0 TO END"
290 INPUT "H, V, AND D SCALING FACTORS"; HS,VS,DS
300 IF HS = 0 AND VS = 0 AND DS = 0 THEN 1230
310     '********************** CALCULATE NEW POINTS ****************
320 XCONST = XF * (1 - HS)
330 YCONST = YF * (1 - VS)
340 ZCONST = ZF * (1 - DS)
350 FOR S = 1 TO NS
360     FOR P = 1 TO NV(S)
370         XS(S,P) = X(S,P) * HS + XCONST
380         YS(S,P) = Y(S,P) * VS + YCONST
390         ZS(S,P) = Z(S,P) * DS + ZCONST
400         D(S,P) = SQR((XS(S,P)-XV)^2 + (YS(S,P)-YV)^2 + (ZS(S,P)-ZV)^2)
410         XP(S,P) = XV + (XS(S,P) - XV) * -ZV / (ZS(S,P) - ZV)
420         YP(S,P) = YV + (YS(S,P) - YV) * -ZV / (ZS(S,P) - ZV)
430         IF XP(S,P)<0 OR XP(S,P)>319 OR YP(S,P)<0 AND YP(S,P)>199 THEN
               PRINT "OFF SCREEN. TRY AGAIN": GOTO 340
440     NEXT
450 NEXT
460 GOSUB 480            'DRAW FIGURE
470 GOTO 280
480 '############################## DRAW ROUTINE ##############################
490 CLS
500 FOR S = 1 TO 8 STEP 2        'LOOK AT EACH PAIR OF SYMMETRIC SURFACES
510     IF D(S,1) < D(S+1,1) AND ZS(S,1) > ZV THEN GOSUB 600
520     IF D(S+1,1) < D(S,1) AND ZS(S+1,1) < ZV THEN GOSUB 710
530     IF D(S,1) < D(S,4) AND XS(S,1) < XV THEN COLOUR = 1: GOSUB 750:
              FACE = S: GOSUB 1070: COLOUR = 3: GOSUB 750
              FACE = S: GOSUB 1070: COLOUR = 3: GOSUB 750
540     IF D(S,4) < D(S,1) AND XS(S,4) > XV THEN COLOUR = 1: GOSUB 830:
              FACE = S+1: GOSUB 1070: COLOUR = 3: GOSUB 830
```

Program 11–2 (cont.)

```
550     IF D(S,1) < D(S,2) AND YS(S,1) > YV THEN COLOUR = 1: GOSUB 910:
            FACE = S: GOSUB 1070: COLOUR = 3: GOSUB 910
560     IF D(S,2) < D(S,1) AND YS(S,2) < YV THEN COLOUR = 1: GOSUB 990:
            FACE = S+1: GOSUB 1070: COLOUR = 3: GOSUB 990
570 NEXT
580 RETURN
590     '******************* FRONT AND BACK FACES ************************
600     'DRAW FRONT FACE OUTLINE (FACE WITH POINT (S,1))
610 LINE (XP(S,1),YP(S,1)) - (XP(S,3),YP(S,3)),0,BF
620 LINE (XP(S,1),YP(S,1)) - (XP(S,3),YP(S,3)),3,B
630 IF S <> 7 THEN 700              'ELSE DRAW IN EYES
640 FOR K = 1 TO 3
650     LINE (XP(9,K),YP(9,K)) - (XP(9,K+1),YP(9,K+1))
660     LINE (XP(10,K),YP(10,K)) - (XP(10,K+1),YP(10,K+1))
670 NEXT
680 LINE (XP(9,4),YP(9,4)) - (XP(9,1),YP(9,1))
690 LINE (XP(10,4),YP(10,4)) - (XP(10,1),YP(10,1))
700 RETURN
710     'DRAW BACK FACE (FACE WITH POINT (S+1,1))
720 LINE (XP(S+1,1)YP(S+1,1) - (XP(S+1,3,YP(S+1,3)),0,BF
730 LINE (XP(S+1,1)YP(S+1,1) - (XP(S+1,3,YP(S+1,3)),3,BF
740 RETURN
750     'DRAW RIGHT FACE OUTLINE (FACE WITH POINT (S,1))
760 LINE (XP(S,1),YP(S,1)) - (XP(S,2),YP(S,2)),COLOUR
770 LINE - (XP(S+1,2),YP(S+1,2)),COLOUR
780 LINE - (XP(S+1,1),YP(S+1,1)),COLOUR
790 LINE - (XP(S,1),YP(S,1)),COLOUR
800 XINNER = (XP(S,1) + XP(S,2) + XP(S+1,2) + XP(S+1,1)) / 4
810 YINNER = (YP(S,1) + YP(S,2) + YP(S+1,2) + YP(S+1,1)) / 4
820 RETURN
830     'DRAW LEFT FACE OUTLINE (FACE WITH POINT (S,4))
840 LINE (XP(S,4),YP(S,4)) - (XP(S,3),YP(S,3)),COLOUR
850 LINE - (XP(S+1,3),YP(S+1,3)),COLOUR
860 LINE - (XP(S+1,4),YP(S+1,4)),COLOUR
870 LINE - (XP(S,4),YP(S,4)),COLOUR
880 XINNER = (XP(S,4) + XP(S,3) + XP(S+1,3) + XP(S+1,4)) / 4
890 YINNER = (YP(S,4) + YP(S,3) + YP(S+1,3) + YP(S+1,4)) / 4
900 RETURN
910     'DRAW TOP FACE OUTLINE (FACE WITH POINT (S,1))
920 LINE (XP(S,1),YP(S,1)) - (XP(S,4),YP(S,4)),COLOUR
930 LINE - (XP(S+1,4),YP(S+1,4)),COLOUR
940 LINE - (XP(S+1,1),YP(S+1,1)),COLOUR
950 LINE - (XP(S,1),YP(S,1)),COLOUR
960 XINNER = (XP(S,1) + XP(S,4) + XP(S+1,1) + XP(S+1,4)) / 4
970 YINNER = (YP(S,1) + YP(S,4) + YP(S+1,1) + YP(S+1,4)) / 4
980 RETURN
990     'DRAW BOTTOM FACE OUTLINE (FACE WITH POINT (S,2))
1000 LINE (XP(S,2),YP(S,2)) - (XP(S,3),YP(S,3)),COLOUR
1010 LINE - (XP(S+1,3),YP(S+1,3))COLOUR
1020 LINE - (XP(S+1,2),YP(S+1,2)),COLOUR
1030 LINE - (XP(S,2),YP(S,2)),COLOUR
1040 XINNER = (XP(S,2) + XP(S,3) + XP(S+1,2) + XP(S+1,3)) / 4
1050 YINNER = (YP(S,2) + YP(S,3) + YP(S+1,2) + YP(S+1,3)) / 4
1060 RETURN
1070    'FIND INTERIOR POINT
1080 PAINT (XINNER,YINNER),2,1
1090 PAINT (XINNER,YINNER),0,1
1100 RETURN
1110    '*********************************************************************
```

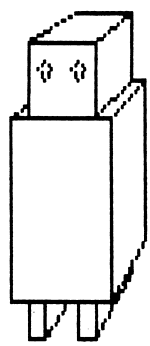
Program 11–2 (cont.)

```
1120 DATA 10
1130 DATA 4,60,140,0,60,150,0,55,150,0,55,140,0
1140 DATA 4,60,140,15,60,150,15,55,150,15,55,140,15
1150 DATA 4,75,140,0,75,150,0,70,150,0,70,140,0
1160 DATA 4,75,140,15,75,150,15,70,150,15,70,140,15
1170 DATA 4,80,90,0,80,140,0,50,140,0,50,90,0
1180 DATA 4,80,90,15,80,140,15,50,140,15,50,90,15
1190 DATA 4,75,70,0,75,90,0,55,90,0,55,70,0
1200 DATA 4,75,70,15,75,90,15,55,90,15,55,70,15
1210 DATA 4,70,75,0,72,77,0,70,80,0,68,77,0
1220 DATA 4,60,75,0,62,77,0,60,80,0,58,77,0
1230 END
```

This program scales a robot figure to any specified size relative to a fixed point on the robot. The robot is originally placed in the lower left corner of the screen, facing toward us. A viewing position, for perspective, is chosen to the right and above the robot. Visibility of the robot surfaces is determined relative to this point, and the scaled robot is drawn from the feet up, left to right. Figure 11–2 shows the output for both a size increase and decrease. Input to the program can be set up to allow alternate picture definitions, variations of the fixed point and viewing point, and various scaling factors, HS, VS, and DS.

We can apply these same scaling techniques to graphs in three dimensions.

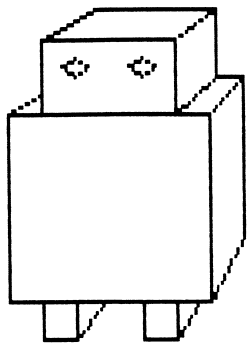
```
ENTER 0,0,0 TO END
H, V, AND D SCALING FACTORS? ■
```



(a)

Figure 11–2 Scaling a figure in three-dimensions. This output of Prog. 11–2 shows (a) the original, (b) an enlarged perspective view, and (c) a reduced perspective view. The viewing position is at coordinates (160,10,−100).

```
ENTER 0,0,0 TO END
H, V, AND D SCALING FACTORS? ■
```



(b)

```
ENTER 0,0,0 TO END
H, V, AND D SCALING FACTORS? ■
```

(c)

Figure 11–2 (cont.)

But in many cases we can set up a graph showing three or more variable relationships with all screen coordinate positions specified in just two dimensions. For these situations, two-dimensional scaling methods are sufficient to handle reduction or enlargement requirements, as discussed in Chapter 7.

11–3 ROTATION

We have seen that an object can be rotated through an angle A about a specified point (XO,YO) in the X,Y plane by transforming all X and Y coordinates to rotated values (XR,YR) through the calculations

$$XR = XO + (X - XO) * COS(A) + (Y - YO) * SIN(A)$$
$$YR = YO + (Y - YO) * COS(A) - (X - XO) * SIN(A) \quad (11\text{–}3)$$

The angle A in these calculations must be specified in radians and is measured in a counterclockwise direction from position (X,Y) to position (XR,YR). Considering a three-dimensional object, such as the box in Fig. 11–3, we can visualize this rotation as occurring about a line through the point (XO,YO) that is parallel to the Z axis. All points of the object would be rotated about this line. This rotation would leave all Z coordinates unchanged.

In a similar way we can consider rotating three-dimensional objects about axes in other directions. Figure 11–4 shows a rotation axis that is parallel to the Y axis and through a point (XO,ZO). Rotation about this line would change each of the X and Z coordinates of the box to rotated values XR and ZR. We calculate XR and ZR from the equations

$$XR = XO + (X - XO) * COS(A) - (Z - ZO) * SIN(A)$$
$$ZR = ZO + (Z - ZO) * COS(A) + (X - XO) * SIN(A) \quad (11\text{–}4)$$

This rotation leaves Y values unchanged. Angle A in these calculations is measured in a counterclockwise direction as we look down on the "top" of the

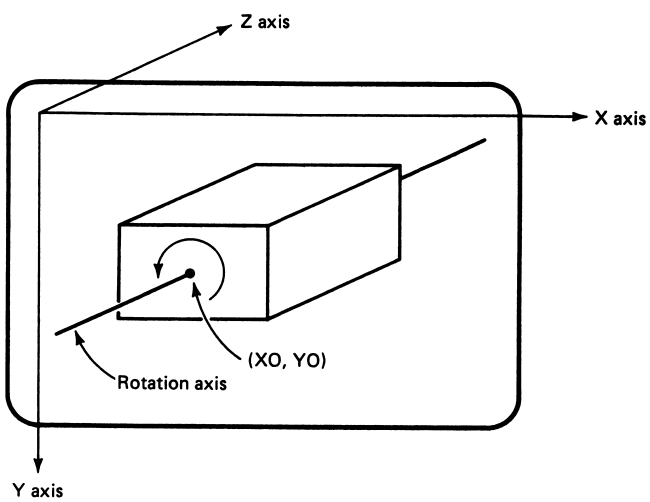


Figure 11–3 Rotating an object about an axis in the Z direction changes X and Y coordinates, but leaves Z coordinates unchanged.

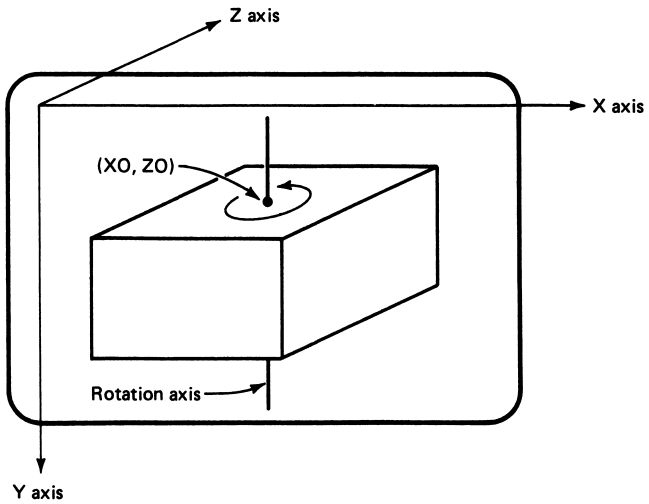


Figure 11–4 Rotating an object about an axis in the Y direction changes X and Z coordinates, but leaves Y coordinates unchanged.

box. Another way to describe this direction of rotation is to imagine that we are standing at the origin of the coordinate system and looking along the positive Y axis. An object rotates counterclockwise for this viewing direction. If we want objects to rotate the other way, we use negative radian values for the angle A in calculations (11–4).

In Fig. 11–5, we take the rotation axis to be parallel to the X axis. If we rotate the box about this axis, each of the Y and Z coordinates of the box would be

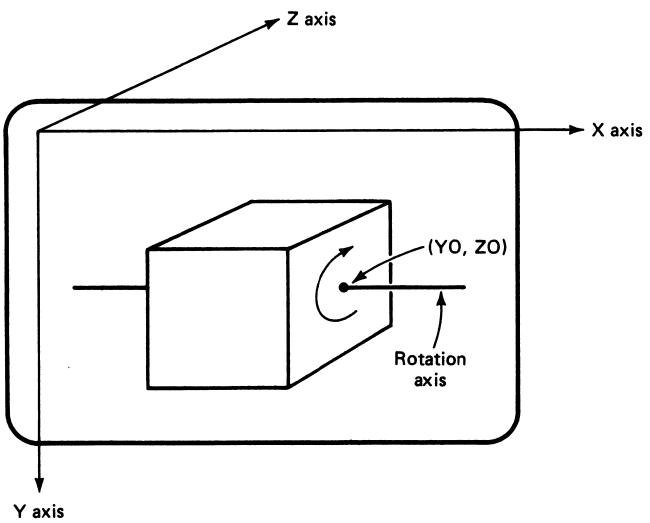


Figure 11–5 Rotating an object about an axis in the X direction changes Y and Z coordinates, while leaving X values unchanged.

rotated to coordinates YR and ZR. These new coordinates are calculated from the relations

$$YR = YO + (Y - YO) * COS(A) + (Z - ZO) * SIN(A)$$
$$ZR = ZO + (Z - ZO) * COS(A) - (Y - YO) * SIN(A) \quad (11\text{–}5)$$

The X values for this rotation are unchanged. Viewing the box on its "left" side (or standing at the origin and looking along the positive X axis), we have a counterclockwise direction of rotation for the angle A.

Many other rotation axes could be selected, but we can accomplish any other type of rotation with a combination of the three we have discussed. Rotating an object about an axis parallel to the Z axis (Fig. 11–3) simply "spins" the object, always presenting the same side to our view. Rotating an object about an axis parallel to the Y axis (Fig. 11–4) gives us side views of the object. Rotating about an axis parallel to the X axis (Fig. 11–5) gives us top and bottom views. To produce any combination of views, such as top and left side, we can transform an object through two or more rotations. The order in which rotations are applied is important, since we may get a different view by reversing the order of any two rotations.

Program 11–3 rotates a die in any combination of the three rotational

Program 11–3 Three-dimensional rotations (die).

```
10 'PROGRAM 11-3. ROTATION IN 3 DIMENSIONS
20      'DEFINES A SINGLE DIE IN THREE DIMENSIONS. POINTS FOR
30      'THE SIX FACES AND ALL SPOTS ARE STORED IN XR,YR,ZR.
40      'DIE CAN BE ROTATED ANY NUMBER OF DEGREES AROUND ANY
50      'AXIS. SYMMETRY OF THE SHAPE IS USED TO DETERMINE
60      'WHICH SIDES ARE VISIBLE.
70      '*****************************************************************
80  SCREEN 1: CLS
90  DIM XR(29), YR(29), ZR(29)
100 READ XO, YO, ZO
110 FOR K = 1 TO 29
120       READ XR(K), YR(K), ZR(K)
130       IF XR(K)<0 OR XR(K)>319 OR YR(K)<0 OR YR(K)>199 THEN 1020
140 NEXT
150 GOSUB 460
160 LOCATE 1,1: PRINT "ENTER Q TO QUIT"
170 INPUT "ROTATE AROUND WHAT AXIS"; R$
180 IF R$ = "Q" THEN 1020
190 INPUT "HOW MANY DEGREES"; A
200 A = A * 3.14159 / 180          'CONVERT A TO RADIANS
210 COSA = COS(A): SINA = SIN(A)
220      '***************** CALCULATE NEW POINTS *********************
230 IF R$ = "X" THEN 260
240 IF R$ = "Y" THEN 320
250 IF R$ = "Z" THEN 380
260 FOR K = 1 TO 29                'AROUND X AXIS
270      YS = YR(K)
280      YR(K) = INT(YO + (YR(K) - YO) * COSA + (ZR(K) - ZO) * SINA + .5)
290      ZR(K) = INT(ZO + (ZR(K) - ZO) * COSA - (YS - YO) * SINA + .5)
```

Program 11–3 (cont.)

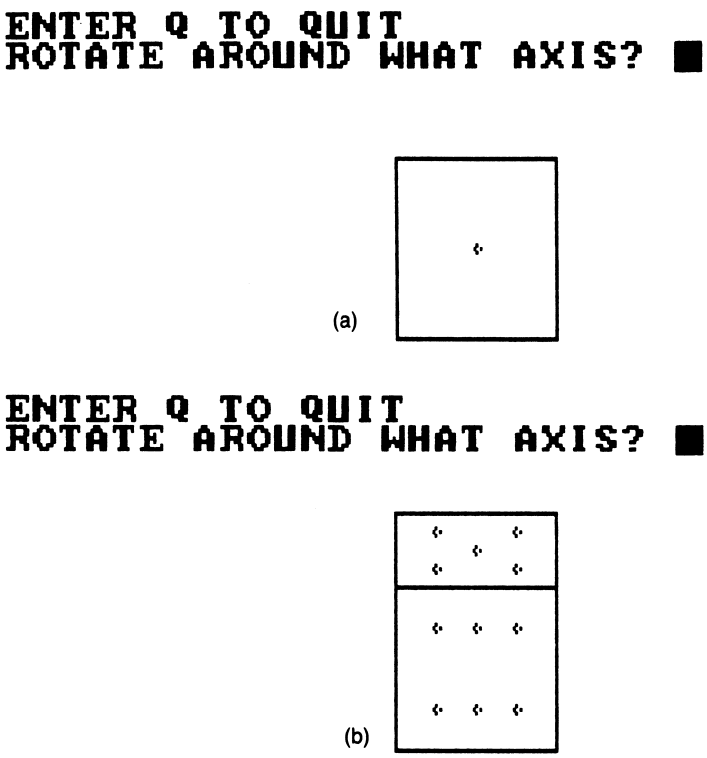
```
300 NEXT
310 GOTO 430
320 FOR K = 1 TO 29                'AROUND Y AXIS
330     XS = XR(K)       'SAVE XR(K) FOR USE IN ZR CALCULATION
340     XR(K) = INT(XO + (XR(K) - XO) * COSA - (ZR(K) - ZO) * SINA + .5)
350     ZR(K) = INT(ZO + (ZR(K) - ZO) * COSA + (XS - XO) * SINA + .5)
360 NEXT
370 GOTO 430
380 FOR K = 1 TO 29                'AROUND Z AXIS
390     XS = XR(K)       'SAVE XR(K) FOR USE IN YR CALCULATION
400     XR(K) = INT(XO + (XR(K) - XO) * COSA + (YR(K) - YO) * SINA + .5)
410     YR(K) = INT(YO + (YR(K) - YO) * COSA - (XS - XO) * SINA + .5)
420 NEXT
430 GOSUB 450
440 GOTO 160
450     '****************** DRAW ONLY VISIBLE FACES ******************
460 CLS
470 IF ZR(1) = ZR(5) THEN 630                'NEITHER SIDE IS VISIBLE
480 IF ZR(1) > ZR(5) THEN 560
490 FOR K = 1 TO 3                           'DRAW FACE CONTAINING 1 SPOT
500     LINE (XR(K),YR(K)) - (XR(K+1),YR(K+1))
510 NEXT
520 LINE (XR(4),YR(4)) - (XR(1),YR(1))
530 CIRCLE (XR(9),YR(9)),1
540 GOTO 630
550     '
560 FOR K = 5 TO 7                           'DRAW FACE CONTAINING 6 SPOTS
570     LINE (XR(K),YR(K)) - (XR(K+1),YR(K+1))
580 NEXT
590 LINE (XR(8),YR(8)) - (XR(5),YR(5))
600 FOR K = 24 TO 29
610     CIRCLE (XR(K),YR(K)),1
620 NEXT
630 IF ZR(1) = ZR(4) THEN 770                'NEITHER SIDE IS VISIBLE
640 IF ZR(1) > ZR(4) THEN 720
650 LINE (XR(1),YR(1)) - (XR(2),YR(2))       'DRAW FACE CONTAINING 4 SPOTS
660 LINE - (XR(6),YR(6)): LINE - (XR(5),YR(5)): LINE (XR(1),YR(1)
670 FOR K = 15 TO 18
680     CIRCLE ( XR(K), YR(K)),1
690 NEXT
700 GOTO 770
710     '
720 LINE (XR(4),YR(4)) - (XR(3),YR(3))       'DRAW FACE CONTAINING 3 SPOTS
730 LINE - (XR(7),YR(7)): LINE - (XR(8),YR(8)): LINE - (XR(4),YR(4))
740 FOR K = 12 TO 14
750     CIRCLE (XR(K),YR(K)),1
760 NEXT
770 IF ZR(1) = ZR(2) THEN 910                'NEITHER SIDE IS VISIBLE
780 IF ZR(1) > ZR(2) THEN 860
790 LINE (XR(1),YR(1)) - (XR(4),YR(4))       'DRAW FACE CONTAINING 2 SPOTS
800 LINE - (XR(8),YR(8)): LINE - (XR(5),YR(5)): LINE - (XR(1),YR(1))
810 FOR K = 10 TO 11
820     CIRCLE (XR(K),YR(K)),1
830 NEXT
840 GOTO 910
850     '
860 LINE (XR(2),YR(2)) - (XR(3),YR(3))       'DRAW FACE CONTAINING 5 SPOTS
870 LINE - (XR(7),YR(7)): LINE - (XR(6),YR(6)): LINE - (XR(2),YR(2))
880 FOR K = 19 TO 23
890     CIRCLE (XR(K),YR(K)),1
```

Program 11–3 (cont.)

```
900 NEXT
910 RETURN
920     '*****************************************************************
930 DATA 140,80,124
940 DATA 164,56,100,164,104,100,116,104,100,116,56,100
950 DATA 164,56,148,164,104,148,116,104,148,116,56,148
960 DATA 140,80,100
970 DATA 128,56,136,152,56,112
980 DATA 116,68,136,116,80,124,116,92,112
990 DATA 164,68,112,164,68,136,164,92,112,164,92,136
1000 DATA 128,104,112,128,104,136,140,104,124,152,104,112,152,104,136
1010 DATA 128,68,148,140,68,148,152,68,148,128,92,148,140,92,148,152,92,148
1020 END
```

transformations. Figure 11–6 shows the output of this program for rotation angles about each of the three axes we have discussed. In each case, an axis was chosen that passed through the center of the die. No perspective transformation was applied to the displayed view.

Figure 11–6 Rotated views of a three-dimensional object (Prog. 11–3): (a) original view, (b) rotated 155 degrees about an X axis, (c) rotated 28 degrees about a Z axis, then (d) rotated 45 degrees about a Y axis.



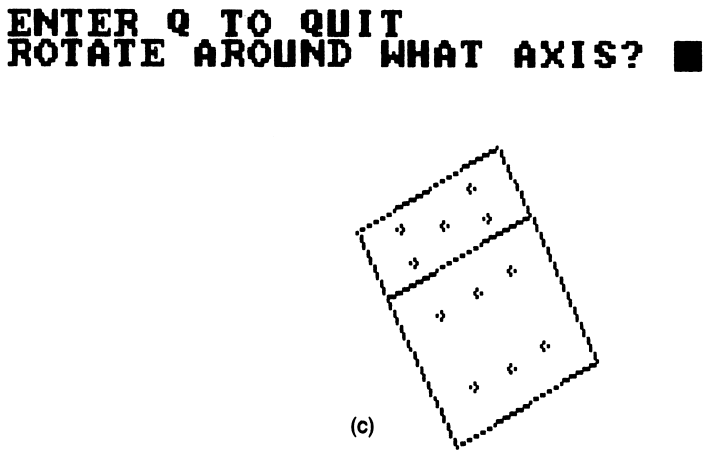


Figure 11–6 (cont.)

11–4 COMBINED TRANSFORMATIONS

We can collect the programs for the various transformations into one set of general routines. This general program could be organized so that an input figure is displayed and transformed through any combination of translation, scaling, rotation, and perspective views. We can use an interactive selection procedure for the transformations to be applied, with a termination signal to end the program. A hidden line (or surface) routine can be used to display only the visible sides of the object after a rotation or perspective transformation.

PROGRAMMING PROJECTS

11–1. Animate an object in three dimensions. The object is to be translated back and forth across the screen with alternating positive and negative increments for the X and Z directions. Initially move the object from left to right and increase the Z coordinate. When the object reaches the right side of the screen, reverse the X motion, leaving Y and Z coordinates constant. When the object returns to the left side of the screen, reverse the X direction and begin incrementing the Z coordinate in a negative direction. Repeating this motion, in perspective, moves the object along a figure 8 pattern in the X,Z plane.

11–2. Revise the program in Project 11–1 to animate an object along any three-dimensional curve.

11–3. Animate an object along a straight-line path in three dimensions that passes "behind" a rectangular wall drawn on the screen. Do not display parts of the object that are hidden by the wall. The program can be generalized to include several "wall" areas of various shapes.

11–4. Write a program to scale any two-dimensional or three-dimensional object by increasing its Z-coordinate position and applying the perspective equations.

11–5. Write an interactive animation program that moves an airplane (or other object) around in the Z direction as it flies back and forth across the screen in the X direction. Use either joystick or keyboard input to control the Z motion. For joystick input, the Y coordinate input can be converted to Z coordinates. For keyboard input, the up and down arrows on the numeric keypad can be used. Either scale the airplane or apply perspective transformations to change its size to correspond to the Z movement. To make this into a game, the Z movement can be used to dodge objects that get into the airplane's path. The Z coordinates for both the airplane and the other objects can be printed on the screen as a means for determining the relative depth of objects. The airplane can then be made to fly in front or in back of the other object. Erase the airplane when it flies in back of an object; erase an object when the airplane flies in front of it.

11–6. Set up a program to scale a three-dimensional object in an arbitrary direction. This is done by rotating the object to any specified position, then scaling in the X, Y, and Z directions.

11–7. Write a program that displays words written in large letters that decrease in size into the distance. This can be accomplished by rotating the word about a vertical line (parallel to the Y axis) through the left side of the word and applying the perspective equations. The word can then be translated (and scaled) to any screen position.

11–8. Write a program to display a globe formed with a rotated circle. Rotating a circle in three 30-degree steps about a diameter parallel to the Y axis produces a sphere shape with meridian lines. Taking smaller angular rotation steps, from the original position to 90 degrees, creates more meridian lines. The lines of latitude are drawn horizontally between the boundaries of the original circle position.

11–9. Repeat the method of Project 11–8 to form a cylinder by rotating a rectangle about a vertical axis, then about a horizontal axis to show the top or bottom. Erase hidden lines.

11–10. Write a program to display a "solid of revolution" formed from a straight line. Draw a diagonal line with different Z values for each endpoint. Then rotate the line about a vertical axis (parallel to the Y axis) through its midpoint. Rotating about a horizontal axis through the line midpoint then displays the ends of the hourglass figure. A hidden-line method can be applied to produce a realistic solid figure.

11–11. Set up a program to display a "solid of revolution" formed from a parabola. A displayed parabola can first be rotated about a vertical line (parallel to the Y axis) through its center in small angular steps from its original position to 180 degrees. Then, rotating it about a horizontal line (parallel to the X axis) that passes through the parabola will display the concave interior. Lines can be drawn joining the endpoints of the rotated parabola to give the figure a cup appearance.

11–12. Set up a three-dimensional bar chart at the screen center and apply rotations, translation, and perspective to position it at any other screen location and orientation.

11–13. Devise a general three-dimensional transformation program that combines the various transformations, the perspective equations, and a hidden line (or surface) method, each written as a separate subroutine. For any object input to the program, a particular transformation is to be selected from a displayed menu and the object is then transformed and drawn in perspective with hidden lines removed. Transformations are to be repeatedly selected until a termination input (such as typing STOP) is given. Menu options may be chosen from the keyboard or with a light pen or joystick. Similarly, the input object may be specified in data statements or constructed interactively.

Part V
APPLICATIONS

The methods for creating and manipulating pictures and graphs that we have explored in the preceding chapters have a wide range of applications. We will look at some of the ways we can use graphics in business, in the classroom, and at home.

Chapter 12

Business Graphics

Business use of computer graphics represents one of the largest and most diversified applications areas. Computer-generated graphs, charts, and pictures are commonly used as aids to financial analysis, marketing studies, planning, and decision making. These are typical functions in many different kinds of organizations. We can create graphs and charts to provide information on budgets, inventories, cash flow, net income, interest rates, return on investments, or portfolio analysis. We can plot graphs to show comparisons in pricing or product characteristics between competitors, to show regional buying habits, or to show sales trends plotted by region, salesperson, or year. Graphs of demographic data can be useful for locating potential customers or adjusting sales territories. Network graphs and time charts provide aids for project management. Pictorial layouts can help in planning for facility and equipment placement.

We can set up business graphs and charts so as to present data by geographic area or by division within an organization. We can also graphically correlate new data with old in order to provide quick comparison or for indicating future trends. Such graphs are used in internal reports, status reports to customers, or presentations. In this chapter, we discuss some of the methods for producing various types of business graphs and charts.

12–1 GENERAL TECHNIQUES

Graph-plotting methods were introduced in Chapters 4 and 5. We now consider some extensions to these basic methods that are commonly used in business graphics.

A common technique used with pie charts is to emphasize one or more

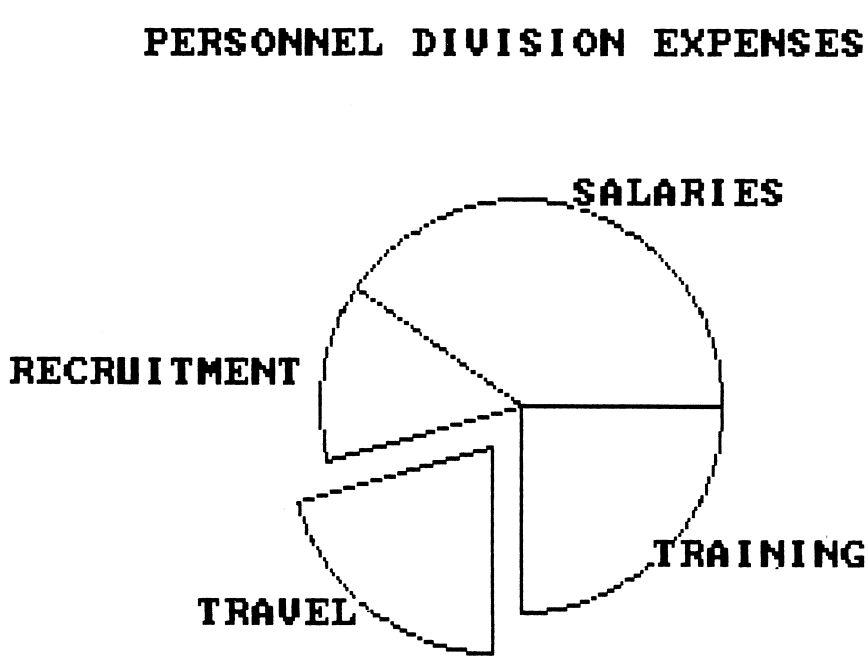


Figure 12–1 Exploded pie chart displayed by Prog. 12–1.

sections by displacing the sections radially out from the center. An example of such an exploded pie chart is shown in Fig. 12–1. The technique used in Prog. 12–1 to produce this chart is based on the method we discussed in Section 5–4 for positioning labels on pie charts. We first determine the angular bounds of the section to be exploded. Then we calculate the angle of the radius line for the middle of this section. Finally, we locate a center point for the exploded section out from the chart center along this radius and draw the section from that point.

Program 12–1 Exploded pie chart.

```
10 'PROGRAM 12-1. EXPLODED PIE CHART.
20      'MAKES LABELED PIE CHART WITH ANY SINGLE SECTION EXPLODED.
30  DIM LABEL$(8), VALUE(8)
40  SCREEN 0: WIDTH 80: CLS
50  PIXELSPERCOLUMN = 8: PIXELSPERROW = 8
60  INPUT "CENTER COORDINATES FOR PIECHART"; XNORMAL,YNORMAL
70  INPUT "RADIUS"; RADIUS
80  IF XNORMAL+RADIUS > 319 OR XNORMAL-RADIUS < 0 OR YNORMAL+RADIUS > 199 OR
        YNORMAL-RADIUS < 0 THEN 570
90      '************************* INPUT DATA **********************
100 INPUT "TITLE"; TITLE$
110 INPUT "NUMBER OF DIVISIONS (UP TO 8)"; DIVISIONS
120 PRINT "ENTER NAME AND VALUE FOR EACH DIVISION"
130 TOTAL = 0
140     'INPUT DATA. FIND TOTAL OF ALL VALUES.
150 FOR K = 1 TO DIVISIONS
160     INPUT LABEL$(K), VALUE(K)
170     TOTAL = TOTAL + VALUE(K)
180 NEXT
```

Program 12–1 (cont.)

```
190 PRINT "EXPLODE WHICH DIVISION (1 -"; STR$(DIVISIONS); ")";
200 INPUT EXPLODER
210 IF EXPLODER < 1 OR EXPLODER > DIVISIONS THEN 190
220     '************************* MAKE PIE CHART ******************
230 SCREEN 1: CLS
240 LOCATE 1,20-LEN(TITLE$)/2: PRINT TITLE$
250 BEFORE = 0
260 FOR K = 1 TO DIVISIONS
270     XCENTER = XNORMAL
280     YCENTER = YNORMAL
290         'LINE TO PLOT IS BASED ON THE VALUE OF THIS
300         'DIVISION PLUS PRECEDING DIVISIONS
310     ANGLE = BEFORE + 6.28318 * VALUE(K) / TOTAL
320     BISECTANGLE = 6.28318 - ((ANGLE + BEFORE) / 2)
330     IF K <> EXPLODER THEN 370               'IS THIS SECTION TO EXPLODE?
340     XEXPLODE = XCENTER + INT(RADIUS / 5 * COS(BISECTANGLE) + .5)
350     YEXPLODE = YCENTER + INT(RADIUS / 5 * SIN(BISECTANGLE) * .9199999 + .5)
360     XCENTER = XEXPLODE: YCENTER = YEXPLODE
370     CIRCLE (XCENTER,YCENTER),RADIUS,,-BEFORE,-ANGLE,.9199999
380         'PUT LABEL ON DIVISION
390         'FIND A POINT 4 UNITS OUTWARD FROM THE CENTER
400         'POINT OF THE ARC BELONGING TO THE DIVISION
410     XLABEL = XCENTER + (RADIUS + 4) * COS(BISECTANGLE)
420     YLABEL = YCENTER + (RADIUS + 4) * SIN(BISECTANGLE) * .9199999
430         '(XLABEL,YLABEL) IS THE POINT USED TO ANCHOR LABEL
440         'USE THE POINT AS START OF LABEL IF IT'S ON RIGHT
450         'SIDE OF CIRCLE, AS END OF LABEL IF IT'S ON LEFT,
460         'AS MIDPOINT IF IT'S ON TOP OR BOTTOM OF CIRCLE
470     IF XLABEL > XCENTER - 10  AND XLABEL < XCENTER + 10 THEN
           XLABEL = XLABEL - LEN(LABEL$(K)) / 2 * PIXELSPERCOLUMN: GOTO 490
480     IF XLABEL < XCENTER - 10 THEN
           XLABEL = XLABEL - LEN(LABEL$(K)) * PIXELSPERCOLUMN
490         'CONVERT THE PIXEL LOCATION (XLABEL,YLABEL) TO THE CLOSEST
500         'CORRESPONDING PRINT POSITION
510     ROW = INT(YLABEL / PIXELSPERROW) + 1
520     COLUMN = INT(XLABEL / PIXELSPERCOLUMN) + 1
530     LOCATE ROW,COLUMN: PRINT LABEL$(K);
540     BEFORE = ANGLE
550 NEXT
560 GOTO 580
570 PRINT "COORDINATE OUT OF RANGE"
580 IF INKEY$ = "" THEN 580
590 END
```

Program 12–1 exploded sections by taking the displacement distance out from the center to be one-fifth of the radius of the chart. Color and shading patterns can be added, as illustrated in Fig. N of the color insert, to aid in separating and identifying the pie sections.

There are times when we would like to produce graphs showing negative quantities, such as financial losses. We can do this by plotting negative values below a horizontal axis, as shown in Fig. 12–2. These graphs are produced by simply extending the vertical axis below the horizontal axis and labeling both positive and negative values.

Plotting two types of graphs on the same axis is an effective means for condensing charts or comparing data. Program 12–2 gives an example of plotting a bar chart on one part of the horizontal axis and a curve on another part. Different scaling is used for each part of the horizontal axis, while one vertical axis serves

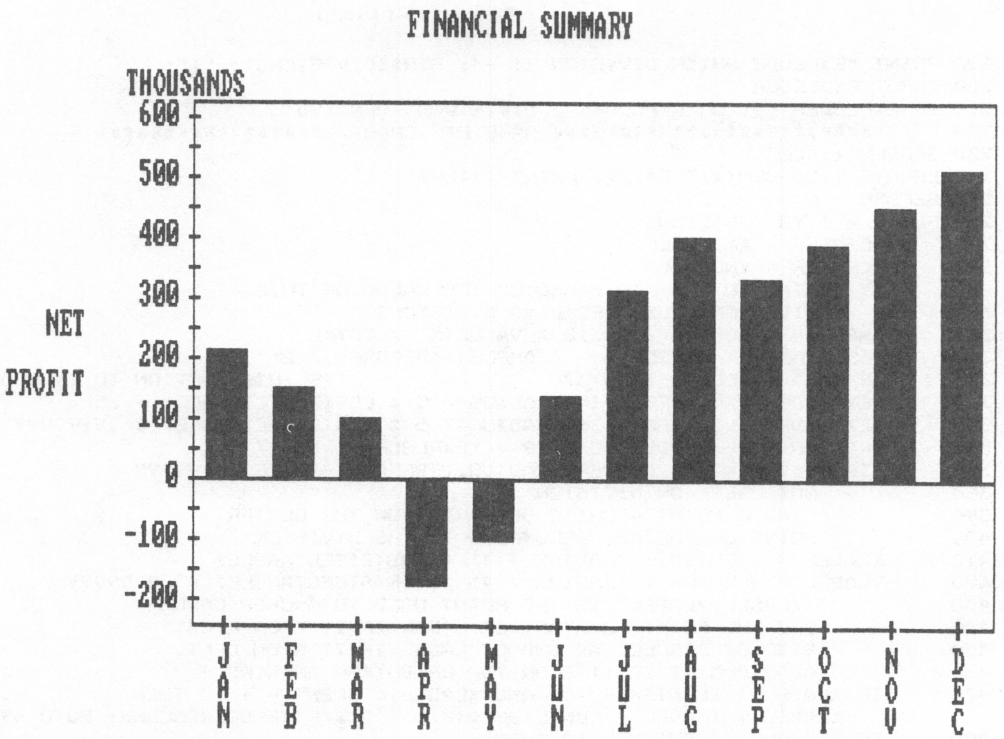


Figure 12–2 Graph scaled to show both positive and negative quantities.

Program 12–2 Combination graphs: bar chart and line graph.

```
10 'PROGRAM 12-2. COMBINED BAR AND CURVE CHART ON X AXIS,
20      'WITH SAME SCALING ON Y AXIS.
30      'SALES VALUES (in the range of 0 - 800) ARE SCALED TO PIXELS
40      '28 - 156.  MONTHS USE EVERY 40 PIXELS, STARTING AT 148.
50      '*****************************************************************
60  DIM X(12), Y(12)
70  SCREEN 2: CLS
80  LOCATE 1,34: PRINT "SALES FIGURES"
90  LOCATE 3,10: PRINT "THOUSANDS";
100  LINE (128,24) - (608,163),,B
110     'MAKE NOTCHES FOR SALES MAGNITUDES
120 FOR Y = 28 TO 156 STEP 8
130     LINE (125,Y) - (131,Y)
140 NEXT
150     'LABEL THE NOTCHES
160 ROW = 20
170 FOR S = 0 TO 800 STEP 100
180     LOCATE ROW,13: PRINT USING "###"; S
190     ROW = ROW - 2
200 NEXT
210     'MAKE NOTCHES FOR QUARTERS AND WEEKS
220 FOR COLUMN = 24 TO 40 STEP 8
230     LOCATE 21,COLUMN: PRINT "+";
240 NEXT
250 FOR COLUMN = 48 TO 73 STEP 5
260     LOCATE 21,COLUMN: PRINT "+";
```

Program 12–2 (cont.)

```
270 NEXT
280 PRINT TAB(22);"FIRST   SECOND   THIRD    40   42   44   46   48   50"
290 PRINT TAB(22);"     QUARTERS                          WEEKS"
300 LOCATE 13,3: PRINT " SALES"
310     'DRAW CHART BARS
320 RANGERATIO = (156 - 28) / (800 - 0)
330 X = 163
340 FOR K = 1 TO 3
350     READ SALES
360     Y = INT ((800 - SALES) * RANGERATIO + 28.5)
370     LINE (X,Y) - (X+40,156),,BF
380     X = X + 64
390 NEXT
400     'DRAW WEEKLY CURVE
410 X = 384
420 FOR K = 1 TO 6
430     READ SALES
440     Y(K) = INT((800-SALES) * RANGERATIO + 28.5)
450 NEXT
460 FOR K = 1 TO 5
470     LINE (X,Y(K)) - (X+40,Y(K+1))
480     X = X + 40
490 NEXT
500 DATA 310,420,599,598,623,592,670,740,695
510 IF INKEY$ = "" THEN 510
520 END
```

both graphs. This allows us to plot portions of a data set, for example, in two different forms. Figure 12–3 shows the resulting output.

Presenting data in different graphical forms can aid in interpreting the data. Program 12–3 outlines a modular design that allows us to choose the graph or

Program 12–3 General graphing program—allowing graph type to be chosen.

```
10 'PROGRAM 12-3. GENERALIZED GRAPHING PROGRAM.
20      'ALLOWS INTERACTIVE INPUT OF DATA AND CHOICE OF
30      'CHART FORMAT (LINE, BAR, OR PIE CHARTS).
40  DIM LABEL$(10), VALUE(10)
50  SCREEN 2: CLS
60  PC = 8          'PC IS HORIZONTAL PIXELS PER CHARACTER
70  PR = 8          'PR IS VERTICAL PIXELS PER CHARACTER
80  YADJUST = .46               'YADJUST IS RESOLUTION ADJUSTMENT
90  PRINT "1 - LINE       2 - BAR       3 - PIE"
100 INPUT "WHAT KIND OF CHART"; CHART$
110 INPUT "TITLE OF CHART";TITLE$
120 IF CHART$ = "1" OR CHART$ = "2" THEN GOSUB 170
130 IF CHART$ = "1" THEN GOSUB 510
140 IF CHART$ = "2" THEN GOSUB 610
150 IF CHART$ = "3" THEN GOSUB 700
160 GOTO 1130
170 '############################ LINE OR BAR CHART ##############################
180 INPUT "ENTER NUMBER OF DIVISIONS";N
190 D = INT(568 / N)
200 PRINT "ENTER NAME AND VALUE OF EACH DIVISION"
210 FOR K = 1 TO N
220     INPUT LABEL$(K),VALUE(K)
230     IF LEN(LABEL$(K)) < D/8 THEN 260    'WILL THE LABEL FIT?
240     PRINT "LABEL TOO LONG. MAXIMUM LENGTH IS";D/8
250     INPUT "NEW LABEL"; LABEL$(K)
260 NEXT
```

Program 12–3 (cont.)

```
270 INPUT "MINIMUM AND MAXIMUM VALUES FOR VERTICAL AXIS"; LO,HI
280 RANGE = HI - LO                          'R IS RANGE OF VALUES
290 RS = (180 - 20) / RANGE                  'RS IS RATIO TO USE IN SCALING
300 CLS
310 LINE (71,20) - (639,180),,B
320 P = 40 - LEN(TITLE$) / 2          'CENTER THE TITLE
330 LOCATE 1,P: PRINT TITLE$
340 ROW = 23
350 Y = 180
360 FOR K = 0 TO 5                  'LABEL VERTICAL AXIS WITH SUCCESSIVE FIFTHS
370     LOCATE ROW,1                  'OF THE SCALING RANGE
380     L = LO + RANGE * K / 5
390     PRINT USING "###.###";L
400     LINE (70,Y) - (639,Y)
410     ROW = ROW - 4
420     Y = Y - 32
430 NEXT
440    'LABEL THE DIVISIONS
450 LOCATE 25,1
460 FOR K = 1 TO N
470     P = INT((72 + (K-1) * D + D/2) / PC - LEN(LABEL$(K))/2 + .5) + 1
480     PRINT TAB(P);LABEL$(K);
490 NEXT
500 RETURN
510 '################################### LINE CHART ##########################
520 FOR K = 1 TO N
530     Y(K) = INT((HI - VALUE(K)) * RS + 20.5)
540 NEXT
550 X = 71 + D/2
560 FOR K = 1 TO N-1
570     LINE (X,Y(K)) - (X+D,Y(K+1))
580     X = X + D
590 NEXT
600 RETURN
610 '################################### BAR CHART ###########################
620 YO = INT(HI * RS + 20.5)                 'FIND WHERE 0 IS
630 X = 71 + D/6
640 FOR K = 1 TO N
650     Y = INT((HI - VALUE(K)) * RS + 20.5)
660     LINE (X,Y) - (X+D*2/3,YO),,BF
670     X = X + D
680 NEXT
690 RETURN
700 '################################### PIE CHART ###########################
710 XCENTER = 320
720 YCENTER = 110
730 RADIUS = 160
740 INPUT "NUMBER OF DIVISIONS (UP TO 6)"; N
750 PRINT "ENTER NAME AND VALUE FOR EACH DIVISION"
760 TOTAL = 0
770     'INPUT DATA. FIND TOTAL (T) OF ALL VALUES.
780 FOR K = 1 TO N
790     INPUT LABEL$(K), VALUE(K)
800     IF VALUE(K) > 0 THEN 820
810     IF VALUE(K)<0 THEN PRINT "MUST BE POSITIVE. DO OVER":
              INPUT VALUE(K): GOTO 810
820     TOTAL = TOTAL + VALUE(K)
830 NEXT
840 CLS
850 P = 40 - LEN(TITLE$) / 2          'CENTER THE TITLE
860 LOCATE 1,P: PRINT TITLE$
870 BEFORE = 0              'B IS ANGLE THAT DETERMINED PRECEDING LINE
880 FOR K = 1 TO N
```

Program 12–3 (cont.)

```
890          'LINE TO PLOT IS BASED ON THE VALUE OF THIS
900          'DIVISION PLUS PRECEDING DIVISIONS
910      ANGLE = BEFORE + 6.28318 * VALUE(K) / TOTAL
920      CIRCLE (XCENTER,YCENTER),RADIUS,,-BEFORE,-ANGLE,YADJUST
930          'PUT LABEL ON DIVISION
940          'FIND A POINT 4 UNITS OUTWARD FROM THE CENTER
950          'POINT OF THE ARC BELONGING TO THE DIVISION
960      BISECTANGLE = 6.28318 - ((BEFORE + ANGLE) / 2)
970      XLABEL = XCENTER + (RADIUS + 4) * COS(BISECTANGLE)
980      YLABEL = YCENTER + (RADIUS + 4) * SIN(BISECTANGLE) * YADJUST
990          '(XLABEL,YLABEL) IS THE POINT USED TO ANCHOR LABEL
1000         'USE THE POINT AS START OF LABEL IF IT'S ON RIGHT
1010         'SIDE OF CIRCLE, AS END OF LABEL IF IT'S ON LEFT,
1020         'AS MIDPOINT IF IT'S ON TOP OR BOTTOM OF CIRCLE
1030     IF XLABEL > XCENTER - 10 AND XLABEL < XCENTER + 10 THEN
                 XLABEL = XLABEL - LEN(LABEL$(K)) / 2 * PC: GOTO 1050
1040     IF XLABEL < XCENTER - 10 THEN
                 XLABEL = XLABEL - LEN(LABEL$(K)) * PC
1050         'CONVERT THE PIXEL LOCATION (XLABEL,YLABEL) TO THE CLOSEST
1060         'CORRESPONDING PRINT POSITION
1070      ROW = INT(YLABEL / PR)+1
1080      COLUMN = INT(XLABEL / PC)+1
1090      LOCATE ROW,COLUMN: PRINT LABEL$(K);
1100      BEFORE = ANGLE
1110 NEXT
1120 RETURN
1130 IF INKEY$ = "" THEN 1130
1140 END
```

Figure 12–3 A combination bar chart and line graph produced by Prog. 12–2, showing one vertical scale and two different horizontal scales.

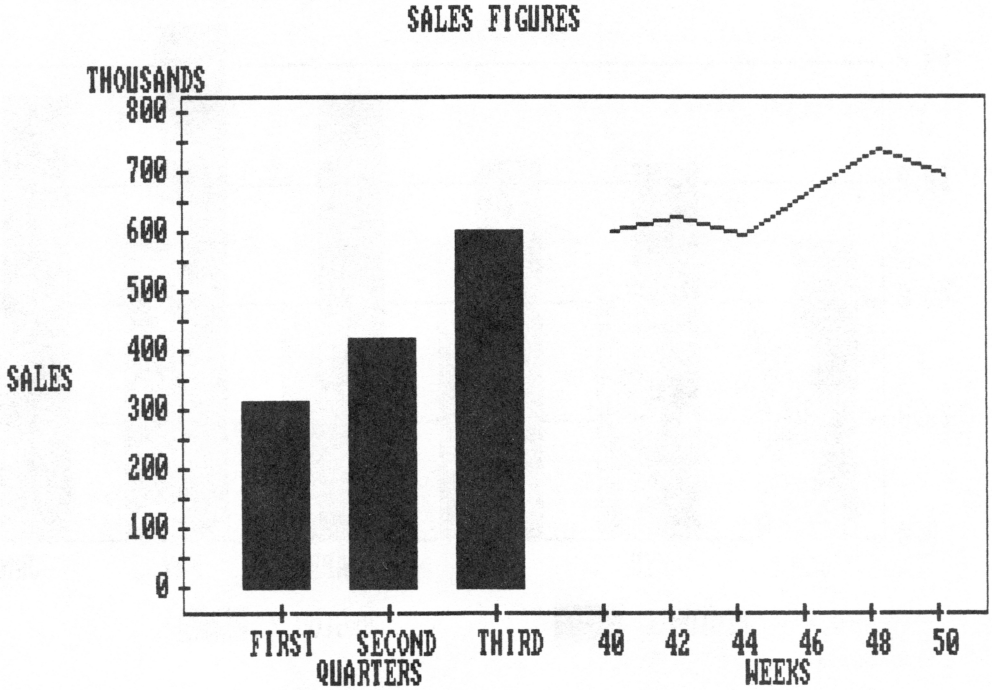


chart type, as well as the data to be plotted. This general design can be useful for producing different graph forms with the same data and to experiment with various parameter changes.

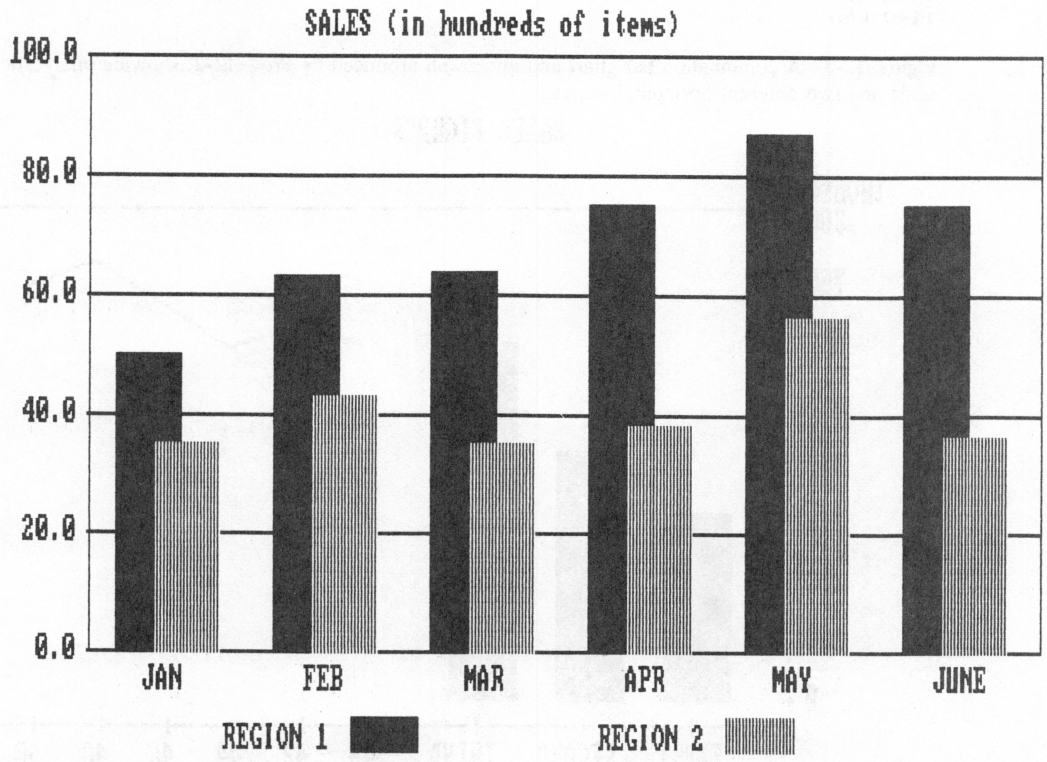
12–2 COMPARATIVE GRAPHS

We can plot two or more sets of data within one graph in order to compare relationships between variables, such as sales of products in relation to district or in relation to salesperson. There are several ways we can set up such comparative graphs.

In a bar graph, we can compare two sets of data by overlapping the bars (Fig. 12–4). Programs to produce this type of comparative graph can plot each data set in a different shading as in Prog. 12–4. The bars for one data set are shifted slightly and drawn over the bars of the other set. We could also construct such graphs in SCREEN 1, and plot the bars as different-colored, filled-in boxes.

Another type of comparative bar chart is shown in Fig. 12–5. This graph was

Figure 12–4 Comparative bar graph from Prog. 12–4, showing overlapping bars from two data sets plotted on the same axes.



produced by Prog. 12–5. In this example, the two data sets share the same X axis, but have different Y axes. This arrangement allows us to use a different vertical scaling for each data set.

Curves can be compared by simply plotting multiple sets of data points on

Program 12–4 Comparative graph: overlaid bar charts.

```
10 'PROGRAM 12-4. TWO SETS OF DATA WITH OVERLAPPING BARS.
20      'PLOTS BARS FOR TWO SETS OF DATA SHARING THE SAME X AND Y AXIS.
30  DIM LABEL$(12), VALUE1(12), VALUE2(12)
40  SCREEN 2: CLS
50  PC = 8        'PC IS HORIZONTAL PIXELS PER CHARACTER
60      '********************** INPUT CHART LABELS AND DATA ***************
70  INPUT "TITLE OF CHART";TITLE$
80  INPUT "NAME OF FIRST SET OF DATA"; TITLE1$
90  INPUT "NAME OF SECOND SET OF DATA"; TITLE2$
100 INPUT "NUMBER OF DIVISIONS"; N
110 D = INT(568 / N)     'D IS AREA FOR EACH DIVISION
120 PRINT "ENTER LABEL AND TWO VALUES FOR EACH DIVISION"
130 FOR K = 1 TO N
140     INPUT LABEL$(K),VALUE1(K),VALUE2(K)
150     IF LEN(LABEL$(K)) > D/PC THEN PRINT "TOO LONG. MAXIMUM LENGTH IS"; D/PC:
              INPUT "NEW LABEL"; LABEL$(K): GOTO 150
160 NEXT
170 PRINT "ENTER MINIMUM AND MAXIMUM VALUES FOR VERTICAL AXIS"
180 INPUT LO, HI
190 RANGE = HI - LO
200 RS = (172 - 12) / RANGE              'RS IS RATIO TO USE IN SCALING
210     '************************** DRAW GRID AND LABELS ********************
220 CLS
230 LINE (71,12) - (639,172),,B
240 P = 40 - LEN(TITLE$) / 2          'CENTER THE TITLE
250 LOCATE 1,P: PRINT TITLE$
260 ROW = 22
270 Y = 172
280 FOR K = 0 TO 5                 'LABEL VERTICAL AXIS WITH SUCCESSIVE FIFTHS
290     L = LO + RANGE * K / 5
300     LOCATE ROW,3: PRINT USING "####.#";L
310     LINE (70,Y) - (639,Y)
320     ROW = ROW - 4
330     Y = Y - 32
340 NEXT
350    'LABEL THE DIVISIONS
360 FOR K = 1 TO N
370     P = INT((71+(K-1)*D+D/2)/PC - LEN(LABEL$(K))/2 + .5) + 1
380     LOCATE 23,P: PRINT LABEL$(K);
390 NEXT
400     'MAKE CODE AREAS
410 LOCATE 25,28-LEN(TITLE1$): PRINT TITLE1$;
420 LINE (228,190) - (268,199),,BF
430 LOCATE 25,56-LEN(TITLE2$): PRINT TITLE2$;
440 FOR X = 452 TO 492 STEP 2
450     LINE (X,190) - (X,199)
460 NEXT
470     '***************************** MAKE BARS ****************************
480 YZERO = INT(HI * RS + 12.5)              'FIND WHERE 0 IS
490 X1 = 71 + D/6
500 FOR K = 1 TO N
510     Y = INT((HI - VALUE1(K)) * RS + 12.5)
```

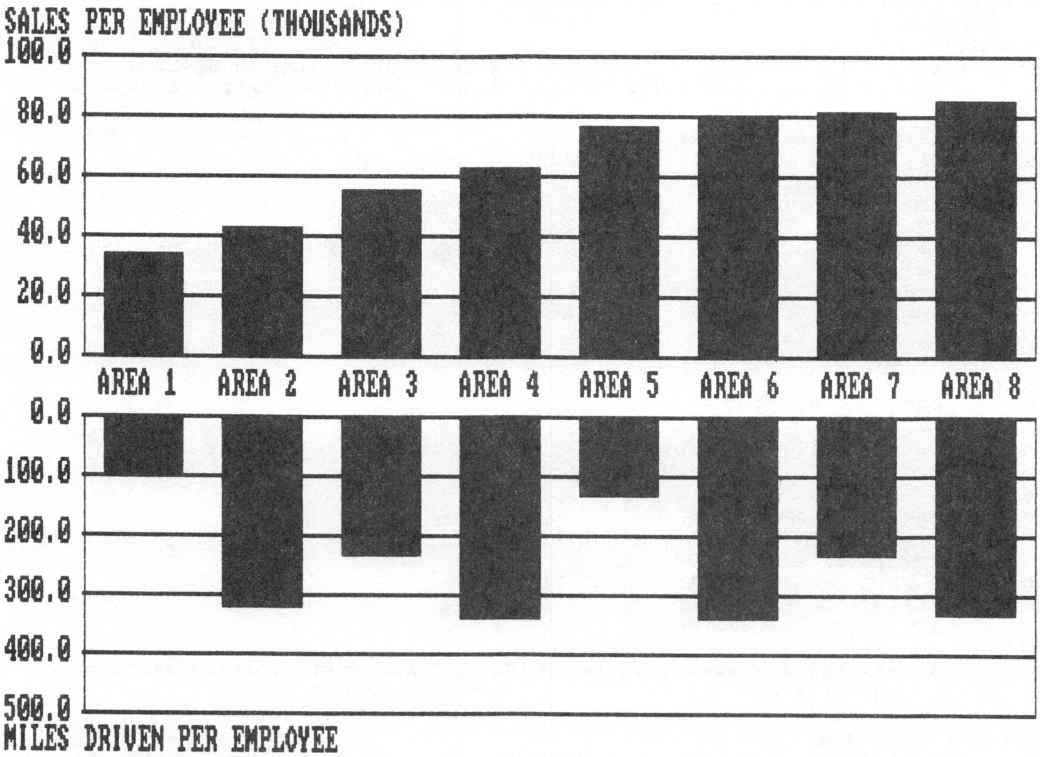
Program 12–4 (cont.)

```
520      LINE (X1,Y) - (X1+D*5/12,YZERO),,BF
530      X2 = INT(X1 + D * 3/12 + .5)          'MOVE OVER FOR SECOND BAR
540      Y = INT((HI - VALUE2(K)) * RS + 12.5)
550      FOR X = X2 TO X2 + D * 5/12 STEP 2
560          LINE (X,Y) - (X,YZERO)
570          LINE (X+1,Y) - (X+1,YZERO),0
580      NEXT
590      X1 = X1 + D
600 NEXT
610 IF INKEY$ = "" THEN 610
620 END
```

the same graph. We can also construct a cumulative surface chart with methods demonstrated in Prog. 12–6. The resulting graph, shown in Fig. 12–6, plots the upper curve as the sum of the two data sets and uses shading to distinguish the areas. This type of graph can often lead to misinterpretations, since the upper line shows the sum of the two data sets and not actual data values.

Shading patterns or color can be used to emphasize areas between curves, as

Figure 12–5 A comparative graph displayed by Prog. 12–5, with bars drawn up for one data set and down for the other.



Program 12–5 Comparative graph of two bar charts: one up, one down.

```
10 'PROGRAM 12-5. COMPARATIVE BAR GRAPH WITH TWO Y SCALES.
20      'INPUTS TITLES AND TWO SETS OF DATA FOR ANY NUMBER
30      'OF DIVISIONS. DRAWS BARS FOR ONE SET OF DATA ON
40      'A VERTICAL AXIS IN THE TOP HALF OF THE SCREEN AND
50      'BARS FOR THE OTHER SET OF DATA ON A VERTICAL AXIS
60      'IN THE BOTTOM HALF OF THE SCREEN.
70      '************************************************************
80  DIM L$(15), T(15), B(15)
90  SCREEN 2: CLS
100 PC = 8          'PC IS HORIZONTAL PIXELS PER CHARACTER
110     '********************* INPUT DATA ***********************
120 INPUT "NUMBER OF DIVISIONS"; N
130 D = INT(568 / N)                'D IS NUMBER OF PIXELS ALLOWED PER DIVISION
140 PRINT "ENTER NAME AND VALUES OF EACH DIVISION"
150 FOR K = 1 TO N
160     INPUT LABEL$(K),TOP(K), BOTTOM(K)
170     IF LEN(LABEL$(K)) < D/PC THEN 200       'WILL THE LABEL FIT?
180     PRINT "LABEL TOO LONG. MAXIMUM LENGTH IS";D/PC
190     INPUT "NEW LABEL"; LABEL$(K): GOTO 170
200 NEXT
210 INPUT "LABEL FOR TOP VERTICAL AXIS"; TOPTITLE$
220 INPUT "MINIMUM AND MAXIMUM VALUES FOR TOP VERTICAL AXIS"; LOTOP, HITOP
230 RANGE1 = HITOP - LOTOP
240 RSTOP = (92 - 12) / RANGE1          'RATIO TO USE IN SCALING TOP
250 INPUT "LABEL FOR BOTTOM VERTICAL AXIS"; BOTTOMTITLE$
260 INPUT "MINIMUM AND MAXIMUM VALUES FOR BOTTOM VERTICAL AXIS"; LOBOT,HIBOT
270 RANGE2 = HIBOT - LOBOT
280 RSBOTTOM = (188 - 108) / RANGE2         'RATIO TO USE IN SCALING BOTTOM
290     '************************** DRAW BACKGROUND ***********************
300 CLS
310 LINE (71,12) - (639,92),,B
320 LINE (71,108) - (639,188),,B
330 LOCATE 1,4: PRINT TOPTITLE$
340 ROW = 12
350 Y = 92
360 FOR K = 0 TO 5              'LABEL VERTICAL AXIS WITH SUCCESSIVE FIFTHS
370     L = LOTOP + RANGE1 * K / 5
380     LOCATE ROW,3: PRINT USING "####.#";L;
390     LINE (70,Y) - (639,Y)
400     ROW = ROW - 2
410     Y = Y - 16
420 NEXT
430 LOCATE 25,4: PRINT BOTTOMTITLE$;
440 ROW = 14
450 Y = 108
460 FOR K = 0 TO 5
470     L = LOBOT + RANGE2 * K / 5
480     LOCATE ROW,3: PRINT USING "####.#";L;
490     LINE (70,Y) - (639,Y)
500     ROW = ROW + 2
510     Y = Y + 16
520 NEXT
530     'LABEL THE DIVISIONS
540 LOCATE 13,1
550 FOR K = 1 TO N
560     P = INT((72 + (K-1) * D+ D/2) / PC - LEN(LABEL$(K))/2 + .5) + 1
570     PRINT TAB(P);LABEL$(K);
580 NEXT
590     '************************** MAKE BARS *********************
600 X = 71 + D/6
```

Program 12–5 (cont.)

```
610 FOR K = 1 TO N
620     YTOP = INT((HITOP - TOP(K)) * RSTOP + 12.5)
630     YBOT = INT(BOTTOM(K) * RSBOTTOM + 108.5)
640     LINE (X,YTOP) - (X+D*2/3,92),,BF
650     LINE (X,YBOT) - (X+D*2/3,108),,BF
660     X = X + D
670 NEXT
680 IF INKEY$ = "" THEN 680
690 END
```

Program 12–6 Cumulative surface chart, plotting two data sets.

```
10 'PROGRAM 12-6. SURFACE CHART WITH TWO SETS OF CUMULATIVE DATA.
20     'PLOTS LOWER CURVE FOR ONE SET OF DATA. PLOTS UPPER CURVE
30     'AS THE SUM OF FIRST AND SECOND SETS OF DATA.
40 DIM LABEL$(20), VALUE1(20), VALUE2(20)
50 SCREEN 2: CLS
60 PC = 8          'PC IS HORIZONTAL PIXELS PER CHARACTER
70     '********************** INPUT CHART LABELS AND DATA ***************
80 INPUT "TITLE OF CHART"; TITLE$
90 INPUT "NAME OF FIRST SET OF DATA"; TITLE1$
100 INPUT "NAME OF SECOND SET OF DATA"; TITLE2$
110 INPUT "ENTER NUMBER OF HORIZONTAL DIVISIONS"; N
120 D = INT(568 / N)
130 IF D/6 <> INT(D/6) THEN D = INT(D/6+.5)*6
140 PRINT "ENTER LABEL AND TWO VALUES FOR EACH DIVISION"
150 MAX2 = 0                'MAX2 IS THE MAXIMUM VALUE IN ARRAY S2
160 FOR K = 1 TO N
170     INPUT LABEL$(K),VALUE1(K),VALUE2(K)
180     VALUE2(K) = VALUE1(K) + VALUE2(K)
190     IF VALUE2(K) > MAX2 THEN MAX2 = VALUE2(K)
200     IF LEN(LABEL$(K)) < D/PC THEN 230     'WILL THE LABEL FIT?
210     PRINT "LABEL TOO LONG. MAXIMUM LENGTH IS";D/PC
220     INPUT "NEW LABEL"; LABEL$(K): GOTO 200
230 NEXT
240 INPUT "ENTER MINIMUM AND MAXIMUM VALUES FOR VERTICAL AXIS"; LO, HI
250 IF HI >= MAX2 THEN 260 ELSE
       INPUT "MAXIMUM VALUE NOT ENOUGH. ENTER MAXIMUM AGAIN"; HI: GOTO 250
260 RANGE = HI - LO
270 RS = (172 - 12) / RANGE              'RS IS RATIO TO USE IN SCALING
280     '************************* DRAW GRID AND LABELS ********************
290 CLS
300 LINE (71,12) - (639,172),,B
310 P = 40 - LEN(TITLE$) / 2          'CENTER THE TITLE
320 LOCATE 1,P: PRINT TITLE$;
330 ROW = 22
340 Y = 172
350 FOR K = 0 TO 5                'LABEL VERTICAL AXIS WITH SUCCESSIVE FIFTHS
360     L = LO + RANGE * K / 5
370     LOCATE ROW,3: PRINT USING "####.#"; L;
380     LINE (70,Y) - (639,Y)
390     ROW = ROW - 4
400     Y = Y - 32
410 NEXT
420    'LABEL THE DIVISIONS
430 LOCATE 23,1
440 FOR K = 1 TO N
450     P = INT((71+(K-1)*D+D/2)/PC - LEN(LABEL$(K))/2 + .5) + 1
460     PRINT TAB(P); LABEL$(K);
```

Program 12–6 (cont.)

```
470 NEXT
480     'MAKE CODE AREAS
490 LOCATE 25,28-LEN(TITLE1$): PRINT TITLE1$;
500 FOR X = 228 TO 288 STEP 2
510     LINE (X,190) - (X,199)
520 NEXT
530 LINE (228,190) - (288,190)'OUTLINE CODE BAR
540 LINE (228,199) - (288,199)
550 LOCATE 25,56-LEN(TITLE2$): PRINT TITLE2$;
560 FOR X = 452 TO 512 STEP 3
570     LINE (X,190) - (X,199)
580 NEXT
590 LINE (452,190) - (512,190)'OUTLINE CODE BAR
600 LINE (452,199) - (512,199)
610     '****************************** MAKE BARS ****************************
620 Y0 = 171
630 X1 = 71 + D/2       'PUT FIRST POINT HALFWAY ACROSS THE FIRST DIVISION
640 Y1 = INT((HI - VALUE2(1)) * RS + 12.5)
650 LINE (X1,Y1) - (X1,Y0)
660 LINE (X1+1,Y1) - (X1+1,Y0),0
670 LINE (X1+2,Y1) - (X1+2,Y0),0
680 FOR K = 2 TO N
690     X2 = X1 + D
700     Y2 = INT((HI - VALUE2(K)) * RS + 12.5)
710     SLOPE = (Y2-Y1) / (X2-X1)
720     INTERCEPT = Y1 + 1 - SLOPE * X1
730     FOR X = X1+3 TO X2 STEP 3
740         Y = SLOPE * X + INTERCEPT
750         LINE (X,Y) - (X,Y0)
760         Y = SLOPE * (X+1) + INTERCEPT
770         LINE (X+1,Y) - (X+1,Y0),0
780         Y = SLOPE * (X+2) + INTERCEPT
790         LINE (X+2,Y) - (X+2,Y0),0
800     NEXT
810     LINE (X1,Y1) - (X2,Y2)
820     X1 = X2
830     Y1 = Y2
840 NEXT
850 X1 = 71 + D/2
860 Y1 = INT((HI - VALUE1(1)) * RS + 12.5)
870 LINE (X1,Y1) - (X1,Y0)
880 LINE (X1+1,Y1) - (X1+1,Y0),0
890 FOR K = 2 TO N
900     X2 = X1 + D
910     Y2 = INT((HI - VALUE1(K)) * RS + 12.5)
920     SLOPE = (Y2 - Y1) / (X2 - X1)
930     INTERCEPT = Y1 + 1 - SLOPE * X1
940     FOR X = X1 TO X2
950         Y = SLOPE * X + INTERCEPT
960         LINE (X+1,Y) - (X+1,Y0),0
970     NEXT
980      FOR X = X1+2 TO X2 STEP 2
990         Y = SLOPE * X + INTERCEPT
1000        LINE (X,Y) - (X,Y0)
1010    NEXT
1020    LINE (X1,Y1) - (X2,Y2)
1030    X1 = X2
1040    Y1 = Y2
1050 NEXT
1060 IF INKEY$ = "" THEN 1060
1070 END
```

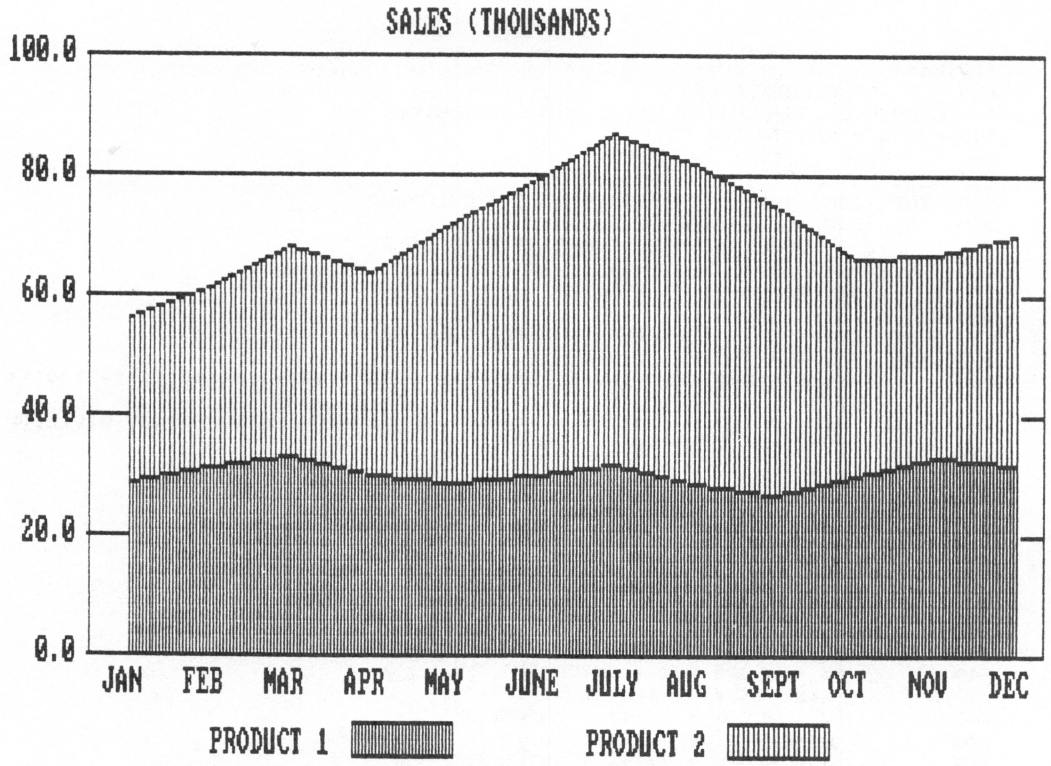


Figure 12–6 Cumulative surface chart produced by Prog. 12–6, with one data set plotted as the lower line and the other data set added to the first to obtain the upper line.

shown in Fig. 12–7. This band chart was produced by Prog. 12–7. A color band chart is shown in Fig. O of the color insert. We can use this technique to help in estimating magnitude differences. Two different shading patterns are used in this example to identify profits from losses (that is, when one curve falls below the other).

Program 12–7 Band chart, shading the area between two curves.

```
10 'PROGRAM 12-7. BAND CHART WITH TWO SETS OF DATA.
20      'PLOTS CURVES FOR TWO SETS OF DATA. SHADES IN AREAS WHERE CURVES
30      'CROSS. DESIGNED FOR SCREEN SIZE OF 640 x 200 PIXELS.
40  DIM L$(20), S1(20), S2(20)
50  SCREEN 2: CLS
60  PC = 8          'PC IS HORIZONTAL PIXELS PER CHARACTER
70      '********************* INPUT CHART LABELS AND DATA ***************
80  INPUT "TITLE OF CHART"; T$
90  INPUT "NAME OF FIRST SET OF DATA"; T1$
100 INPUT "NAME OF SECOND SET OF DATA"; T2$
```

Program 12–7 (cont.)

```
110 INPUT "NAME OF AREA WHEN FIRST DATA IS GREATER THAN SECOND"; T3$
120 INPUT "NAME OF AREA WHEN SECOND DATA IS GREATER THAN FIRST"; T4$
130 INPUT "NUMBER OF HORIZONTAL DIVISIONS"; N
140 D = INT(568 / N)     'D IS NUMBER OF PIXELS PER DIVISION
150 IF D/2 <> INT(D/2) THEN D = INT(D/2+.5)*2   'MAKE D AN EVEN NUMBER
160 PRINT "ENTER LABEL AND TWO VALUES FOR EACH DIVISION"
170 FOR K = 1 TO N
180     INPUT L$(K),S1(K),S2(K)
190     IF LEN(L$(K)) < D/PC THEN 220    'WILL THE LABEL FIT?
200     PRINT "LABEL TOO LONG. MAXIMUM LENGTH IS"; D/PC
210     INPUT "NEW LABEL"; L$(K): GOTO 190
220 NEXT
230 PRINT "ENTER MINIMUM AND MAXIMUM VALUES FOR VERTICAL AXIS"
240 INPUT LO, HI
250 R = HI - LO                          'R IS RANGE OF VALUES
260 RS = (172 - 12) / R                  'RS IS RATIO TO USE IN SCALING
270     '************************** DRAW GRID AND LABELS ********************
280 CLS
290 LINE (71,12) - (639,172),,B
300 P = 40 - LEN(T$) / 2          'CENTER THE TITLE
310 LOCATE 1,P: PRINT T$
320 ROW = 22
330 Y = 172
340 FOR K = 0 TO 5                'LABEL VERTICAL AXIS WITH SUCCESSIVE FIFTHS
350     L = LO + R * K / 5
360     LOCATE ROW,3: PRINT USING "####.#";L
370     LINE (70,Y) - (639,Y)
380     ROW = ROW - 4
390     Y = Y - 32
400 NEXT
410     'LABEL THE DIVISIONS
420 LOCATE 23,1
430 FOR K = 1 TO N
440     P = INT((71 + (K-1) * D + D/2) / PC - LEN(L$(K))/2 + .5) + 1
450     PRINT TAB(P);L$(K);
460 NEXT
470     'MAKE CODE AREAS
480 LOCATE 25,28-LEN(T3$): PRINT T3$;
490 FOR X = 228 TO 288 STEP 3
500     LINE (X,190) - (X,199)
510 NEXT
520 LOCATE 25,56-LEN(T4$): PRINT T4$;
530 FOR X = 452 TO 512 STEP 2
540     LINE (X,190) - (X,199)
550 NEXT
560     '****************************** MAKE BARS ***************************
570 Y0 = 171
580 X1 = 71 + D/2        'PUT FIRST POINT HALFWAY ACROSS THE FIRST DIVISION
590 Y1 = INT((HI - S1(1)) * RS + 12.5)
600 R1 = INT(Y1/PC + .5)
610 C1 = INT(X1/PC + .5)                 'X1,Y1 AND X2,Y2 ARE FIRST CURVE
620 X3 = X1                              'X3,Y3 AND X4,Y4 ARE OTHER CURVE
630 Y3 = INT((HI - S2(1)) * RS + 12.5)
640 R3 = INT(Y3/PC + .5)
650 C3 = INT(X3/PC + .5)
660 FOR K = 2 TO N
670     X2 = X1 + D
680     Y2 = INT((HI - S1(K)) * RS + 12.5)
690     M1 = (Y2-Y1) / (X2-X1)  'FIND SLOPE & INTERCEPT OF FIRST CURVE
```

Program 12–7 (cont.)

```
700     B1 = Y1 - M1 * X1
710     X4 = X2
720     Y4 = INT((HI - S2(K)) * RS + 12.5)
730     M3 = (Y4-Y3) / (X4-X3)  'FIND SLOPE & INTERCEPT OF OTHER CURVE
740     B3 = Y3 - M3 * X3
750     'SHADING PATTERN (EVERY OTHER LINE OR EVERY THIRD LINE) IS
760     'DETERMINED BY WHICH CURVE IS ON TOP. DRAW EVERY SECOND LINE
770     'WHEN 3-4 CURVE IS ON TOP, EVERY THIRD LINE WHEN 1-2 IS TOP.
780     IF Y3 <= Y1 AND Y4 <= Y2 THEN 890       '3-4 CURVE IS HIGHER THAN 1-2
790     IF Y3 > Y1 AND Y4 > Y2 THEN 910         '1-2 CURVE IS HIGHER THAN 3-4
800     'OTHERWISE THE TWO CURVES CROSS
810     XP = INT((B1-B3) / (M3-M1) + .5)     'FIND X INTERSECTION OF CURVES
820     IF Y3 > Y1 THEN 860
830     B = X3: E = XP: I = 2: GOSUB 1040        '3-4 CURVE IS ON TOP 'TIL XP
840     B = XP: E = X4: I = 3: GOSUB 1040        'FROM XP ON, 1-2 CURVE IS TOP
850     GOTO 920
860     B = X3: E = XP: I = 3: GOSUB 1040
870     B = XP: E = X4: I = 2: GOSUB 1040
880     GOTO 920
890     B = X3: E = X4: I = 2: GOSUB 1040
900     GOTO 920
910     B = X3: E = X4: I = 3: GOSUB 1040
920     LINE (X1,Y1) - (X2,Y2)
930     LINE (X3,Y3) - (X4,Y4)
940     X1 = X2
950     Y1 = Y2
960     X3 = X4
970     Y3 = Y4
980 NEXT K
990 LOCATE R1,C1: PRINT T1$;             'LABEL THE START OF THE CURVES
1000 LOCATE R3,C3: PRINT T2$;
1010 GOTO 1130
1020 '
1030 '################### SHADE IN AREA ############################
1040 FOR X = B TO E
1050     YF = M1 * X + B1
1060     YS = M3 * X + B3
1070     LINE (X,YF) - (X,YS),0
1080     IF X/I <> INT(X/I) THEN 1100
1090     LINE (X,YF) - (X,YS)
1100 NEXT
1110 RETURN
1120 '#############################################################
1130 IF INKEY$ = "" THEN 1130
1140 END
```

12–3 MULTIPLE FORMATS

A useful technique for comparing two sets of data is to plot the data in several formats within the same graph. This allows us to display various types of relationships between data sets. The combination bar chart, line graph, and pie chart shown in Fig. 12–8 was output by Prog. 12–8. Relative magnitude between the two data sets is displayed with the bar chart. Cumulative totals are shown with the line graph. The pie chart shows total percentage for each data set. Color coding is used to identify the two sets of data.

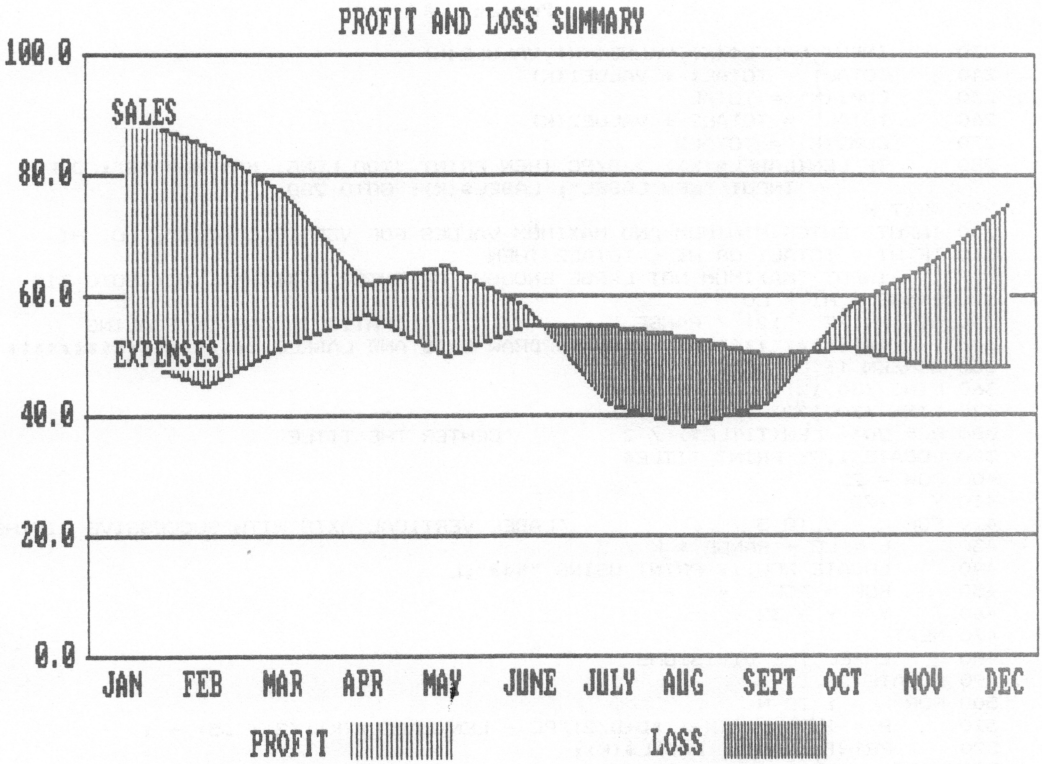


Figure 12–7 Band chart plotted by Prog. 12–7, illustrating the use of shading between data curves.

Program 12-8 Multiple formats: overlapping bar charts, cumulative line graphs, and pie chart.

```
10 'PROGRAM 12-8. TWO SETS OF DATA IN MULTIPLE FORMATS.
20      'INPUTS TWO SETS OF DATA. PLOTS DATA IN BAR GRAPH FORM, IN
30      'PIE CHART FORM TO SHOW PERCENTAGES OF TOTAL, AND IN CURVE
40      'FORM TO SHOW CUMULATIVE VALUES OVER TIME.
50      '*************************************************************************
60  DIM LABEL$(5), VALUE1(5), VALUE2(5), CUM1(5), CUM2(5)
70  SCREEN 0: WIDTH 80: CLS
80  PC = 8        'PC IS HORIZONTAL PIXELS PER CHARACTER
90  XC = 73       'XC,YC IS CENTER OF PIE CHART
100 YC = 40
110 RADIUS = 30
120 YADJUST = .9199999
130       '********************** INPUT CHART LABELS AND DATA ****************
140 INPUT "TITLE OF CHART"; TITLE$
150 INPUT "NAME OF FIRST SET OF DATA"; TITLE1$
160 INPUT "NAME OF SECOND SET OF DATA"; TITLE2$
170 INPUT "NUMBER OF DIVISIONS"; N
180 D = INT(290 / N)     'D IS NUMBER OF PIXELS PER DIVISION
190 TOTAL1 = 0              'TOTAL1 AND TOTAL2 ARE USED TO TOTAL DATA VALUES
200 TOTAL2 = 0
210 PRINT "ENTER LABEL AND TWO VALUES FOR EACH BAR"
220 FOR K = 1 TO N
```

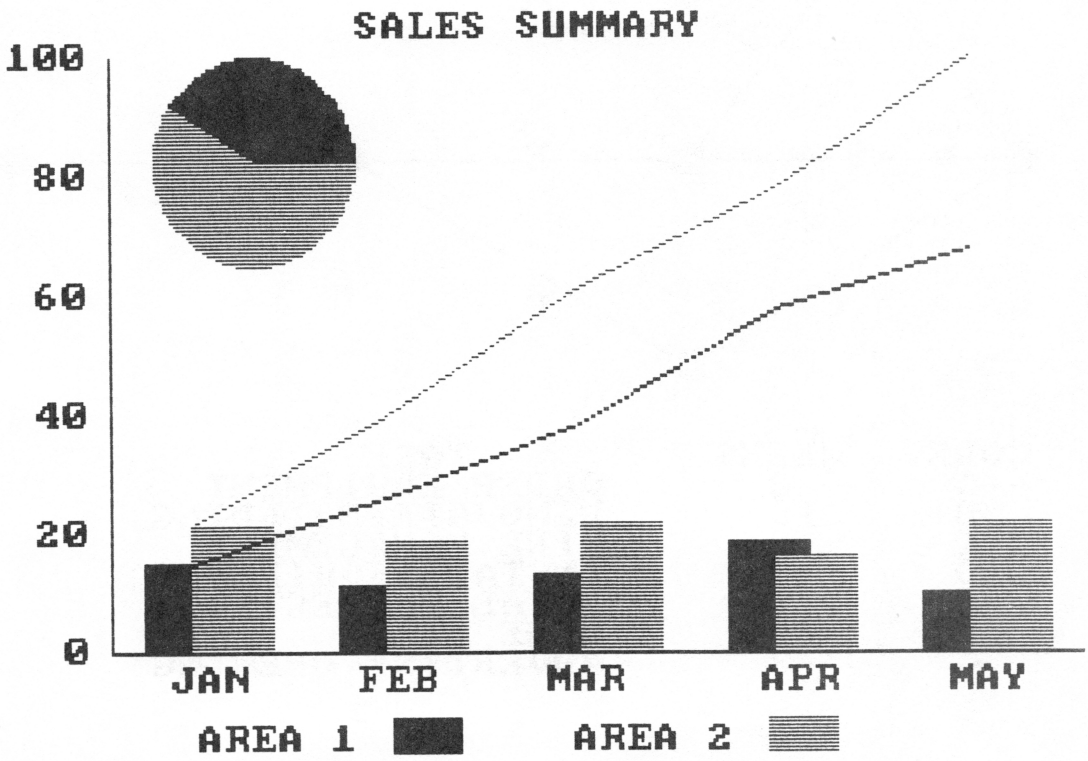
Program 12–8 (cont.)

```
230     INPUT LABEL$(K),VALUE1(K),VALUE2(K)
240     TOTAL1 = TOTAL1 + VALUE1(K)
250     CUM1(K) = TOTAL1
260     TOTAL2 = TOTAL2 + VALUE2(K)
270     CUM2(K) = TOTAL2
280     IF LEN(LABEL$(K)) > D/PC THEN PRINT "TOO LONG. MAXIMUM IS"; D/PC:
            INPUT "NEW LABEL"; LABEL$(K): GOTO 280
290 NEXT K
300 INPUT "ENTER MINIMUM AND MAXIMUM VALUES FOR VERTICAL AXIS"; LO, HI
310 IF HI < TOTAL1 OR HI < TOTAL2 THEN
        INPUT "MAXIMUM NOT LARGE ENOUGH. RE-ENTER MAXIMUM"; HI: GOTO 310
320 RANGE = HI - LO
330 RS = (172 - 12) / RANGE                    'RATIO TO USE IN SCALING
340     '************************** DRAW GRID AND LABELS ********************
350 SCREEN 1: COLOR 0,0: CLS
360 LINE (30,12) - (30,172)
370 LINE (30,172) - (319,172)
380 P = 20 - LEN(TITLE$) / 2          'CENTER THE TITLE
390 LOCATE 1,P: PRINT TITLE$
400 ROW = 22
410 Y = 172
420 FOR K = 0 TO 5                'LABEL VERTICAL AXIS WITH SUCCESSIVE FIFTHS
430     L = LO + RANGE * K / 5
440     LOCATE ROW,1: PRINT USING "###";L
450     ROW = ROW - 4
460     Y = Y - 32
470 NEXT
480     'LABEL THE DIVISIONS
490 LOCATE 23,1
500 FOR K = 1 TO N
510     P = INT((30+(K-1)*D+D/2)/PC - LEN(LABEL$(K))/2 + .5) + 1
520     PRINT TAB(P); LABEL$(K);
530 NEXT
540     'MAKE CODE AREAS
550 LOCATE 25,14-LEN(TITLE1$): PRINT TITLE1$;
560 LINE (114,190) - (134,199),3,BF
570 LOCATE 25,28-LEN(TITLE2$): PRINT TITLE2$;
580 LINE (226,190) - (246,199),2,BF
590     '
600     '***************************** MAKE BARS ****************************
610 YO = 171
620 X = 30 + D/6
630 FOR K = 1 TO N
640     Y = INT((HI - VALUE1(K)) * RS + 12.5)
650     LINE (X,Y) - (X+D*5/12,YO),3,BF
660     X1 = X + D * 3/12          'START SECOND BAR 3/12 OVER FROM FIRST
670     Y = INT((HI - VALUE2(K)) * RS + 12.5)
680     LINE (X1,Y) - (X1+D*5/12,YO),2,BF
690     X = X + D
700 NEXT K
710     '
720     '***************************** MAKE LINES ****************************
730 X = 30 + D * 5 / 12
740 Y1 = INT((HI - CUM1(1)) * RS + 12.5)
750 Y2 = INT((HI - CUM2(1)) * RS + 12.5)
760 FOR K = 2 TO N
770     Y3 = INT((HI - CUM1(K)) * RS + 12.5)
780     Y4 = INT((HI - CUM2(K)) * RS + 12.5)
790     LINE (X,Y1) - (X+D,Y3),3
800     LINE (X,Y2) - (X+D,Y4),2
810     X = X + D
```

Program 12–8 (cont.)

```
820      Y1 = Y3
830      Y2 = Y4
840 NEXT K
850     '
860     '************************** MAKE PIE CHART ************************
870 ALTOGETHER = TOTAL1 + TOTAL2          'TOTAL OF BOTH DATA SETS
880      'ANGLE IS PERCENTAGE OF CIRCLE (IN RADIANS) CORRESPONDING TO TOTAL1
890 BEFORE = 0
900 ANGLE = 6.28318 * TOTAL1 / ALTOGETHER
910 CIRCLE (XC,YC),RADIUS,3,-BEFORE,-ANGLE,YADJUST
920 INTANGLE = -ANGLE/2
930 XINTERIOR = XC + RADIUS/2 * COS(INTANGLE)
940 YINTERIOR = YC + RADIUS/2 * SIN(INTANGLE) * YADJUST
950 PAINT (XINTERIOR,YINTERIOR),3,3
960 CIRCLE (XC,YC),RADIUS,2,-ANGLE,-6.28318,YADJUST
970 INTANGLE = (6.28318 - ANGLE) / 2
980 XINTERIOR = XC + RADIUS/2 * COS(INTANGLE)
990 YINTERIOR = YC + RADIUS/2 * SIN(INTANGLE) * YADJUST
1000 PAINT (XINTERIOR,YINTERIOR),2,2
1010 IF INKEY$ = "" THEN 1010
1020 END
```

Figure 12–8 A multiple format graph produced by Prog. 12–8, providing several types of comparative information.

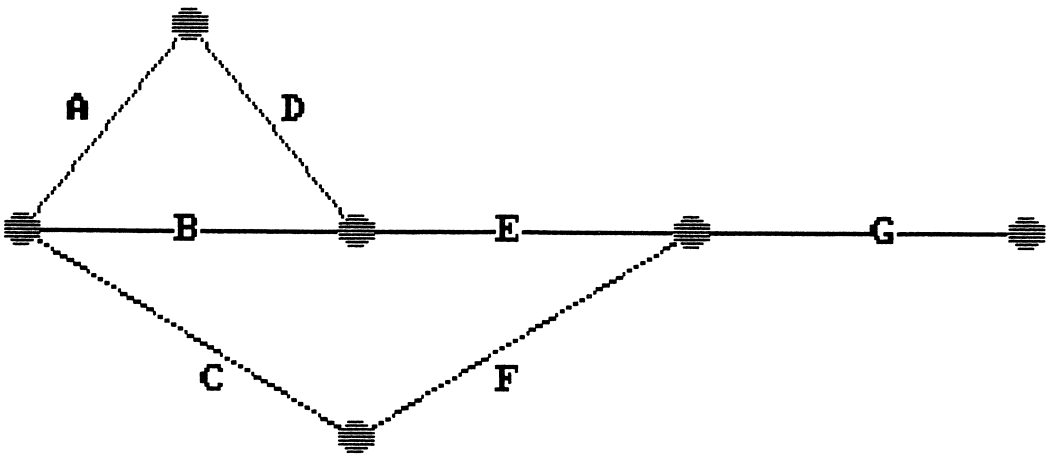


12–4 PROJECT MANAGEMENT GRAPHS

Graphs displaying a network of project tasks, as in Fig. 12–9, can be used as an aid in scheduling and monitoring the tasks. This type of graph is particularly useful with PERT-CPM project scheduling techniques. Tasks are represented in the network graph by lines and ordered from left to right, according to when they can be started. Circles are used to indicate the beginning and ending of tasks. The leftmost circle shows the start of the project, where tasks A, B, and C can be initiated simultaneously. Task D cannot be started until task A is completed, and task E must wait for tasks A, B, and D.

Project tasks can also be listed on a time chart to show actual starting and ending dates. The project time chart shown in Fig. 12–10 was produced by Prog. 12–9. This chart shows relative starting and ending weeks for each task in the project. Tasks are represented as horizontal bars, with the length of each bar proportional to the scheduled task time. Both network and time charts are useful for planning and managing projects.

Figure 12–9 Network graph showing the sequencing of various tasks in a project.



CODE	WEEKS	TASK
A	2	ORDER EQUIPMENT
B	16	RENOVATE BUILDING
C	8	HIRE MANAGEMENT
D	8	INSTALL EQUIPMENT
E	3	LOCAL INSPECTIONS
F	4	HIRE STAFF
G	3	ADVERTISE OPENING

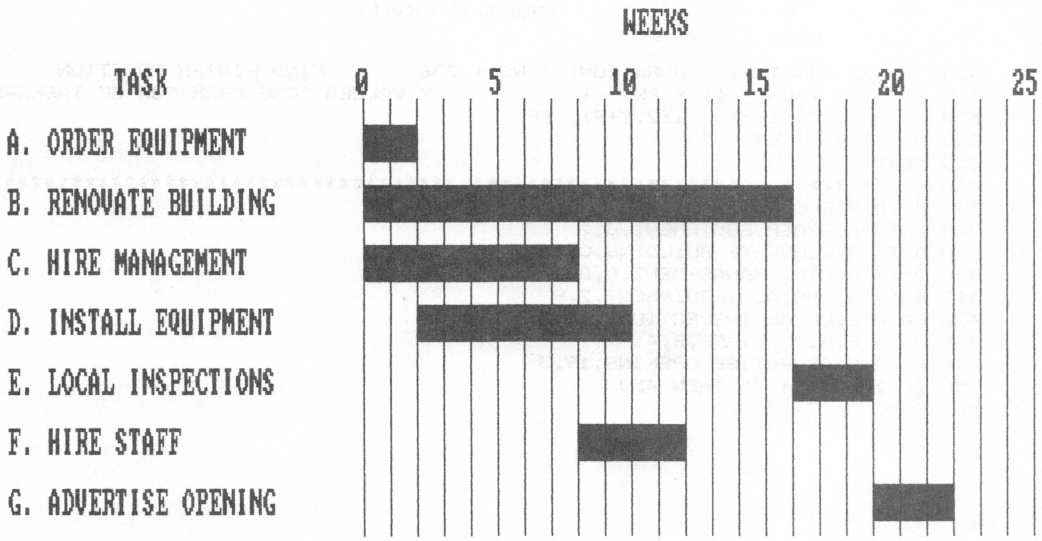


Figure 12–10 Time chart, produced by Prog. 12–9, displaying the starting and ending dates for scheduled project tasks.

Program 12–9 Time chart for scheduling tasks.

```
10 'PROGRAM 12-9. TIME DIAGRAM
20      'TIME BARS ARE SCALED TO PIXELS 236 - 636
30  SCREEN 2: CLS
40  PC = 8                  'PC IS NUMBER OF HORIZONTAL PIXELS PER CHARACTER
50  PR = 8                  'PR IS NUMBER OF VERTICAL PIXELS PER CHARACTER
60  READ TIMEDIVISION$
70  READ FIRSTTIME,LASTTIME
80  RANGETIME = LASTTIME - FIRSTTIME
90  RS = (636 - 236) / RANGETIME
100 LOCATE 2,50: PRINT TIMEDIVISION$;
110 LOCATE 4,12: PRINT "TASK";
120 COLUMN = 28
130 FOR K = 0 TO 5          'LABEL COLUMNS IN SUCCESSIVE FIFTHS OF TIME RANGE
140     T = FIRSTTIME + RANGETIME * K/5
150     LOCATE 4,COLUMN: PRINT USING "###";T;
160     COLUMN = COLUMN + 10
170 NEXT
180 INCREMENT = 400/25
190 X = 236
200 FOR L = 1 TO 26                 '26 LINES IN ALL
210     LINE (X,32) - (X,148)
220     X = X + INCREMENT
230 NEXT
240 ROW = 6
250 FOR K = 1 TO 7
260     READ CODE$, TASK$, START, DURATION
270     LOCATE ROW,4
280     PRINT CODE$; ". "; TASK$
290     X1 = START * RS + 236           'FIND START POSITION OF BAR
```

Program 12–9 (cont.)

```
300     X2 = (START + DURATION) * RS + 236      'FIND FINISH POSITION
310     Y = (ROW - 1) * PR - 1          'Y VALUES COME FROM ROW OF TASKNAME
320     LINE (X1,Y) - (X2,Y+9),,BF
340     ROW = ROW + 2
350 NEXT
360     '*************************************************************************
370 DATA WEEKS,0,25
380 DATA A,ORDER EQUIPMENT,0,2
390 DATA B,RENOVATE BUILDING,0,16
400 DATA C,HIRE MANAGEMENT,0,8
410 DATA D,INSTALL EQUIPMENT,2,8
420 DATA E,LOCAL INSPECTIONS,16,3
430 DATA F,HIRE STAFF,8,4
440 DATA G,ADVERTISE OPENING,19,3
450 IF INKEY$ = "" THEN 450
460 END
```

Chapter 13

Educational Graphics

The availability of low-cost microcomputer graphics systems provides a powerful educational resource at all levels, from grade schools to graduate schools. We can develop graphics programs for classroom demonstrations or for self-study projects in a lab. Such computer-assisted instruction (CAI) programs can be broadly classified as either drill and practice programs, tutorial and inquiry programs, or as simulations.

13–1 DRILL AND PRACTICE PROGRAMS

With drill and practice programs, we can repeatedly present practice problems on a video screen and ask for the answers. The problems could be questions about sentence structure, foreign languages, historical personalities, art forms, or geological eras. Answers could be chosen from a menu or typed in. We can design a drill and practice program to respond to an answer by displaying a simple message, such as "THAT'S RIGHT" or "THAT'S WRONG. TRY AGAIN." Usually, it is better to have the program be a bit more helpful by furnishing additional information when a wrong answer is given. A chemistry drill on the periodic table might ask for the atomic number of a randomly selected element and respond to a wrong answer by citing which element, if any, has the stated atomic number. More elaborate programs could respond to a wrong answer with a series of "leading" questions or by displaying pictures and text to help in getting a right answer.

Graphics displays can aid in the statement of many drill and practice problems. A spelling drill could draw a picture of the object to be spelled (car, boat, tree, house), and questions for an economics class could use graphs and

charts. Simply to make a more entertaining display, we can add pictures to accompany the statement of a problem or as part of the response to an answer. Program 13–1, an addition drill, uses pictures both for entertainment and as visual aids in the statement of the problem presented. Figure 13–1 shows the possible

Program 13–1 Arithmetic practice, presenting additional problems with prompts and pictures.

```
10 'PROGRAM 13-1. ARITHMETIC PRACTICE.
20      'USING RANDOM NUMBER FUNCTION, GENERATES ADDITION
30      'PROBLEMS AND PRESENTS THEM IN TEXT FORM (3 + 4 = ?).
40      'PRESENTS THE PROBLEM AS CIRCLES TO COUNT IF TWO
50      'INCORRECT RESPONSES ARE GIVEN. IF AN INCORRECT
60      'RESPONSE IS STILL GIVEN (TWICE MORE), WE GO ON TO
70      'ANOTHER PROBLEM. A COUNT (R) IS KEPT OF THE NUMBER OF
80      'PROBLEMS ANSWERED CORRECTLY ON THE FIRST TRY (WHEN
90      'C = 1). AFTER 5 PROBLEMS, A SMILEY FACE AND MESSAGE ARE
100     'DISPLAYED IF R IS GREATER THAN 3 (4 OR 5 RIGHT OUT OF 5).
110     'AFTER TEN PROBLEMS, A SECOND DISPLAY AND MESSAGE OCCURS
120     'IF R IS GREATER THAN 8.
130 '*************************************************************************
140 SCREEN 1: COLOR 0,0: CLS
150 INPUT "HI!  WHAT'S YOUR NAME"; N$
160 PRINT "OKAY, "; N$; ", HERE WE GO!"
170 PRINT "SEE HOW MANY PROBLEMS YOU CAN GET RIGHT!"
180 FOR DELAY = 1 TO 2000: NEXT
190     '******************** GENERATE 10 PROBLEMS ********************
200 SEED = VAL(RIGHT$(TIME$,2))
210 FIRSTRAND = RND(-SEED)
220 FOR P = 1 TO 10
230     C = 0           'C IS COUNT OF HOW MANY ANSWERS GIVEN TO THIS PROBLEM
240     J = INT(RND * 9 + .5)
250     K = INT(RND * 9 + .5)
260     CLS
270     LOCATE 16,8: PRINT J;"+";K;"= ";
280     INPUT A
290     C = C + 1
300     IF A = J + K THEN 450           'RIGHT?
310     IF C <> 1 THEN 360
320     LOCATE 18,5: PRINT "THAT'S NOT RIGHT, ";N$      'ANSWERED ONCE, DO OVER
330     PRINT "    TRY AGAIN  "
340     FOR DELAY = 1 TO 2000: NEXT
350     GOTO 260
360     IF C <> 2 THEN 390
370     GOSUB 530               'HAS ANSWERED TWICE. GO TO DRAWING CIRCLES
380     GOTO 270
390     IF C <> 3 THEN 430      'HAS ANSWERED THREE TIMES. ONE MORE TRY
400     GOSUB 530
410     LOCATE 4,1: PRINT "TRY ONE MORE TIME";
420     GOTO 270
430     LOCATE 20,1: PRINT "LET'S TRY ANOTHER";
440     FOR DELAY = 1 TO 2000: NEXT
450     IF C = 1 THEN RIGHT = RIGHT + 1         'RIGHT ON FIRST TRY
460     IF P = 5 AND RIGHT > 3 THEN GOSUB 680           'DRAW SMILEY FACE
470     IF P = 10 AND RIGHT > 8 THEN GOSUB 780          'DRAW BALLOONS
480 NEXT
490 CLS
500 LOCATE 10,19: PRINT "BYE,"
510 LOCATE 12,20 - LEN(N$) / 2: PRINT N$
520 GOTO 1050
```

Program 13–1 (cont.)

```
530 '#################### DRAW CIRCLES TO HELP GET ANSWER ##################
540 CLS
550 Y = 76: X = 8
560 FOR W = 1 TO J
570     CIRCLE (X,Y),5,2
580     X = X + 16
590 NEXT
600 LOCATE 10,X/8+1: PRINT "+";
610 X = X + 20
620 FOR W = 1 TO K
630     CIRCLE (X,Y),5,2
640     X = X + 16
650 NEXT
660 RETURN
670 '############################### SMILEY FACE ############################
680 COLOR 0,0: CLS
690 LOCATE 25,4: PRINT "SO FAR, SO GOOD, "; N$
700 CIRCLE (160,100),80,2,,,.9199999
710 CIRCLE (130,80),5,2,,,.9199999
720 CIRCLE (190,80),5,2,,,.9199999
730 CIRCLE (160,100),3,2,,,.9199999
740 CIRCLE (160,80),60,2,3.92,5.55,.9899999
750 FOR DELAY = 1 TO 2000: NEXT
760 RETURN
770 '################################ BALLOONS ##############################
780 CLS
790 LOCATE 4,17: PRINT N$
800 LOCATE 6,18: PRINT "THE"
810 LOCATE 8,17: PRINT "GREAT!";
820 CIRCLE (60,50),40,1
830 CIRCLE (60,88),5,1
840 LINE (60,91) - (60,170),3          'PUT STRING ON BALLOON
850 CIRCLE (280,50),30,1
860 CIRCLE (280,78),3,1
870 LINE (280,79) - (280,180),3
880 CIRCLE (140,130),35,2
890 CIRCLE (140,163),3,2
900 LINE (140,164)-(140,199),3
910 FOR Y = 2 TO 199 STEP 5            'MAKE CONFETTI
920     X = RND(1) * 319
930     CIRCLE (X,Y),1,1
940     FOR DELAY = 1 TO 50: NEXT
950 NEXT
960 X = 240: Y = 20: A = 0: R = 20: DA = 1/R/.5
970 FOR Y = 20 TO 199                  'MAKE SPIRAL
980    XP = X + R * COS(A)
990    YP = Y + R * SIN(A)
1000    PSET (XP,YP),2
1010    A = A + DA
1020 NEXT Y
1030 RETURN
1040 '#########################################################################
1050 END
```

program outputs for the question 4 + 5 = ?. To provide some variety in the program responses, we could randomly choose different face designs and text phrases (GOOD, SWELL, NO, TOO BAD). We could also select a different-shaped object for the problem statement each time.

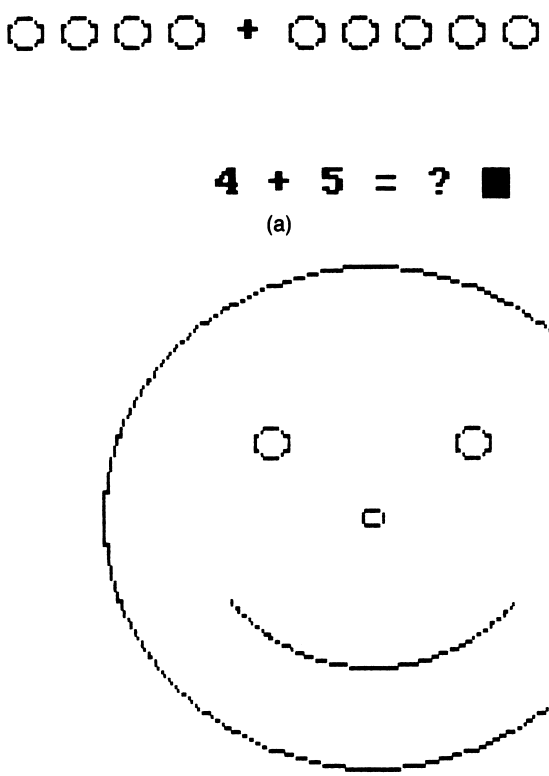
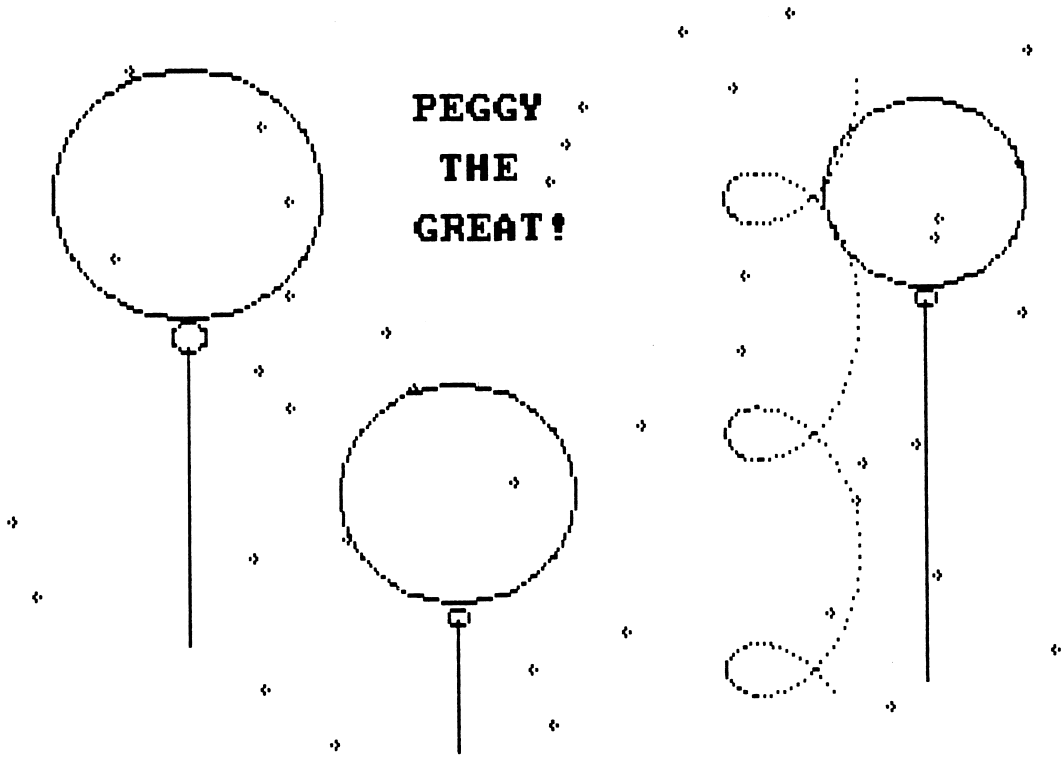


Figure 13–1 An arithmetic drill output from Prog. 13–1, displaying (a) prompts when a wrong answer is given, (b) a happy face for a series of right answers, and (c) balloons and streamers for a good final score.



(c)

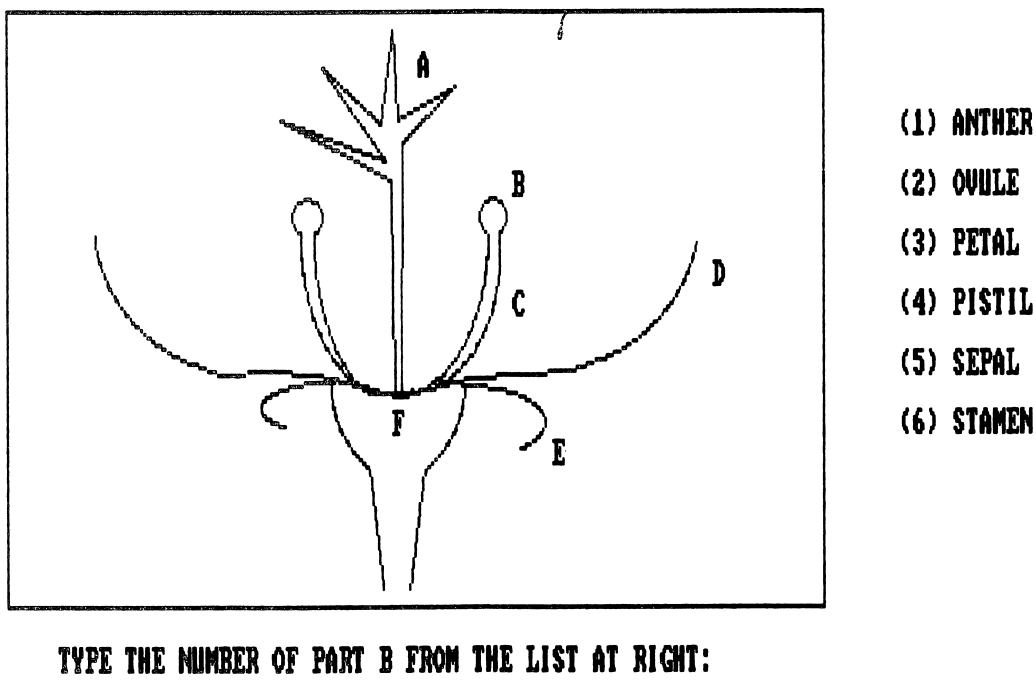
Figure 13–1 (cont.)

13–2 TUTORIAL AND INQUIRY PROGRAMS

In tutorial and inquiry programs, we employ more extensive conversational methods. Tutorials can provide instructions to read and study certain materials (books, articles, films) for a self-paced course, then give a test over that material. Inquiry programs can carry on a conversation by both answering and asking questions. For example, an inquiry program can help train medical interns by responding to questions as if the program were a patient. The program's answers could then be used to make a medical diagnosis. Exams and questions provided by inquiry and tutorial programs can use graphics in much the same way as drill and practice programs.

For a self-study type of program we might produce a diagram of plant parts and a menu of terms, as in Fig. 13–2. The program could produce several such displays, evaluate the responses, and output a grade for the total performance. We can use this type of display as part of an examination program or for review and study in a tutorial or inquiry program.

Figure 13–2 Picture displayed as part of a self-study program.



13–3 SIMULATION PROGRAMS

We use simulation programs to demonstrate the behavior of various types of systems. An actual physical or biological model (or a hypothetical system) can be displayed in a lecture demonstration or as part of an individual study program. Simulation programs are particularly effective for studying systems with many parameters or systems that we cannot actually observe. Models of geopolitical systems, atomic and molecular structures, or relativistic motion of objects are examples of such systems. We can vary the parameters involved in these systems and watch the changes occur on the screen.

In many graphics simulations, we want to demonstrate complex motions, as in Prog. 13–2. Here, we produce an animated model of the solar system. Figure 13–3 shows several positions along the paths of motion of the moon and earth as they rotate about the sun. The actual output of Prog. 13–2 presents only one

Program 13–2 Simulation: modeling the solar system with rotating moon and earth.

```
10 'PROGRAM 13-2. SOLAR SYSTEM.
20      'ILLUSTRATES MOTION OF SOLAR SYSTEM. MOON REVOLVES AROUND
30      'EARTH 12 TIMES FOR EVERY ONE REVOLUTION OF THE EARTH
40      'AROUND THE SUN. THE MOON'S ORBIT IS CHOSEN TO BE ONLY
50      'ONE-SIXTH AS LARGE AS THE EARTH'S -- SO THE MOON'S
60      'ANGULAR INCREMENT (1 / RADIUS) IS 6 TIMES AS LARGE.
70      'USING STEP 1/EARTHORBIT AND 1/MOONORBIT, THE MOON WOULD TRAVEL
80      'THROUGH ITS ORBIT SIX TIMES AS FAST AS THE EARTH. WE PLOT
90      'A NEW POSITION FOR THE MOON TWICE FOR EVERY POSITION OF THE
100     'EARTH, SO THE MOON TRAVELS 12 TIMES AS FAST.
110     '*****************************************************************
120 SCREEN 1: COLOR 0,0: CLS
130 XSUN = 160
140 YSUN = 100
150 EARTHORBIT = 84      'EARTHORBIT IS RADIUS OF EARTH'S ORBIT
160 MOONORBIT = 14       'MOONORBIT IS RADIUS OF MOON'S ORBIT
170 SUN = 20
180 EARTH = 7
190 MOON = 4
200 CIRCLE (XSUN,YSUN),SUN,3                              'DRAW SUN
210 PAINT (XSUN,YSUN),SUN,3
220 MOONPOSITION = 1/MOONORBIT            'ANGULAR INCREMENT FOR MOON
230 FOR EARTHPOSITION = 1/EARTHORBIT TO 6.28318 STEP 1/EARTHORBIT
240     PAINT (XEARTH,YEARTH),0,0                         'ERASE EARTH
250     XEARTH = XSUN + EARTHORBIT * COS(EARTHPOSITION)
260     YEARTH = YSUN + EARTHORBIT * SIN(EARTHPOSITION)
270     CIRCLE (XEARTH,YEARTH),EARTH,1                    'DRAW EARTH
280     PAINT (XEARTH,YEARTH),1,1
290     PAINT (XMOON,YMOON),0,0                           'ERASE MOON
300     XMOON = XEARTH + MOONORBIT * COS(MOONPOSITION)
310     YMOON = YEARTH + MOONORBIT * SIN(MOONPOSITION)
320     CIRCLE (XMOON,YMOON),MOON,2                       'DRAW MOON
330     PAINT (XMOON,YMOON),2,2
340     MOONPOSITION = MOONPOSITION + 1/MOONORBIT
350     PAINT (XMOON,YMOON),0,0                           'ERASE MOON
360     XMOON = XEARTH + MOONORBIT * COS(MOONPOSITION)
```

Program 13–2 (cont.)

```
370     YMOON = YEARTH + MOONORBIT * SIN(MOONPOSITION)
380     CIRCLE (XMOON,YMOON),MOON,2                    'DRAW MOON
390     PAINT (XMOON,YMOON),2,2
400     MOONPOSITION = MOONPOSITION + 1/MOONORBIT
410 NEXT
420 GOTO 220
430 END
```

position at a time, erasing previous positions. Circular paths are used, and the objects are not drawn to scale.

Simulation programs can be designed as educational games. A gunnery game that plots the trajectory to a target, based on choices for projection angle, can be an effective learning program. This program can help in grasping the geometrical meaning of angular values, as well as demonstrating the relation between trajectory and angle of projection.

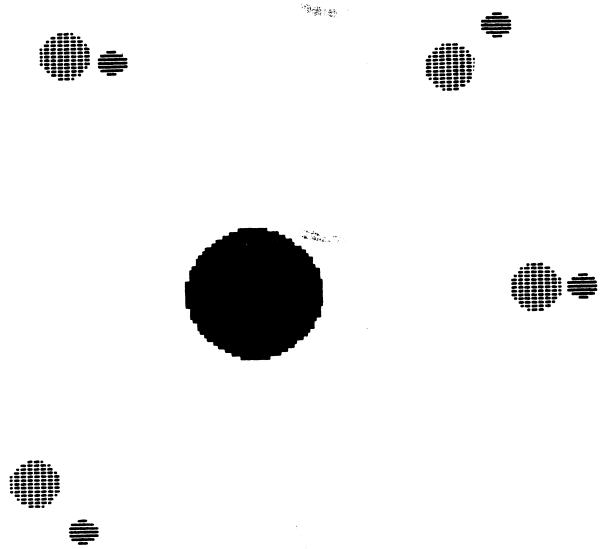


Figure 13–3 Simulation of the motion of the earth and moon about the sun, showing several positions from the output of Prog. 13–2.

13–4 COMPUTER-MANAGED INSTRUCTION

Graphics programs can be used as aids in record keeping and grading. Computer-managed instruction (CMI) programs can be designed to maintain records of grades, calculate grades, and provide statistics. We can use CMI programs to output graphs of grading distributions for a single exam, for a single course, for a particular course over several years, or for all courses taught during any time interval.

Chapter 14

Personal Graphics

For our final look at applications, we will discuss a few ways that we can put graphics to work for us personally. We can create personal graphics programs for recreation, education, or profit.

14–1 HOUSEHOLD GRAPHICS

There are many kinds of financial applications of graphics that can be useful for our home use. We can plot our various expenses or interest payments, analyze returns on investments, or graph potential savings due to energy conservation improvements to the home. Program 14–1 is an example of a monthly budget

Program 14–1 Household budget bar chart.

```
10 'PROGRAM 14-1. HOUSEHOLD BUDGET SUMMARY
20      'PLOTS A BAR GRAPH OF MONTHLY HOUSEHOLD EXPENSES.
30      'BARS PLOTTED ARE THE PERCENTAGE OF THE TOTAL
40      'EXPENSES FOR THE MONTH. ALSO PRINTS THE ACTUAL
50      'EXPENSES IN EACH CATEGORY -- FOOD, CLOTHING,
60      'HOUSING, AND RECREATION.
70  SCREEN 0: WIDTH 80: COLOR 7,0,0: CLS
80      '************************** INPUT DATA **************************
90  D = INT(290 / 4)                'FOUR CATEGORIES OF EXPENSES
100 RS = (172 - 12)                 'RS IS RATIO TO USE IN SCALING
110 TOTAL = 0                       'T IS TOTAL OF ALL EXPENSES
120 INPUT "ENTER MONTH NAME"; MONTH$
130 PRINT
140 PRINT "F -FOOD       H -HOUSING & UTILITIES    C -CLOTHING    R -RECREATION"
150 PRINT: PRINT "ENTER 0,Q TO QUIT"
160 INPUT "EXPENSE AND CATEGORY CODE"; EXPENSE, CODE$
170 IF EXPENSE = 0 AND CODE$ = "Q" THEN 270
```

Program 14–1 (cont.)

```
180 IF CODE$ = "F" OR CODE$ = "H" OR CODE$ = "C" OR CODE$ = "R" THEN 200
190 PRINT "INCORRECT EXPENSE CODE": GOTO 160
200 TOTAL = TOTAL + EXPENSE
210     'ADD UP EXPENSES FOR EACH CATEGORY
220 IF CODE$ = "F" THEN FOODTOTAL = FOODTOTAL + EXPENSE
230 IF CODE$ = "H" THEN HOUSETOTAL = HOUSETOTAL + EXPENSE
240 IF CODE$ = "C" THEN CLOTHINGTOTAL = CLOTHINGTOTAL + EXPENSE
250 IF CODE$ = "R" THEN RECTOTAL = RECTOTAL + EXPENSE
260 GOTO 160
270     '************************* DRAW GRID AND LABELS ********************
280 SCREEN 1: COLOR 0,0: CLS
290 LINE (25,12) - (25,172)
300 LINE (25,172) - (315,172)
310 P = 20 - LEN(MONTH$) / 2            'CENTER THE MONTH
320 LOCATE 1,P: PRINT MONTH$
330 ROW = 22
340 Y = 172
350 FOR K = 0 TO 4                  'LABEL VERTICAL AXIS WITH SUCCESSIVE FIFTHS
360     L = LO + 100 * K / 5
370     LOCATE ROW,1: PRINT USING "###";L
380     ROW = ROW - 4
390     Y = Y - 32
400 NEXT
410 LOCATE 2,1: PRINT "PERCENT";
420    'LABEL THE DIVISIONS
430 LOCATE 23,1: PRINT "      FOOD      HOUSE     CLOTHES  LEISURE";
440     '***************************** MAKE BARS ***************************
450 YO = 171
460 X = 25 + D/4        'START FIRST BAR 1/4 OVER IN FIRST DIVISION
470 Y = INT((1 - FOODTOTAL/TOTAL) * RS + 12.5)      'EACH EXPENSE CATEGORY IS
480 GOSUB 600                                       'WHAT % OF TOTAL? SUBTRACT
490 Y = INT((1 - HOUSETOTAL/TOTAL) * RS + 12.5)     'FROM 1 (SO BARS WILL GO UP
500 GOSUB 600                                       'AND MULTIPLY BY RANGE OF
510 Y = INT((1 - CLOTHINGTOTAL/TOTAL) * RS + 12.5)      'PIXELS USED FOR BARS
520 GOSUB 600
530 Y = INT((1 - RECTOTAL/TOTAL) * RS + 12.5)
540 GOSUB 600
550 LOCATE 24,1
560 PRINT USING " $#####.##   ####.##   ####.##   ####.##"; FOODTOTAL;
        HOUSETOTAL; CLOTHINGTOTAL; RECTOTAL;
570 LOCATE 3,24: PRINT "TOTAL EXPENSES";
580 LOCATE 4,30: PRINT USING "$#####.##"; TOTAL
590 GOTO 640
600     'MAKE BAR
610 LINE (X,Y) - (X+D/2,YO),3,BF
620 X = X + D
630 RETURN
640 IF INKEY$ = "" THEN 640
650 END
```

program to compare expenses. We have included only four categories of expenses in this example (Fig. 14–1). The bars are drawn for this graph to show percentage of expenditures in each category for one month. Many other types of expense graphs could be plotted. We could also accumulate weekly and monthly expenses in a data file. Then a budget program could be used to compare expenses over several months or years.

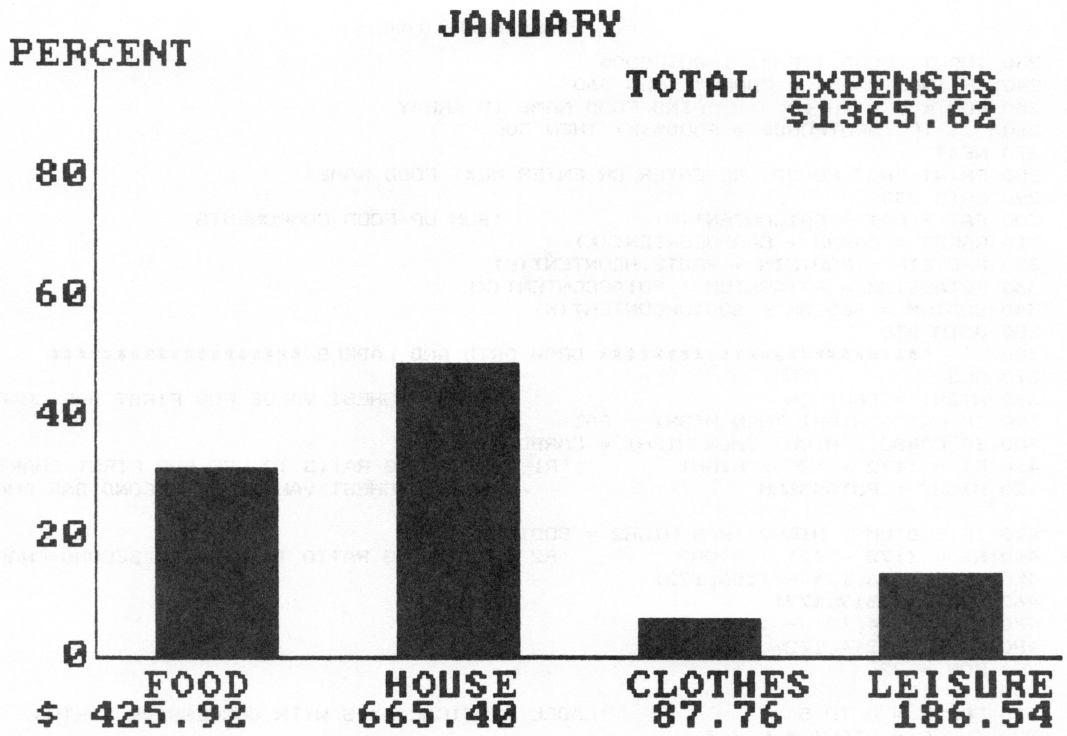


Figure 14–1 A display of household expenses by Prog. 14–1.

Program 14–2 Nutrition graph, plotting calories and nutrients in two bar charts and a pie chart.

```
10 'PROGRAM 14-2. NUTRITION ANALYSIS.
20      'READS FOOD NAMES & RELATED QUANTITIES FROM DATA STATEMENTS
30      'INTO ARRAYS. READS THE NUMBER OF GRAMS OF FAT,
40      'CARBOHYDRATES, PROTEIN, POTASSIUM, AND SODIUM FOR
50      'AN AVERAGE SIZE SERVING. WE ENTER THE FOOD NAMES WE'VE
60      'EATEN FOR THE DAY. GRAMS OF FAT, PROTEIN, ETC. ARE ADDED
70      'FOR ALL THE DIFFERENT FOODS. GRAMS OF PROTEIN, FAT, AND
80      'CARBOHYDRATE ARE CONVERTED TO TOTAL NUMBER OF CALORIES
90      'CONSUMED. A PIE CHART SHOWING PERCENT OF TOTAL CALORIES
100     'FROM CARBOHYDRATE, FAT, AND PROTEIN SOURCES IS DISPLAYED
110     'ALSO DRAWS TWO BAR CHARTS -- ONE SHOWING NUMBER GRAMS OF
120     'FAT, CARBOHYDRATE, AND PROTEIN; SECOND SHOWS NUMBER OF
130     'GRAMS OF POTASSIUM AND SODIUM CONSUMED.
140     '************************************************************
150 SCREEN 1: COLOR 0,0: CLS
160 DIM FOOD$(10),FATCONTENT(10),CARBOCONTENT(10),PROTEINCONTENT(10),
      POTASCONTENT(10),SODIUMCONTENT(10)
170 YADJUST = .9199999
180 FOR K = 1 TO 10
190       READ FOOD$(K),FATCONTENT(K),CARBOCONTENT(K),PROTEINCONTENT(K),
            POTASCONTENT(K),SODIUMCONTENT(K)
200 NEXT
210     '********************** INPUT CHART LABELS AND DATA ***************
220 PRINT "ENTER DONE TO QUIT"
```

Program 14–2 (cont.)

```
230 INPUT "FOOD ITEM"; INPUTFOOD$
240 IF INPUTFOOD$ = "DONE" THEN 360
250 FOR K = 1 TO 10      'FIND FOOD NAME IN ARRAY
260     IF INPUTFOOD$ = FOOD$(K) THEN 300
270 NEXT
280 PRINT "NOT FOUND. RE-ENTER OR ENTER NEXT FOOD NAME"
290 GOTO 230
300 FAT = FAT + FATCONTENT(K)            'SUM UP FOOD COMPONENTS
310 CARBO = CARBO + CARBOCONTENT(K)
320 PROTEIN = PROTEIN + PROTEINCONTENT(K)
330 POTASSIUM = POTASSIUM + POTASCONTENT(K)
340 SODIUM = SODIUM + SODIUMCONTENT(K)
350 GOTO 230
360     '************************* DRAW GRID AND LABELS ********************
370 CLS
380 HIGH1 = PROTEIN                      'FIND HIGHEST VALUE FOR FIRST BAR CHART
390 IF FAT > HIGH1 THEN HIGH1 = FAT
400 IF CARBO > HIGH1 THEN HIGH1 = CARBO
410 R1 = (172 - 12) / HIGH1          'R1 IS SCALING RATIO TO USE FOR FIRST CHART
420 HIGH2 = POTASSIUM                    'FIND HIGHEST VALUE FOR SECOND BAR CHART

430 IF SODIUM > HIGH2 THEN HIGH2 = SODIUM
440 R2 = (172 - 12) / HIGH2          'R2 IS SCALING RATIO TO USE FOR SECOND CHART
450 LINE (256,12) - (256,172)
460 LINE - (319,172)
470 LINE (144,12) - (144,172)
480 LINE - (214,172)
490 ROW = 22
500 Y = 172
510 FOR K = 0 TO 5               'LABEL VERTICAL AXIS WITH SUCCESSIVE FIFTHS
520     L = HIGH1 * K / 5
530     LOCATE ROW,17: PRINT USING "###";L;
540     L = HIGH2 * K / 5
550     LOCATE ROW,31: PRINT USING "#.##";L;
560     ROW = ROW - 4
570     Y = Y - 32
580 NEXT
590 LOCATE 1,18: PRINT "GRAMS       GRAMS";
600    'LABEL THE DIVISIONS
610 LOCATE 23,21: PRINT "F C P          P S";
620 LOCATE 24,21: PRINT "A A R          O O";
630 LOCATE 25,21: PRINT "T R O          T D";
640 '
650     '****************************** MAKE BARS ****************************
660 YO = 171
670 X = 160
680 Y = INT((HIGH1 - FAT) * R1 + 12.5)
690 C = 3: GOSUB 800                                  'GO MAKE BAR
700 Y = INT((HIGH1 - CARBO) * R1 + 12.5)
710 C = 1: GOSUB 800                                  'GO MAKE BAR
720 Y = INT((HIGH1 - PROTEIN) * R1 + 12.5)
730 C = 2: GOSUB 800                                  'GO MAKE BAR
740 X = 280                                           'MOVE OVER FOR SECOND BAR CHART
750 Y = INT((HIGH2 - SODIUM) * R2 + 12.5)
760 C = 2: GOSUB 800                                  'GO MAKE BAR
770 Y = INT((HIGH2 - POTASSIUM) * R2 + 12.5)
780 GOSUB 800                                         'GO MAKE BAR
790 GOTO 850
800     'MAKE BAR
810 LINE (X,Y) - (X+8,YO),C,BF
820 X = X + 16                                        'MOVE OVER FOR NEXT BAR
```

Program 14–2 (cont.)

```
830 RETURN
840 '
850 '############################## MAKE PIECHART ########################
860 XC = 45: YC = 78: R = 40
870 FATCALRES = FAT * 9                    'CONVERT GRAMS TO CALORIES
880 CARBOCALRES = CARBO * 4
890 PROCALRES = PROTEIN * 4
900 CALORIES = FATCALRES + CARBOCALRES + PROCALRES
910 LOCATE 3,1: PRINT "CALORIES - ";
920 LOCATE 4,3: PRINT USING "####";CALORIES;
930 LOCATE 17,1: PRINT "% OF TOTAL";
940 LOCATE 18,1: PRINT "  FROM -";
950 S = 0: B = 0
960 S = FATCALRES: C = 3: GOSUB 1030            'SECTION FOR FAT CALORIES
970 S = CARBOCALRES: C = 1: GOSUB 1030          'SECTION FOR CARBOHYDRATE
980 S = PROCALRES: C = 2: GOSUB 1030            'SECTION FOR PROTEIN CALORIES
990 LOCATE 20,1: PRINT "FAT";TAB(10);USING"##";FATCALRES/CALORIES*100;
1000 LOCATE 21,1: PRINT "CARBO";TAB(10);USING "##";CARBOCALRES/CALORIES*100;
1010 LOCATE 22,1: PRINT "PROTEIN";TAB(10);USING "##";PROCALRES/CALORIES*100;
1020 GOTO 1240
1030 '################## DRAW AND FILL IN PIECHART AREA #################
1040 ANGLE = BEFORE + 6.28318 * S / CALORIES
1050 CIRCLE (XC,YC),R,C,-BEFORE,-ANGLE,YADJUST
1060 INTERIORANGLE = 6.28318 - (ANGLE + BEFORE) / 2
1070 XINTERIOR = XC + R/2 * COS(INTERIORANGLE)
1080 YINTERIOR = YC + R/2 * SIN(INTERIORANGLE) * YADJUST
1090 PAINT (XINTERIOR,YINTERIOR),C,C
1100 BEFORE = ANGLE
1110 RETURN
1120 '#########################################################################
1130 DATA MILK,12,9,9,.122,.351
1140 DATA BACON,1,8,5,.163,.038
1150 DATA HADDOCK,5,6,20,.177,.348
1160 DATA TUNA,0,18,21,.688,.259
1170 DATA EGG,0,6,7,.066,.070
1180 DATA SPINACH,3,0,2,.040,.259
1190 DATA CORN,1.4,26.32,4.48,0,.231
1200 DATA DATES,.89,129,3.9,.002,1.15
1210 DATA LIVER,.6,.3,1.5,.022,3.04
1220 DATA CHILI,5.2,10.4,6.4,.001,.45
1230 '#########################################################################
1240 IF INKEY$ = "" THEN 1240
1250 END
```

We can use graphs and charts for various nonfinancial home uses. A daily nutrition chart is shown in Fig. 14–2. This chart is the output of Prog. 14–2, which calculates the daily caloric intake and plots the percentages of calories from protein, carbohydrate, and fat sources in our diet. We could store this daily information in a data file and modify Prog. 14–2 to plot long-term comparisons. A more complete file of food types and food components could also be set up for such a program.

Program 14–3 produces a biorhythm graph, as shown in Fig. 14–3. This graph plots the theoretical ups and downs of our physical, emotional, and intellectual energy levels. The algorithm used for this graph assumes a 23-day cycle for the physical curve, a 28-day cycle for the emotional curve, and a 33-day

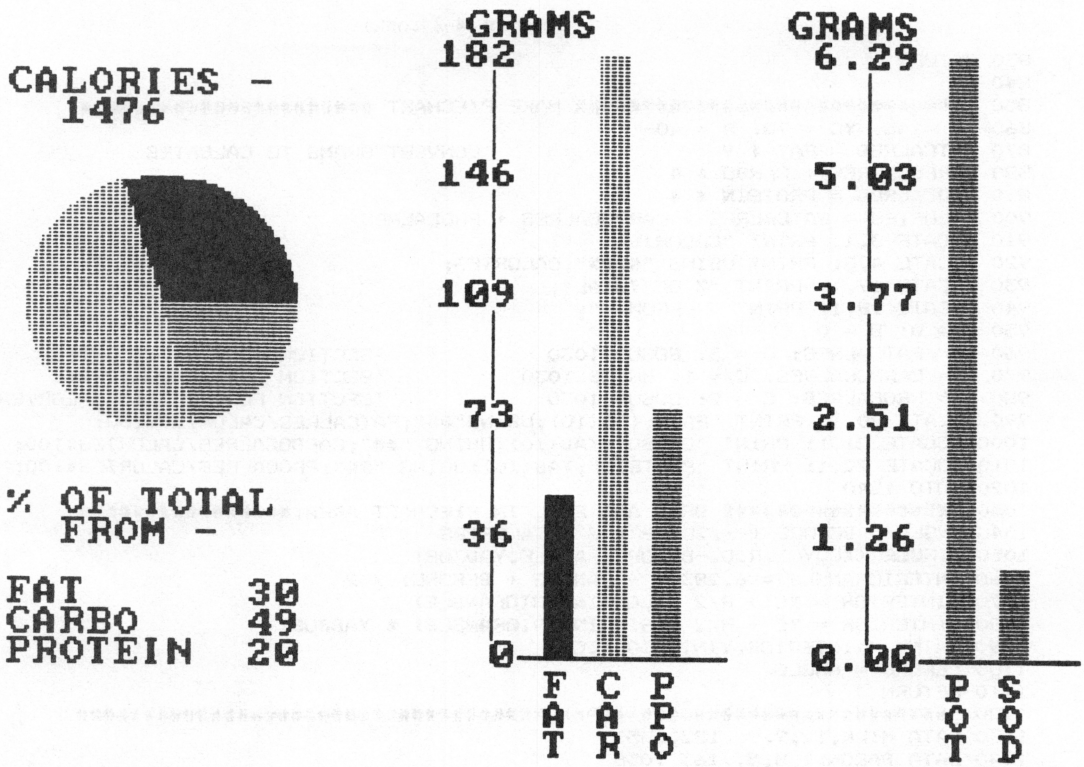


Figure 14–2 Nutrition chart produced by Prog. 14–2.

Program 14–3 Biorhythm graph.

```
10 'PROGRAM 14-3. BIORHYTHM
20      'GIVEN AN INDIVIDUAL'S BIRTHDATE AND A STARTING DATE,
30      'CONSTRUCTS BIORHYTHM CHART (WITH CURVES FOR EMOTIONAL
40      'PHYSICAL, AND INTELLECTUAL CYCLES) FOR THE NEXT 30 DAYS.
50      'CURVES ARE DRAWN IN DIFFERENT COLORS WITH LABELS PLACED
60      'NEAR THE CURVES. LABELS ARE PLACED ON THE LEFT SIDE UNLESS
70      'ANY TWO CURVES START TOO CLOSE TOGETHER  -- THEN ONE OF
80      'THE LABELS IS MOVED TO THE RIGHT SIDE OF CHART.
90      '*****************************************************************
100 SCREEN 0: WIDTH 80: COLOR 7,0: CLS
110 DIM MONTH$(12), DAYSINMONTH(12), ACCUMDAYS(13)
120 FOR K = 1 TO 12                 'READ MONTH NAME, # OF DAYS IN MONTH, AND
130      READ MONTH$(K), DAYSINMONTH(K), ACCUMDAYS(K)        'ACCUMULATED DAYS
140 NEXT
150 PR = 8        'PR IS NUMBER OF VERTICAL PIXELS PER CHARACTER
160 PI = 3.14159
170 H = 50                  'H IS HEIGHT OF THE CURVES
180 RR = 240/30             'RR IS SCALING RATIO - 30 X VALUES OVER 240 PIXELS
190     '
200     '************************ INPUT DATE DATA **********************
210 INPUT "BIRTHDATE (MONTH,DAY,YEAR)"; BIRTHMONTH,BIRTHDAY,BIRTHYEAR
220 INPUT "START DATE FOR CHART (MO,DA,YR)"; CHARTMONTH,CHARTDAY,CHARTYEAR
```

Program 14–3 (cont.)

```
230     'FIND TIME BETWEEN CHART DATE AND BIRTH DATE
240 TIME = (CHARTYEAR-BIRTHYEAR) * 365.25 + (ACCUMDAYS(CHARTMONTH) -
        ACCUMDAYS(BIRTHMONTH)) + (CHARTDAY - BIRTHDAY)
250     'FIND DISPLACEMENT FOR EACH CURVE
260 PHYDISP = 2 * PI * (TIME / 23 - INT(TIME / 23))
270 EMODISP = 2 * PI * (TIME / 28 - INT(TIME / 28))
280 INTDISP = 2 * PI * (TIME / 33 - INT(TIME / 33))
290     'FIND FREQUENCY FOR EACH CURVE
300 PHYFRE = 2 * PI / 23
310 EMOFRE = 2 * PI / 28
320 INTFRE = 2 * PI / 33
330     '************************* DRAW GRID **********************
340 SCREEN 1: COLOR 8,0: CLS
350 LOCATE 1,15: PRINT "BIORHYTHM"
360 LOCATE 3,20-(13+LEN(MONTH$(CHARTMONTH)))/2
370 PRINT "STARTING "; MONTH$(CHARTMONTH); CHARTYEAR;
380 DAY = CHARTDAY
390 X = 44
400 FOR COLUMN = 5 TO 35 STEP 3
410     LOCATE 22,COLUMN: PRINT USING "##";DAY;
420     DAY = DAY + 3
430     IF DAY <= DAYSINMONTH(CHARTMONTH) THEN 450      'STILL THE SAME MONTH?
440     DAY = DAY - DAYSINMONTH(CHARTMONTH)     'SET DAY TO START OF NEXT MONTH
450     LINE (X,40) - (X,160),1
460     X = X + 24
470 NEXT
480 LINE (44,100) - (284,100),1                 'MAKE LINE AT GRAPH 0
490 LOCATE 24,8: PRINT "BIRTHDATE:";BIRTHDAY;MONTH$(BIRTHMONTH);BIRTHYEAR;
500     '************************* DRAW CURVES ************************
510 FOR X = 0 TO 30 STEP .1
520     PHYSY = INT(H * SIN(PHYFRE * X + PHYDISP) + .5)
530     XG = X * RR + 44                    'SCALE THIS X TO AN X FOR THE GRAPH
540     PSET (XG,PHYSY+100),1
550     EMOY = INT(H * SIN(EMOFRE * X + EMODISP) + .5)
560     PSET (XG,EMOY+100),2
570     INTY = INT(H * SIN(INTFRE * X + INTDISP) + .5)
580     PSET (XG,INTY+100),3
590     IF X = 0 THEN GOSUB 640         'GO DO LABELING
600 NEXT
610 IF PUTEMORIGHT$ = "YES" THEN EMOPLACE = INT((EMOY + 100) / PR + .5):
    LOCATE EMOPLACE,38: PRINT "EMO";
620 IF PUTINTRIGHT$ = "YES" THEN INTPLACE = INT((INTY + 100) / PR + .5):
    LOCATE INTPLACE,38: PRINT "INT";
630 GOTO 790
640 '################ LABELING ON LEFT ##########################
650 PHYSPLACE = INT((PHYSY + 100) / PR + .5)     'PRINT POSITION FOR PHYSICAL
660 LOCATE PHYSPLACE,2: PRINT "PHY";
670 EMOPLACE = INT((EMOY + 100) / PR + .5)       'PRINT POSITION FOR EMOTIONAL
680     'WILL LABELING FIT HERE OR SHOULD WE PUT IT ON THE RIGHT$
690 IF EMOPLACE = PHYSPLACE THEN PUTEMORIGHT$ = "YES"
700 IF PUTEMORIGHT$ <> "YES" THEN LOCATE EMOPLACE,2: PRINT "EMO";
710 INTPLACE = INT((INTY + 100) / PR + .5)
720 IF INTPLACE = PHYSPLACE OR INTPLACE = EMOPLACE THEN PUTINTRIGHT$ = "YES"
730 IF PUTINTRIGHT$ <> "YES" THEN LOCATE INTPLACE,2: PRINT "INT";
740 RETURN
750     '*****************************************************************
760 DATA JANUARY,31,0,FEBRUARY,28,31,MARCH,31,59,APRIL,30,90
770 DATA MAY,31,120,JUNE,30,151,JULY,31,181,AUGUST,31,212
780 DATA SEPTEMBER,30,243,OCTOBER,31,273,NOVEMBER,30,304,DECEMBER,31,334
790 IF INKEY$ = "" THEN 790
800 END
```

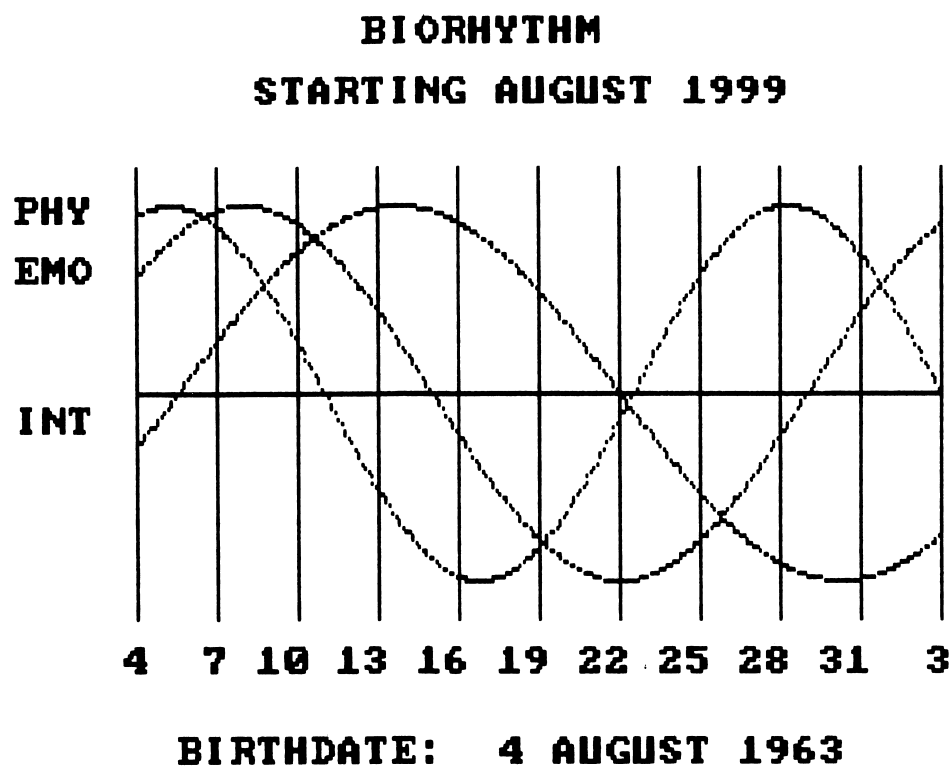


Figure 14–3 Biorhythm graph displayed by Prog. 14–3 for a birth date of August 4, 1963.

cycle for the intellectual curve. Drawing the lines in different colors, as in Fig. P of the color insert, can help to identify the different curves.

Computer-generated pictures can be devised for many types of home use. We can create pictures for inclusion in graphs, in educational programs, as parts of games, or for decoration. A printed picture can be used as a wall decoration or on personally designed greeting cards. We could also create small pictures or designs for personalized stationery.

14–2 GAME PLAYING

Games can be both fun and educational. We can devise games that teach about numbers, arithmetic, letters, words, or spelling. Some games can help develop coordination.

With Prog. 14–4, we produce a ball and paddle game that requires some coordination to keep the ball in play. This program takes the box and bouncing ball of Chapter 8 and adds a paddle in place of the left wall. The ball starts at a

Program 14–4 Bouncing ball and paddle game.

```
10 'PROGRAM 14-4. BOUNCING BALL GAME.
20      'DRAWS THREE-SIDED BOX AND A PADDLE ON THE LEFT SIDE.
30      'BALL BOUNCES IN BOX (DX = DY = 5 AT GAME START) AND
40      'AGAINST PADDLE. PADDLE MUST BE POSITIONED, THROUGH
50      'KEYBOARD INPUT (HITTING THE CURSOR KEYS FOR UP & DOWN).
60      'SPEED OF BALL INCREASES (BY INCREASING DX AND DY) WHEN-
70      'EVER BALL IS HIT BY PADDLE 5 TIMES AT THE SAME SPEED.
80      'SCORE IS UPDATED BY 1 FOR EVERY SUCCESSFUL BOUNCE OFF
90      'THE PADDLE. GAME CONTINUES UNTIL BALL IS MISSED BY PADDLE.
100     '*****************************************************************
110 SCREEN 1
120 STARTER = RND(-VAL(RIGHT$(TIME$,2)))          'RANDOMIZE RND FUNCTION
130 XL = 50: XR = 300: YT = 20: YB = 160          'BOUNDARIES OF BOX
140 YP = 100                                      'YP IS TOP POINT OF PADDLE
150 R = 3                                         'R IS RADIUS OF BALL
160 DX = 5: DY = 5       'BALL INITIALLY TRAVELS 5 UNITS IN EACH STEP
170     '***************************** DRAW BOX ********************************
180 CLS
190 LINE (XL,YT) - (XR,YT): LINE - (XR,YB): LINE - (XL,YB)
200 LINE (XL,YP) - (XL,YP+40)                     'DRAW PADDLE
210     '************************** BOUNCE BALL ********************************
220 XNEW = XL + INT((XR-XL-6) * RND + .5)
230 YNEW = YT + INT((YB-YT-6) * RND + .5)
240 SCORE = 0
250 LOCATE 1,36: PRINT USING "##"; SCORE
260 CIRCLE (X,Y),R,0                      'ERASE CURRENT BALL POSITION
270 CIRCLE (XNEW,YNEW),R                  'DRAW NEW POSITION
280     'MOVE PADDLE
290 A$ = INKEY$
300 IF A$ = "" THEN 350
310 LINE (XL,YP) - (XL,YP+40),0           'ERASE CURRENT PADDLE POSITION
320 IF RIGHT$(A$,1) = CHR$(80) THEN YP = YP + 15
330 IF RIGHT$(A$,1) = CHR$(72) THEN YP = YP - 15
340 LINE (XL,YP) - (XL,YP+40)             'DRAW NEW PADDLE
350 X = XNEW                              'SAVE CURRENT POSITION IN X AND Y
360 Y = YNEW
370 BX = 0                          'BX AND BY ARE SWITCHES TO INDICATE
380 BY = 0                          'WHICH WALL WE'RE GOING TO HIT
390 SLOPE = DY / DX
400     '*****************************************************************
410     'WILL WE HIT A VERTICAL WALL?
420 IF DX > 0 AND X + DX + R >= XR THEN BX = 1: XNEW = XR - R - 1
    ELSE IF DX < 0 AND X + DX - R <= XL THEN BX = 1: XNEW = XL + R + 1
430     'WILL WE HIT A HORIZONTAL WALL?
440 IF DY > 0 AND Y + DY + R >= YB THEN BY = 1: YNEW = YB - R -1
    ELSE IF DY < 0 AND Y + DY - R <= YT THEN BY = 1: YNEW = YT + R + 1
450     '*************************************************************
460     'ARE WE BOUNCING OFF NO WALLS, AN X WALL, A Y WALL, OR BOTH WALLS?
470 IF BX = 0 AND BY = 0 THEN 510           'NOT BOUNCING
480 IF BX = 0 AND BY = 1 THEN 540           'BOUNCING OFF Y
490 IF BX = 1 AND BY = 0 THEN 590           'BOUNCING OFF X
500 IF BX = 1 AND BY = 1 THEN 720           'BOUNCING OFF BOTH (IN A CORNER)
510 '########################## NOT BOUNCING ##################################
520 XNEW = X + DX: YNEW = Y + DY
530 GOTO 260
540 '########################## BOUNCE OFF Y WALL #############################
550 XNEW = (YNEW - Y) / SLOPE + X
560 DY = -DY
570 BEEP
580 GOTO 260
590 '########################## BOUNCE OFF X WALL #############################
```

Program 14–4 (cont.)

```
600 YNEW = (XNEW - X) * SLOPE + Y
610 IF DX > 0 THEN 690                          'WE'RE GOING TO THE RIGHT
620     'GOING TOWARDS PADDLE. IS PADDLE IN GOOD POSITION?
630 IF YNEW < YP OR YNEW > YP+40 THEN 800        'BALL IS MISSED BY PADDLE
640 SCORE = SCORE + 1                           'INCREASE SCORE
650 LOCATE 1,36: PRINT USING "##"; SCORE;
660 IF SCORE/5 <> INT(SCORE/5) THEN 690         'IF EQUAL, TIME TO SPEED UP BAL
670 IF DX < 0 THEN DX = DX - 5
680 IF DX > 0 THEN DX = DX + 5
690 DX = -DX
700 BEEP
710 GOTO 260
720 '######################### GOING INTO A CORNER ######################
730     'WHICH WALL WOULD IT HIT FIRST?
740 IF ABS(XNEW - X) < ABS(YNEW - Y) THEN 590          'BOUNCE OFF X
750 IF ABS(YNEW - Y) < ABS(XNEW - X) THEN 540          'BOUNCE OFF Y
760     'BALL IS EQUAL DISTANCE FROM X AND Y WALLS ON EACH SIDE OF CORNER
770 DX = -DX
780 DY = -DY
790 GOTO 260
800 '###################### MISSED THE BALL ############################
810 XNEW = XL - R - 8           'DROP THE BALL DOWN TO THE GROUND ALONG
820 YNEW = (XNEW - X) * SLOPE + Y               'LEFT SIDE OF BOX
830 CIRCLE (X,Y),R,0
840 FOR YNEW = YNEW TO 190 STEP 5
850     CIRCLE (X,Y),R,0
860     CIRCLE (XNEW,YNEW),R
870     FOR DELAY = 1 TO 100: NEXT
880     Y = YNEW: X = XNEW
890 NEXT
900 '###################### PRINT THE SCORE ############################
910 LOCATE 10,10: PRINT "YOUR SCORE IS ";
920 IF SCORE >= 24 THEN PRINT "INCREDIBLE!": GOTO 980
930 IF SCORE >= 19 THEN PRINT "OUTSTANDING!": GOTO 980
940 IF SCORE >= 14 THEN PRINT "PRETTY GOOD": GOTO 980
950 IF SCORE >= 9 THEN PRINT "FAIR": GOTO 980
960 IF SCORE >= 4 THEN PRINT "IMPROVING";: GOTO 980
970 IF SCORE < 4 THEN PRINT "AWFUL!"
980     '********************** PLAY AGAIN OR END? ******************
990 LOCATE 13,10: INPUT "LIKE TO PLAY AGAIN"; C$
1000 IF C$ = "Y" THEN 160
1010 END
```

random position within the box. Whenever the ball gets to the left side, we must bounce it back into the box. If we miss, the ball goes outside the box and stops. The number of consecutive times that we are able to bounce the ball off the paddle is our score. After every five paddle bounces, the ball speeds up a little. Figure 14–4 shows the display at the end of the game for one possible score. We could play this game alone and see how high we can score in a single game, or we could play in teams, adding the scores each time until one team reaches 100.

An archery game is given in Prog. 14–5. A box is placed at a random position on the right half of the screen, and we try to hit it with an arrow shot from the lower left corner. We choose an angle and an initial speed, and the arrow travels along a parabolic path, as discussed in Chapter 8. We have three shots from a quiver of arrows (Fig. 14–5) for each box. The program generates five boxes for

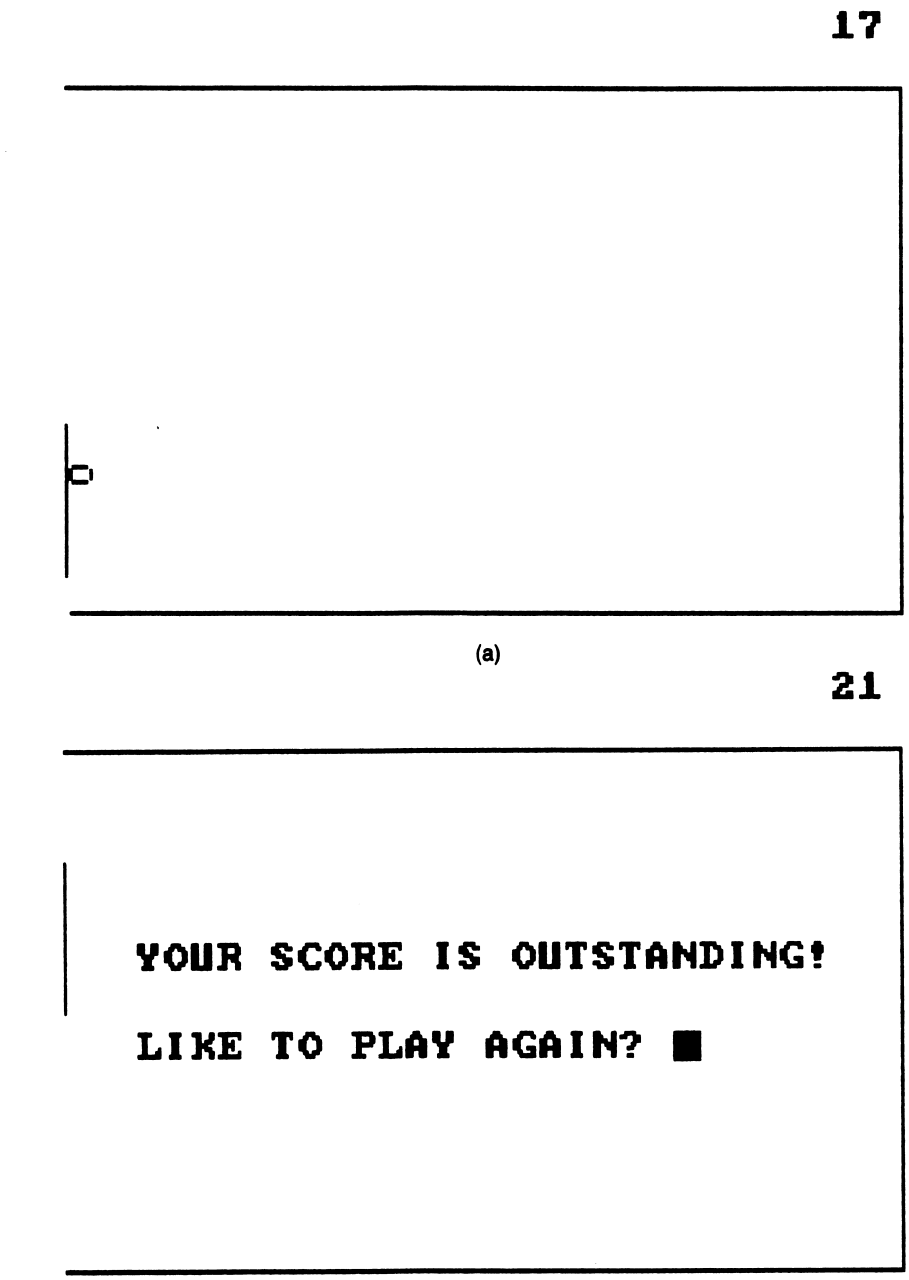


Figure 14–4 Two displays from the ball and paddle game of Prog. 14–4.

each game, and displays the current box number and accumulated score. We count a hit on the first shot as 10 points, a hit on the second shot as 5 points, and a hit on the third shot as 2 points.

We can include sound using the SOUND and PLAY commands with our game programs. Sounds could be used to coincide with the bounce of a ball or the landing of an arrow. We could even play a tune at key points in a game, such as when we score or at the end.

Program 14–5 Arrow and target game.

```
10 'PROGRAM 14-5. ARROW AND TARGET GAME.
20      'REPEATEDLY DRAWS AND ERASES AN ARROW WHOSE TAIL
30      'IS A POINT ON A PARABOLA. REMAINDER OF THE ARROW
40      'IS FOUND USING THIS TAIL POINT, THE SLOPE OF THE
50      'LINE TANGENT TO THE CURVE AT THIS POINT, AND THE
60      'LENGTH OF THE ARROW
70      '*****************************************************************
80  SCREEN 1: COLOR 1,1
90  XO = 15                 'XO,YO IS STARTING POSITION OF ARROW
100 YO = 180
110 G = 980                         'G IS FORCE OF GRAVITY
120 ARROWLENGTH = 30
130 TIPLENGTH = 6
140 STARTRND = RND(-VAL(RIGHT$(TIME$,1)))       'START RND FUNCTION
150 TARGET = 1
160 SCORE = 0
170     '**************** DRAW BOX TARGET AND PLAY ******************
180 CLS
190 GOSUB 880                               'DRAW QUIVER
200 SHOT = 1                                'NUMBER OF ATTEMPTS AT THIS TARGET
210 LOCATE 25,30: PRINT "SCORE";
220 LOCATE 25,35: PRINT SCORE;
230 XL = 100 + RND * 180                    'RANDOMLY PLACE LEFT EDGE OF BOX
240 XR = XL + 35
250 YT = RND * 165                          'RANDOMLY PLACE TOP EDGE OF BOX
260 YB = YT + 35
270 LINE (XL,YT) - (XR,YB),1,BF                     'DRAW TARGET
280     '************** CHOOSE ARROW ANGLE, SPEED ********************
290 LOCATE 1,1: PRINT STRING$(40," ");
300 LOCATE 1,1: INPUT "ANGLE (0-90)"; ANGLE
310 IF ANGLE < 0 OR ANGLE > 90 THEN PRINT "RE-ENTER ANGLE": GOTO 290
320 ANGLE = ANGLE * 3.14159 / 180
330 LOCATE 1,19: INPUT "SPEED"; SPEED
340 RANGE = SPEED * SPEED * SIN(2 * ANGLE) / G
350     'DETERMINE COEFFICIENTS FOR PARABOLA'S EQUATION
360 C1 = G / (2 * (SPEED * COS(ANGLE)) ^ 2)
370 TWOC1 = 2 * C1
380 C2 = - TAN(ANGLE)
390 GOSUB 1040                              'REMOVE ARROW FROM QUIVER
400     '********************* MOVE ARROW ************************
410     'FIND ARROW TAIL POINTS ALONG THE PARABOLA AND DRAW ARROW
420 FOR X = 30 TO RANGE STEP 5
430     Y = C1 * X * X + C2 * X + YO
440     'X AND Y ARE THE TAILPOINTS ON THE PARABOLA
450     'FIND OTHER ENDPOINT OF ARROW
460     M = TWOC1 * X + C2              'M IS SLOPE OF THE ARROW
470     A1 = ATN(M)                     'INVERSE TANGENT OF M GIVES ANGLE A1
480     Y1 = Y + ARROWLENGTH * SIN(A1)
```

Program 14–5 (cont.)

```
490     X1 = X + ARROWLENGTH * COS(A1)
500     IF X1 > 319 OR Y1 > 199 THEN 670          'IS OTHER ENDPOINT ON SCREEN?
510     IF POINT(X1,Y1) = 1 THEN HIT = 1
520     IF INT(X/15) <> X/15 THEN 660             'DON'T DRAW THIS ARROW
530     IF X > 25 THEN GOSUB 1250                 'ELSE ERASE ARROW & DRAW
540     'CALCULATE ARROW TIP
550     M2 = M + .75                              'SLOPE OF ONE TIP
560     A2 = ATN(M2)
570     X2 = X1 - TIPLENGTH * COS(A2)
580     Y2 = Y1 - TIPLENGTH * SIN(A2)
590     M3 = M - .75                              'SLOPE OF SECOND TIP
600     A3 = ATN(M3)
610     X3 = X1 - TIPLENGTH * COS(A3)
620     Y3 = Y1 - TIPLENGTH * SIN(A3)
630     GOSUB 1190                                    'DRAW ARROW
640     IF HIT THEN 740
650     XS = X: YS = Y: X1S = X1: Y1S = Y1            'SAVE CURRENT POSITION
660 NEXT
670 SHOT = SHOT + 1
680 IF SHOT <= 3 THEN GOSUB 1250: GOTO 280            'GO ON TO NEW TARGET
690 IF TARGET = 5 THEN 1310                           'GAME OVER
700 TARGET = TARGET + 1                           'ELSE, GO ON TO NEXT TARGET
710 LOCATE 25,1: PRINT "TOO BAD. TRY ANOTHER";
720 FOR DELAY = 1 TO 600: NEXT
730 GOTO 180
740     '************** ARROW HAS HIT BOX *************************
750     'INCREASE SCORE
760 IF SHOT = 1 THEN SCORE = SCORE + 10
770 IF SHOT = 2 THEN SCORE = SCORE + 5
780 IF SHOT = 3 THEN SCORE = SCORE + 2
790 LOCATE 25,35: PRINT SCORE;
800 FOR TB = 1 TO 2                               'BLINK "BULLSEYE"
810     LOCATE 25,1: PRINT "BULLSEYE!";
820     FOR DELAY = 1 TO 300: NEXT
830     LOCATE 25,1: PRINT "          ";
840     FOR DELAY = 1 TO 300: NEXT
850 NEXT
860 HIT = 0
870 IF TARGET = 5 THEN 1310 ELSE TARGET = TARGET + 1: GOTO 180
880 '********************** DRAW QUIVER ********************************
890 LOCATE 22,1: PRINT TARGET;
900 CIRCLE (10,150),10,,,,.3
910 CIRCLE (10,185),10,,,,.3
920 LINE (0,150) - (0,185)
930 LINE (20,150) - (20,185)
940 LINE (5,130) - (5,152)
950 LINE (5,130) - (2,133)
960 LINE (5,130) - (8,133)
970 LINE (10,125) - (12,153)
980 LINE (10,125) - (7,128)
990 LINE (10,125) - (13,128)
1000 LINE (18,128) - (15,152)
1010 LINE (18,128) - (15,131)
1020 LINE (18,128) - (21,131)
1030 RETURN
1040 '******************** REMOVE ARROW FROM QUIVER ********************
1050 C = 1
1060 IF SHOT <> 1 THEN 1100
1070 LINE (10,125) - (12,153),0          'REMOVE FIRST ARROW
1080 LINE (10,125) - (7,128),0
```

Program 14–5 (cont.)

```
1090 LINE (10,125) - (13,128),0
1100 IF SHOT <> 2 THEN 1140
1110 LINE (5,130) - (5,152),0          'REMOVE SECOND ARROW
1120 LINE (5,130) - (2,133),0
1130 LINE (5,130) - (8,133),0
1140 IF SHOT <> 3 THEN 1180
1150 LINE (18,128) - (15,152),0        'REMOVE THIRD ARROW
1160 LINE (18,128) - (15,131),0
1170 LINE (18,128) - (21,131),0
1180 RETURN
1190 '########################## DRAW ARROW ##########################
1200 C = 3
1210 LINE (X,Y) - (X1,Y1),C
1220 LINE (X1,Y1) - (X2,Y2),C
1230 LINE (X1,Y1) - (X3,Y3),C
1240 RETURN
1250 '########################## ERASE ARROW ##########################
1260 C = 0
1270 LINE (XS,YS) - (X1S,Y1S),C
1280 LINE (X1S,Y1S) - (X2,Y2),C
1290 LINE (X1S,Y1S) - (X3,Y3),C
1300 RETURN
1310  '******************** PLAY AGAIN OR STOP ****************************
1320 CLS
1330 LOCATE 12,30: PRINT "FINAL SCORE -- "; SCORE
1340 PRINT
1350 PRINT TAB(10);"WANT TO PLAY AGAIN";
1360 INPUT C$
1370 IF C$ = "N" THEN 1390
1380 GOTO 150
1390 END
```

Figure 14–5 Initial and bullseye positions of the arrow for the archery game (Prog. 14–5).

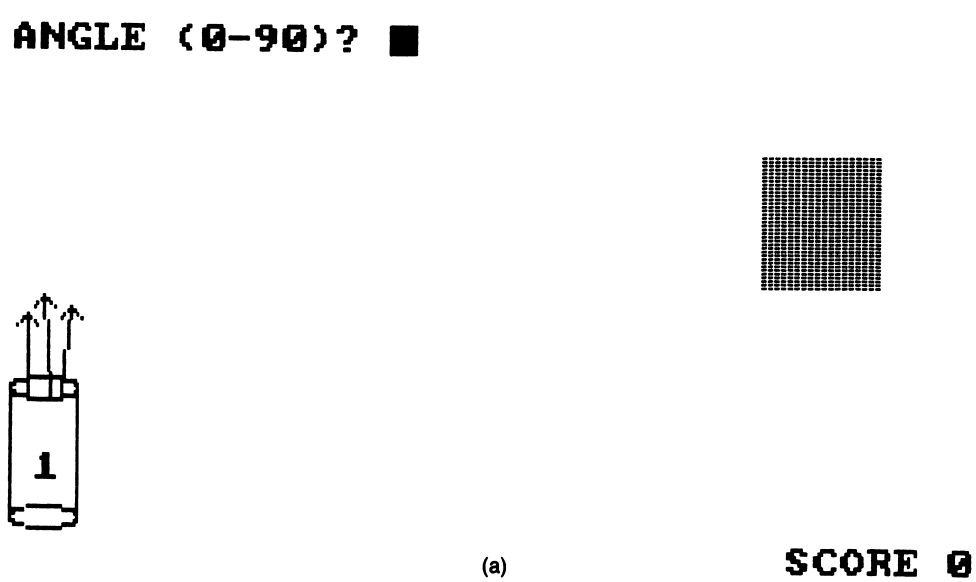
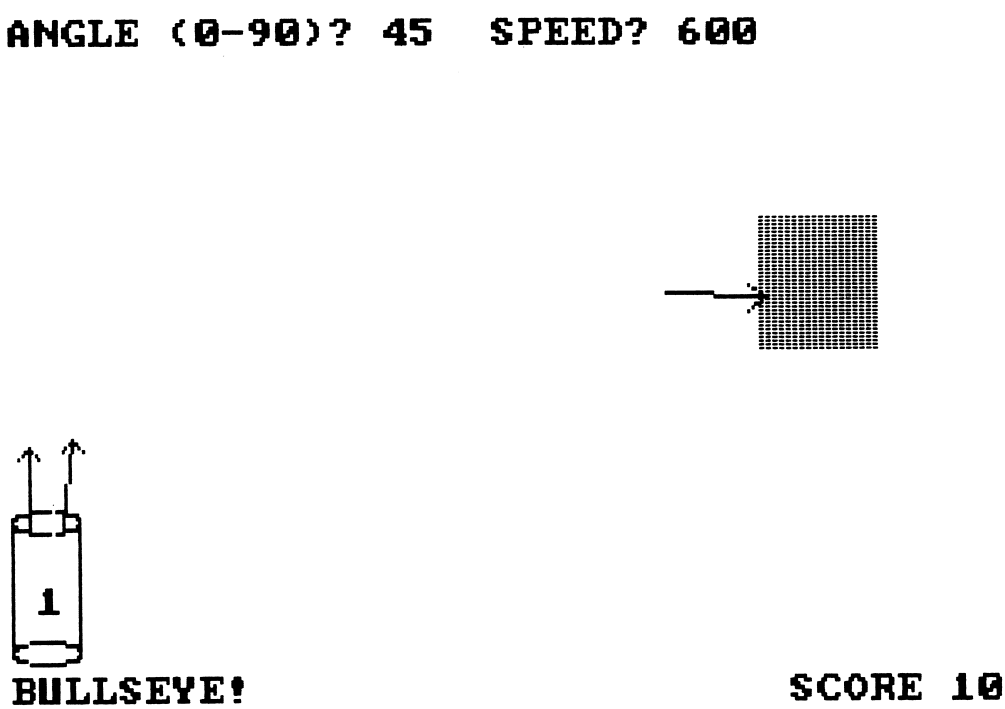


Figure 14–5 (cont.)



(b)

Appendix A

PC Graph Paper

Customized graph paper for the PC can be constructed by printing out a grid of horizontal and vertical lines for each mode of operation. Figures A–1 and A–2 provide examples of such graph paper that can be used for laying out pictures for each of the two text modes, WIDTH 40 and WIDTH 80. Figures A–3 and A–4 show examples of graph paper that can be used for picture layouts with SCREEN 1 and SCREEN 2.

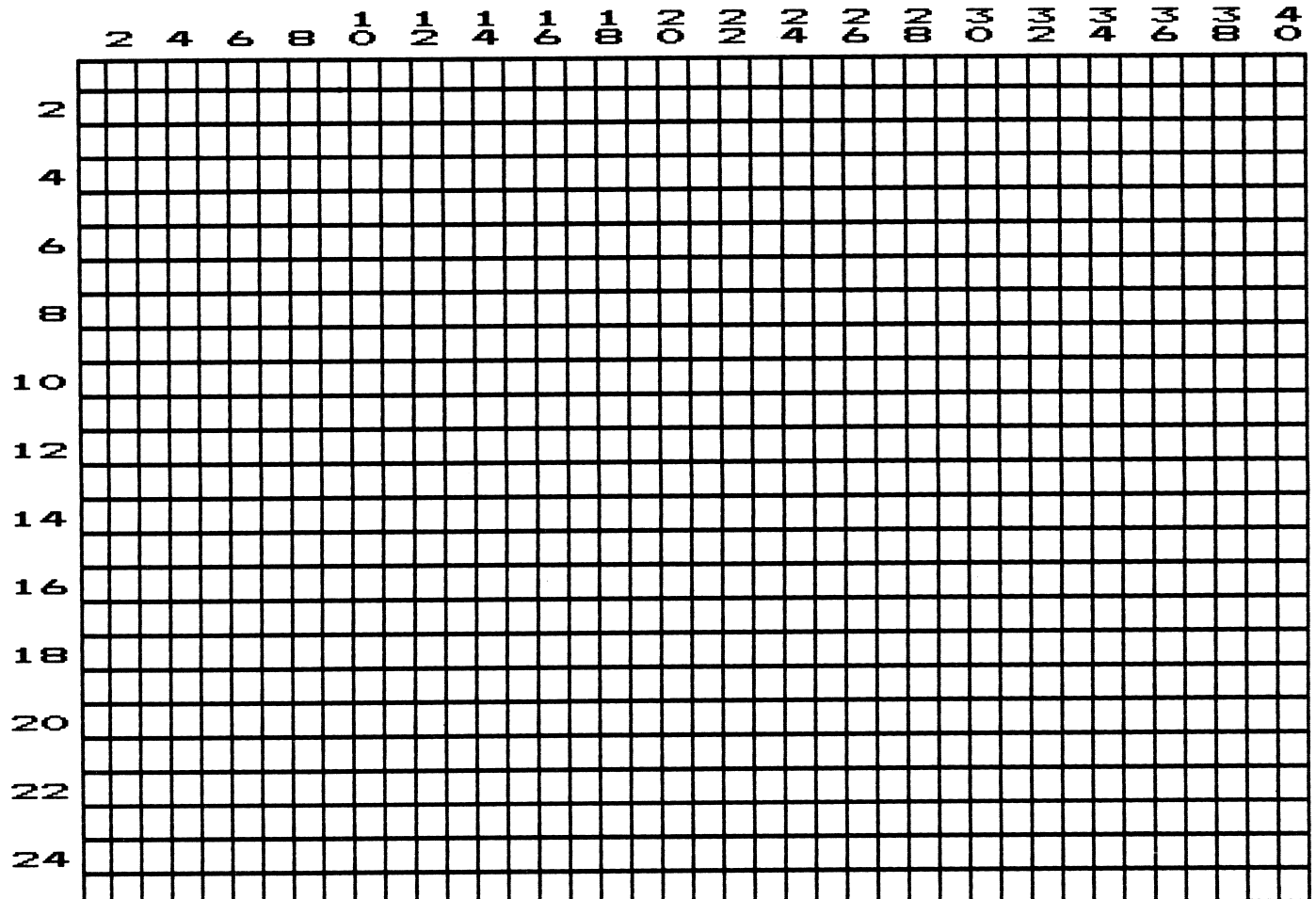


Figure A–1 A customized graph paper grid that can be used for layouts of character pictures in WIDTH 40. Horizontal and vertical lines are drawn on character boundaries, and every second row and column position is labeled.

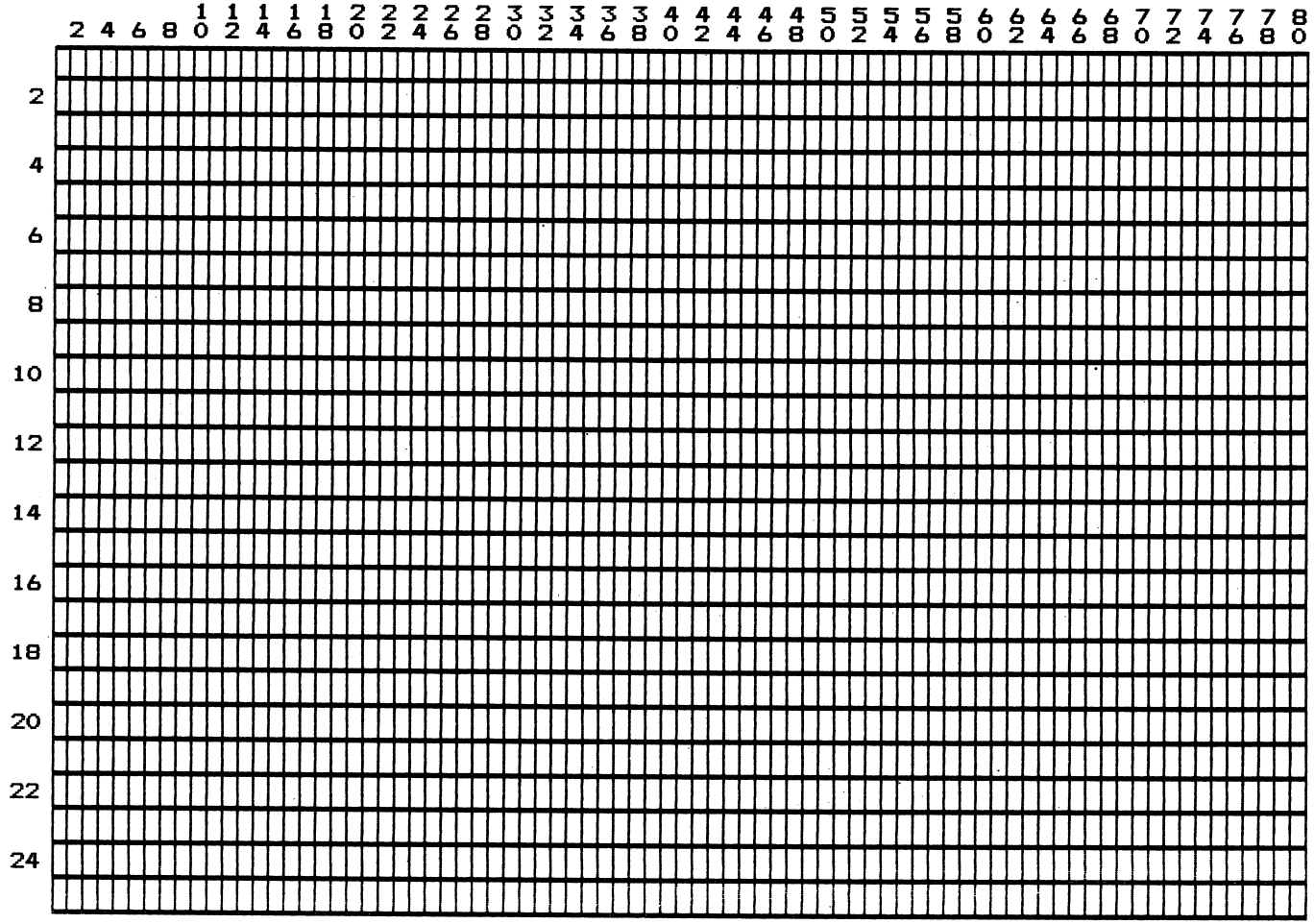


Figure A–2 A customized graph paper grid that can be used for layouts of character pictures in WIDTH 80. Horizontal and vertical lines are drawn on character boundaries, and every second row and column position is labeled.

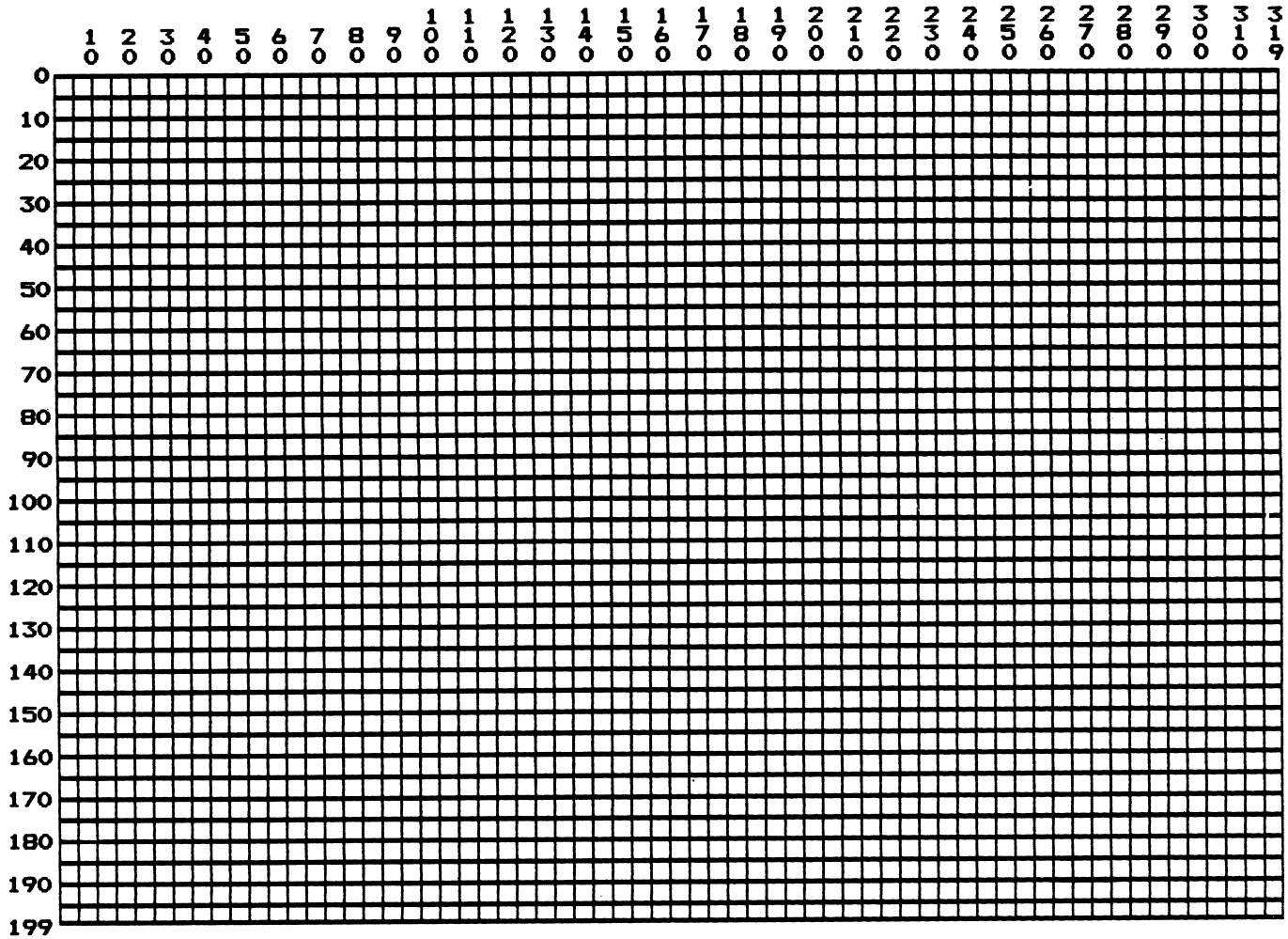


Figure A–3 A customized graph paper grid that can be used for layouts of pixel pictures in medium resolution (SCREEN 1). Horizontal and vertical lines are spaced five pixels apart, starting from position (0,0).

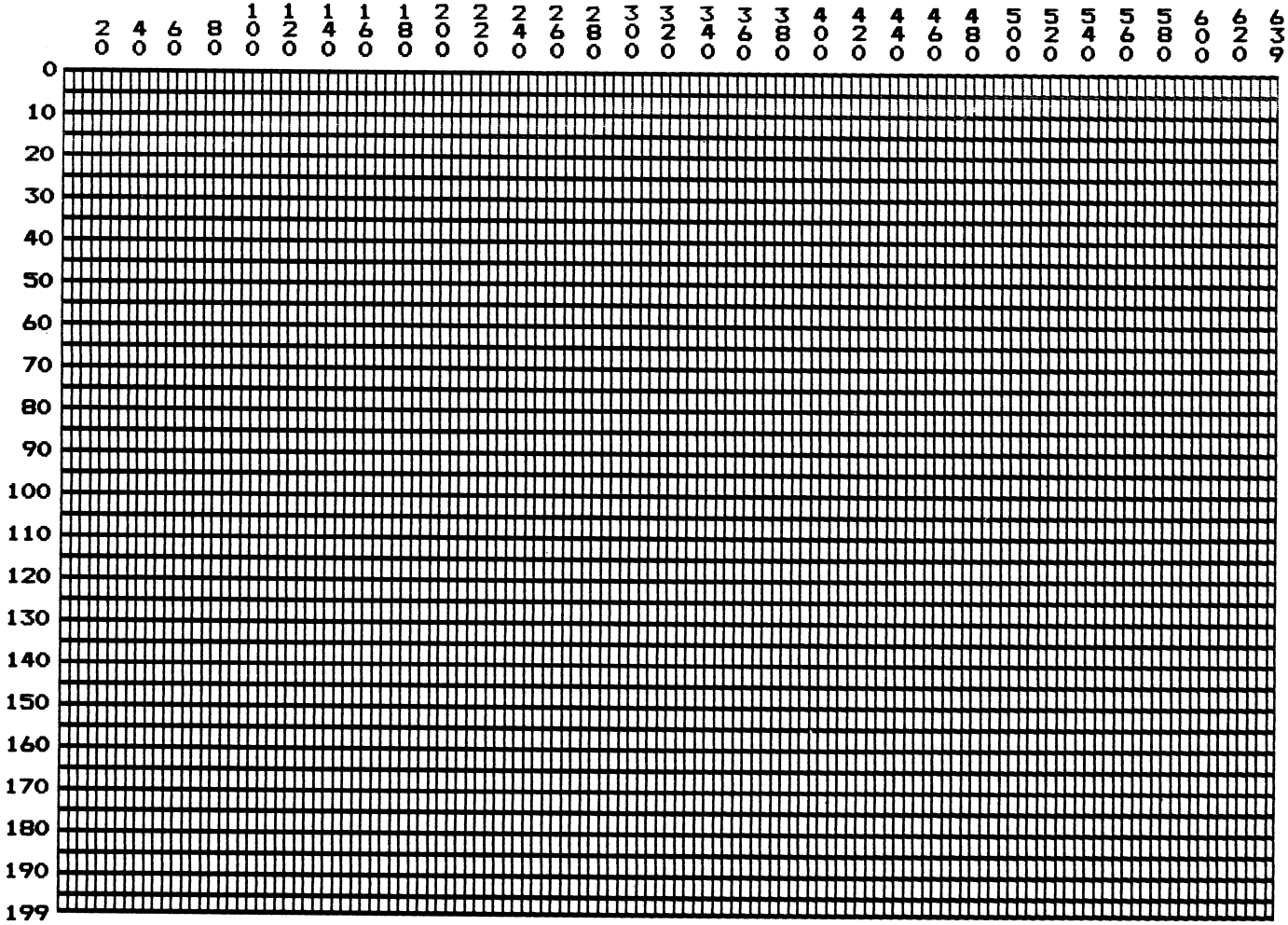


Figure A–4 A customized graph paper grid that can be used for layouts of pixel pictures in high resolution (SCREEN 2). Horizontal and vertical lines are spaced five pixels apart, starting from position (0,0).

Appendix B

PC Character Set and ASCII Codes

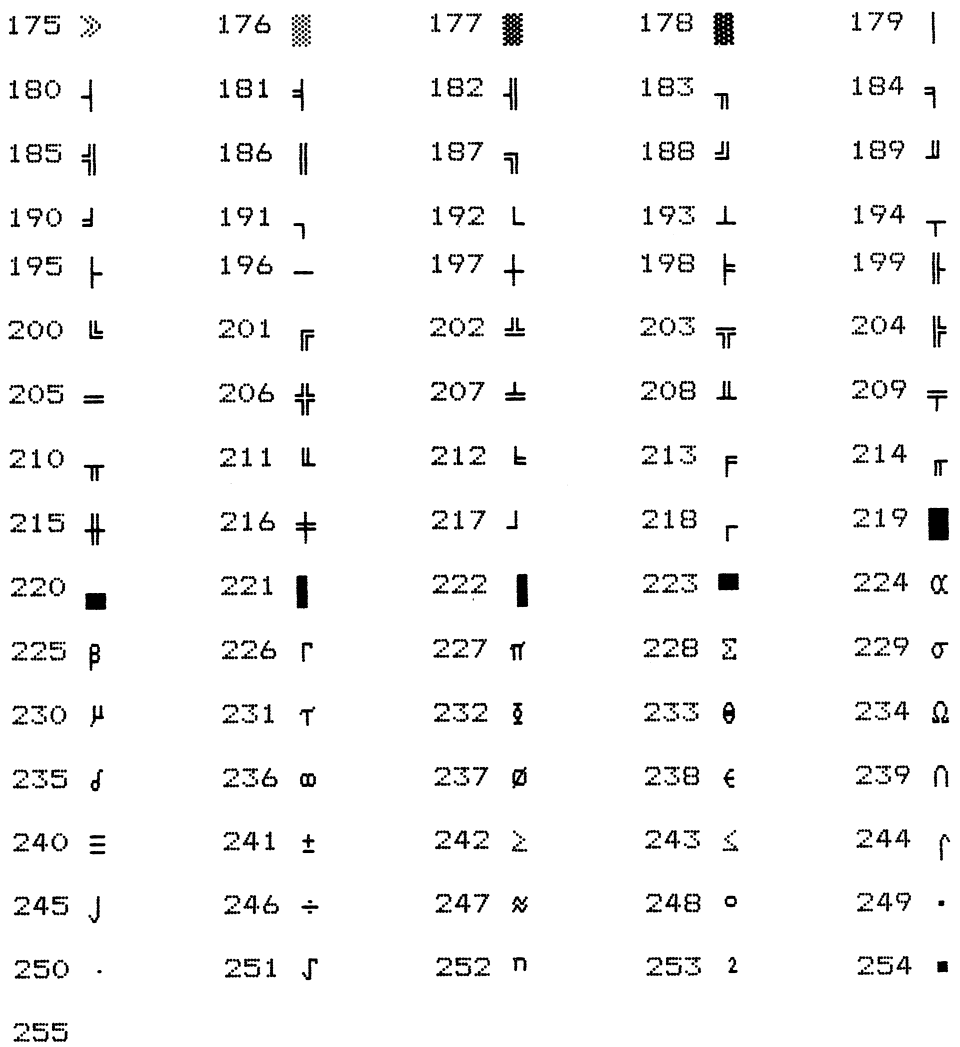
Figure B–1 Table of ASCII codes and the corresponding characters displayed by these codes. ASCII code 32 produces a space, and the control codes listed in Fig. B–2 do not produce visible characters.

0	1 ☺	2 ☻	3 ♥	4 ♦
5 ♣	6 ♠	7	8 ◘	9
10	11	12	13	14 ♫
15 ☼	16 ►	17 ◄	18 ↕	19 ‼
20 ¶	21 §	22 ▬	23 ↨	24 ↑
25 ↓	26 →	27 ←	28	29
30	31	32	33 !	34 "
35 #	36 $	37 %	38 &	39 '
40 (	41)	42 *	43 +	44 ,
45 -	46 .	47 /	48 0	49 1
50 2	51 3	52 4	53 5	54 6
55 7	56 8	57 9	58 :	59 ;
60 <	61 =	62 >	63 ?	64 @

Figure B–1 (cont.)

65 A	66 B	67 C	68 D	69 E
70 F	71 G	72 H	73 I	74 J
75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T
85 U	86 V	87 W	88 X	89 Y
90 Z	91 [	92 \	93]	94 ^
95 _	96 `	97 a	98 b	99 c
100 d	101 e	102 f	103 g	104 h
105 i	106 j	107 k	108 l	109 m
110 n	111 o	112 p	113 q	114 r
115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124 |
125 }	126 ~	127 ⌂	128 Ç	129 ü
130 é	131 â	132 ä	133 à	134 å
135 ç	136 ê	137 ë	138 è	139 ï
140 î	141 ì	142 Ä	143 Å	144 É
145 æ	146 Æ	147 ô	148 ö	149 ò
150 û	151 ù	152 ÿ	153 Ö	154 Ü
155 ¢	156 £	157 ¥	158 ₧	159 ƒ
160 á	161 í	162 ó	163 ú	164 ñ
165 Ñ	166 ª	167 º	168 ¿	169 ⌐
170 ¬	171 ½	172 ¼	173 ¡	174 «

Figure B–1 (cont.)



ASCII value	Action
0	null
7	beep
8	backspace
9	tab
10	line feed
11	home
12	form feed
13	carriage return
28	move cursor right
29	move cursor left
30	move cursor up
31	move cursor down
255	blank

Figure B–2 Table of ASCII control codes and their purpose. These codes do not produce visible characters.

Index

COMPUTER GRAPHICS FOR THE IBM PERSONAL COMPUTER

DONALD HEARN AND M. PAULINE BAKER

In this new book, the authors discuss the basic concepts and techniques of computer graphics and explore the capabilities of the IBM Personal Computer for graphics applications. They examine methods for creating two- and three-dimensional pictures and graphs and show how to manipulate and animate displays. They also analyze the make-up of the PC and the graphics features of the PC's BASIC in detail.

The book is presented in five parts:

PART I is about the IBM Personal Computer—what makes the system tick, how the different hardware components function, and what options are available for expansion boards, video monitors, and other input/output devices.

PART II introduces fundamental methods for constructing pictures and graphs in two dimensions.

PART III presents techniques for manipulating displays.

PART IV covers three-dimensional graphics.

PART V surveys applications of computer graphics in business, education, and the home. Topics include additional graph-drawing techniques, simulations, computer-assisted instruction, household budget charts, nutrition charts, and game playing.

Also available . . .

MICROCOMPUTER GRAPHICS: Techniques and Applications by Donald Hearn and M. Pauline Baker

Published 1983 *320 pages*

PRENTICE-HALL, INC.
Englewood Cliffs, N.J. 07632

ISBN 0-13-164327-4