TEK SPS BASIC
VØ2/VØ2XM
Graphics Package

CP57002/CP57502

Tektronix®

COMMITTED TO EXCELLENCE

Tektronix®
COMMITTED TO EXCELLENCE

TEK SPS BASIC
VØ2/VØ2XM
Graphics Package
CP57002/CP57502

INSTRUCTION MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

Serial Number _____

First Printing FEB 1979 PRODUCED BY SPS DOCUMENTATION GROUP 070-2737-01

SOFTWARE SUPPORT POLICY

Unless otherwise provided, Tektronix, Inc., agrees that during the one (1) year period following installation, if the customer encounters a problem with this software which the customer's diagnosis indicates is caused by a software defect, the customer may submit a Software Performance Report to Tektronix, Inc. For problems occurring in current, unaltered releases of software, Tektronix, Inc., will respond to Software Performance Reports via a software maintenance periodical. The software maintenance periodical will be provided at no cost to the customer for one year following installation and will contain information for correcting or bypassing verified problems where possible, and will give notice of availability of corrected software.

Any software updates released by Tektronix, Inc., to correct problems during the one (1) year period will be provided to the customer at no charge on the standard distribution media specified in the software documentation. If media other than the standard distribution media is requested, the customer will only be charged for the current cost of the optional media.

SOFTWARE LICENSE

This software product, including subsequent improvements or updates, is furnished under a license for use on a single controller. It may only be copied, in whole or in part (with the proper inclusion of the Tektronix, Inc., copyright notice on the software), for use on that specific controller.

Specification and price change privileges are reserved.

Although the material in this manual has been thoroughly edited and checked for accuracy, Tektronix, Inc., makes no guarantees against typographical or human errors. Also, Tektronix, Inc., assumes no responsibility or liability, consequential or otherwise, of any kind arising from misinterpretation or misuse of the material in this manual. The contents of this manual are subject to change without notice.

PREFACE

This manual describes the Graphics Package for TEK SPS BASIC VØ2 and VØ2XM software. Any exception to an option or a capability of a command in this package being supported by a specific release of the software is noted where appropriate.

The manual has been organized according to users' instructional needs in learning to use the TEK SPS BASIC Graphics Package. Section 1 begins with a brief discussion of graphics hardware and an introduction to the two-axis Cartesian coordinate system. This is followed by a tutorial overview of each of the MOVE and DRAW commands used in plotting vectors on a graphics terminal. Finally, the more complex commands for graphing arrays, waveforms, and X-Y plots are discussed.

Section 2 is a reference section and contains a detailed description of each of the graphics commands, listed in alphabetical order. Each command description includes a syntax listing and several examples. This is followed by a discussion of how the command works and its interaction with other graphics commands.

Section 3 provides some advanced techniques and applications to demonstrate the utility of the graphics commands. Several example programs are provided, illustrating such concepts as gray-scale plotting, hidden-line techniques, and interactive graphics. For your convenience, a glossary of graphics terms and a concise summary of command functions are included in the Appendices.

We hope that this manual, though necessarily limited in scope, will provide enough ideas for you to generate your own applications software using graphics. Further information on the mathematical foundations of computer graphics can be found in the books listed in Appendix C. Help is also available through HANDSHAKE (a signal-processing newsletter published by Tektronix, Inc.) and the TEK SPS BASIC program library. If you discover a novel application for your software and would be willing to share it via the program library, please let us know.* Good luck and happy programming!

*Contributions to HANDSHAKE or the program library can be mailed to the SPS DOCUMENTATION GROUP (Group 157) P.O. Box 500, Beaverton, OR 97077.

NOTE

The prerequisite software for executing the commands in this package is the corresponding version of the TEK SPS BASIC System Software. The VØ2 package (CP57002) requires the VØ2 System Software (CP57000); the VØ2XM package (CP57502) requires the VØ2XM System Software (CP57500). Once the proper TEK SPS BASIC Monitor/Interpreter has been bootstrapped and loaded into computer memory with the graphics capability retained, any of the graphics commands can be executed immediately.

As an example, the GRAPH command -- one of the most commonly used graphics routines -- can be executed either in immediate mode (by entering a statement like GRAPH A) or in program mode (by entering a line such as 1ØØ GRAPH B). In either case, the GRAPH command automatically becomes resident in computer memory when it is first referenced. However, a command so autoloaded may be automatically released from memory if required to make room for new commands to be autoloaded.

Any of the graphics commands can also be loaded into memory via the LOAD command (e.g., LOAD "GRAPH") and then executed. Each command so LOADed will then be permanently resident in computer memory.

TABLE OF CONTENTS

@

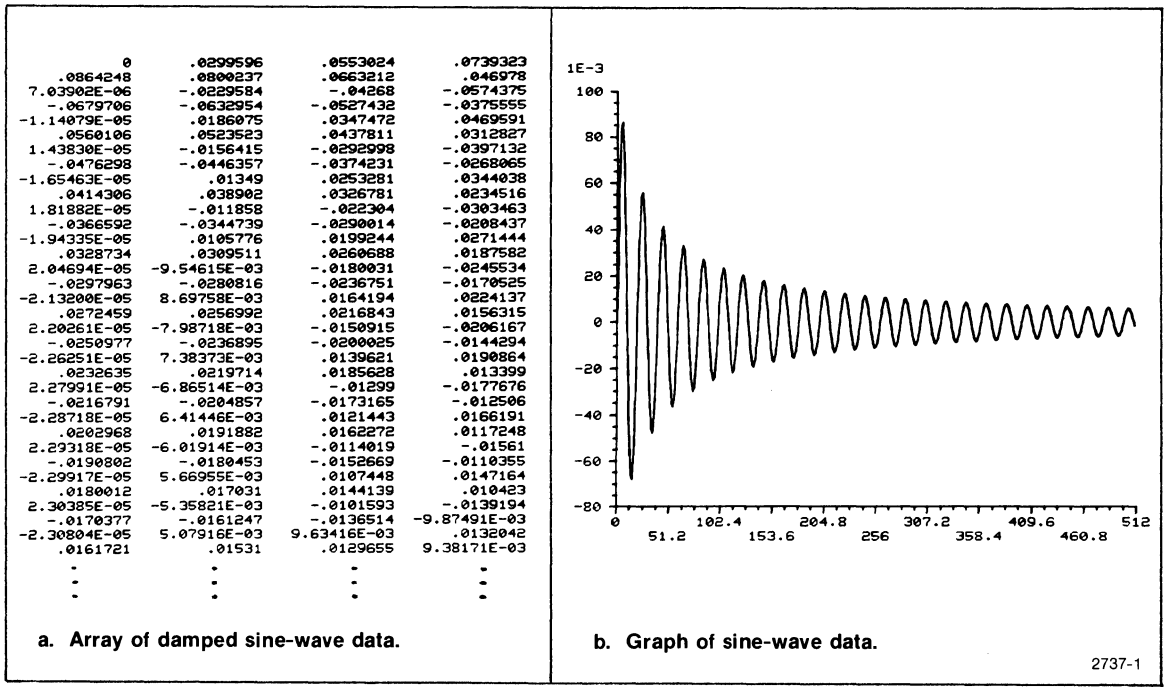
SECTION 1

THE BASICS OF GRAPHICS

The old adage "a picture is worth a thousand words" may be a bit hackneyed but it is still true -- particularly in the realm of science and engineering. An important relationship involving two or more variables may be difficult, if not impossible, to understand when presented as an array of numbers. Yet, by means of a graph or chart, the same relationship or trend can often be recognized at a glance.

As an example, consider the array of numbers shown in Fig. 1-1a. From examining the raw data, it is difficult to see much of a trend except that the data is cyclic about the x-axis and that it eventually degenerates to a small value. However, from the graph of the data shown in Fig. 1-1b, it is immediately obvious that the data is a damped sine wave. In particular, it is a function of the form:

$$F(x) = \frac{\sin x}{x + k}$$



0	.0299596	.0553024	.0739323
.0864248	.0800237	.0663212	.046978
7.03902E-06	-.0229584	-.04268	-.0574375
-.0679706	-.0632954	-.0527432	-.0375555
-1.14079E-05	.0186075	.0347472	.0469591
.0560106	.0523523	.0437811	.0312827
1.43830E-05	-.0156415	-.0292998	-.0397132
-.0476298	-.0446357	-.0374231	-.0268065
-1.65463E-05	.01349	.0253281	.0344038
.0414306	.038902	.0326781	.0234516
1.81882E-05	-.011858	-.022304	-.0303463
-.0366592	-.0344739	-.0290014	-.0208437
-1.94335E-05	.0105776	.0199244	.0271444
.0328734	.0309511	.0260688	.0187582
2.04694E-05	-9.54615E-03	-.0180031	-.0245534
-.0297963	-.0280816	-.0236751	-.0170525
-2.13200E-05	8.69758E-03	.0164194	.0224137
.0272459	.0256992	.0216843	.0156315
2.20261E-05	-7.98718E-03	-.0150915	-.0206167
-.0250977	-.0236895	-.0200025	-.0144294
-2.26251E-05	7.38373E-03	.0139621	.0190864
.0232635	.0219714	.0185628	.013399
2.27991E-05	-6.86514E-03	-.01299	-.0177676
-.0216791	-.0204857	-.0173165	-.012506
-2.28718E-05	6.41446E-03	.0121443	.0166191
.0202968	.0191882	.0162272	.0117248
2.29318E-05	-6.01914E-03	-.0114019	-.01561
-.0190802	-.0180453	-.0152669	-.0110355
-2.29917E-05	5.66955E-03	.0107448	.0147164
.0180012	.017031	.0144139	.010423
2.30385E-05	-5.35821E-03	-.0101593	-.0139194
-.0170377	-.0161247	-.0136514	-9.87491E-03
-2.30804E-05	5.07916E-03	9.63416E-03	.013204
.0161721	.01531	.0129655	9.38171E-03
.	.	.	.
.	.	.	.
.	.	.	.

a. Array of damped sine-wave data.

b. Graph of sine-wave data.

2737-1

Fig. 1-1. Comparison of tabular and graphic data presentation.

As recently as 15 years ago, computer graphics of the type shown in Fig. 1-1 were either very crude or very expensive. Since the primary input/output device was usually a TELETYPE or similar device, graphics had to be simulated by typing a series of dots or X's in the shape of the data plot (see Fig. 1-2).

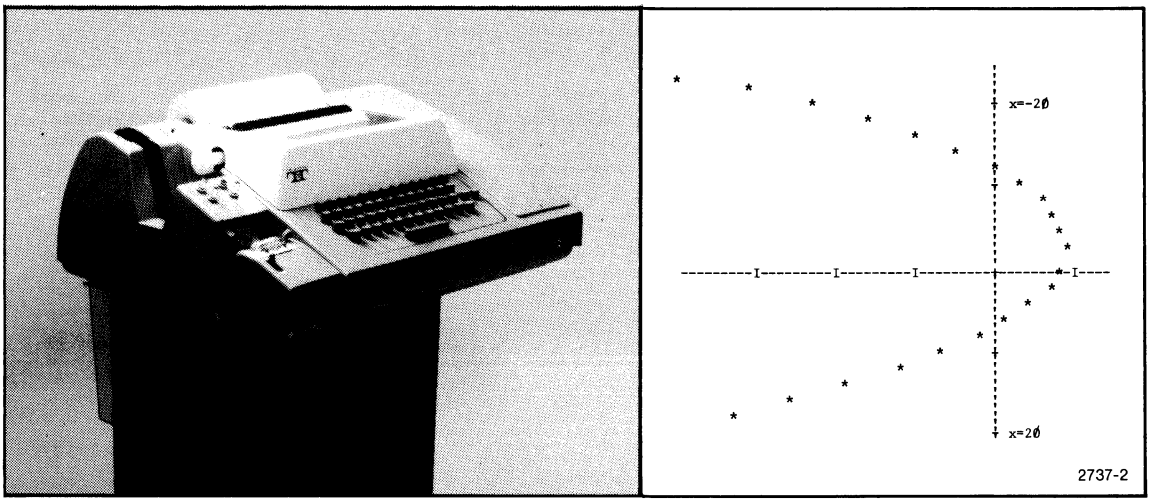


Fig. 1-2. A TELETYPE and its simulated data plot.

Eventually, several computer terminals were engineered with graphics capability. Most of these used television tubes and produced graphics by continually refreshing the image momentarily stored on the phosphorus screen. These terminals were a big improvement over the older methods in that the display could be produced and updated very quickly. Also, they generally provided better resolution and did not require a hard copy to be generated (costly and time-consuming) for every data plot. However, these improvements came only at a cost that was prohibitive for most applications.

A major breakthrough came in 1969 when Tektronix introduced the T4002 Computer Display Terminal. This was the first commercially available graphics terminal for under $10,000 -- a price breakthrough made possible by the Direct View Storage Tube (DVST). Since the display tube itself was the primary storage device, the terminal did not require vast amounts of memory for refreshing the display. Today, a DVST-type terminal can be purchased at less than a third of the original cost of the T4002.

Graphics Hardware

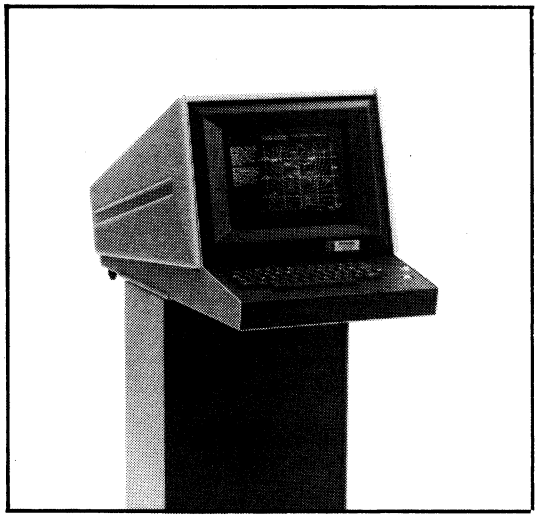
Choices of Output Hardware

At present, there is a wide variety of commercially available graphics terminals. Many of these still use the DVST to create a stored image. However, the refreshed graphics terminal has staged a dramatic comeback due to the falling prices of semiconductor memory chips (required for storing the graphic information). Also, the refreshed graphics terminal permits selective erasure of the terminal screen, thereby allowing the display to be updated quickly.

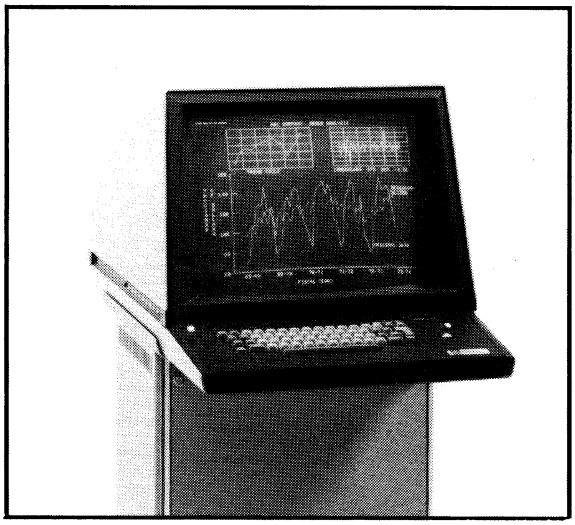
TEK SPS BASIC was specifically designed to support the TEKTRONIX 4010 Computer Display Terminal (Fig. 1-3). However, most features of the larger DVST-type terminals such as the TEKTRONIX 4014 Graphic Display Terminal can also be used with this software. TEK SPS BASIC was not specifically designed for the newer refreshed-type terminals such as the TEKTRONIX 4025, and its color version, the 4027. However, as these terminals become increasingly popular, they may see some use with Tektronix signal processing systems.

While most examples in this text assume the use of a graphics terminal as the primary output device, there are other possibilities. If you want to make a permanent copy of the display produced on the terminal, this can be done with a hard copy unit such as the TEKTRONIX 4631 (Fig. 1-4). The 4631 Hard Copy Unit produces an 8 1/2" X 11" sheet which contains an exact black and white replica of the information produced on the terminal.

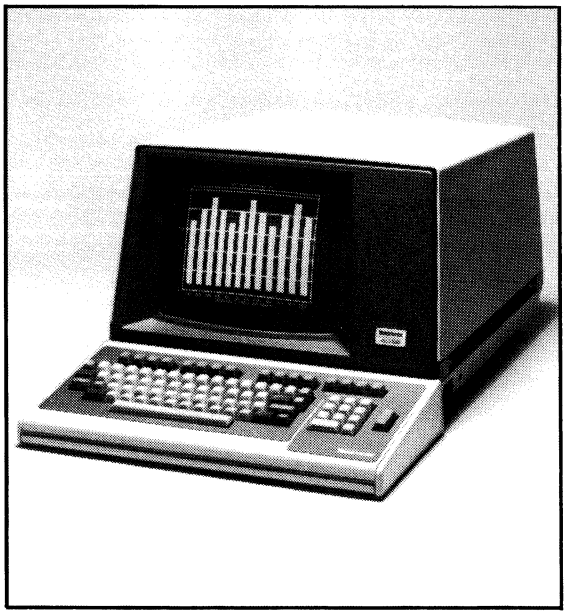
Another output possibility is a digital plotter, such as the TEKTRONIX 4662 Interactive Digital Plotter (Fig. 1-5). When interfaced to a TEKTRONIX Computer Display Terminal, the plotter can produce 11" X 17" sheets that contain all alphanumeric and graphic data displayed on the terminal screen. One advantage of the digital plotter is that different colors can be incorporated in the hard copy. For example, a graph showing instantaneous voltage, current, and power in a transistor could be color coded to permit easy identification of each plot. Another advantage of the plotter is that the information can be written on special graph paper such as log-log, semi-log, or Smith charts. The plots can also be created on mylar so that information can be shown on an overhead projector. Section 3 contains more information on these applications.



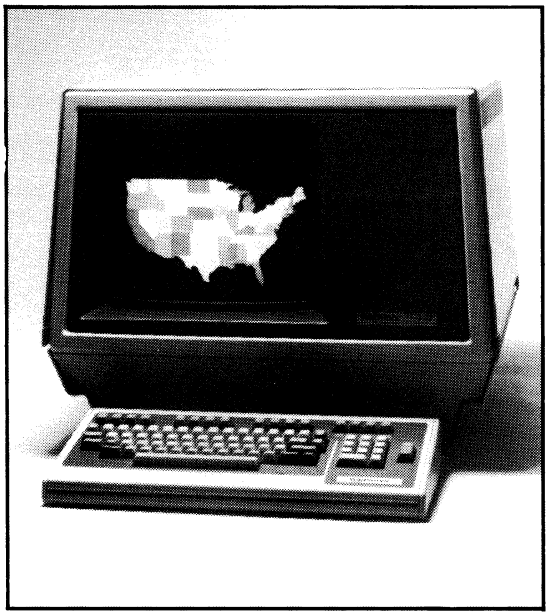
a. TEKTRONIX 4010 -- standard DVST graphics terminal.



b. TEKTRONIX 4014 -- large screen DVST graphics terminal.



c. TEKTRONIX 4025 -- refreshed graphics terminal, black & white.



d. TEKTRONIX 4027 -- refreshed graphics terminal, color.

2737-3

Fig. 1-3. Some common TEKTRONIX graphics terminals.

Fig. 1-4. The TEKTRONIX 4631 Hard Copy Unit.

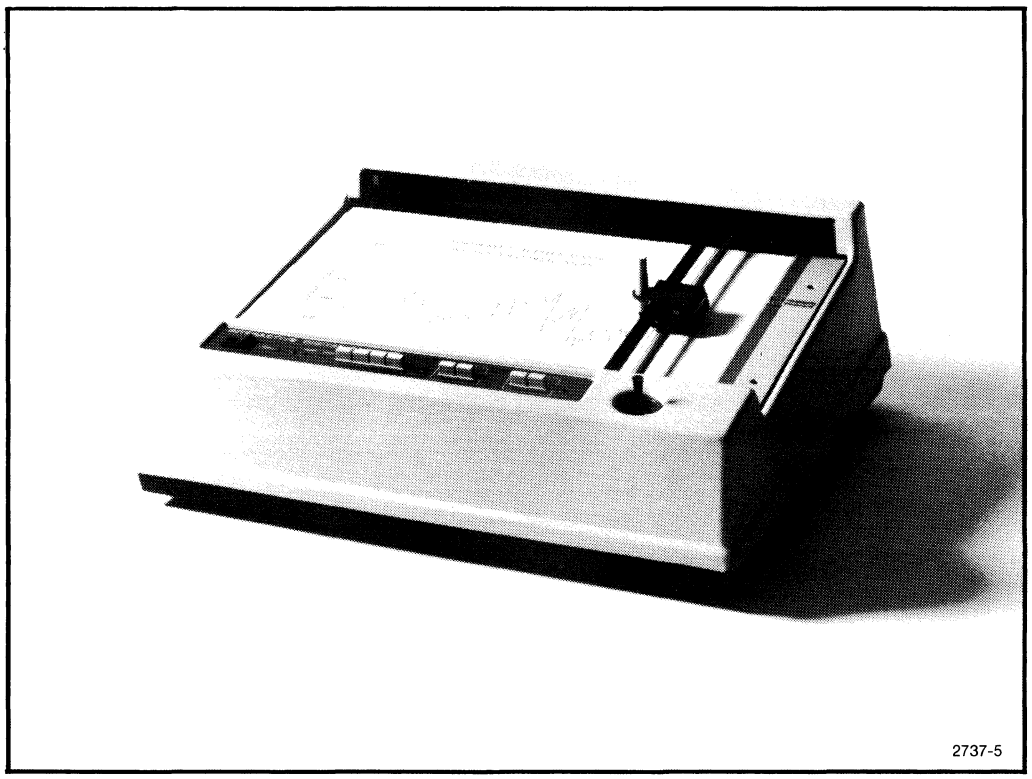


Fig. 1-5. The TEKTRONIX 4662 Interactive Digital Plotter.

Choices of Input Hardware

The data to be displayed via the Graphics Package can arise from a variety of inputs. In some cases, it may be generated internally by the controller (minicomputer). Normally, though, the data is generated from a signal digitizer. The controller is then used to convert the data into a more palatable form. Most signal digitizers directly acquire the analog input signal, via an appropriate transducer, and then perform the required analog-to-digital conversion. Examples of these types of instruments are the TEKTRONIX Digitizing Oscilloscope and TEKTRONIX 7912AD Programmable Digitizer.

There is another type of digitizer, known as the graphics tablet, that digitizes the data from a previously produced graph, chart, or CRT photo. Any type of relationship that can be represented in graphic form can then be input to the controller for computer processing. In some cases, the graphics tablet can provide a "quick and dirty" method for inputting data from an instrument that might be difficult to interface to a computer. Figure 1-6 shows a picture of the TEKTRONIX 4953 Graphics Tablet.

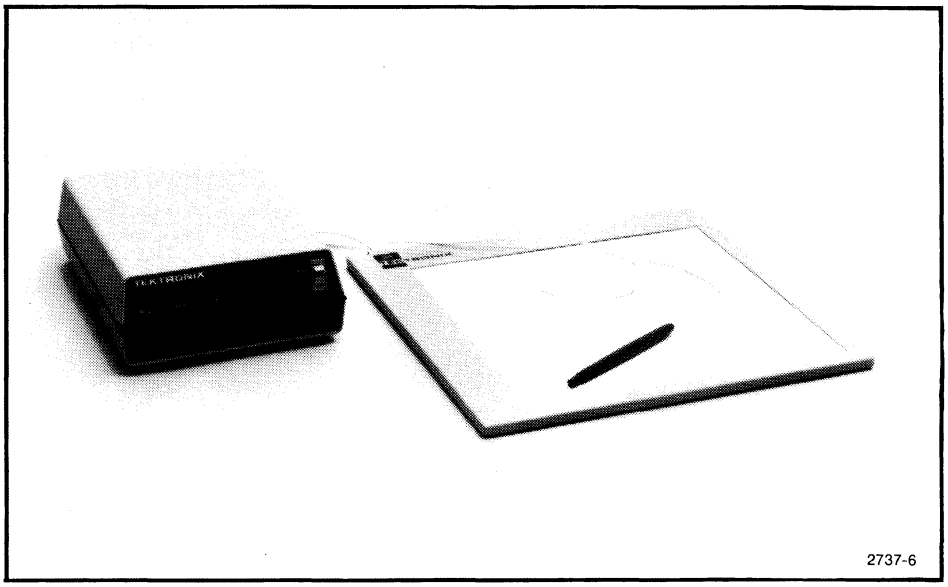


Fig. 1-6. The TEKTRONIX 4953 Graphics Tablet.

This has been only a brief introduction to graphics hardware and its applications. If you would like more information on these and other graphics input and output devices, contact your nearest Tektronix Field engineer.

The Cartesian Coordinate System

The Cartesian coordinate system is briefly reviewed here because of its importance in understanding the various types of TEK SPS BASIC move and draw commands. It is really just a convenient way of denoting the location of a point in a plane.

A 2-axis Cartesian system consists of two perpendicular lines called axes. By convention, the vertical line is called the Y-axis, and the horizontal line the X-axis. Each axis is divided into an arbitrary number of equally spaced divisions which serve as the reference grid for locating points in the plane. Fig. 1-7 shows a typical Cartesian system.

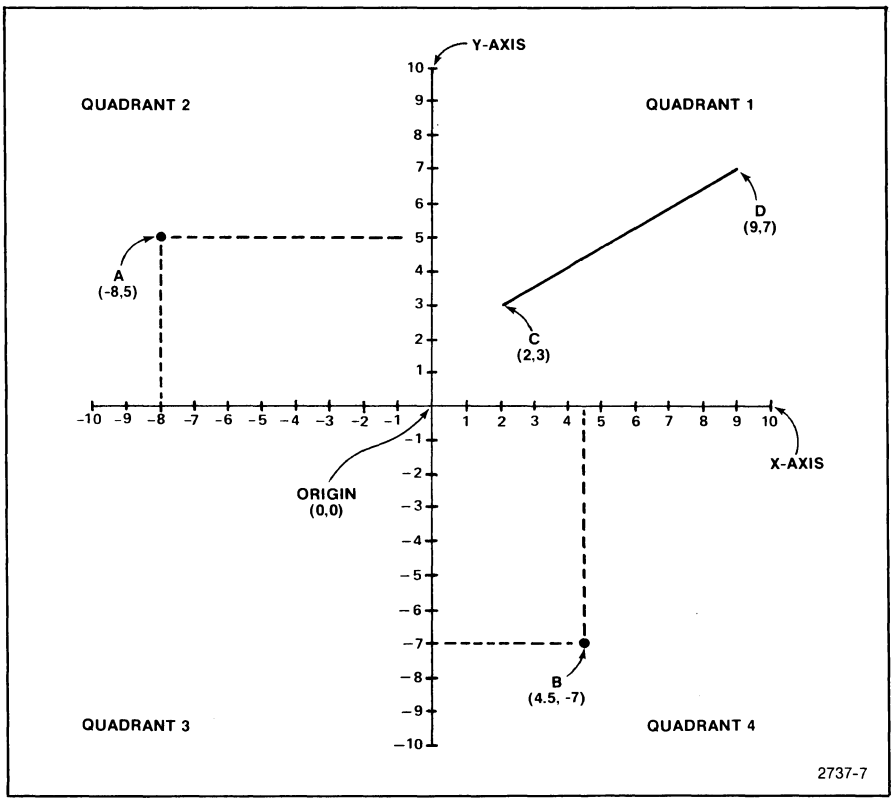


Fig. 1-7. A typical Cartesian coordinate system.

The location of any point P in the coordinate system can be specified by a unique set of ordered pairs (x_p,y_p), where x_p denotes the horizontal location of the point and y_p denotes the vertical location of the point. Both of these distances are measured from the origin (0,0). As an example, the coordinates of point A in Fig. 1-7 are (-8,5). That is, the horizontal location, x_a, is -8 and the vertical location, y_a, is 5. Similarly, the coordinates of point B are (4.5,-7).

In addition to specifying the location of points, coordinate pairs can also be used to designate the position and length of vectors or line segments. For example, the vector $\overrightarrow{CD}$ can be specified by the coordinate pairs (2,3) and (9,7). The horizontal distance between C and D is:

$$x_d - x_c = 9-2 = 7$$

The vertical distance between C and D is:

$$y_d - y_c = 7-3 = 4$$

The total length of the vector, as given by the Pythagorean theorem, is:

$$CD = \sqrt{(x_d - x_c)^2 + (y_d - y_c)^2} = \sqrt{7^2 + 4^2} = \sqrt{65} \simeq 8.1$$

The scales for the X and Y axes on a Cartesian system can have any convenient range, provided that linearity is maintained. For example, the scales shown in Fig. 1-7 could be multiplied by a factor of 10, if desired. Then the coordinates of point A would be (-80,50) rather than (-8,5), and the length of $\overrightarrow{CD}$ would be about 81 rather than 8.1.

The Terminal Coordinate System

Screen Coordinates

The concepts just discussed are fairly elementary, but are essential to a clear understanding of the TEK SPS BASIC graphics commands. This is because the TEKTRONIX 4010 Computer Display Terminal is addressed as the first quadrant of a Cartesian coordinate system. The range of this system is 0 to 1023 in the X-axis and 0 to 779 in the Y-axis, as shown in Fig. 1-8. (In reality, the Y-axis addressable range is 0 to 1023, but only points 0 through 779 are normally displayable on a calibrated terminal.)

To further clarify, the boundary of the screen of the computer display terminal is formed by four sets of coordinates:

LOWER-LEFT CORNER: (0,0)
LOWER-RIGHT CORNER: (1023,0)
UPPER-RIGHT CORNER: (1023,779)
UPPER-LEFT CORNER: (0,779)

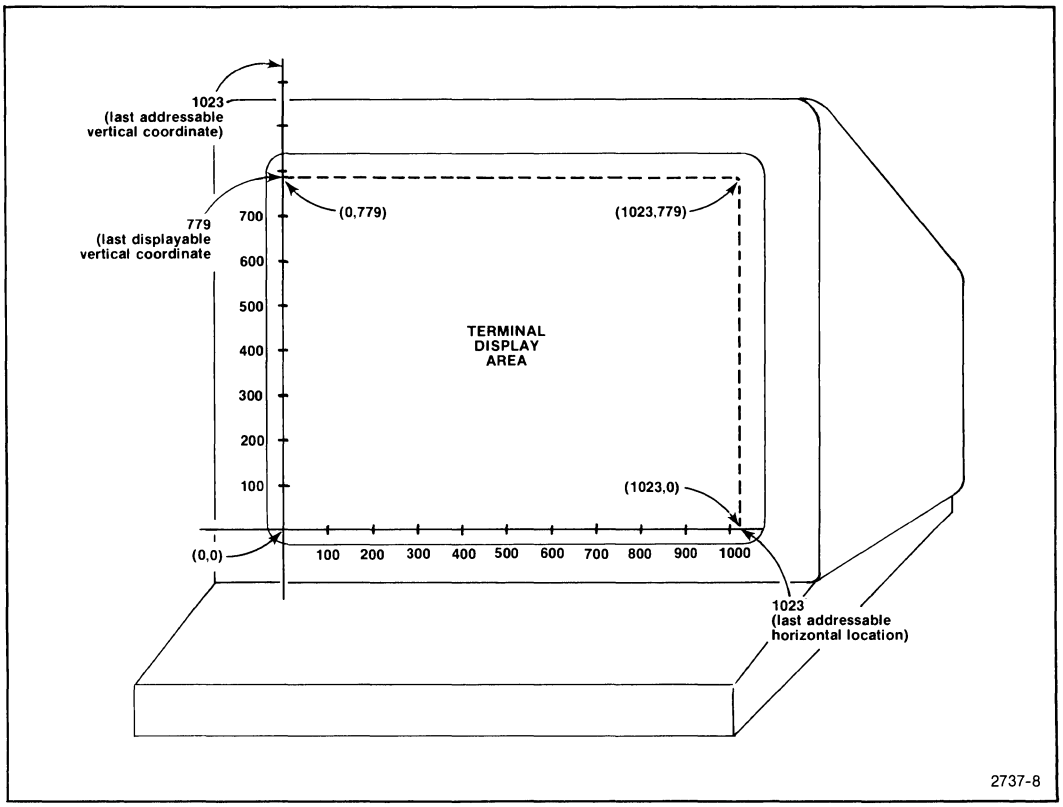


Fig. 1-8. The TEKTRONIX Computer Display Terminal and its addressable display area.

Since the TEKTRONIX 4010 terminal has an address range of 0 to 1023 in the X-axis and 0 to 779 in the Y-axis, it is convenient to refer to these individual points as "screen coordinates" or "device coordinates". Similarly, we shall hereafter refer to the distance between successive "device coordinates" as a "device unit".

A device unit is defined as 1/1023 of the horizontal address area of the terminal screen. In other words, the displayable area of the terminal screen is 1023 device units wide and 779 device units high. Since the display area of the screen is about 19.1 cm X 14.3 cm (7.52 in. X 5.63

in.) a device unit corresponds to about 0.183 mm (0.0072 in.). As we shall see shortly, the device unit is an important unit of measure on Tektronix terminals.

Enhanced Graphics. A TEKTRONIX 4014 Computer Display Terminal with the Enhanced Graphics Module has screen coordinates that range from 0 to 4095 for both the X and Y axes. (Only 0 through 3114 in the Y-axis are viewable.) This high-resolution graphics capability is supported by the Enhanced Graphics Keyboard driver (KBE.SPS), which can be optionally selected as the system keyboard driver. (See the SYSBLD command in Section 4 of the System Software manual for details.) KBE.SPS maps the TEK SPS BASIC device units to the device units of an enhanced 4014. So, when using KBE.SPS, there is no need to alter your BASIC program.

Absolute Moves and Draws

In discussing how the Computer Display Terminal plots images, it is helpful to use the analogy of a pencil and paper. Let's suppose you wanted to print the letter "X" in the space provided below:

Though this is a very simple process (learned by children in grammar school), it involves four separate steps. One possible method is as follows:

1) Move your pencil to the upper-right corner of the box.
2) Draw a line to the lower-left corner of the box.
3) Move your pencil to the upper-left corner of the box.
4) Draw a line to the lower-right corner of the box.

Another possible method is:

1) Move pencil to lower-left corner.
2) Draw line to upper-right corner.
3) Move pencil to upper-left corner.
4) Draw line to lower-right corner.

There are several other ways to draw the "X" depending on where the pencil is initially moved to and which directions the lines are drawn. The method is not really of concern here; what is of importance is to note

that there are two essential processes: move and draw. Not only is this true for printing letters on a paper or blackboard, but the same basic process is used in programming a graphics terminal or digital plotter. A digital plotter demonstrates this concept very dramatically by lifting its pen and **mov**ing it to the desired position, then lowering its pen and **draw**ing a line to the desired position. In a DVST graphics terminal, the idea is the same except that an electron beam is turned on or off while being deflected to the desired position.

As one might guess, there are two TEK SPS BASIC commands for drawing lines on a graphics terminal: MOVE and DRAW. The MOVE command positions the electron beam at the start of the line to be drawn. The DRAW command then instructs the terminal to draw a line from this location to the location specified in the DRAW command.

The MOVE and DRAW commands operate on device coordinates (mentioned previously) until a WINDOW or VIEWPORT command has been executed; from then on, they operate on user coordinates (more about that later). An example of the MOVE command is:

```
MOVE 1Ø23,779
```

Assuming that WINDOW or VIEWPORT have not been executed, the above statement would move the pointer to the upper-right corner (1023,779) of the terminal display area. Notice that the argument preceding the comma, 1023, specifies the X-coordinate, and the argument following the comma, 779, specifies the Y-coordinate.

The DRAW command has a similar format. An example is:

```
DRAW 1Ø23,Ø
```

This statement draws a line segment from the previous pointer position (1023,779) to the position specified in the DRAW command (1023,0). In this case, a vertical line is drawn along the right-hand boundary of the display area. As with the MOVE command, the argument preceding the comma specifies the X-coordinate, and the argument following the comma specifies the Y-coordinate.

As another example, consider the following program. It draws a big "X" on the terminal screen extending from corner to corner. The intersection indicates the approximate center of the display area.

```
1Ø MOVE Ø,Ø
2Ø DRAW 1Ø23,779
3Ø MOVE 1Ø23,Ø
4Ø DRAW Ø,779
```

Notice that the preceding program began with a MOVE command. Not only does it move the pointer to the lower-left corner of the display, but it also sets the terminal to graphics plot mode. And this is an important point to remember: **It is permissable to mix graphics and alphanumerics in the same program, but a MOVE command must be executed whenever returning the terminal to graphics plot mode.**

It is possible to plot more than one vector with a single DRAW command. In this case, the DRAW command will have pairs of arguments corresponding to the coordinates of each vector to be plotted. This is demonstrated in the following program which draws a square near the center of the screen. The size of the square will be 100 device units to a side.

```
1ØØ REM MOVE TO CENTER OF SCREEN
11Ø MOVE 511,39Ø
12Ø REM NOW DRAW THE SQUARE
13Ø DRAW 611,39Ø,611,49Ø,511,49Ø,511,39Ø
```

As previously mentioned, the first graphics command executed must be a form of the MOVE command (i.e., MOVE, RMOVE, RSMOVE, or SMOVE). Line 110 moves the writing beam to center screen. No line is drawn, however. Line 130 does the actual drawing. This command draws four separate vectors, one for each side of the square (see Fig. 1-9).

Relative Moves and Draws

The previously discussed MOVE and DRAW commands are often called **absolute** MOVE and DRAW commands because they draw a vector to an absolute coordinate set on the terminal screen. It is frequently convenient, however, to move the pointer or draw a vector relative to the previous position of the pointer. The RMOVE and RDRAW commands provide this capability. The arguments specified in these commands are added to the current coordinates of the pointer to create a new coordinate pair.

The RMOVE and RDRAW commands free the user from having to figure out the absolute coordinates of each new data point. For example, if the next vector to be drawn is a point 5 device units to the right and 10 device

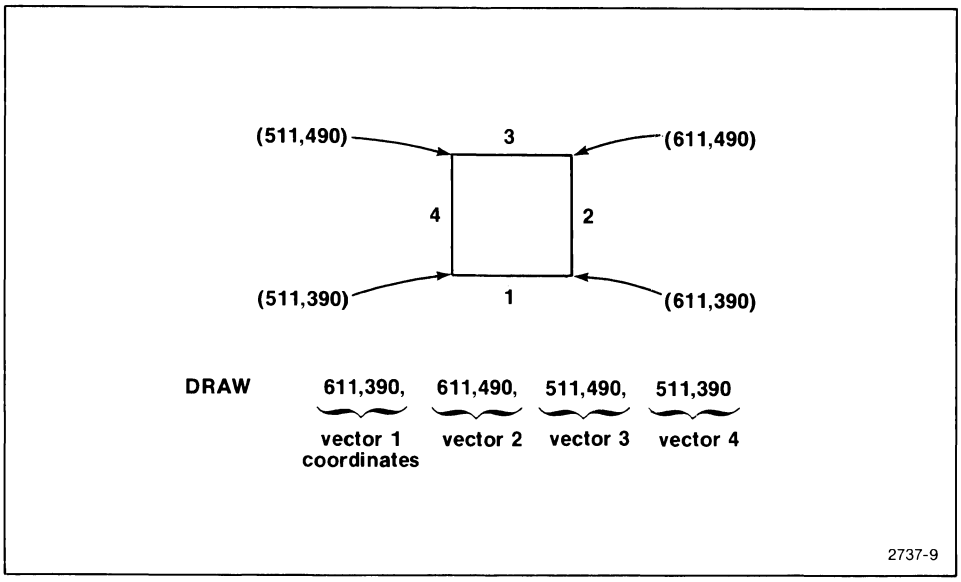


Fig. 1-9. The arguments of the DRAW command are a set of ordered pairs that specify the ending coordinates of vectors to be drawn.

units above the current pointer position, then these increments can be specified by the following statement:

```
RDRAW 5,1Ø
```

TEK SPS BASIC computes the absolute coordinates of the destination point and then draws a vector to that point. Like the MOVE and DRAW commands, RMOVE and RDRAW operate on device coordinates until a WINDOW or VIEWPORT command has been executed. From then on, they operate on user coordinates.

The first argument in an RMOVE or RDRAW command specifies the horizontal adjustment in the X-coordinate. A positive value causes a displacement to the right, and a negative value causes a displacement to the left. The second argument specifies the vertical adjustment in the Y-coordinate. A positive value causes an upward displacement, and a negative value causes a downward displacement. All movement is relative to the present position of the pointer.

To demonstrate these concepts, let's review the square-drawing program, this time using RDRAW. Here is the revised program:

```
1ØØ REM MOVE POINTER TO CENTER SCREEN
11Ø MOVE 511,39Ø
12Ø REM DRAW BOX RELATIVE TO CURRENT POSITION
13Ø RDRAW 1ØØ,Ø,Ø,1ØØ,-1ØØ,Ø,Ø,-1ØØ
```

Up to line 120, the program is essentially the same as the previous example. At line 130, the RDRAW command has been substituted for the DRAW command. The arguments for RDRAW are displacements for both the X and Y location of the beam. The first X and Y coordinate set (100,0) instructs the software to draw a vector to a point 100 units to the right of the beam, and no units up or down. The next set (0,100) moves the beam up 100 units, but doesn't move it horizontally. The third coordinate pair (-100,0) moves the beam 100 units to the left, but doesn't move it vertically. The last coordinate pair (0,-100) returns the beam back to its starting point. Remember, each time the beam moves (once for each coordinate pair), its new position becomes the base for the next relative coordinate pair.

Windows and Viewports

Windowing Your Data

It was noted previously that the Computer Display Terminal has a data range of 0 to 1023 in the X-axis and 0 to 779 in the Y-axis. While this data range provides sufficient resolution for most applications, it rarely provides the most convenient data range. That is, the data is usually much larger or much smaller than the terminal coordinate system allows. Thus some sort of transformation must be performed that will scale the user data to fit the terminal coordinate system. With the WINDOW command, this transformation is accomplished automatically in both the X and Y axes.

The WINDOW command specifies what data values mark the boundaries of the display. It requires four arguments: a minimum and maximum X-coordinate and a minimum and maximum Y-coordinate, respectively. An example of a typical WINDOW command is:

```
WINDOW -5Ø,5Ø,-1,1
```

This statement creates the user-coordinate system that ranges from -50 to 50 in the X-axis and -1 to 1 in the Y-axis. Upon executing this statement, the transformation shown in Fig. 1-10 is performed.

After executing the preceding WINDOW command, a MOVE -50,-1 command would place the pointer at the lower-left corner of the terminal display area, and a MOVE 50,1 command would place the pointer at the upper-right corner. Furthermore, the concatenated commands:

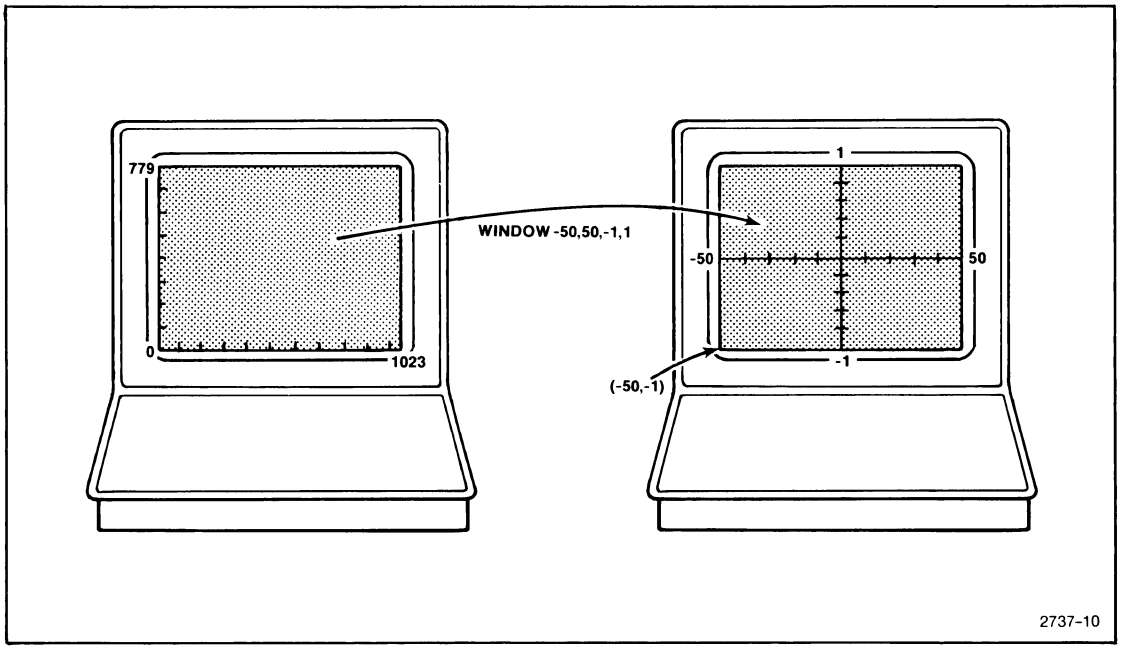


Fig. 1-10. The transformation performed by WINDOW -50,50,-1,1.

```
MOVE Ø,Ø\DRAW 5Ø,-1
```

would draw a vector from center screen (0,0) to the lower-right corner of the display (50,-1).

Let's try another example. Suppose that we wanted to plot the average income of dentists from the years 1958 through 1978. In this case, our X-values would range from 1958 to 1978, and our Y-values (average income) might range from 10,000 to 100,000. Thus the following statement would establish an appropriate data window.

```
WINDOW 1958,1978,1ØE+4,1ØE+5
```

Even though this statement has specified an X-axis 20 user-units in width, the terminal is still capable of drawing up to 1023 discrete device units in the X-axis. This means that you still have a resolution of about 51 device units per user unit. In other words, if you were to draw a vertical line at each year (1958, 1959, 1960, etc.), the lines would be 51 device units apart. Likewise, since there are about 779 displayable vertical units, it takes about 115 vertical user units (dollars in this case) to equal one device unit.

The data window is set to its default values (0 to 1023 in the X-axis and 0 to 779 in the Y-axis) whenever a RESETG or INITG command is executed. Both commands also set the terminal to alpha mode. The main difference between the two is that INITG erases the terminal screen while RESETG does not. Incidentally, the Tektronix terminal can be erased either manually (by pressing the PAGE key) or under program control (by executing the PAGE command).

Setting the Viewport

In all of the preceding examples, it has been assumed that the entire display screen is to be used as the area for plotting data. It is sometimes necessary, however, to restrict a plot to a designated portion of the terminal screen. The VIEWPORT command allows this to be done; it specifies the minimum and maximum X and Y coordinates (in device units) that will serve as the boundaries for the plotted data. Thus, the VIEWPORT command allows you to treat a selected portion of the screen as though it were the entire screen.

The VIEWPORT command is particularly useful when graphing more than one array or waveform on the same display. By means of the VIEWPORT command, each waveform can be graphed with the same set of commands; only the viewport is changed preceding each plot.

The VIEWPORT command requires four arguments. These arguments are always specified in device coordinates, and designate a rectangular area of the screen where all graphic output occurs. An example of a typical VIEWPORT command is:

```
VIEWPORT 2ØØ,8ØØ,1ØØ,6ØØ
```

The first two arguments (200 and 800) define the minimum and maximum X-coordinates of the viewport, and the last two arguments (100 and 600) define the minimum and maximum Y-coordinates, respectively. After executing the above command, a mapping of the type shown in Fig. 1-11 is performed.

After a VIEWPORT command has been executed, a MOVE or DRAW command will operate just as though the newly defined viewport is the entire screen. In the case of the preceding example (and assuming a WINDOW command has not been executed), a MOVE 0,0 command would move the graphics pointer to absolute screen coordinates (200,100). Similarly,

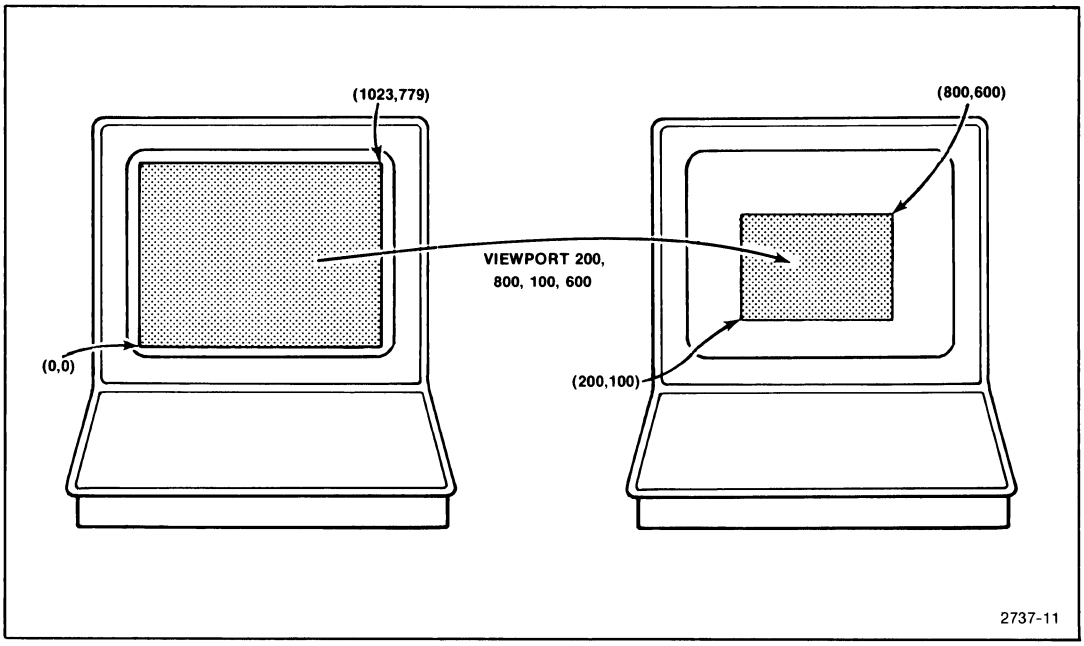


Fig. 1-11. The mapping effected by VIEWPORT 200,800,100,600.

MOVE 1023,779 would move the pointer to absolute screen coordinates (800,600). Furthermore, the concatenated commands:

```
MOVE Ø,779\DRAW 1Ø23,Ø
```

would draw a line from absolute coordinates (200,600) to absolute coordinates (800,100).

The viewport can be defined to any rectangular portion of the screen; it can be tall or short, narrow or wide. Whatever the size, all graphic output is treated such that the newly defined viewport is the entire screen. However, the VIEWPORT command does not clip any data that results in an out-of-viewport condition. The out-of-viewport data is still displayed as long as it is on screen.

Interaction between WINDOW and VIEWPORT

The VIEWPORT command does not affect the current data window (if a WINDOW command has been executed). Likewise, the VIEWPORT is not affected by the WINDOW command. The WINDOW and VIEWPORT commands are mutually independent, but work in unison to affect the range, size, and aspect ratio of the displayed data. In short, we can think of the window as

being the image of the viewport mapped onto the user data space. This is illustrated in Fig. 1-12.

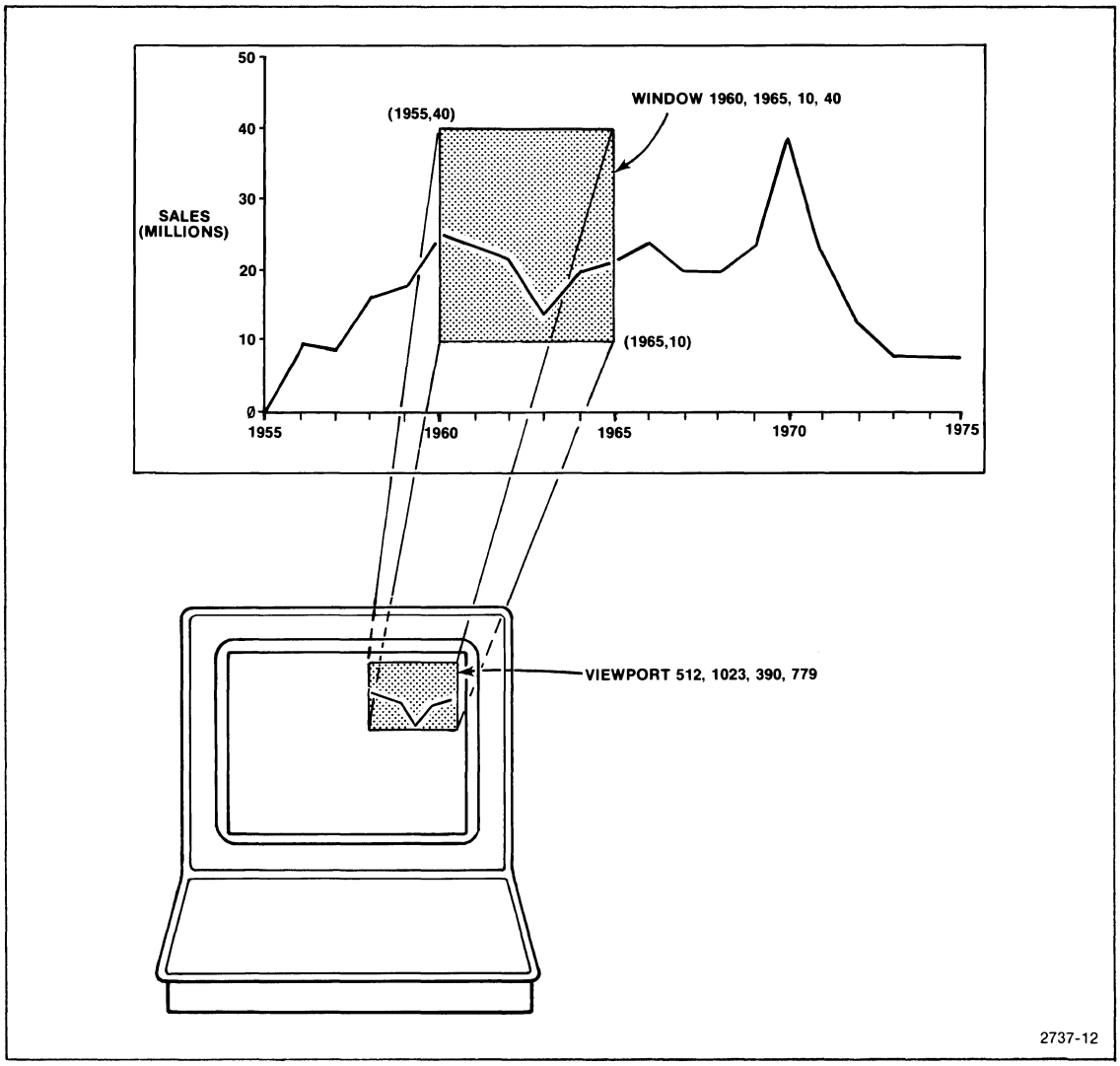


Fig. 1-12. WINDOW and VIEWPORT interact to display the desired data in the selected area of the terminal screen.

In the example of Fig. 1-12, a portion of a sales graph is mapped into a viewport which is defined in the upper-right corner of the terminal screen. The appropriate commands are:

```
VIEWPORT 512,1Ø23,39Ø,779
WINDOW 196Ø,1965,1Ø,4Ø
```

Notice that the viewport parameters are specified in device coordinates. This defines the physical size and location of the drawing area (the upper-right, or first, quadrant of the terminal screen). The viewport is

then divided into user-coordinates with the WINDOW command. In this case, the width of the viewport starts at the year 1960 and ends at 1965. The vertical axis of the viewport is labeled in millions of dollars and ranges from $10 million to $40 million.

To plot a particular vector in the preceding example, you simply specify the year and the amount of sales dollars in a MOVE or DRAW command. For example, to draw a portion of the graph, you could enter

```
MOVE 1962,21\DRAW 1963,14
```

This indicates that the sales were 14 million dollars in 1963 and thus a line is drawn to point (1963,14). The origin of the line segment is the present position of the pointer (the 1962 sales figure).

NOTE

If the arguments of a MOVE, RMOVE, DRAW or RDRAW command specify a destination point outside the designated window, TEK SPS BASIC executes a move or draw to that point, provided the point is on screen. If a draw is executed, that portion of the vector outside the window is still drawn -- it is **not** clipped at the edge of the viewport.

Some Example Programs

The VIEWPORT command is quite useful when you want to create a series of plots, all with the same general shape, but each having a different size, location, or aspect ratio. This is demonstrated in the following program which simulates the appearance of a tunnel on the terminal screen.

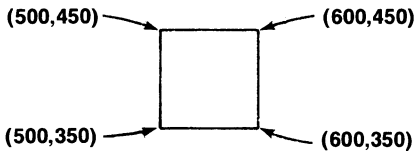
```
1Ø X=55Ø\Y=4ØØ
2Ø Z=5Ø
3Ø WINDOW Ø,1ØØ,Ø,1ØØ
4Ø VIEWPORT X-Z,X+Z,Y-Z,Y+Z
5Ø MOVE Ø,Ø
6Ø DRAW Ø,1ØØ,1ØØ,1ØØ,1ØØ,Ø,Ø,Ø
7Ø Z=Z+1Ø
```

```
8Ø IF X+Z<9ØØ THEN 4Ø
9Ø MOVE 44,5Ø
1ØØ PRINT "TUNNEL"
```

The program begins by defining a viewport which is initially set to:

```
VIEWPORT 5ØØ,6ØØ,35Ø,45Ø
```

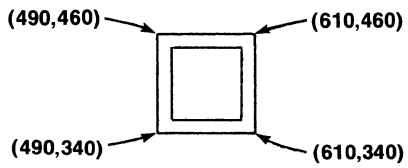
as shown in the following illustration:



The WINDOW, VIEWPORT, MOVE, and DRAW commands at lines 30 through 60 result in the drawing of a square that shows the actual boundaries of the viewport. At line 70, variable Z is incremented by 10 (from 50 to 60). Since X+Z is less than 900, control passes back to line 40 which redefines the viewport as:

```
VIEWPORT 49Ø,61Ø,34Ø,46Ø
```

Again, lines 40 through 60 replot the boundaries of the new viewport as shown in the illustration below:



After Z is incremented a second time, control passes back to line 40 and the loop is executed a third time.

The process of incrementing the viewport and drawing its boundaries is repeated again and again until X+Z>900. At that point, the loop is terminated, and a MOVE to user coordinates (44,50) occurs where the word "TUNNEL" is printed. Figure 1-13 shows the completed drawing.

As a second example, let's try a program in which the height of the viewport is constantly decreasing while the width of the viewport is constantly increasing. Also, rather than plotting the boundary of the viewport, let's plot a circle that just touches the viewport boundaries.

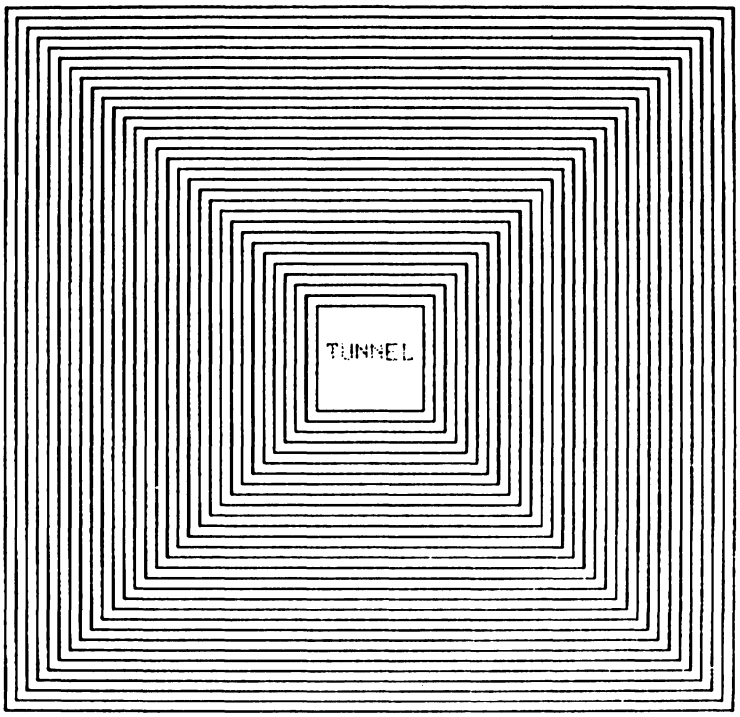


Fig. 1-13. Output of program showing a changing viewport.

The following program does just that:

```
1Ø X=Ø
2Ø VIEWPORT 3ØØ-X,8ØØ+X,2ØØ+X,7ØØ-X
3Ø WINDOW -1,1,-1,1
4Ø MOVE 1,Ø
5Ø FOR I=Ø TO 6.3 STEP .Ø2          } plots
6Ø DRAW COS(I),SIN(I)                  circle
7Ø NEXT I                           }
8Ø X=X+1Ø
9Ø IF X<18Ø THEN 2Ø
1ØØ END
```

Lines 10 and 20 initially result in the square viewport defined by:

```
VIEWPORT 3ØØ,8ØØ,2ØØ,7ØØ
```

Lines 30 through 70 then draw a circle that just touches the boundaries of the viewport. The WINDOW command at line 30 defines the data range of the viewport and line 40 then moves the graphics pointer to the middle of the right-hand boundary of the viewport. Lines 50 through 70 do the actual plotting of the circle. (The range of variable I is 0 to 2π since the SIN and COS functions operate on radians rather than degrees.)

Line 80 increments X which results in a change in the viewport on each pass through the loop. When X has reached a value of 180, the program ends at line 100. Figure 1-14a shows the output produced by the program.

Figure 1-14b shows a similar display. This second display was produced by the same program, but with lines 10 and 20 modified as follows:

```
1Ø X=-1ØØ
2Ø VIEWPORT 3ØØ-X,8ØØ+X,15Ø+X,65Ø-X
```

Variations of Fig. 14 can be produced merely by changing the initial parameters specified in lines 10 and 20.

Seeing the Window and Viewport

The preceding program illustrates the fact that the window or viewport often changes during the course of program execution. It is sometimes convenient to be able to ascertain the window or viewport parameters, at any given time. The SEEWINDOW and SEEVIEW commands provide this capability. Each command has an argument string consisting of four variables.

When the SEEWINDOW command is executed, the first two variables contain the minimum and maximum X-values of the window, and the last two variables contain the minimum and maximum Y-values, respectively. The SEEVIEW command has a similar format. Its first two variables contain the minimum and maximum X-coordinates of the viewport, and the last two variables contain the minimum and maximum Y-coordinates, respectively. For more information on these commands, see the section on command descriptions.

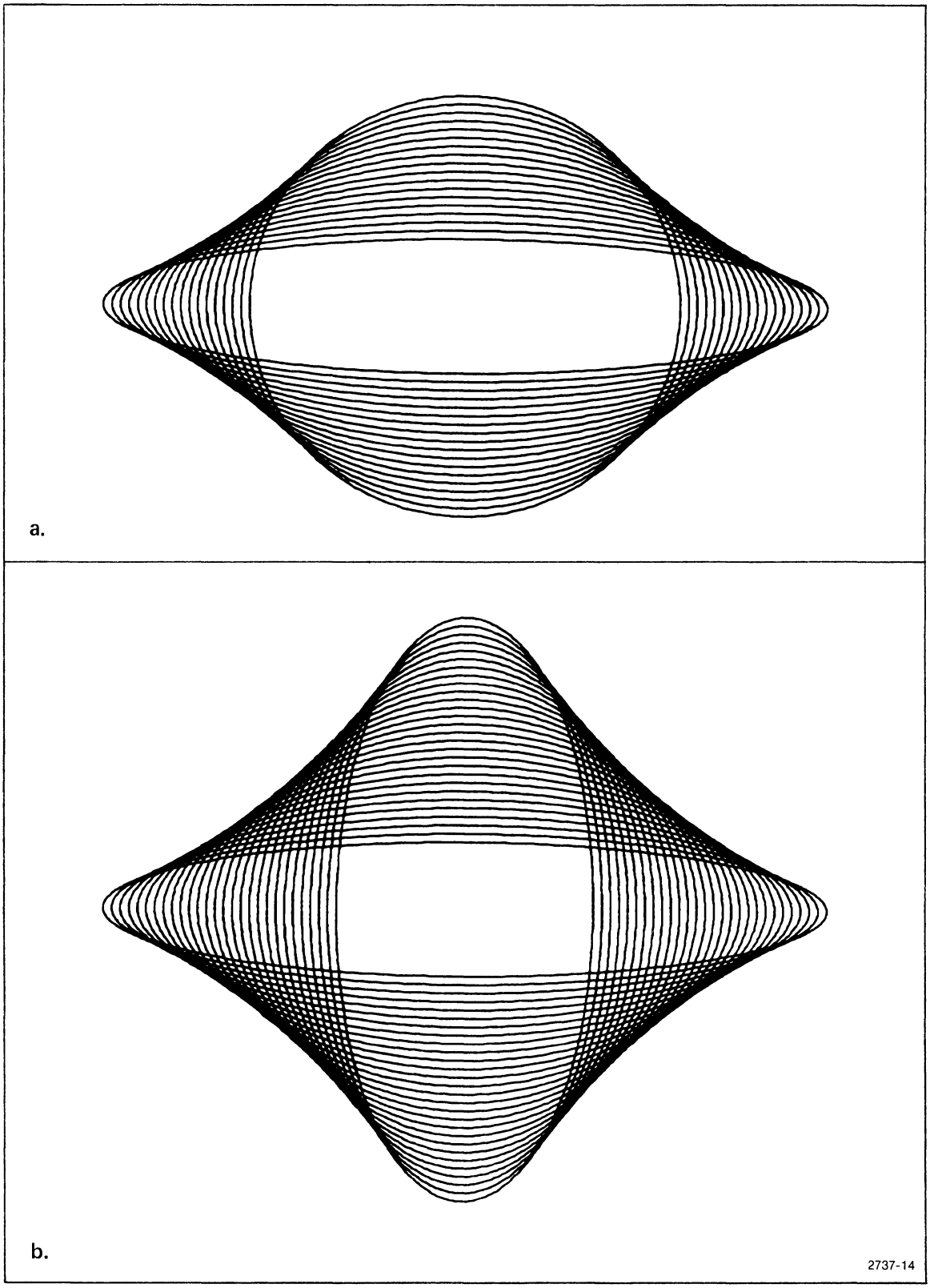


Fig. 1-14. The VIEWPORT command allows a perfect circle to be portrayed as an ellipse.

Moves and Draws in Screen Coordinates

As previously noted, the MOVE, RMOVE, DRAW, and RDRAW commands all operate on device coordinates (also called screen coordinates) until a WINDOW or VIEWPORT command has been executed. After execution of WINDOW or VIEWPORT, the moves and draws are then made with respect to user coordinates until a RESETG or INITG command is executed -- at which time user coordinates are identical to screen coordinates.

It is often desirable to operate with respect to either user coordinates or screen coordinates without having to change the window or viewport when switching from one system to the other. This is made possible by four special commands: SMOVE, SDRAW, RSMOVE, and RSDRAW. These commands operate just like the MOVE, DRAW, RMOVE, and RDRAW commands except the moves and draws are always made in terms of screen (device) coordinates. The following program will demonstrate this more graphically:

```
200 WINDOW -1,1,-1,1
210 MOVE -1,1
220 DRAW 0,0
230 SDRAW 0,0
240 RDRAW 0,2
```

Line 200 assigns a user-coordinate system to the terminal screen; the data range is -1 to 1 in both the X and Y axes (see Fig. 1-15). Line 210 then performs an absolute move to user coordinates (-1,1). This is followed by an absolute draw to user coordinates (0,0), which is the center of the terminal display area. Next, line 230 does an absolute draw to screen coordinates (0,0) -- the lower-left corner of the display area. Finally, line 240 performs a relative draw back to user coordinates (-1,1).

Graphing and Displaying Data

In addition to the more primitive MOVE and DRAW commands just discussed, TEK SPS BASIC includes several "smart" commands specifically designed for graphing arrays and waveforms. One of these commands, GRAPH, plots the entire array or waveform -- complete with labeled axes and tic marks. Another command, DISPLAY, works just like GRAPH except that only the array or waveform plot is drawn -- but no axes or labels. A third command, SETGR, allows you to vary the type and number of tic marks (or divisions) that accompany the axes produced by GRAPH. These three commands provide a great deal of versatility as to the types of graphic

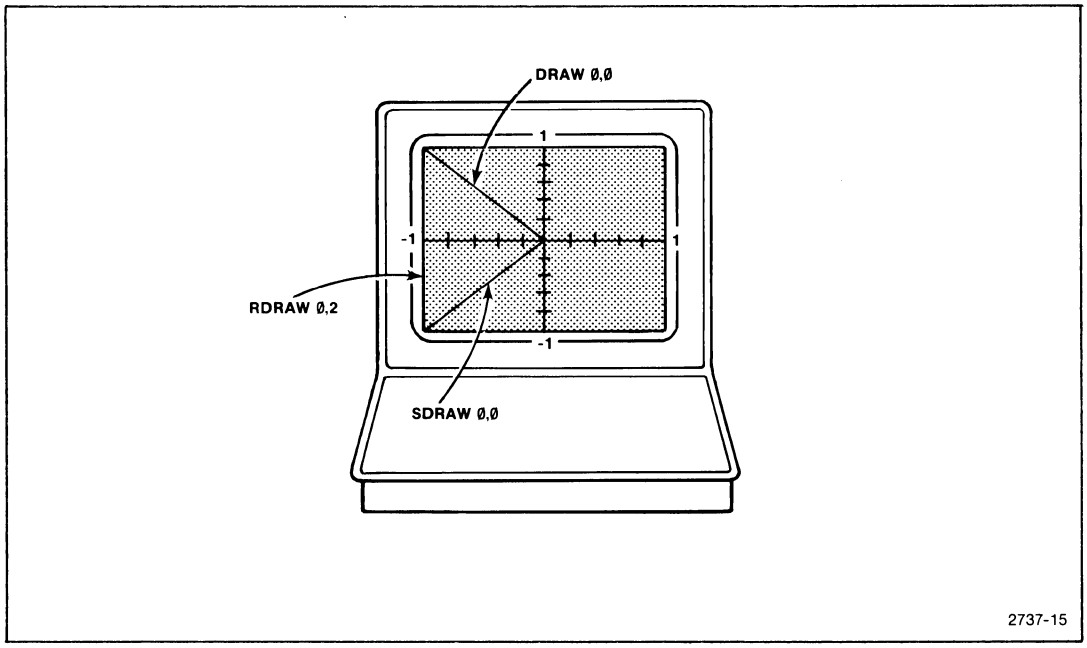


Fig. 1-15. Using the DRAW, SDRAW, and RDRAW commands to create an isosceles triangle.

presentations that are possible. Before discussing these three commands, however, let's briefly review the structure of TEK SPS BASIC arrays and waveforms. This will make the following command descriptions much more meaningful.

A Review of Arrays and Waveforms

Graphs drawn by TEK SPS BASIC present either an array of data or a waveform. Each of these terms has a special meaning with regard to this software. As used in the context of this manual,

> An **array** is a collection of data, each having a common name, data-range, and data-type.

An array can consist of either integer data or floating-point data. An array of integer data can be created by a statement such as the following:

```
INTEGER IN(2Ø)
```

This statement creates an array of 21 elements. The data range of each element is -32768 to 32767 (15 bits plus a sign bit).

A second type or array is the floating-point array. It is created by a DIMension command such as the following:

```
DIM FL(1Ø)
```

This statement creates an 11-element floating-point array. Each element has a 24-bit mantissa and an 8-bit exponent. The largest expressible floating-point number is 1.70141E+38 and the smallest expressible floating-point fraction is ±2.93874E-39.

The concept of a waveform is similar to that of an array. As used in this manual,

> A **waveform** is an array, data-sampling variable, vertical-units string variable, and horizontal-units string variable which have been associated with a variable name for manipulation as a single entity.

The need for a "waveform" (as used in the context of TEK SPS BASIC) arises when acquiring signals from acquisition instruments such as the TEKTRONIX Digitizing Oscilloscope. The array component of the waveform holds the digitized values produced by the instrument. Inherent in this array data is the vertical scale factor of the acquisition instrument. The data-sampling variable holds the timing information related to the instrument's horizontal scale factor. (The data-sampling variable denotes the time interval between acquisition of each successive array element.) The horizontal and vertical string variables simply hold the ASCII units applicable to the horizontal and vertical scale factors, respectively. Typically these string variables will contain information like "V" (for "volts") or "S" (for "seconds").

A TEK SPS BASIC waveform can be classified as either integer or floating-point depending on whether its array component is integer or floating-point. A floating-point waveform can be created by a statement such as:

```
WAVEFORM WA IS A(511), SA, HA$, VA$
```

This statement creates waveform WA which consists of:

1) The 512-element floating-point array A, holding the digitized signal data.

2) The simple variable SA, holding the data-sampling interval.

3) The string variable HA$, holding the horizontal units.

4) The string variable VA$, holding the vertical units.

In the preceding example, the WAVEFORM statement actually dimensions the 512-element array A. Let's suppose, however, that array A had already been dimensioned, and that variable SA and string variables HA$ and VA$ had already been defined. You could still create a TEK SPS BASIC waveform that would associate these entities. For example:

```
WAVEFORM C IS A, SA, HA$, VA$
```

After executing the above statement, the previously defined entities A, SA, HA$, and VA$ would be associated as waveform C. If A was dimensioned with a DIM command, then C will be a floating-point waveform. However, if A was dimensioned with an INTEGER command, then C will be an integer waveform.

Graphing Arrays and Waveforms

Once an array or waveform has been defined (regardless of whether the data represents simulated data or real signal data acquired from an instrument), it can be graphed on the terminal screen. When a waveform is graphed, the horizontal axis is labeled with respect to the data-sampling interval. That is, the left side of the axis (the origin) corresponds to time zero, and each element graphed to the right of the origin represents one increment (the data-sampling interval -- DSI) in time. If the array is 101 elements in length, then the time at the right end of the axis is DSI*101.

The GRAPH command automatically prints time intervals at each major horizontal tic mark. Also, if the horizontal unit of the waveform is defined, the string containing the unit is printed below the horizontal axis. Vertical labeling of the graph includes the vertical unit (if any) and the amplitude of the signal at the major vertical tic marks.

When an array (rather than a waveform) is graphed, the display is much the same, but there are no horizontal or vertical units accompanying the display. (An array is unitless.) Also, since there is no

applicable sampling interval, the horizontal axis of the graph is labeled according to the element numbers of the array.

To demonstrate the graphing of a waveform, consider the following program. It defines a 512-element waveform, fills it with simulated sine-wave data, then graphs it on the terminal screen. (The INTegrate command at line 30 is part of the Signal Processing Package.)

```
1Ø WAVEFORM WC IS C(511), SC, HC$, VC$
2Ø C=.Ø6283
3Ø INT C,C
4Ø C=SIN(C)
5Ø SC=.ØØ2
6Ø HC$="SECONDS"
7Ø VC$="VOLTS"
8Ø GRAPH WC
9Ø END
```

After running the preceding program, the display shown in Fig. 1-16a is produced. Notice that the X-axis has been labeled in units of "SECONDS", and the Y-axis in units of "VOLTS". Also, notice that the horizontal scale ranges from 0 to 1.024. The endpoint of 1.024 is equal to the data sampling interval (0.002) times the number of array elements (512).

Figure 1-16b is an example of graphing just an array. In this case, the array was C, the 512-element array component defined in line 10 of the preceding program. Notice that no horizontal or vertical units are displayed. Also, the horizontal scale ranges from 0 to 512. (The endpoint of 512 corresponds to the number of elements in the array.) Notice, however, that the vertical scale is the same as in Fig. 1-16a. This is because the vertical scale factor is inherent in the array component of a waveform.

It is possible to graph more than one array or waveform with the GRAPH command. For example, the following program defines three 512-element waveforms -- D, E, and F -- and fills them with simulated data representing a sine wave (D), a cosine wave (E), and a power (D*E) waveform.

```
1Ø WAVEFORM D IS WD(511),SD,HD$,VD$
2Ø WAVEFORM E IS WE(511),SE,HE$,VE$
3Ø WAVEFORM F IS WF(511),SF,HF$,VF$
4Ø SD=.1\SE=.1\SF=.1
5Ø HD$="MILLISECONDS"\HE$=HD$\HF$=HD$
6Ø WD=.6283\INT WD,WD
7Ø WD=SIN(WD*.Ø5)\WD=WD*5
8Ø DIFF WD,WE\WE=WE*5
9Ø WF=ABS(WD*WE)
1ØØ GRAPH D,E,F
11Ø SMOVE 3ØØ,75Ø
12Ø PRINT "INSTANTANEOUS POWER PLOT"
13Ø SMOVE 925,6ØØ
14Ø PRINT "VOLTS"
15Ø SMOVE 925,5ØØ
16Ø PRINT "AMPS"
17Ø SMOVE 925,4ØØ
18Ø PRINT "WATTS"
19Ø WAIT\END
```

Lines 10 through 50 define the waveforms, their data-sampling intervals, and the horizontal units. Lines 60 through 100 simulate the actual array data and graph the three waveforms. Lines 110 through 190 then label the top of the graph as well as printing the applicable vertical units (VOLTS, AMPS, and WATTS) along the right side of the graph. Figure 1-17 shows the results of running this program. (For more information on graphing multiple waveforms or arrays, refer to the description of the GRAPH command in Section 2.)

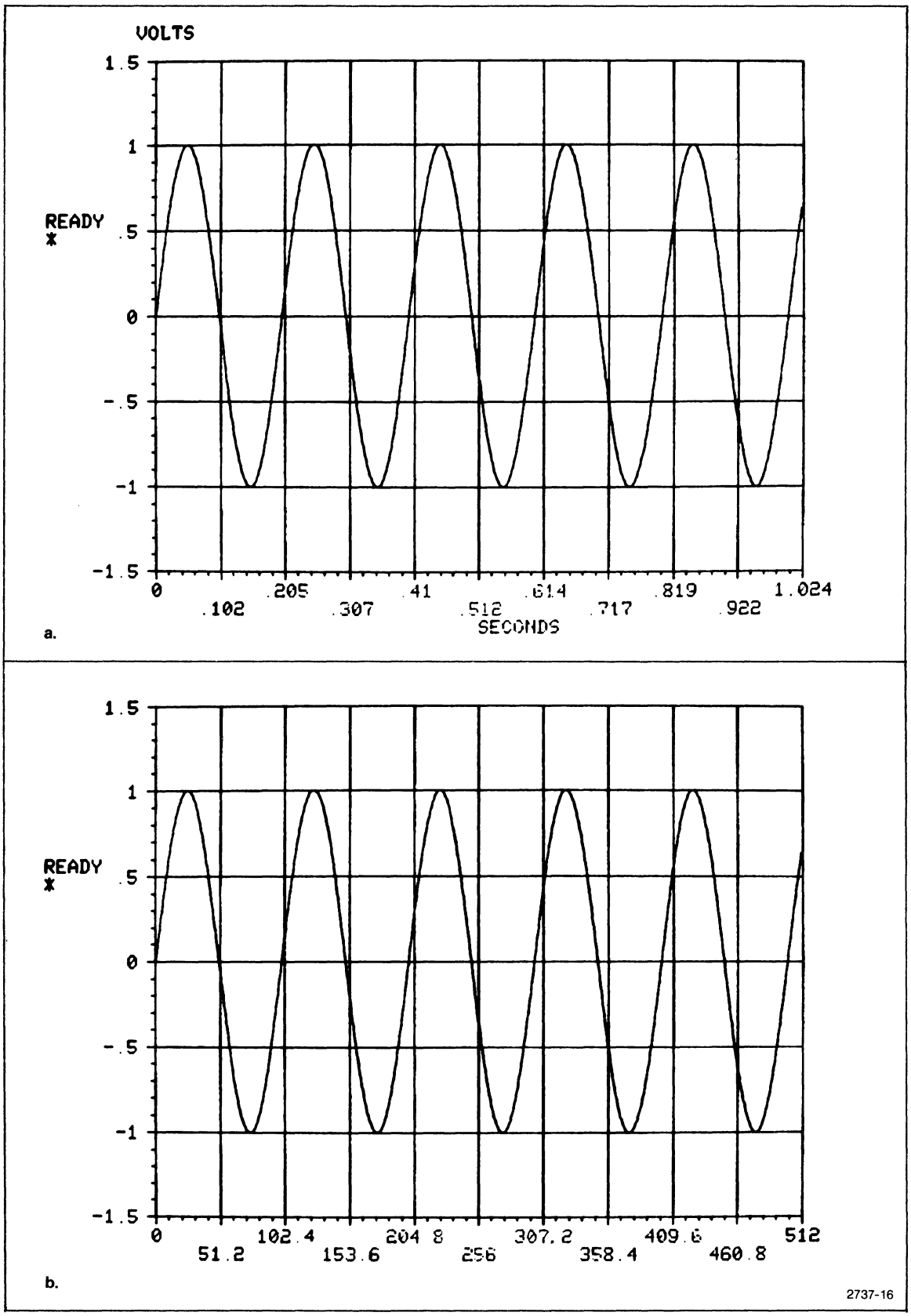


Fig. 1-16. An example graph of a waveform (a) and the array component of a waveform (b).

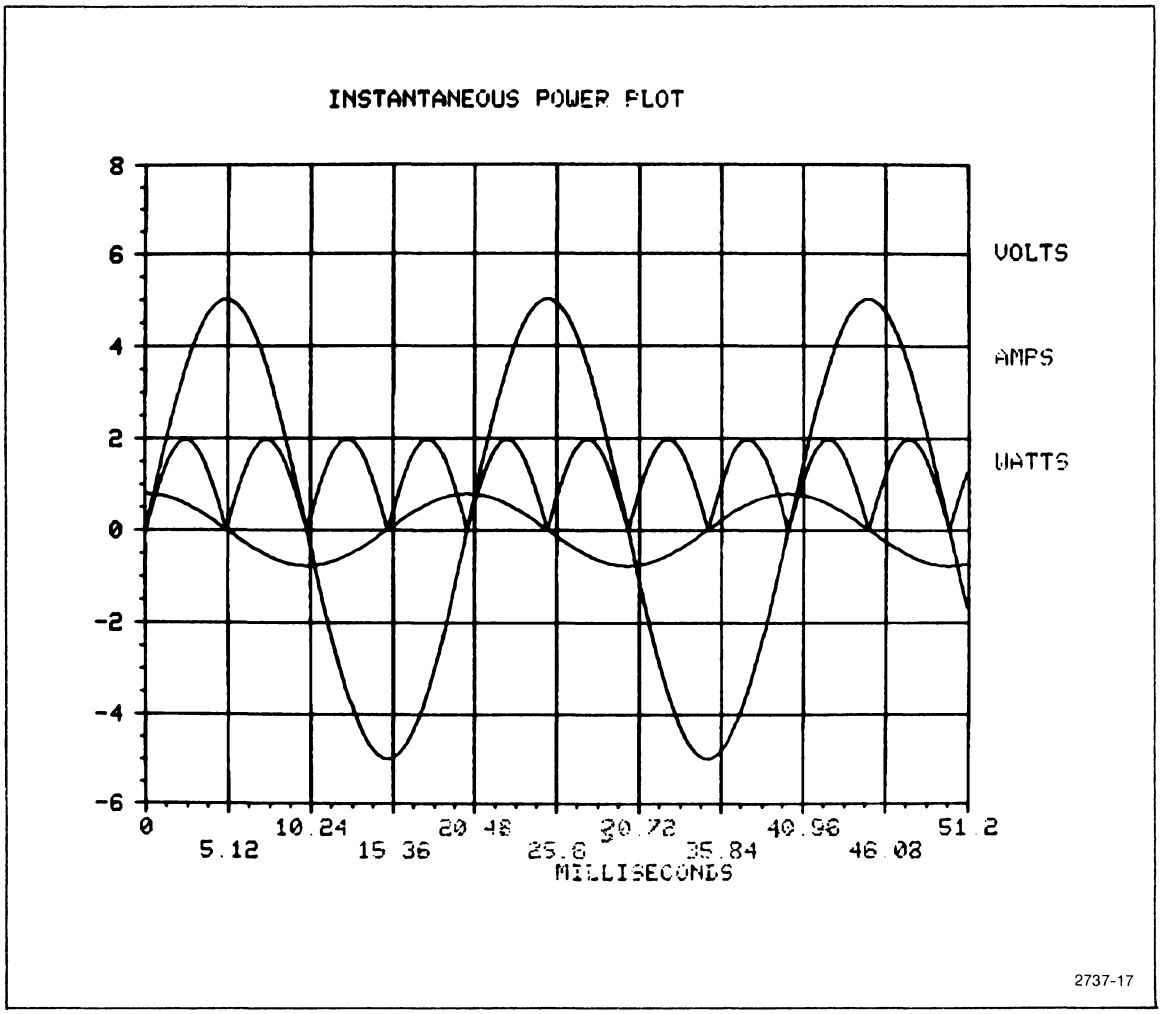


Fig. 1-17. An example of a multi-waveform graph.

Displaying Arrays and Waveforms

It is sometimes necessary to plot an array or waveform without the accompanying graticule and units information. The DISPLAY command provides this capability. Like the GRAPH command, DISPLAY works with arrays and waveforms only.

The DISPLAY command has the ability to create different types of plots: a regular line plot, a point plot (lines are not drawn between individual points), and line plots with various symbols. Figure 1-18 shows the different ways in which the DISPLAY command can be used. This capability is particularly useful when coding waveforms in multiple waveform plots.

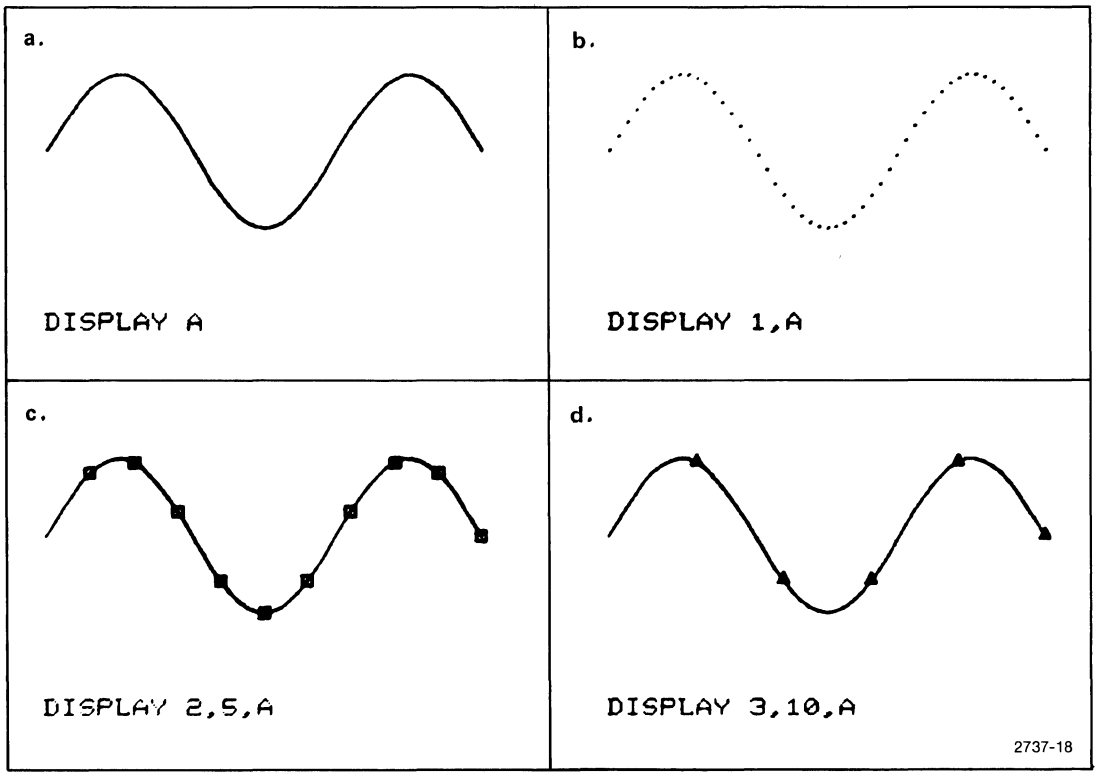


Fig. 1-18. Results of various DISPLAY formats.

The DISPLAY command has several formats. If you want to display just a solid-line plot, as shown in Fig. 1-18a, then the following command will work:

```
DISPLAY A
```

where A is the array or waveform to be plotted. On the other hand, you might want to create a point plot as shown in Fig. 1-18b. In that case you would use the command:

```
DISPLAY 1,A
```

To create a line plot with symbols superimposed, you could execute the command:

```
DISPLAY 2,5,A
```

This creates the display shown in Fig. 1-18c. The first argument in the command (2) designates a line-plot with squares, and the second argument (5) specifies the number of array elements separating the squares. A similar display (Fig. 1-18d) is produced by the command:

```
DISPLAY 3,1Ø,A
```

In this case, the line plot has triangles superimposed (as designated by the first argument, 3) and the triangles are 10 array elements apart.

Care must be taken when using this symbol mode for plotting an array with many elements. Consider a 1000-element array, for example. Since the default viewport for the display command is only 770 device units wide, several data points are plotted in the same space. If the symbol-spacing variable in the DISPLAY command has a small value, the symbols may overlap.

Similarly, when large arrays are displayed with the point-plot mode, the points may be so close together that the result appears to be a solid line.

The size and location of DISPLAYed arrays can be controlled with the VIEWPORT command. Likewise, the WINDOW command can be used to convert your data to screen coordinates. However, if no WINDOW command has been executed, the DISPLAY command automatically adjusts the windows for a "best fit" with the data.

Using the SETGR Command

In the discussion about the GRAPH command, mention was made of the SETGR (for SET GRaticule) command. This command is used to modify the default parameters used when the GRAPH command is executed. The SETGR command can also be used to modify the type of graticule that accompanies an X-Y plot (the XYPLOT command is discussed immediately following SETGR).

Normally, when an array, waveform, or relation is plotted, the GRAPH or XYPLOT command determines what kind of graticule is to be displayed, and what the window and viewport should be. (This is true even if you have defined a viewport or window.) However, the SETGR command provides the capability to change any of these default values.

The SETGR command has six keywords, any of which may be included or deleted from the command line. Each of these parts is discussed separately below. It is important to remember that the SETGR command affects only the **next** GRAPH or XYPLOT command. It does not retain the information. Also, only the **last** SETGR command executed before the GRAPH or XYPLOT is used. Each time the SETGR command is executed, any old information is lost.

The keywords and their definitions are listed below. All keywords may be abbreviated to the first four characters.

GRATICULE. The graticule keyword is associated with two or four arguments which describe the major and minor tic mark types. The first two arguments are required; they specify the type of major tic marks to be used in both the X and Y axes. The second two arguments are optional; they specify the type of minor tic marks for the X and Y axes. Figure 1-19 shows the different types available. Tic lines with arrows indicate that the line is drawn to the opposite side of the graticule area. Figure 1-20 is the result of several different tic types used with the same array.

If the second pair of arguments is not present (the minor tic mark types for the X and Y axes) the default tic types are used.

Default types are type five for the major and type two for the minor tics.

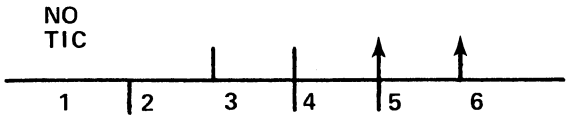


Fig. 1-19. Various tic types available with the SETGR command. (Arrows indicate that lines are drawn to opposite side of graticule.)

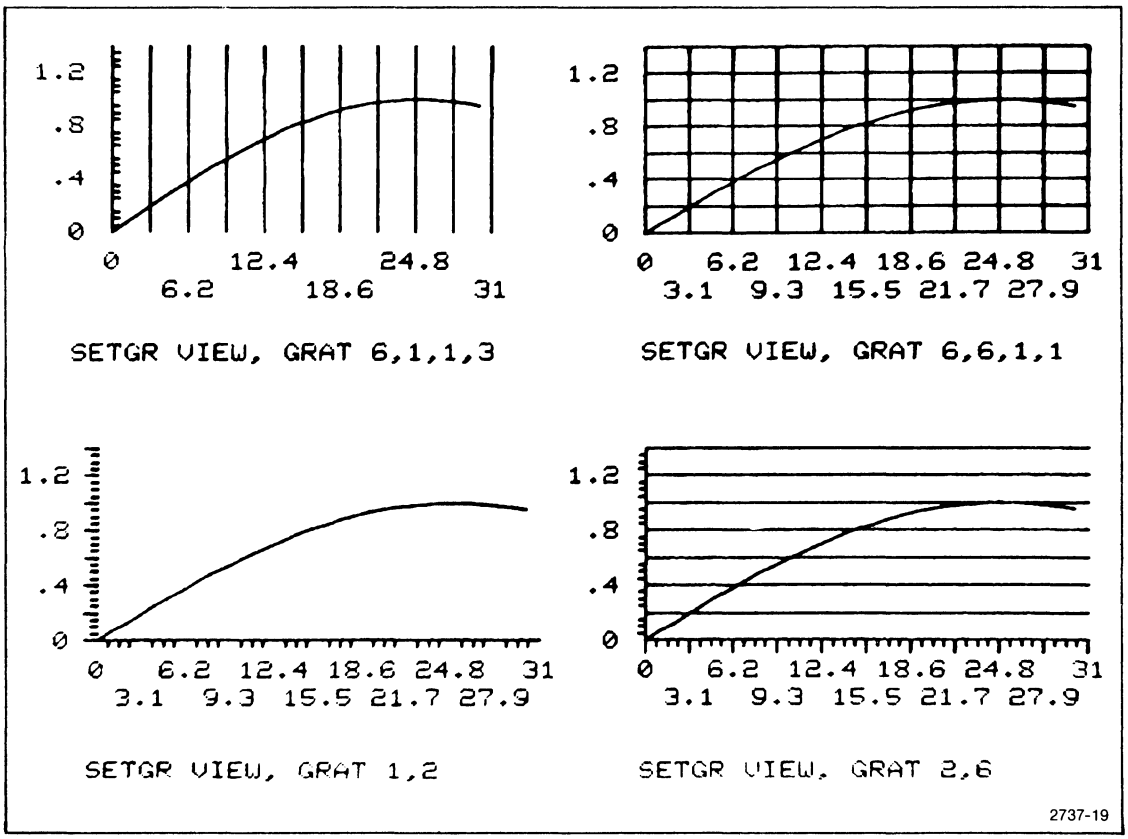


Fig. 1-20. Resultant graphs using different GRATICULE specifications.

NOGRAT. This specifies that no graticule be drawn when the next GRAPH command is executed.

NOLABEL. This specifies that no labels be displayed beside the graticule or axes. Both the numeric axis labeling and the units are suppressed by this keyword.

NOPLOT. The NOPLOT keyword tells the SETGR command to suppress actual plotting when the next GRAPH or XYPLOT command is executed. A graticule is generated, however, if the NOGRAT keyword is not included in the command. This allows a graticule to be drawn, which can be used with the DISPLAY command to plot several arrays with symbols. Figure 1-21 demonstrates this method of displaying arrays.

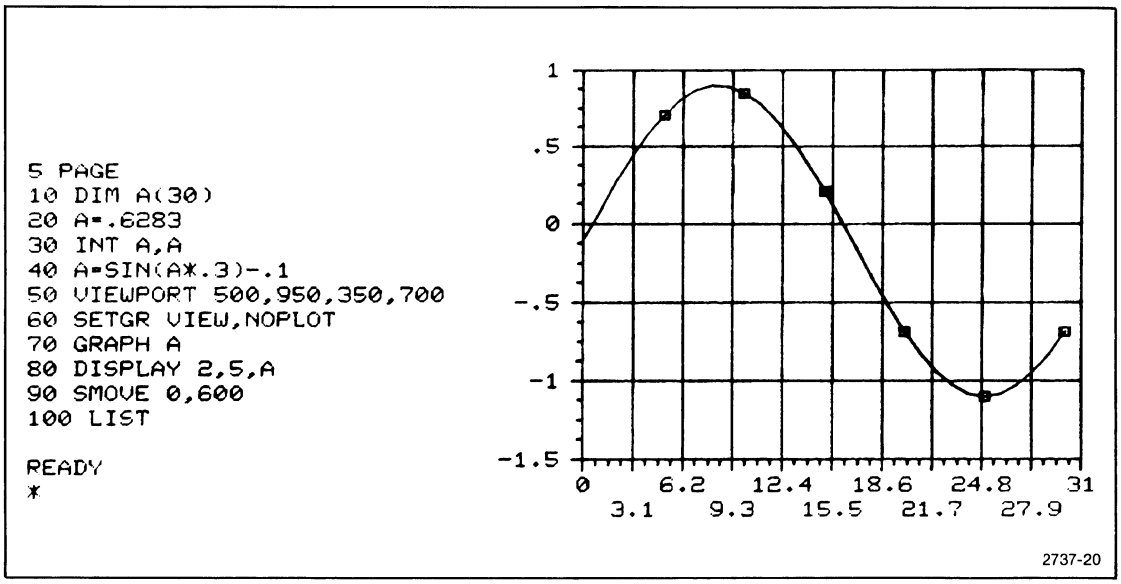


Fig. 1-21. Using NOPLOT to generate graticule for displayed array.

TICS. This keyword specifies the intervals for the major and minor tic marks in both the X and Y axes. The first two arguments must be present. They represent the major tic mark intervals for the X and Y axes, respectively. The second, optional argument pair specifies minor tic mark intervals in the X and Y axes. If the Y value for the major tic mark interval is zero, the GRAPH or XYPLOT command automatically chooses the tic interval that best fits the array or waveform. Figure 1-22 shows the result of four different tic specifications on an array of 31 elements.

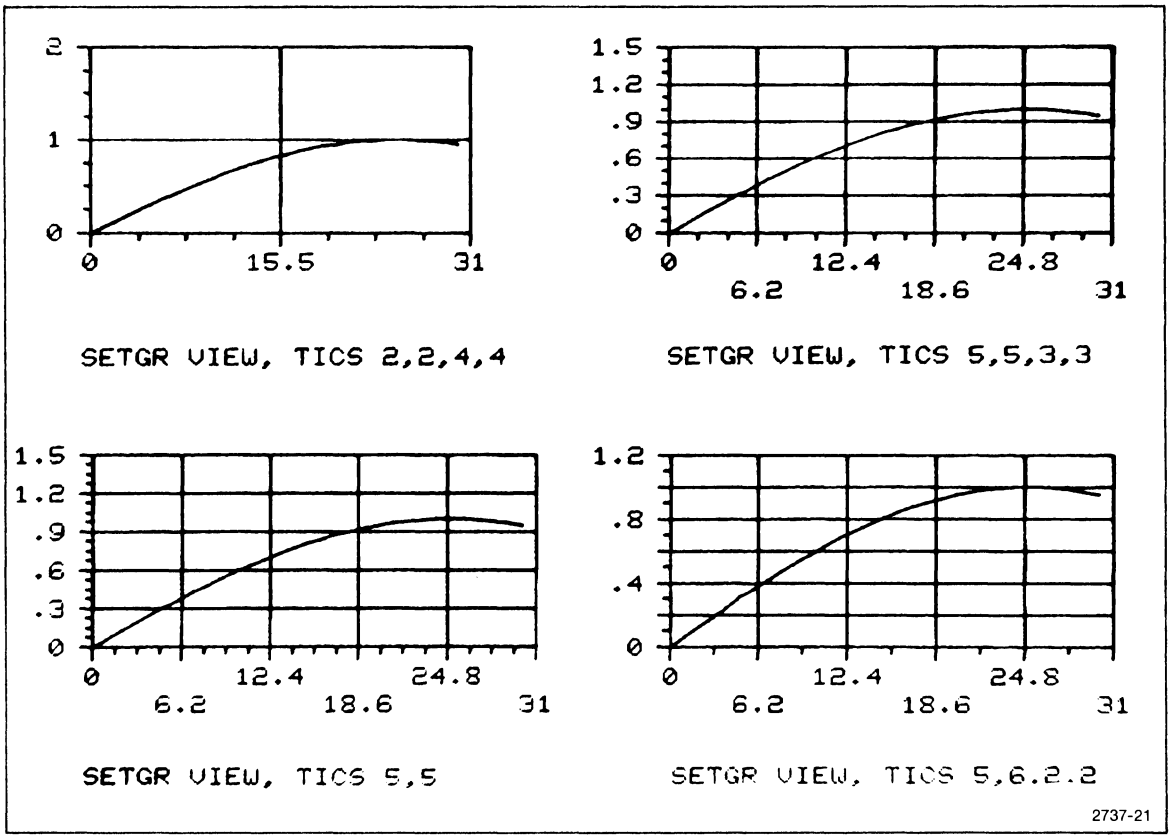


Fig. 1-22. An array graphed with four different TICS specified.

VIEWPORT. This keyword tells the next GRAPH or XYPLOT command to plot the data in the current viewport, defined by the last VIEWPORT command executed.

WINDOW. When the WINDOW keyword appears in the SETGR command line, the current WINDOW is used to define the data ranges of the next GRAPH or XYPLOT. This keyword causes the current window to override the normal autoscaling done by the GRAPH or XYPLOT commands.

XOFFSET. This keyword specifies a base value to be added to the X-axis values of the graphics window (and thus to the labels) by the next GRAPH or DISPLAY command. The graph of the data is shifted by the base value, but the data itself is not altered. XOFF does not affect the XYPLOT command. The XOFF option is very useful in applications where data is being acquired from a spectrum analyzer -- where the left side of the graph corresponds to some non-zero frequency.

Remember that only the **last** SETGR command executed prior to a GRAPH or XYPLOT command is effective. This means that if more than one keyword specification is to be used for the next GRAPH or XYPLOT, all must appear in one SETGR command. Some example programs using SETGR are included later in this section, as well as in Section 2 under the description of the SETGR command.

Creating X-Y Plots

Another useful command is XYPLOT. The XYPLOT command creates a display similar to that of GRAPH. However, whereas GRAPH plots array data against corresponding element numbers (multiplied by a data sampling interval in the case of a waveform), XYPLOT plots the data in one array against the data in a second array. In more mathematical terms, the GRAPH command is used for plotting just functions, whereas XYPLOT is used in plotting any kind of mathematical relation. (A function is defined as a mathematical relation in which for every X-value there is one, and only one, Y-value called f(x).) The XYPLOT command is particularly useful when creating X-Y plots such as Lissajous figures, stress-strain data, and hysteresis plots.

In its simplest form, the XYPLOT command contains two arguments. Either one or both of these arguments may be either an array or a waveform. In any case, the first argument is assumed to be the X-axis data, and the second argument is assumed to be the Y-axis data. Both arguments must be of the same array length.

When two waveforms are used as the arguments for XYPLOT, the graticule is labeled with the **vertical units** from each waveform. For example:

```
XYPLOT WX,WY
```

This statement plots waveform WY (the Y-data) against waveform WX (the X-data); the vertical axis is labeled with the vertical units from WY, and the horizontal axis is labeled with the vertical units from WX. However, the horizontal units and sampling intervals from both waveforms are ignored since they are not applicable to an X-Y data plot.

Let's consider a second example in which the waveform WY is plotted against the array X. This could be accomplished with:

```
XYPLOT X,WY
```

In this case, the vertical units from WY will accompany the vertical axis, but no units will be printed with the horizontal axis. Similarly, if we were to plot array X against the array component of waveform WY, neither axis would be labeled with units.

The XYPLOT command allows more than one pair of arguments to be plotted on the same graticule, provided that each argument pair has arguments of the same array length. As an example, the following statement is legal, provided that arguments X and Y are of the same length and arguments Z and W are of the same length:

```
XYPLOT X,Y,Z,W
```

This command would first create an X-Y plot of X and Y, followed by an X-Y plot of Z and W. Any or all of these arguments may be arrays or waveforms, and X and Y may be of different length than Z and W. The only stipulation is that X and Y must be of equal lengths as must Z and W. (For more information on graphing multiple relations, see the description of XYPLOT in Section 3.)

The following program demonstrates the concept of an X-Y plot by generating and plotting a typical Lissajous pattern. In this case, the X-data represents a single cycle of a cosine wave, and the Y-data represents three cycles of a sine wave. (A similar pattern can be seen on an oscilloscope by setting the time base to the 60 Hz AC line voltage, and connecting the vertical input to a 180 Hz sine generator.) Here is the program:

```
1Ø WAVEFORM WX IS X(199),SX,HX$,VX$
2Ø WAVEFORM WY IS Y(199),SX,HX$,VY$
3Ø X=.Ø31416\Y=3*X
4Ø INT X,X\INT Y,Y
5Ø X=COS(X)\Y=SIN(Y)
6Ø VX$="VOLTS"\VY$="AMPS"
7Ø VIEWPORT 2ØØ,8ØØ,1ØØ,7ØØ
8Ø SETGR VIEW\XYPLOT WX,WY
9Ø WAIT\PAGE
1ØØ SETGR VIEW\XYPLOT WY,WX
```

The program begins by defining two waveforms, WX and WY. Lines 30 and 40 simulate a ramp in WX and WY by means of the INTegrate command (part of the Signal Processing Package). The slope of the WX ramp is such that one cycle of cosine data is generated (line 50). Likewise, the slope of the WY ramp is such that three cycles of sine data are generated (also at line 50). Line 60 merely sets the vertical units for both waveforms.

Line 70 defines a square viewport for use with the SETGR VIEW commands at lines 80 and 100. (If this had not been done, a rectangular default viewport would have been used.) Line 80 then creates the X-Y plot of WY against WX, shown in Fig. 1-23a. After the terminal's RETURN key is pressed, the screen is erased and line 100 executes. This line creates an X-Y plot of WX agains WY, shown in Fig. 1-23b. Notice that Fig. 1-23b is essentially the same as Fig. 1-23a except that the Lissajous pattern has been rotated 90°. Also, the units ("AMPS" and "VOLTS") accompanying each axis have been interchanged as a result of switching the command arguments.

Graphic Input and Output

All of the commands described thus far are used for output of data to a terminal or other graphics device. The remainder of this section describes three commands that can be used for inputting data from a graphics device to computer memory or to some peripheral storage device. These are the GIN, SGIN, and DRAWON commands.

Using Graphic Input

Two of the graphics input commands (GIN and SGIN) allow you to obtain the position of the cross-hair cursor (on a TEKTRONIX 4010-Series terminal) in either user coordinates or screen coordinates. These commands also return a string containing the character typed when the cross-hairs were displayed. This character can be used to determine what action to take after getting the cross-hair position -- more about that later.

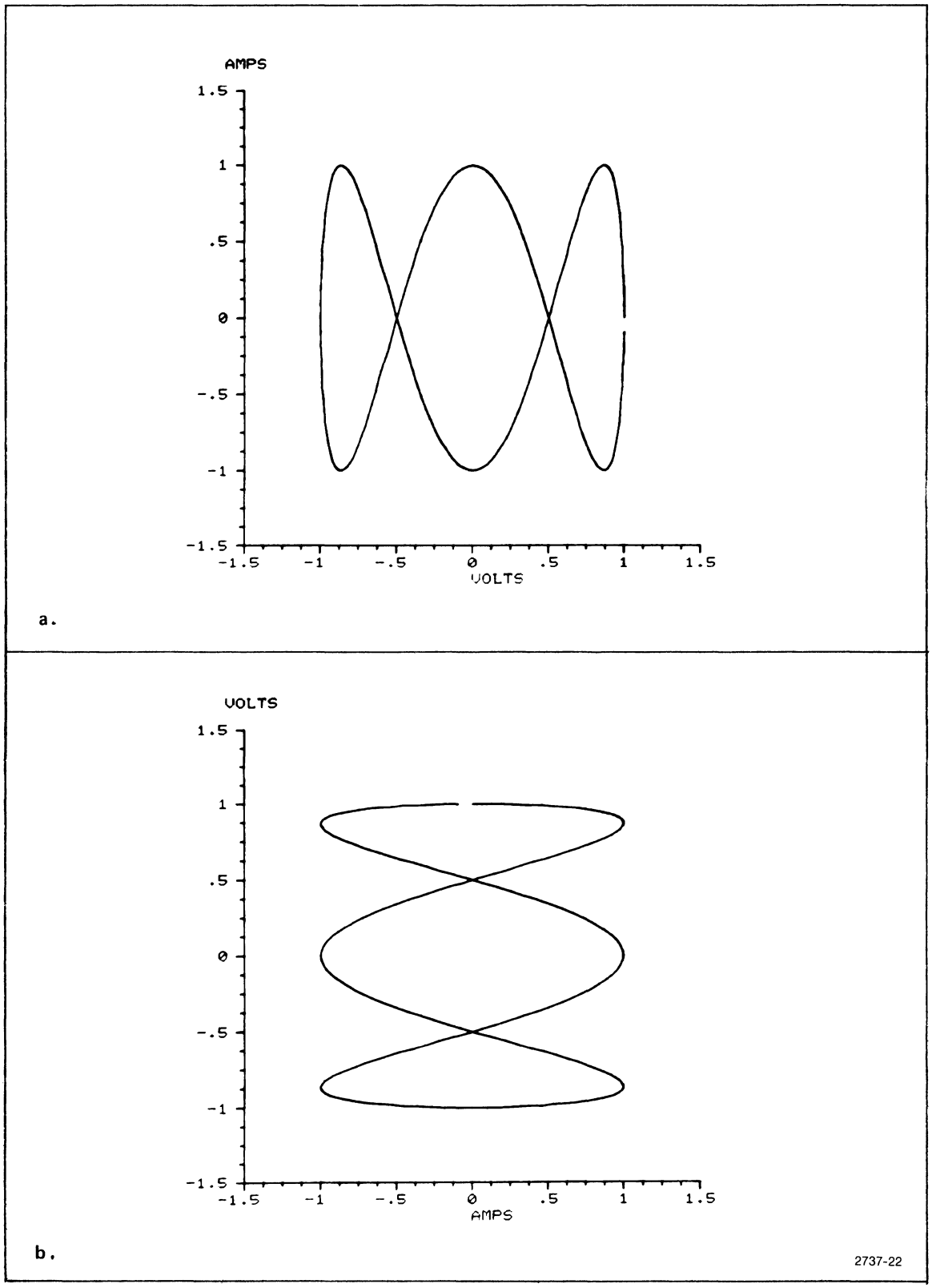


Fig. 1-23. Typical Lissajous patterns displayed with the XYPLOT command.

The format of the GIN command (the one that returns the cross-hair position in user coordinates) is illustrated by the following example:

```
GIN A$,X1,Y1
```

Executing the above statement immediately calls up the terminal's cross-hair cursor. The position of the cursor is controlled by two thumb-wheels at the right side of the terminal keyboard. (The GIN and SGIN commands will not finish executing until both cross-hairs are visible on the terminal screen.) After the cursor has been set to the desired position, any alphanumeric key may be struck to complete the command's execution. In the above example, the key just struck is entered into the string variable A$. Also, the X-coordinate of the cursor position is entered into X1, and the Y-coordinate is entered into Y1. These values can then be printed with the statement:

```
PRINT A$,X1,Y1
```

to indicate the current position (X1,Y1) of the cross-hairs.

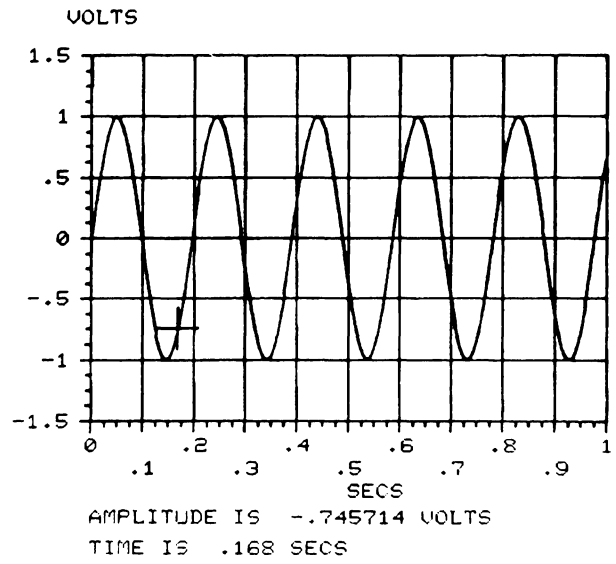
The SGIN command has the same command format and operates like the GIN command. The only difference is that SGIN always obtains the cursor position in terms of screen (device) coordinates, rather than user coordinates. Thus the coordinates obtained by SGIN are not affected by the current window or viewport.

The program shown in Fig. 1-24 demonstrates how the GIN command can be used to obtain the amplitude and time corresponding to a point on a waveform. A waveform (AA) is defined, and the data sampling interval (DS) is set to 0.1/51.2 (which would be the time per element if the waveform had been acquired from a 512-element digitizer set to a horizontal scale factor of 0.1 seconds per division). The VIEWPORT command in line 80 is used in conjunction with the SETGR command to place the graph on the right side of the screen. Since no WINDOW keyword is used in the SETGR command line, the GRAPH command automatically autoscales the waveform data and sets up a default window.

Line 110 is the GIN command. When this line is executed, the cross-hairs are displayed, and the program waits for you to type a character at the keyboard. You should now move the cross-hairs to a point on the waveform display and strike a key. (The terminal should be strapped to send an automatic carriage return after a key is struck.) A$, a string variable, contains the character just typed. Variable X holds the

```
10 WAVEFORM AA IS A(511),DS,H$,V$
30 A=.06283
40 INT A,A
50 A=SIN(A)
60 H$="SECS"\V$="VOLTS"
70 DS=.1/51.2
80 VIEWPORT 500,1000,200,550
90 SETGR VIEW
100 GRAPH AA
110 GIN A$,X,Y
120 MOVE X,Y
130 RSDRAW 0,20,0,-40
140 RSDRAW 0,20,-20,0,40,0
150 SMOVE 500,100
160 PRINT "AMPLITUDE IS ",
170 PRINT Y;" ";V$
180 SMOVE 500,70
190 PRINT "TIME IS ";X;" ";H$
200 SMOVE 0,650
210 LIST

READY
*
```



2737-23

Fig. 1-24. Using the GIN command to select a point on a waveform.

horizontal value, and variable Y the vertical value. These values are in user coordinates by virtue of the GIN command. Since a waveform was graphed, the X variable represents time (in seconds) and the Y variable represents amplitude (in volts).

The commands in lines 120 through 140 cause a cross to be drawn at the point of intersection of the cross-hairs. A MOVE command (which moves the pointer in terms of user coordinates) positions the alpha curser at the selected location. To actually make the cross, the RSDRAW command is used to draw a line segment relative to the selected location. The line is drawn in screen (device) coordinates. This allows the cross to be the same size, regardless of how the window or viewport is defined.

Finally, the alpha cursor is moved to a position below the graph, and the information about the time and amplitude is printed. The last two lines (200 and 210) simply list the program at the left side of the screen.

The SGIN command is used in the same manner as GIN, but the coordinates returned in the X and Y variables are always in screen coordinates. This is demonstrated in the following program which uses the SGIN command to input data from the screen into an array. This program allows the terminal to be used as an "electronic sketchpad" so that simple drawings (such as Fig. 1-25) can be easily produced. The program looks at the character typed to determine whether the cross-hair cursor will serve as the terminus for a MOVE or a DRAW operation. (Typing M means MOVE; D means DRAW.) It also checks for the letter E, which signals the end of graphic input. When the E is found, the array contains all the information necessary to redraw the picture. Here is the program:

```
1Ø REM -- PROGRAM TO READ SCREEN COORDINATES INTO AN ARRAY
2Ø INTEGER A(1,5ØØ)
3Ø REM -- SET ARRAY TO -1 FOR END OF DATA SIGN
4Ø A=-1
5Ø REM -- INITIALIZE ØTH ELEMENTS TO POINT (Ø,Ø)
6Ø A(Ø,Ø)=Ø
7Ø A(1,Ø)=Ø
8Ø REM -- START DATA GATHERING LOOP
9Ø FOR P=1 TO 5ØØ
1ØØ SGIN A$,X,Y
11Ø REM -- LOOK FOR END SIGNAL
12Ø IF A$="E" THEN 24Ø\REM -- EXIT LOOP IF "E"
13Ø REM -- OR MOVE COMMAND
14Ø IF A$="M" THEN 2ØØ
15Ø REM -- MUST BE A DRAW
16Ø A(Ø,P)=X
17Ø A(1,P)=Y
18Ø GOSUB 37Ø
19Ø GOTO 22Ø
2ØØ A(Ø,P)=X+2ØØØ
21Ø A(1,P)=Y
22Ø NEXT P
23Ø REM -- END OF LOOP
24Ø END
25Ø REM -- LOOP TO DRAW COORDINATES IN ARRAY
```

```
26Ø REM -- SWITCH TO GRAPHICS MODE
27Ø INITG
28Ø FOR P=1 TO 5ØØ
29Ø IF A(Ø,P)=-1 THEN RETURN
3ØØ IF A(Ø,P)<2ØØØ THEN 33Ø
31Ø MOVE A(Ø,P)-2ØØØ,A(1,P)
32Ø GOTO 34Ø
33Ø DRAW A(Ø,P),A(1,P)
34Ø NEXT P
35Ø RETURN
36Ø REM -- ROUTINE TO DRAW PICTURE AS YOU GO
37Ø RESETG
38Ø IF A(Ø,P-1)<2ØØØ THEN 41Ø
39Ø MOVE A(Ø,P-1)-2ØØØ,A(1,P-1)
4ØØ GOTO 42Ø
41Ø MOVE A(Ø,P-1),A(1,P-1)
42Ø DRAW A(Ø,P),A(1,P)
43Ø RETURN
```

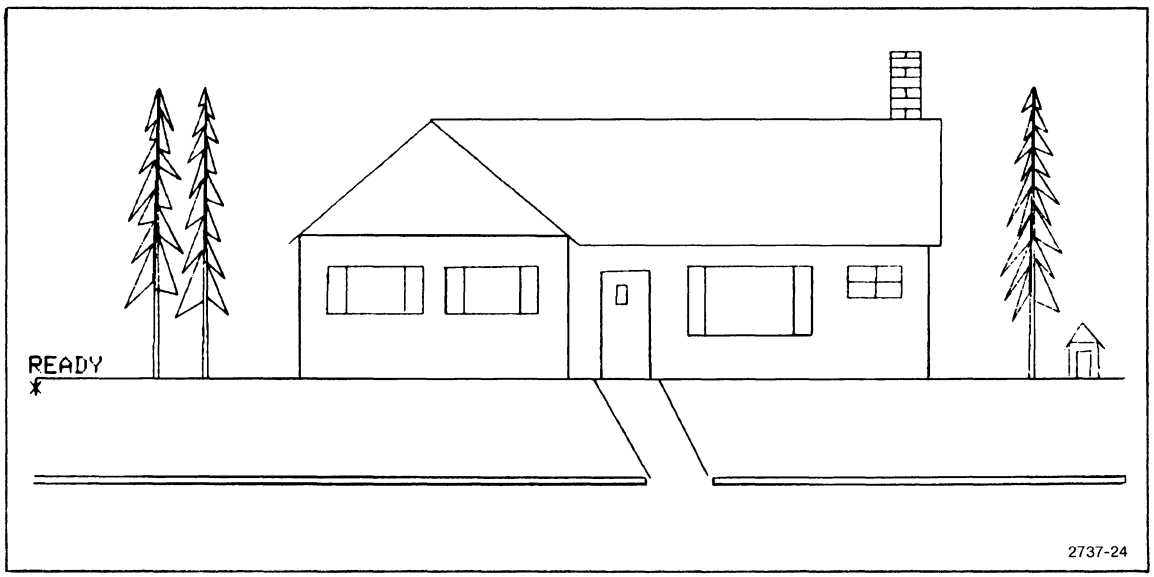


Fig. 1-25. Example output of program using the SGIN command.

The program begins by calling up the cross-hair cursor and waiting for you to enter a character. (An M should be the first character typed so that you can specify the beginning point of the drawing.) After the first character has been typed, any combination of M and D keys can be

typed. These keys, in conjunction with the cross-hair cursor position wheels (right side of keyboard) allow you to create any combination of hidden or displayed lines. If you want just a dot to appear, you can move the cursor to the desired position and then type an M immediately followed by a D. When you are satisfied that the drawing is complete, simply type an E to end the program.

An integer array (A) is used to hold the X and Y data values, since no screen coordinate is larger than 1023 nor contains a fractional part. Up to 501 coordinate values can be stored in this array. Since the values are permanently stored in array A, you can redisplay the picture any number of times simply by entering:

```
GOTO 25Ø
```

This erases the screen and quickly redraws the picture you so painstakingly created.

Saving Your Graphic Output

TEK SPS BASIC allows you to save, in a permanent file, data that would normally be sent to the terminal driver in graphics mode. This data can then be copied back to the driver at a later time for display. The DRAWON command allows this capability.

The format of DRAWON is illustrated by the following example:

```
DRAWON #1
```

The lone argument (#1) specifies the peripheral logical unit number (PLUN) of the destination file. This file does not have to be open when the DRAWON command is executed, but it must be OPENed for WRITE before issuing the first graphics command that outputs data.

The following program is a simple demonstration of how the DRAWON command can be used in a program to create graphics files:

```
1Ø OPEN #1 AS "GRAPH.DAT" FOR WRITE
2Ø DRAWON #1
3Ø INITG
4Ø DIM B(4ØØ)
5Ø B=RND(B)
6Ø GRAPH B
7Ø SMOVE Ø,78Ø
8Ø PRINT #1,"RANDOM ARRAY"
9Ø DRAWON #Ø
1ØØ END
```

The program begins by opening a file called "GRAPH.DAT" on the system storage device for output. The following DRAWON statement instructs the software that all graphic data is to be written out to file number one. The INITG command at line 30 sends the instructions to the file which cause the terminal screen to be erased and the window and viewport to be initialized to their default values.

Lines 40 and 50 create a 401-element array (B) and fill the array with random numbers between zero and one. The GRAPH command at line 60 writes the graticule and graph of array B to the file. Line 70 causes the alpha cursor to move to the top-left (home position) of the terminal screen. The PRINT statement at line 80 then prints the message out to file number one. If the peripheral logical unit number (#1) had not been included in this PRINT statement, the message would have appeared at the terminal when the program was run, since PRINT is not a GRAPHICS command. Line 90 uses the DRAWON command to direct future graphic output back to the terminal (permanently defined at PLUN #0). The last statement, END, closes the file and stops program execution.

To retrieve the stored data, a COPY command can be issued to copy the file from the storage device back to the driver for the graphics device. For example:

```
COPY "GRAPH.DAT" to KB:
```

After entering this statement, the file "GRAPH.DAT" on the system storage device is opened and copied to the terminal. In this case, the output is a screen erasure (because of the INITG command in the program), the graph of array B, and the message RANDOM ARRAY at the top of the screen. Figure 1-26 shows the completed drawing.

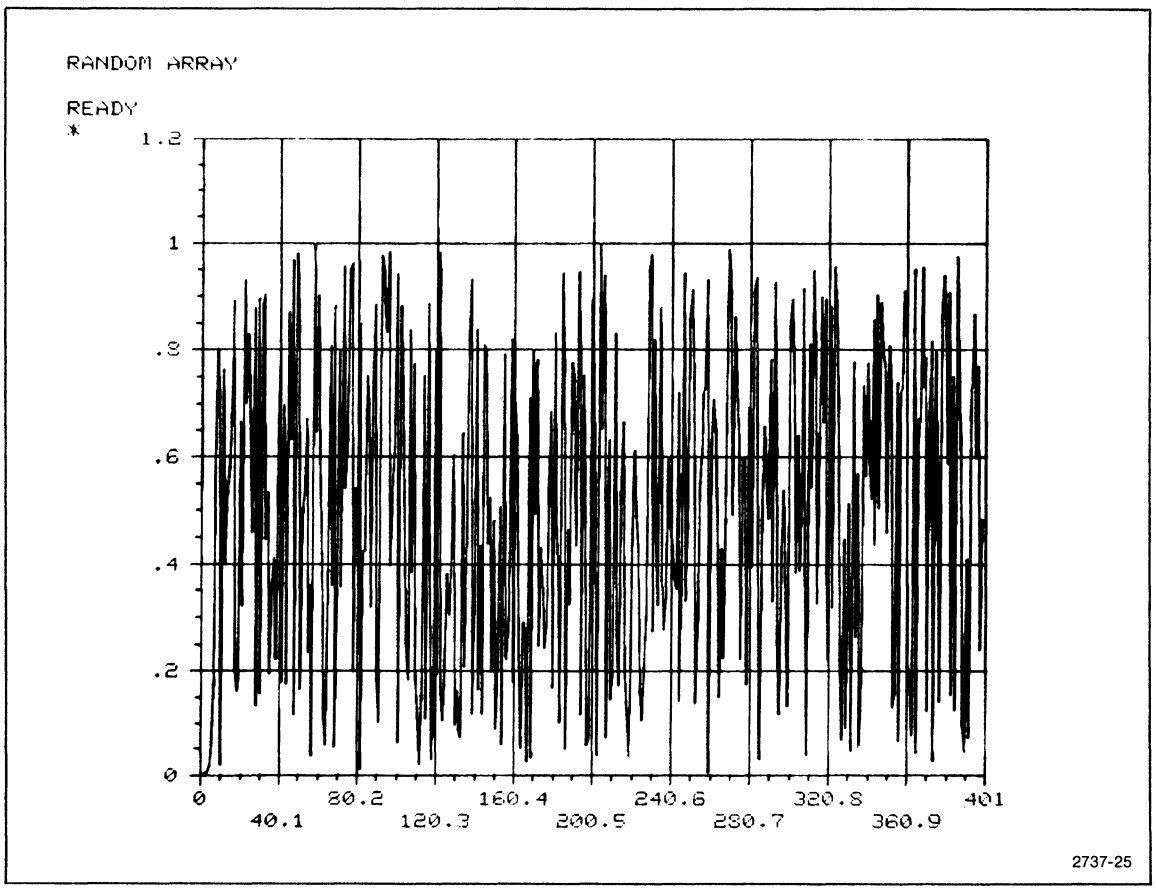


Fig. 1-26. Example of graphic data stored on the system device with the DRAWON command and later COPY'd to the terminal.

While the preceding example is somewhat trivial, the same basic process can be used for storing complicated plots that must be repeatedly accessed during the course of a program. Examples of such plots might be special graticules (log-log or semi-log), Smith chart templates, and special page headers that are too time-consuming to generate during the course of program execution.

SECTION 2

COMMAND DESCRIPTIONS

This part of the manual is a reference section for all of the TEK SPS BASIC Graphics Package commands. Each command description is listed in alphabetical order, and includes examples, syntax listings, and a discussion of how the syntax options are used. Several example programs are also provided to further clarify the information provided in Section 1.

Guide to Syntax Notation

The syntax listings describe how the commands may be typed on the terminal. Upper case characters and punctuation must be typed as shown, but any information in brackets ([]) is optional. Braces ( ) indicate that a choice must be made between one of the listed items. Items followed by an ellipsis (...) may be repeated one or more times.

Memory Requirements

The approximate size of each command is listed in Appendix H of the System Software manual. This size refers to the number of words of memory required to load that particular command. In some cases, the amount of memory needed to execute the command will be considerably more.

DISPLAY (Nonresident)

Examples:

```
   DISPLAY A
100 DISPLAY B,C
200 DISPLAY 1,R,Z(50:200)
300 DISPLAY A,Z,10,B,3,10,C
```

Syntax Form:

[line no.] **DISPLA** [expression[,expression],] { array expression / waveform expression }

[,[expression[,expression],] { array expression / waveform expression }] . . .

Descriptive Form:

[line no.] **DISPLAY** [line type[,horizontal interval],] { array expression / waveform expression }

[,[line type[,horizontal interval],] { array expression / waveform expression }] . . .

Purpose:

To plot an array or waveform into the current graphics window (user coordinate system) as defined by the WINDOW command or the previous GRAPH command autoscaling.

Discussion:

The DISPLAY command allows you to plot data, either with a solid line plot, a point plot, or a solid line plot with symbols. Unlike the GRAPH command, DISPLAY does not generate a graticule. Only arrays, array zones, or waveforms may be plotted. Arrays and waveforms can not be mixed in a single DISPLAY command. Waveforms with a data sampling interval of zero are treated as arrays.

The DISPLAY command plots data into the current viewport defined by the last VIEWPORT or GRAPH command. The DISPLAY command is also affected by the WINDOW command.

Using the Syntax Options:

The DISPLAY command can display one or more arrays or waveforms. For each array or waveform displayed, there are three arguments that define the type of plot to be presented. Two of these arguments are optional.

The first optional argument, a scalar expression, is interpreted as the line or symbol type. It may have one of four values, interpreted as follows:

Ø -- Solid line (default).
1 -- Point plot.
2 -- Solid line with square symbols.
3 -- Solid line with triangle symbols.

The second optional argument, also a scalar expression, is interpreted as the horizontal interval at which the symbol will appear. This argument is valid only if the first argument (line-type) is present and equal to type 2 or 3. The horizontal interval thus defines the frequency of appearance of the square or triangle along the plot. For example, if its value is 20, the symbol appears at every 20th data point (array element). If this argument is not present in the command, and the line type is 2 or 3, the symbol is not plotted.

The third argument is required and simply specifies the array or waveform to be plotted. Line or symbol type and symbol interval expressions may precede each array or waveform expression in the argument list. Notice that the syntax also allows mathematical expressions containing arrays or waveforms to be plotted.

A simple example of the DISPLAY command is:

```
DISPLAY 1,A
```

This statement creates a point-plot of array A. The horizontal interval argument is not present since the point-plot mode is selected. Another example is:

```
DISPLAY 2,2Ø,B
```

This causes a solid line plot of array B to be drawn, with a square symbol output at every 20th array element.

A slightly more complicated example of DISPLAY is:

```
DISPLAY 3,2Ø,A,B,1,1Ø*C
```

This statement first displays array A via a solid plot with triangles superimposed 20 array elements apart. Next, a solid line plot (with no symbols) of array B is displayed. Finally, array C is multiplied by 10 and a point plot of the resulting array is then displayed.

DRAW (Nonresident)

Examples:

```
6ØØ DRAW X,Y
62Ø DRAW A,B,C+D,7ØØ,SIN(X),COS(E(I))
```

Syntax Form:

[line no.] **DRAW** expression,expression[,expression,expression] . . .

Descriptive Form:

[line no.] **DRAW** x coordinate in user units, y coordinate in user units
[,x coordinate in user units, y coordinate in user units] . . .

Purpose:

To draw a line from the current pointer or pen position to the (X,Y) position specified in user coordinates.

Discussion:

In its simplest form, the DRAW command contains two arguments. The first argument specifies the X-coordinate of the point to which a line will be drawn from the current pointer or pen position. The second argument specifies the Y-coordinate of the same point. Both coordinates are in the user space defined by the last WINDOW command. The VIEWPORT command defines the size and location of the user space within the plotting area of the graphics device. It is the user's responsibility to keep the pointer in the graphics space of the display device, since clipping is not provided.

Using the Syntax Options:

Additional ordered pairs of (X,Y) coordinates can be appended to a simple DRAW command to cause additional line segments to be drawn. For example, in line 600 above, a vector would be drawn to point (X,Y) from the present cursor position. Let's suppose, however, that we modified line 600 to read:

```
600 DRAW X,Y,J,K
```

In this case, a vector would first be drawn to point (X,Y). Then a vector would be drawn from point (X,Y) to point (J,K). (This assumes that X,Y,J, and K are all defined to values within the presently defined user space.)

Consider another example. The following program segment first moves the pointer or pen to point (0,0) in the user coordinate system. Then at line 410, a box is drawn on the graphics device; the box is 100 user units square.

```
400 MOVE Ø,Ø
410 DRAW Ø,100,100,100,100,Ø,Ø,Ø
```

The four coordinate pairs specify the four corners of the box. When the command is finished executing, the pointer is back to the position it started at (0,0).

DRAWON (Nonresident)

Example:

```
500 DRAWON #2
```

Syntax Form:

[line no.] **DRAWON #**expression

Descriptive Form:

[line no.] **DRAWON #**target plun

Purpose:

To allow graphic output to be sent to any file currently OPEN for WRITE, and to allow graphic input from any device with that capability.

Discussion:

It is sometimes desirable to send graphic output to a peripheral storage device (hard disk, floppy disk, etc.) rather than immediately displaying the graphic information on the terminal or other device. The DRAWON command allows you to do this. Once the graphic information has been stored, it can then be COPY'd to the graphics device very quickly, and as often as necessary. This is particularly useful for applications that frequently use a given graphics display (Smith chart template, or header for a program).

All graphic output that would normally be sent to the graphics terminal driver is then written to the device associated with the specified peripheral logical unit number (PLUN). The device need not be OPEN for WRITE when the DRAWON command is executed, but must be open before a graphic command is given.

Using the Command Syntax:

The argument for the DRAWON command is an expression that is rounded to a peripheral logical unit number in the range of 0 to n; n is the maximum allowable PLUN specified at initialization time. A PLUN of 0 always corresponds to the graphics terminal, and a non-zero PLUN corresponds to some other peripheral device. Thus when terminating an operation directing graphics output to a peripheral storage device, a DRAWON #Ø command should be executed. Following execution of this command, all graphics output will again be directed to the terminal. (An example program using DRAWON was presented at the end of Section 1.)

GIN (Nonresident)

Example:

```
7ØØ GIN B$,X,Y
```

Syntax Form:

```
[line no.] GIN string variable,variable,variable
```

Descriptive Form:

```
[line no.] GIN target for input character,
               target for x coordinate of crosshairs in user units,
               target for y coordinate of crosshairs in user units
```

Purpose:

To input the position of the terminal's cross-hair cursor in user coordinates.

Discussion:

If the graphics device in use is a TEKTRONIX Computer Display Terminal, GIN enables the cross-hair cursor and waits for you to strike a character on the keyboard. When typed, the character is returned in the specified string variable, and the (X,Y) position of the cross-hairs is returned in the specified variables. The first variable contains the X-coordinate and the second variable contains the Y-coordinate. These values are returned in user-coordinates as defined by the current window.

Note that some terminals may not be strapped to automatically append a carriage return to the input character when a key is struck. If the cross-hairs are still displayed after you type a character, strike the RETURN key also.

If the DRAWON command is in effect (specifying a peripheral other than the terminal), the graphic input is fetched from the specified device. If this device is not capable of returning graphic data, an error results.

GRAPH (Nonresident)

Examples:

```
16Ø GRAPH A
GRAPH A*B,C,SIN(X*3.5)
```

Syntax Form:

[line no.] **GRAPH** { array expression[,array expression] . . . | waveform expression[,waveform expression] . . . }

Purpose:

To graph an array or waveform with graticule and axis labeling.

Discussion:

The GRAPH command automatically creates an array or waveform plot accompanied by a graticule. The horizontal and vertical axes are labeled with the applicable scale factors. If the data graphed is a waveform, the horizontal and vertical units are also displayed.

The array or waveform data is first auto-scaled to the terminal screen before the actual graphing occurs. That is, the vertical scale factor used in scaling the data to the terminal's coordinate system is set to a value that causes a "best fit" in the viewing area. However, this autoscaling may be overridden with the SETGR command. The SETGR command allows you to specify the number and types of major and minor tic marks for the display graticule. It also allows you to specify the size of the window and viewport that is used. However, it should be remembered that SETGR affects only the **next** GRAPH or XYPLOT command executed. It does not have any ensuing effect on further GRAPH commands unless SETGR is executed each time GRAPH is executed. (For more information, refer to the description of the SETGR command.)

The default graticule (the graticule displayed if no SETGR command has been executed) will choose the number of tic intervals which best displays the data in the Y axis. This graticule is also shifted by as many as six screen units each time it is called up. This shifting is done to prevent residual images from appearing on the graphics terminal.

Using the Syntax Options:

In its simplest form, the GRAPH command is used to graph a single array or waveform. If an array is being GRAPHed, the horizontal axis will be labeled such that the left edge of the graph corresponds to element 0 and the right edge of the graph corresponds to the last element in the array plus 1. The vertical axis will be labeled to correspond to the data values of each element in the array.

The graph of a single waveform looks similar to that of a single array except that horizontal and vertical units are also printed beside their respective axes. Also, the horizontal axis is now labeled such that each numeric label corresponds to the product of the element number and the data-sampling interval. (An example of both an array graph and a waveform graph is included in Section 1.)

Like the PRINT command, GRAPH can be used to compute an array or waveform expression before displaying the results. Some simpler examples of this capability are: multiplying an array or waveform by a constant, adding a constant to an array or waveform, and computing the absolute- or rms-value of an array or waveform. It is also possible to perform mathematical operations (such as multiplication and division) on two arrays, two waveforms, or an array and a waveform; however, in each case the array length of the two arguments must be the same. (For more information on legal operations involving waveforms and arrays, see the section on **Expression Evaluation** in the TEK SPS BASIC System Software manual.)

The GRAPH command syntax allows you to graph more than one array or more than one waveform in the same statement. However, GRAPHing arrays and waveforms together can not be done meaningfully unless the data sampling interval of each waveform is 0 or 1 -- in which case it is treated as an array. (When a waveform with a sampling interval of 0 is GRAPHed, the units are not displayed.)

It is possible to graph two or more arrays of different array lengths. The array of longest length is used as the standard for determining the appropriate horizontal scaling of the X-axis. The arrays

of smaller length are then scaled down to fit the existing horizontal scale.

It is also possible to graph two or more waveforms of different array lengths and/or data sampling intervals. In this case, the array length and sampling interval of each waveform are multiplied to get the data span for the corresponding waveform. The waveform with the longest data span is then used as the standard for determining the appropriate horizontal scaling of the X-axis. The waveforms with shorter data spans are then scaled down to fit the existing horizontal scale.

When graphing more than one waveform, the units printed on the graticule are applicable only to the first waveform in the argument list. In general, the units will not apply to other waveforms in the argument list, unless they just happen to be the same.

To illustrate some of the points just mentioned, consider the following program:

```
1Ø WAVEFORM WA IS A(255),SA,HA$,VA$
2Ø SA=.5\HA$="SECONDS"\VA$="VOLTS"
3Ø WAVEFORM WB IS B(511),SB,HB$,VB$
4Ø SB=1\HB$="SECONDS"\VB$="AMPS"
5Ø A=.Ø6283\INT A,A\A=SIN(A)
6Ø B=.Ø6283\INT B,B\B=COS(B)
7Ø GRAPH WA,WB/4
8Ø END
```

The above program defines two waveforms: WA and WB. WA is defined as a 256-element waveform with a sampling interval of 0.5. Thus its data span is 256*0.5 or 128. WB is defined as a 512-element waveform with a sampling interval of 1. Its data span is therefore 512. Lines 50 and 60 cause a sine wave to be created in waveform WA and a cosine wave to be created in waveform WB. Finally, line 70 GRAPHS waveforms WA and WB -- the latter being attenuated by a factor of 4 so that it can be more easily distinguished from waveform WA.

Figure 2-1 shows the results of running the above program. Notice that the units displayed apply to waveform WA which was the first argument listed in line 70. Also notice that the data range on the X-axis corresponds to that of the waveform with the longest data span, WB. The other waveform, WA, has a data span of 128 and is thus plotted in just the first one-fourth of the graticule area.

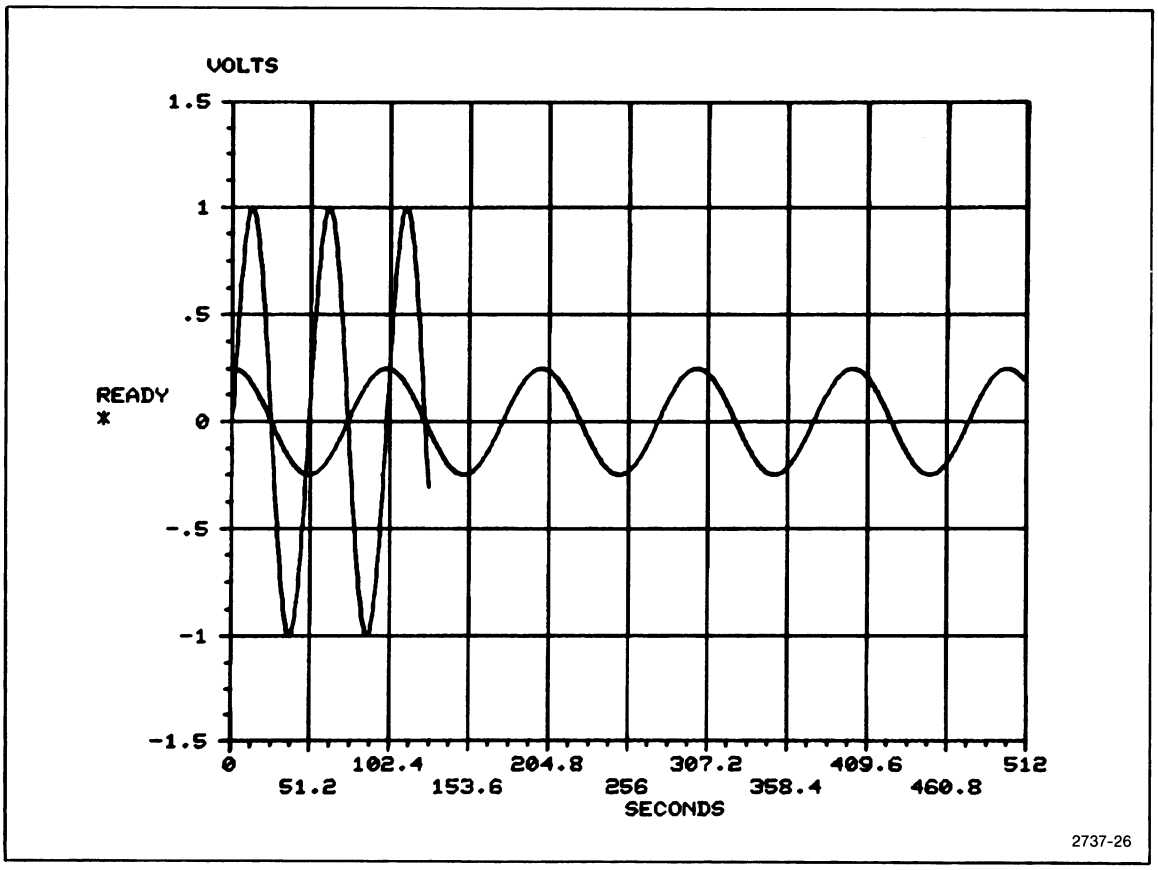


Fig. 2-1. Example of graphing two waveforms with different array lengths and sampling intervals.

INITG (Nonresident)

Examples:

```
15Ø INITG
```

Syntax Form:

[line no.] **INITG**

Purpose:

To initialize graphics terminal or other display device.

Discussion:

This command readies the graphics device by setting it to alpha mode and by moving the alpha cursor to the home position (located near the upper left-hand corner of the terminal screen or plotting surface). Also, the terminal screen is erased. The viewport is set to the full addressable graphics space (0 to 1023 in both the X and Y axes), and the window is set equal to the viewport. These are the conditions that exist just after TEK SPS BASIC has been loaded.

MOVE (Nonresident)

Examples:

```
3ØØ MOVE X,Y
31Ø MOVE 4ØØ,-1ØØ
```

Syntax Form:

[line no.] **MOVE** expression,expression

Descriptive Form:

[line no.] **MOVE** x coordinate in user units, y coordinate in user units

Purpose:

To move the pointer or pen of the graphics device to the (X,Y) position specified in user coordinates.

Discussion:

The MOVE command positions the pointer or pen to the specified coordinates without drawing a line. The location specified must be in user coordinates as determined by the current window. The VIEWPORT command determines the size and location of the user space within the plotting area of the graphics device. Only one set of coordinates are allowed in a MOVE command.

It is the user's responsibility to keep the pointer in the display area of the graphics device.

PAGE (Nonresident)

Example:

```
4ØØ PAGE
```

Syntax Form:

[line no.] **PAGE**

Purpose:

To prepare the graphics device for displaying a new page of text.

Discussion:

This command has the same effect as pressing the PAGE key on a graphics terminal. The screen of the terminal is erased and the alpha cursor is returned to the home position (upper-left corner of screen). The terminal is also set to alpha mode.

RDRAW (Nonresident)

Examples:

```
56Ø RDRAW X,Y
57Ø RDRAW 1ØØ,Ø,Ø,1ØØ,-1ØØ,Ø,Ø,-1ØØ,G(I),H(I)
```

Syntax Form:

[line no.] **RDRAW** expression,expression[,expression,expression] . . .

Descriptive Form:

[line no.] **RDRAW** x displacement in user units, y displacement in user units
[,x displacement in user units, y displacement in user units] . . .

Purpose:

To draw a line from the current pointer or pen position to a point displaced by the specified number of (X,Y) user units.

Discussion:

In its simplest form, the RDRAW command contains two arguments. The first argument specifies the X-displacement of the point to which a line will be drawn. The second argument specifies the Y-displacement of the same point. Both arguments specify the displacement, in user-units, relative to the current pointer or pen position. For example, if the pointer is presently located at (100,100) in the user data space, the command RDRAW 200,200 would draw a line from (100,100) to (300,300) in user coordinates.

Both arguments of RDRAW are in the user space defined by the last WINDOW command. If no WINDOW command has been executed, the default window (0 to 1023 in both X and Y axes) is used. The VIEWPORT command defines the size and location of the user space within the plotting area of the graphics device. It is the user's responsibility to keep the pointer in the graphics space of the display device; the software does not check the validity of the new location.

Using the Syntax Options:

Additional ordered pairs of (X,Y) displacements can be appended to a simple RDRAW command to cause additional line segments to be drawn. For example:

```
900 MOVE 500,500
910 RDRAW Ø,100,100,Ø,Ø,-100,-100,Ø
```

The above statement would draw a box 100 user-units square, with its lower left corner located at point (500,500) in the user data space. The first two arguments in line 910 (0,100) draw the left side of the box; the second pair of arguments (100,0) draw the top side; the third pair of arguments (0,-100) draw the right side; and the last pair of arguments (-100,0) draw the bottom side of the box.

RESETG (Nonresident)

Example:

```
16Ø RESETG
```

Syntax Form:

[line no.] **RESETG**

Purpose:

To initialize the window and viewport to their default values. (Does not erase screen of graphics terminal.)

Discussion:

This command functions the same as the INITG command with the exception that the terminal screen is not erased.

The terminal is set to alpha mode and the alpha cursor is moved to the home position (located near the upper-left corner of the terminal screen). Also, the window and viewport are set for a range of 0 to 1023 in both the X and Y axes.

RMOVE (Nonresident)

Examples:

```
5ØØ RMOVE X,Y
51Ø RMOVE X+3ØØ,E(P)-6Ø
```

Syntax Form:

[line no.] **RMOVE** expression,expression

Descriptive Form:

[line no.] **RMOVE** x displacement in user units, y displacement in user units

Purpose:

To move the pointer or pen of the graphics device to a position displaced from current position by specified number of (X,Y) user units.

Discussion:

The RMOVE command moves the pointer or pen to a location displaced from the current location by the specified number of user units. The values specified are X and Y displacements in user units, defined by the current window. If no window has been executed, the default window (0 to 1023 in both the X and Y axes) is used. The VIEWPORT command determines the size and location of the user space within the plotting area of the graphics device.

It is the user's responsibility to keep the pointer in the graphics area; the software does not check the validity of the new location.

RSDRAW (Nonresident)

Examples:

```
7ØØ RSDRAW X,Y
725 RSDRAW A,B,5Ø,85
```

Syntax Form:

[line no.] **RSDRAW** expression,expression[,expression,expression] . . .

Descriptive Form:

[line no.] **RSDRAW** x displacement in device units, y displacement in device units
[,x displacement in device units, y displacement in device units] . . .

Purpose:

To draw a line from the current pointer or pen position to a point displaced from current position by specified number of (X,Y) screen units.

Discussion:

The RSDRAW command operates just like the RDRAW command except that the arguments are always in screen units rather than user units. Since the arguments specify the X and Y displacements in screen units, the line is drawn without regard to the current window or viewport.

As many coordinate pairs as will fit on one line may be specified in the argument list. Minimum and maximum X and Y displacement values are 0 and 1023, respectively. No check is made of the validity of the specified locations. It is the user's responsibility to keep the pointer in the display area.

RSMOVE (Nonresident)

Examples:

```
1ØØ RSMOVE X,Y
2ØØ RSMOVE 1ØØ,B(I)*2
```

Syntax Form:

[line no.] **RSMOVE** expression,expression

Descriptive Form:

[line no.] **RSMOVE** x displacement in device units, y displacement in device units

Purpose:

To move the pointer or pen of the graphics device to a position displaced from the current position by the specified number of (X,Y) screen units.

Discussion:

The RSMOVE command operates just like the RMOVE command except that the arguments are always screen units rather than user units. Since the arguments specify the X and Y displacements in screen units, the move is made without regard to the current window or viewport.

The minimum and maximum displacement values are 0 and 1023, respectively, for both the X and Y axes. No check is made of the validity of the new location. It is the user's responsibility to keep the pointer in the display area.

SDRAW (Nonresident)

Examples:

```
46Ø SDRAW X,Y
47Ø SDRAW J,K,L+45,M(8)
```

Syntax Form:

[line no.] **SDRAW** expression,expression[,expression,expression] . . .

Descriptive Form:

[line no.] **SDRAW** x coordinate in device units, y coordinate in device units
[,x coordinate in device units, y coordinate in device units] . . .

Purpose:

To draw a line from the current pointer or pen position to the (X,Y) position specified in screen coordinates.

Discussion:

The SDRAW command operates just like the DRAW command except that the arguments are always screen coordinates rather than user coordinates. Since the arguments always refer to screen coordinates, the line segments are drawn without regard to the current window or viewport.

The minimum and maximum coordinates are 0 and 1023, respectively, for both the X and Y axes. No check is made of the validity of the new location. It is the user's responsibility to keep the pointer in the display area.

SEEVIEW (Nonresident)

Examples:

```
     SEEVIEW XL,XH,YL,YH
810  SEEVIEW P,Q,R(I),S(I)
```

Syntax Form:

[line no.] **SEEVIE** variable,variable,variable,variable

Descriptive Form:

[line no.] **SEEVIEW** target for low x value, target for high x value,
target for low y value, target for high y value

Purpose:

To obtain the screen coordinates that define the current graphics viewport.

Discussion:

The SEEVIEW command allows you to readily obtain the minimum and maximum X and Y coordinates that specify the size of the current viewport. The values returned are screen coordinates, and are not affected by the current window.

The argument list for SEEVIEW consists of four variables or array elements. After SEEVIEW has executed, these variables will contain, in order:

1) the minimum X-coordinate
2) the maximum X-coordinate
3) the minimum Y-coordinate
4) the maximum Y-coordinate

The following routine demonstrates the use of SEEVIEW. It draws a rectangle that outlines the boundaries of the current viewport.

```
1Ø SEEVIEW X1,X2,Y1,Y2
2Ø SMOVE X1,Y1
3Ø SDRAW X1,Y2
4Ø SDRAW X2,Y2
5Ø SDRAW X2,Y1
6Ø SDRAW X1,Y1
```

Line 10 enters the minimum and maximum X-values into X1 and X2; the minimum and maximum Y-values are entered into Y1 and Y2, respectively. Line 20 MOVEs the cursor to point (X1,Y1), the lower-left corner of the viewport boundary. Lines 30 through 60 then DRAW the left, upper, right, and lower boundaries of the viewport, respectively.

SEEWINDOW (Nonresident)

Examples:

```
    SEEWINDOW XL,XH,YL,YH
910 SEEWINDOW T,U,V(I),W(I)
```

Syntax Form:

[line no.] **SEEWIN** variable,variable,variable,variable

Descriptive Form:

[line no.] **SEEWINDOW** target for low x value, target for high x value,
target for low y value, target for high y value

Purpose:

To obtain the values that define the current graphics window.

Discussion:

The SEEWINDOW command allows you to readily obtain the minimum and maximum X and Y values that specify the size of the current window. The values returned are not affected by the current viewport.

The argument list for SEEWINDOW consists of four variables or array elements. After SEEWINDOW has executed, these variables will contain, in order:

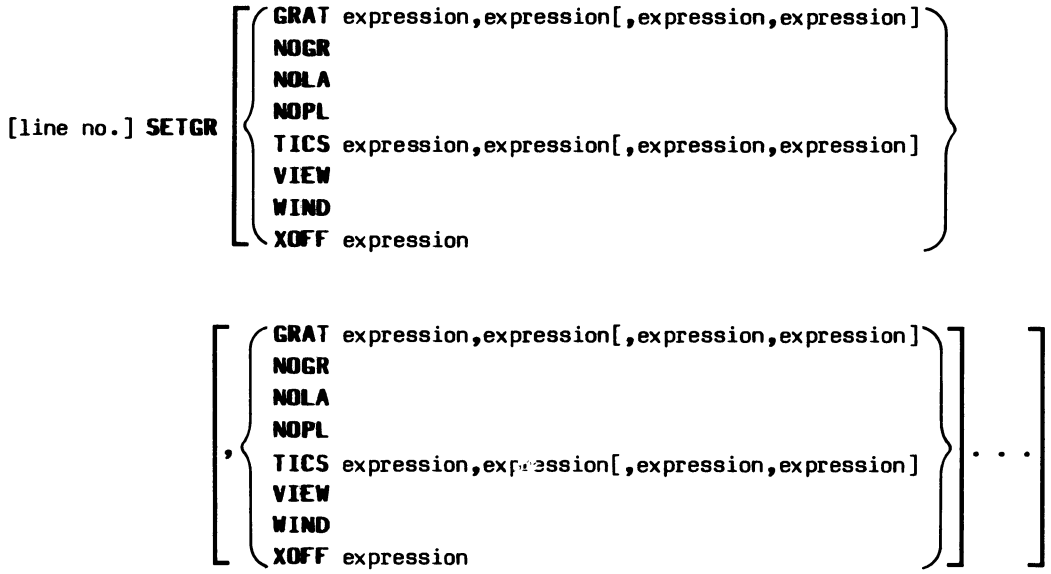
1) the minimum X-value
2) the maximum X-value
3) the minimum Y-value
4) the maximum Y-value

SETGR (Nonresident)

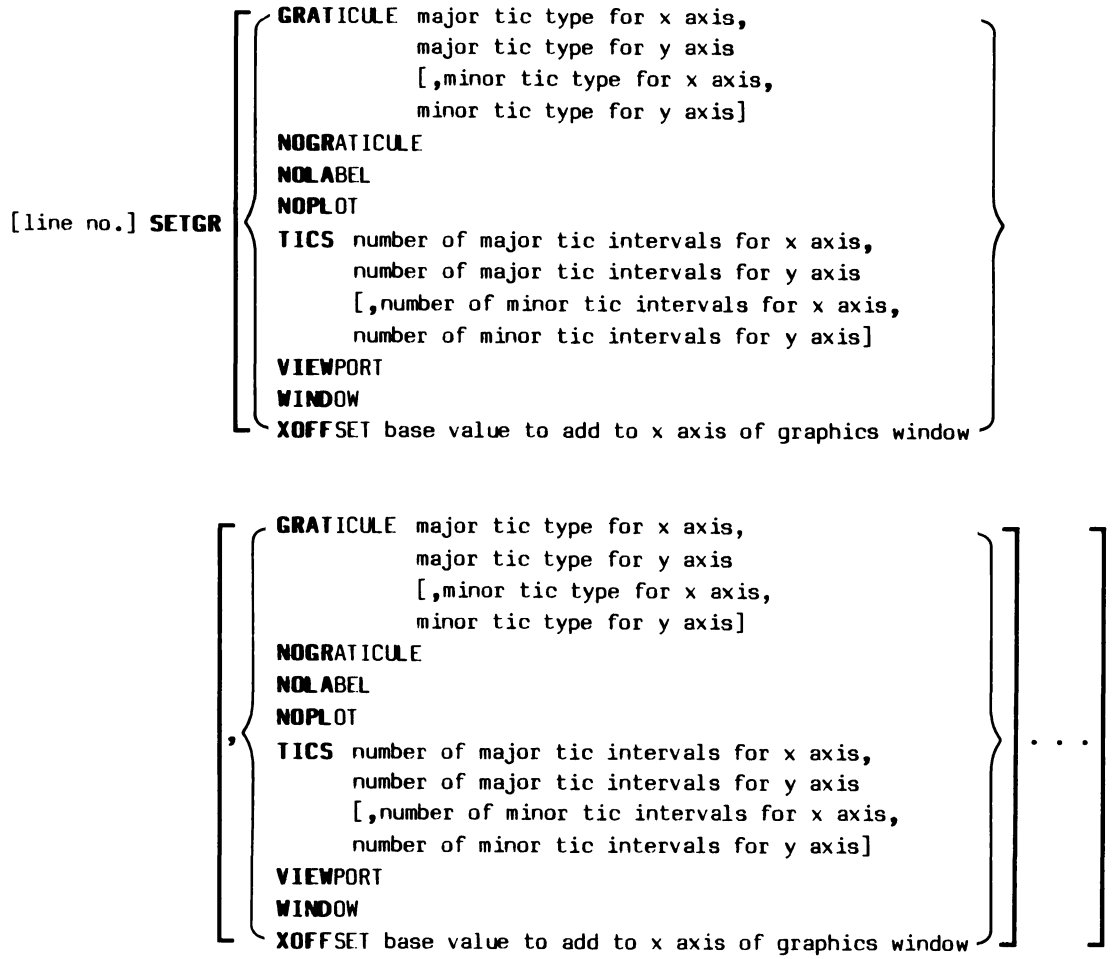
Examples:

```
    SETGR VIEW, WINDOW, TICS 4,3, GRAT 2,2,2,2
67Ø SETGR TICS 4,4,2,3, VIEWPORT, GRAT 5,5
76Ø SETGR NOPLOT,GRAT J,J+1,J,J
```

Syntax Form:



Descriptive Form:



Purpose:

To modify the default plots produced by the GRAPH and XYPLOT commands.

Discussion:

This command controls the graticule, window, and viewport of the next (and only the next) GRAPH or XYPLOT command. If the SETGR command is not executed, or executed with no arguments, the following default values are assumed by GRAPH. (XYPLOT uses the same default values except that the major tic marks default to type 2, rather than type 5.)

	X axis	Y axis	
Major Tic Intervals	10	0	(causes autoscaling)
Minor Tic Intervals	4	4	
Major Tic Type	5	5	
Minor Tic Type	2	2	
Viewport: Minimum	131±3	80±3	
Maximum	901±3	696±3	

Notice that the viewport shifts by as many as six device units each time it is displayed in the default mode. This is to prevent burning the CRT of the graphics terminal.

Since SETGR is only effective on the next GRAPH or XYPLOT command executed, all SETGR arguments must appear in the same statement. Thus:

```
1Ø SETGR VIEWPORT, TICS 4,4
2Ø SETGR GRAT 6,6,2,2
```

is not equivalent to ...

```
1Ø SETGR VIEWPORT, TICS 4,4, GRAT 6,6,2,2
```

but is equivalent to ...

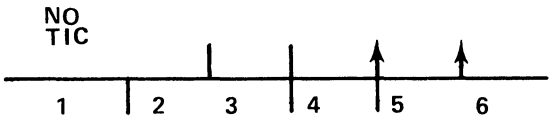
```
1Ø SETGR GRAT 6,6,2,2
```

Explanation of Keywords:

GRAT[ICULE] expression, expression [,expression,expression]

Specifies the type of major and minor graticule tic-mark types. The first argument pair must be present and represents the major tic-mark form for the X and Y axes. The second argument pair is optional and represents the forms of the minor tic marks for the X and Y axes.

The tic-mark types are as follows:



Arrows indicate that a full line is drawn to the opposite graticule edge.

NOGR[AT]

Specifies that no graticule is to be drawn when the next GRAPH or XYPLOT is executed.

NOLA[BEL]

Specifies that no labels be displayed beside the graticule or axes. Both the numeric axis labeling and the units labeling are suppressed by this keyword.

NOPL[OT]

Specifies that the array or waveform data will not be displayed when the next GRAPH or XYPLOT is executed.

TICS expression, expression [,expression,expression]

Specifies the number of major and minor graticule tic-mark intervals. The first two arguments must be present. These represent the major tic-mark intervals for the X and Y axes, respectively. The second optional pair represent the minor tic-mark intervals for the X and Y axes. If the Y value for a major tic-mark is 0, GRAPH or XYPLOT will select the number of tic-mark intervals that best displays the data.

VIEW[PORT]

Specifies that the current graphics viewport is to be used to define the location of the next GRAPH or XYPLOT. This option allows you to specify the size and location of the next plot.

WIND[OW]

Specifies that the current graphics window (defining the limits of the user coordinate system) is to be used to define the data ranges of the next GRAPH or XYPLOT command. This keyword overrides the autoscaling normally performed by GRAPH or XYPLOT.

XOFF[SET] expression

Specifies a base value to be added to the X-axis values of the graphics window (and thus to the labels) by the next GRAPH or DISPLAY command. The graph of the data will be shifted by the base value, but the data itself will not be altered. XOFF does not affect the XYPLOT command.

An Example Program. The following program is a simple routine that demonstrates how the SETGR command works. In this example, a sine wave is created and then graphed by means of the usual default parameters assumed by GRAPH. Following this graph, a SETGR command is executed which specifies a different window and viewport, as well as a different graticule. Also, the base values are changed by means of the XOFF keyword. Here is the program:

```
1Ø DIM X(511)
2Ø X=.Ø6283
3Ø INT X,X
4Ø X=SIN(X)
5Ø GRAPH X
6Ø WAIT 2ØØØ
7Ø WINDOW Ø,511,-2,2
8Ø VIEWPORT 2ØØ,7ØØ,3ØØ,7ØØ
9Ø SETGR WIND,VIEW,GRAT 4,4,2,2,XOFF 1ØØ
1ØØ PAGE
11Ø GRAPH X
```

Lines 10 through 40 create a sine wave by integrating a constant and then taking the SIN function of the resulting ramp. Line 50 graphs the sine wave, and since no SETGR command has been executed, the default parameters previously listed under the SETGR command discussion are used to determine how the graph is drawn. Line 60 simply causes the program to wait 2000 milliseconds (2 seconds) before proceding to line 70.

Line 70 defines a new data window that ranges from 0 to 511 in the X-axis and -2 to 2 in the Y-axis. Similarly, line 80 defines a new viewport that causes the next graph to occupy the space bounded by screen coordinates 200 to 700 (in the X-axis) and screen coordinates 300 to 700 (in the Y-axis). However, the new window and viewport will not take effect until line 90 is executed. In addition to specifying that the current window and viewport are to be used, the SETGR statement specifies a new GRATicule of type 4,4,2,2, and an XOFFset of 100.

After the terminal screen is erased (line 100) a graph of array X is created using the specifications listed in line 90. Figure 2-2 shows the graph of array X before and after executing the SETGR statement. Notice that the second graph not only displays a new window, viewport, and graticule type, but that the X-axis values also range from 100 to 611 -- an offset of 100 from the previous graph.

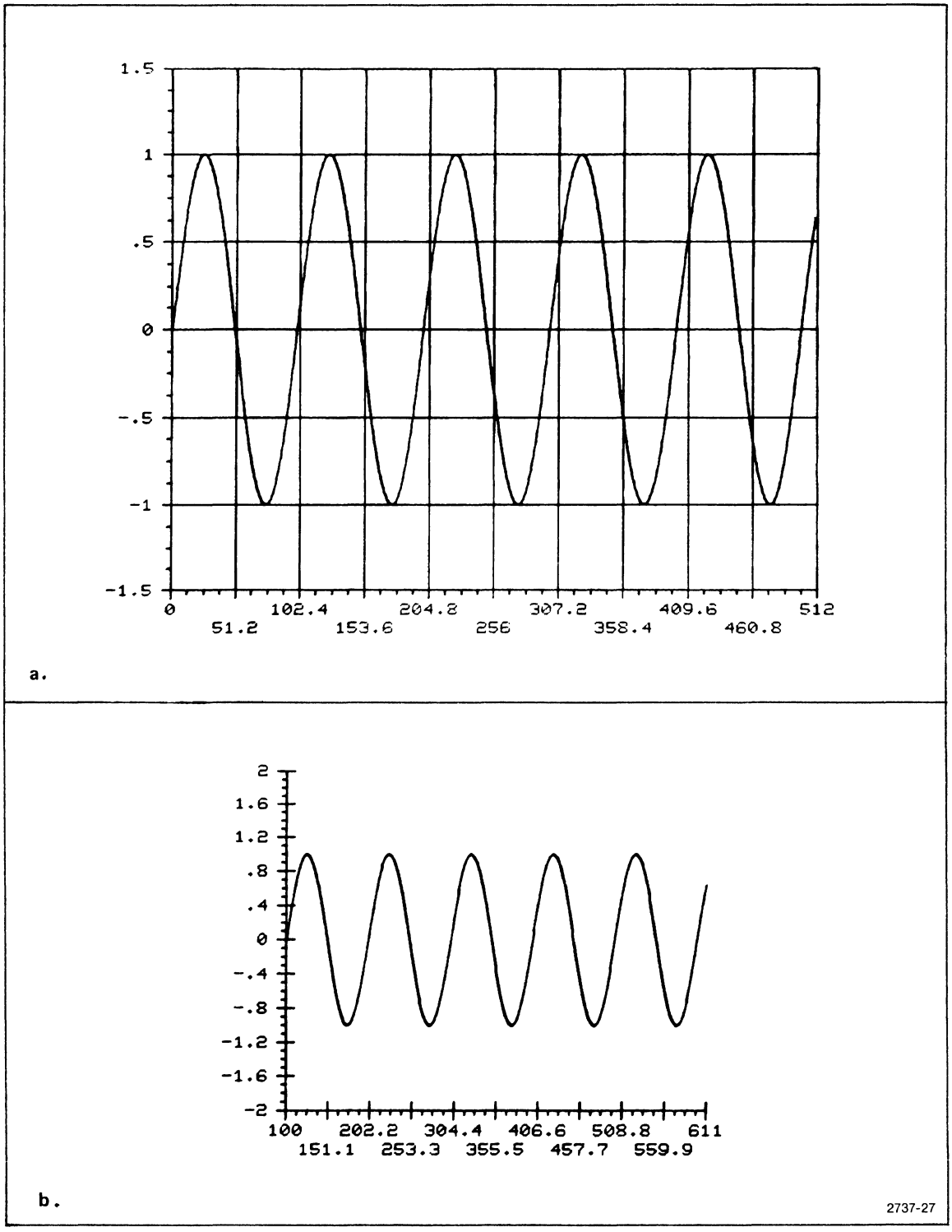


Fig. 2-2. Example of an array graph using (a) default parameters and (b) parameters set by SETGR command.

SGIN (Nonresident)

Examples:

```
255 SGIN C$, X,Y
266 SGIN A$(T), K,A(6)
```

Syntax Form:

[line no.] **SGIN** string variable,variable,variable

Descriptive Form:

[line no.] **SGIN** target for input character,
target for x coordinate of crosshairs in device units,
target for y coordinate of crosshairs in device units

Purpose:

To input the position of the terminal's cross-hair cursor in screen coordinates.

Discussion:

This command operates just like the GIN command except that the X and Y coordinates returned are screen coordinates rather than user coordinates. Since these values are always screen coordinates, they are not affected by the current graphics window or viewport.

SMOVE (Nonresident)

Examples:

```
545 SMOVE X,Y
645 SMOVE J+36,K+67
```

Syntax Form:

[line no.] **SMOVE** expression,expression

Descriptive Form:

[line no.] **SMOVE** x coordinate in device units, y coordinate in device units

Purpose:

To move the pointer or pen of the graphics device to the (X,Y) position specified in screen coordinates.

Discussion:

The SMOVE command operates just like the MOVE command except that the arguments are screen coordinates rather than user coordinates. Since the arguments are always screen coordinates, the move is not affected by the current window or viewport.

The minimum and maximum coordinates are 0 and 1023, respectively, for both the X and Y axes. No check is made of the validity of the new location. It is the user's responsibility to keep the pointer in the display area.

VIEWPORT (Nonresident)

Examples:

```
      VIEWPORT Ø,1Ø23,Ø,779
2ØØ VIEWPORT LX,HX,LY,HY
```

Syntax Form:

[line no.] **VIEWPO** expression,expression,expression,expression

Descriptive Form:

[line no.] **VIEWPO**RT minimum x coordinate in device units,
maximum x coordinate in device units,
minimum y coordinate in device units,
maximum y coordinate in device units

Purpose:

To delimit the portion of the graphics device to be used for drawing.

Discussion:

Normally, the entire usable area of the terminal or display device can be used for plotting data. However, it is sometimes necessary or convenient to restrict a plot to a portion of the total display area. This can be done with the VIEWPORT command.

The VIEWPORT command has four arguments. The first two arguments define the left and right X-axis boundaries; the last two define the bottom and top Y-axis boundaries, respectively. For example, the

following statement would define a viewport bounded by the screen coordinates 200 to 800 (in the X-axis) and 300 to 700 (in the Y-axis):

```
21Ø VIEWPORT 2ØØ,8ØØ,3ØØ,7ØØ
```

To return the viewport to the entire screen, you could execute either a RESETG or INITG command, or execute the following statement:

```
22Ø VIEWPORT Ø,1Ø23,Ø,779
```

Default values for the viewport are 0 and 1023 in both the X and Y axes. These values are the minimum and maximum values allowable in a VIEWPORT statement. (However, a maximum Y-value larger than 779 will not normally be displayed on a properly calibrated 4010-Series terminal.) The viewport arguments are always given in device (screen) coordinates, not user coordinates.

Commands affected by the VIEWPORT command are: DISPLAY, DRAW, RDRAW, MOVE, RMOVE, GIN and SEEVIEW. GRAPH and XYPLOT are also affected by the VIEWPORT command if the viewport has been enabled by a SETGR command.

WINDOW (Nonresident)

Examples:

```
      WINDOW 1965,1976,5ØØØ,1ØE+4
65Ø WINDOW Ø,SIZ(A),MIN(A),MAX(A)
```

Syntax Form:

```
[line no.] WINDOW expression,expression,expression,expression
```

Descriptive Form:

```
[line no.] WINDOW minimum x coordinate in user units,
                  maximum x coordinate in user units,
                  minimum y coordinate in user units,
                  maximum y coordinate in user units
```

Purpose:

To specify the range of user data to be drawn on selected portion of graphics device.

Discussion:

While the graphics terminal has a data range of 0 to 1023 in both the X and Y axes, this is rarely compatible with the range of user data to be graphed. Thus the user is faced with scaling the user data to fit the screen coordinate system. This mapping can be easily performed, however, with the WINDOW command. The WINDOW command automatically scales the data to fit within the default or specified viewport.

The WINDOW command has four arguments. The first two arguments define the minimum and maximum values displayable on the X-axis; the second pair of arguments define the minimum and maximum values displayable on the Y-axis. As an example, let's suppose you wanted to graph the number of employees at a corporation from the years 1958 to 1978. We'll assume that the number of employees increased from 5,000 to 30,000 during the same period. To graph this data with maximum resolution, you could execute the following statement:

```
66Ø WINDOW 1958,1978,5ØØØ,3ØØØØ
```

This would result in a data window ranging from 1958 to 1978 (in the X-axis) and from 5000 to 30000 (in the Y-axis).

Commands affected by the WINDOW command are: DISPLAY, DRAW, RDRAW, MOVE, RMOVE, GIN, and SEEWINDOW. GRAPH and XYPLOT are also affected by the WINDOW command if the window has been enabled by a SETGR command. Arguments required or returned by these commands are always in user coordinates, defined by the WINDOW command.

XYPLOT (Nonresident)

Examples:

```
    XYPLOT XA,YA
100 XYPLOT AX,AY,BX,BY
```

Syntax Form:

[line no.] **XYPLOT** { array / waveform }, { array / waveform } [, { array / waveform }, { array / waveform }] . . .

Purpose:

To plot the data in one array against the data in a second array. A graticule with axis labeling accompanies the plot.

Discussion:

In its simplest form, the XYPLOT command contains two arguments. Either of these arguments may be an array or a waveform since only the array data is used. The first argument in the list is used as the X-axis data; the second is used as the Y-axis data. These arrays (or arrays associated with waveforms) must have the same length.

If waveforms are specified for both arguments, the vertical units string of the first waveform is printed below the horizontal axis, and the vertical units string of the second waveform is printed above the vertical axis. **The data sampling intervals and horizontal units of both waveforms are ignored by XYPLOT.**

The X and Y data is automatically scaled for a "best fit" to the terminal screen or graphics device. However, this autoscaling can be overridden by the SETGR command. The SETGR command allows you to specify the number and types of major and minor tic marks for the display graticule. It also allows you to specify the size of the window and

viewport that is used. However, it should be remembered that SETGR affects only the **next** GRAPH or XYPLOT command executed. It does not have any effect on further XYPLOT commands unless SETGR is executed each time XYPLOT is executed. (For more information refer to the description of the SETGR command.)

The default graticule (displayed when SETGR is not in effect) is shifted by as many as six device units in each axis each time a new graticule is displayed. This shift prevents burning of the CRT terminal screen.

Using the Syntax Options:

Just as the GRAPH command allows you to plot more than one mathematical function, XYPLOT allows you to plot more than one mathematical relation. When plotting more than one relation with XYPLOT, each set of two arguments defines a relation to be plotted -- the first argument being the X-data and the second argument being the Y-data. Both arguments in each pair must be of the same array length, but arguments from different relations need not be of the same length.

Before the actual plotting of multiple relations begins, a computation is made to determine the "best fit" of the data in both the X and Y axes. All data plots are then scaled to fit these axes. Following the data plot, the horizontal and vertical axes are labeled with the vertical units from the first two arguments in the argument list. (However, if these arguments are arrays rather than waveforms, no units will be printed.)

The following program demonstrates some of the concepts just mentioned. It defines four waveforms: WX, WY, WZ, and WW. WX and WY are 100-element waveforms and WZ and WW are 200-element waveforms. Since XYPLOT ignores the horizontal units and data sampling intervals of its arguments, none were defined in this program.

```
1Ø WAVEFORM WX IS X(99),SX,HX$,VX$
2Ø WAVEFORM WY IS Y(99),SY,HY$,VY$
3Ø WAVEFORM WZ IS Z(199),SZ,HZ$,VZ$
4Ø WAVEFORM WW IS W(199),SW,HW$,VW$
5Ø X=.Ø6283\Y=3*X
6Ø Z=.Ø6283\W=Z
7Ø INT X,X\INT Y,Y
```

```
8Ø INT Z,Z\INT W,W
9Ø X=SIN(X)\Y=COS(Y)
1ØØ Z=SIN(Z)\W=COS(W)
11Ø VX$="VOLTS"\VY$="AMPS"
12Ø VZ$="FARADS"\VW$="HENRYS"
13Ø VIEWPORT 2ØØ,8ØØ,1ØØ,7ØØ
14Ø SETGR VIEW
15Ø XYPLOT WW,WZ,WY,WX
16Ø END
```

After defining the four waveforms WX through WW, WX and WZ are filled with one cycle and two cycles of sine-wave data, respectively. Similarly, WY is filled with three cycles of cosine-wave data and waveform WW is filled with two cycles of cosine-wave data. Lines 110 and 120 assign vertical units of "VOLTS" and "AMPS" to waveforms WX and WY, and units of "FARADS" and "HENRYS" to waveforms WZ and WW. Lines 130 and 140 result in a square viewport 600 screen units to a side. Finally at line 150, we see the XYPLOT of the four waveforms, which results in two Lissajous patterns. The WW-WZ relation is plotted first and results in a perfect circle. The WY-WX relation is then plotted which results in the three-lobed figure. Since WW-WZ was plotted first, the plot is labeled with units of "FARADS" in the vertical axis, and "HENRYS" in the horizontal axis. Figure 2-3 shows the results of running the program.

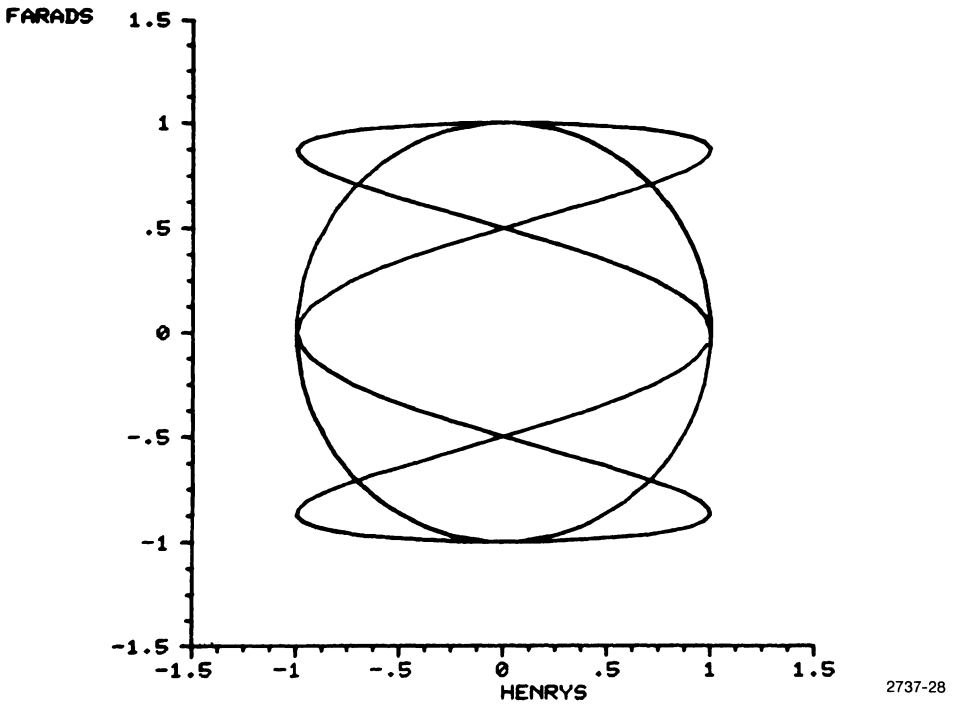


Fig. 2-3. Example using XYPLOT to display two X-Y relations.

SECTION 3

GRAPHICS TECHNIQUES AND APPLICATIONS

This section demonstrates some of the concepts discussed in Sections 1 and 2 by providing some typical applications programs. Each program is accompanied by a short article describing how the program is used, and briefly, how it works. Some of the articles were adapted from the HANDSHAKE column entitled "Getting the Most out of TEK BASIC GRAPHICS". (HANDSHAKE is a Tektronix newsletter provided free to users of signal processing systems.)

While these applications programs are by no means, a complete set of routines intended to solve every situation, it is hoped that the variety of techniques listed will serve as a starting point for developing your own specialized applications software. If, in the process of writing your specialized routines, you discover a novel idea and would be willing to share it with others, please let us know. We would be glad to include it in the TEK SPS BASIC Users Application Library or use it as the basis of a future HANDSHAKE article.

NOTE

Each issue of HANDSHAKE contains helpful information for users of signal processing systems. Features include applications, measurement concepts, new products, interfacing information, and additional graphics techniques and applications. For a free subscription to HANDSHAKE, write to:

HANDSHAKE Editor
Group 157 (94-384)
Tektronix, Inc.
P. O. Box 500
Beaverton, OR 97077

I. Three-dimensional Graphics

Techniques Illustrated:

Hidden-line plotting; time-signature plots; acquiring, logging and reading multiple traces from the DPO.

Optional TEK SPS BASIC Packages Required:

Graphics, DPO Driver.

Abstract:

The following HANDSHAKE article (from Vol. 2 No. 3) demonstrates a time-signature plot by acquiring heartbeat data from a DPO connected to a TEKTRONIX 414 Portable Patient Monitor.

Article:

Studying Long-Term Variations

What happens when a signal is repetitive, but changes slowly over long periods of time? How do we see this change clearly, and how do we record it for later analysis?

TEK SPS BASIC provides an easy solution. Figure 3-1 is the output of a program that captures waveforms from a TEKTRONIX Digitizing Oscilloscope (DPO). The plotted signal is the output of a TEKTRONIX 414 Patient Monitor set to record electrocardiograms (ECGs). Each horizontal trace represents two seconds of time, while the time between traces (moving up the screen) is about one second.

The DPO is set to trigger on the QRS complex of the ECG signal. This gives a common starting point for each trace. Note the smooth lineup of the T wave at the left of the figure.

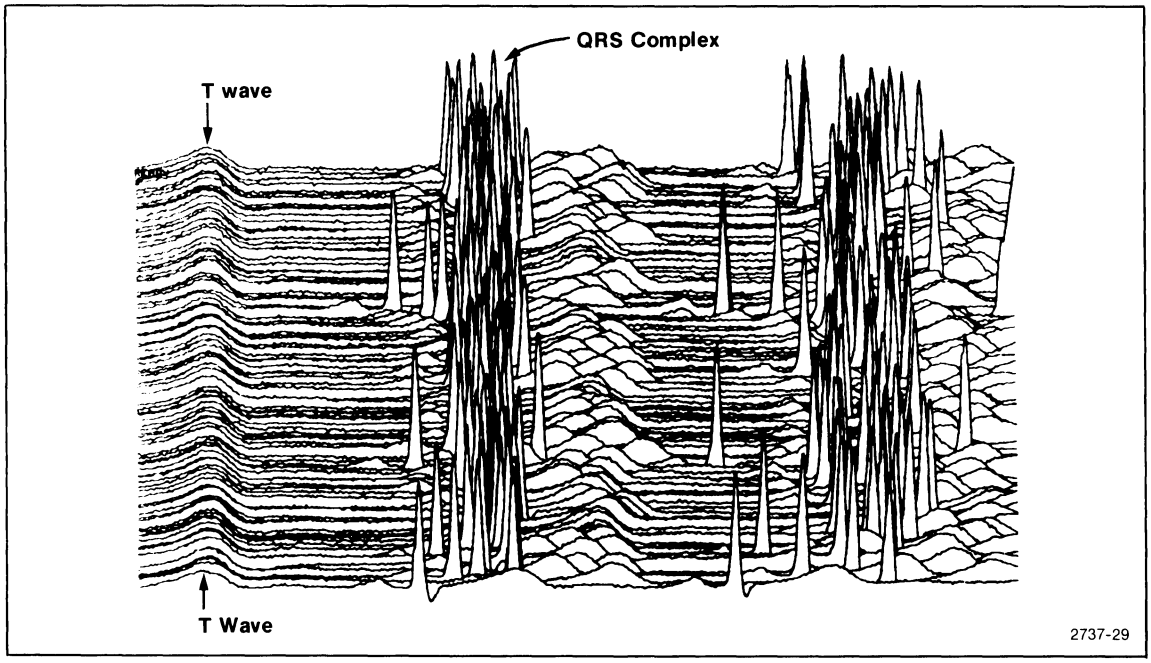


Fig. 3-1. Electrocardiogram plot.

WARNING

Other than 400-Series Patient Monitors, Tektronix equipment is not designed for, nor intended for, direct attachment to human subjects or use in patient-care areas. Electronic equipment, individually or as a system, used for measurement or monitoring of human patients should be certified patient safe and should be used only under the direction of qualified personnel.

This plot represents 100 samples, or about 300 seconds of patient history. To see how it was done, refer to the program listing in Fig. 3-2.

First, the data is acquired and written out to a peripheral file. The acquisition part of the program is from line 3000 to 3240. Not shown is the acquisition of a ground trace from the DPO. The DPO is already in single-sweep mode. Line 3110 clears array A in the DPO and line 3120 arms the single sweep.

The interrupt ability of TEK SPS BASIC is used to tell when the sweep is complete. When the sweep is complete, an interrupt, set by line 3140, causes program control to transfer to line 3230. Here, a switch is turned on and control returns to the mainline program. The program has been looping at line 3150, waiting for the switch to increment. Line 3160 stops the storage of the trace, the waveform is acquired at line 3170, and the array is written out to the peripheral file. This process is repeated until the requested number of samples has been taken.

Next, the file is opened again, this time for reading. The subroutine starting at line 1000 finds the minimum and maximum values of all the array elements. These values are used in the next part of the program.

The plot section starts at line 80, where a hidden-line array (P), and a working hidden-line array (PP) are defined. These arrays are used to determine if a point on the screen is visible or not. Two arrays are necessary; one contains the hidden-line values for the previous plot (what's hidden already), and another is updated as you move across the screen. This second array becomes the next hidden-line array at the end of each plotting cycle.

All data points are in "user" coordinates to save the trouble of mapping the coordinates into screen units. The WINDOW command at line 150 does this for you. We know that the X range of data is from zero to 511 (the size of a DPO array). We also know the maximum and minimum values of the data (DH and DL). These values, along with a step size variable (ST) are used to define the size of the data window.

There are four possible situations encountered in hidden-line plotting: **1)** going from a visible point to a visible point, **2)** going from a visible point to a hidden point, **3)** going from a hidden point to a visible point, and **4)** going from a hidden point to a hidden point. The IF statement starting at line 210 checks for all of these conditions. If condition 2 or 3 is encountered, a subroutine is called to determine the point of intersection (where the visible line meets the hidden-line value).

At the end of each plot, the working hidden-line array is transferred to the previous array and variable Y, which determines the base line of the plot, is updated by the vertical step size ST.

by Garey Fouts, HANDSHAKE Staff
Program and data by David Stubbs, Emergency Medical Technician

```
10 REM - ACQUIRE DATA
20 GOSUB 3000
30 REM - FIND DATA MIN AND MAX
40 OPEN #1 AS "HEART.DAT" FOR READ
50 GOSUB 1000
60 REM - DISPLAY DATA
70 RESET #1
80 DIM P(511),PP(511)
90 P=DL
100 PAGE
110 Y=0
120 ST=(DH-DL)/20
130 WINDOW 0,511,DL,(DH-DL)+ST*(N-1)
140 FOR II=1 TO N
150 READ #1,A
160 A=A+Y
170 MOVE 0,A(0)
180 PP=P
190 FOR X=1 TO 511
200 XM=X-1
210 IF A(XM)<=P(XM) THEN GOTO 240
220 IF A(X)>P(X) THEN GOTO 290
230 IF A(X)<=P(X) THEN GOTO 350
240 IF A(X)<=P(X) THEN GOTO 490
250 IF A(X)>P(X) THEN GOTO 450
260 REM
270 REM - OUTSIDE DRAWING TO OUTSIDE
280 REM
290 DRAW X,A(X)
300 PP(X)=A(X)
310 GOTO 490
320 REM
330 REM - OUTSIDE DRAWING TO INSIDE
340 REM
350 GOSUB 2030
360 DRAW X1,YI
370 GOTO 490
380 REM
390 REM - INSIDE DRAWING TO INSIDE
400 REM
410 REM        SKIP IT
420 REM
430 REM - INSIDE DRAWING TO OUTSIDE
440 REM
450 GOSUB 2030
460 MOVE XI,YI
470 DRAW X,A(X)
480 PP(X)=A(X)
490 NEXT X
500 P=PP
510 Y=Y+ST
520 NEXT II
999 END
1000 REM
1010 REM - FIND DATA MIN AND MAX
1020 REM
1030 DH=-1000000
1040 DL=1000000
1050 FOR I=1 TO N
1060 READ #1,A
1070 IF DH<MAX(A) THEN DH=MAX(A)
1080 IF DL>MIN(A) THEN DL=MIN(A)
1090 NEXT I
1100 RETURN
2000 REM
2010 REM - FIND INTERMEDIATE POINT
2020 REM
2030 S1=(P(X)-P(XM))/(X-XM)
2040 S2=(A(X)-A(XM))/(X-XM)
2050 XI=(P(XM)-S1*XM-A(XM)+S2*XM)/(S2-S1)
2060 YI=S2*X1+A(XM)-S2*XM
2070 RETURN
3000 REM
3010 REM - ACQUIRE DATA
3020 REM
3030 PRINT "ENTER NUMBER OF SAMPLES";
3040 PRINT " TO BE TAKEN: ";
3050 INPUT N
3060 DIM A(511)
3070 WAVEFORM HB IS A,DA,HA$,VA$
3080 CANCEL "HEART.DAT"
3090 OPEN #1 AS "HEART.DAT" FOR WRITE WITH 3
3100 FOR II=1 TO N
3110 PUT "STO" INTO #1,"A"
3120 PUT "SSA" INTO #1
3130 SW=0
3140 WHEN #1 HAS "SSC" GOSUB 3230
3150 IF SW=0 THEN 3150
3160 PUT "HOL" INTO #1,"A"
3170 GET HB FROM #1,"A"
3180 WRITE #1,A
3190 NEXT II
3200 CLOSE #1
3210 RETURN
3220 REM - SET SWITCH
3230 SW=1
3240 RETURN
```

2737-30

Fig. 3-2. TEK SPS BASIC program for studying long term variations.

I. Three-dimensional Graphics (cont.)

Techniques Illustrated:

Hidden-line plotting, plotting a mathematical function in three dimensions.

Optional TEK SPS BASIC Packages Required:

Graphics, Signal Processing.

Abstract:

The following HANDSHAKE article (from Vol. 2 No. 2) demonstrates a routine for creating an orthogonal projection of the sin x/x function.

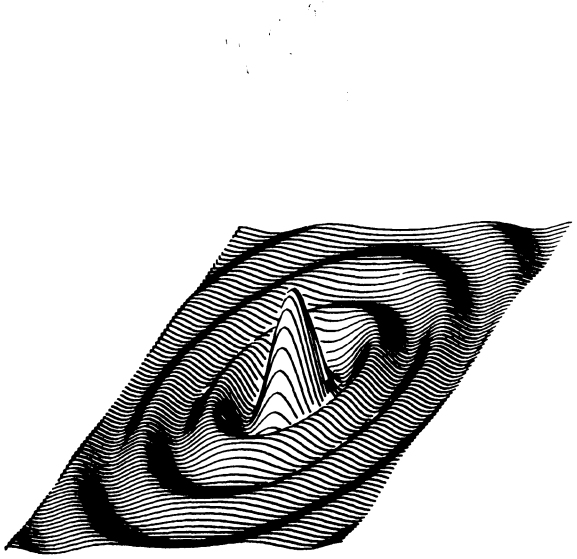
Article:

Orthogonal Projection of Sin x/x

The following program, written in TEK SPS BASIC, uses several of the optional Graphics Package commands.

The WINDOW command is used to keep all plotted lines within the bounds of the display area. Two FOR loops, starting at line 100 and 120, control the Y and X plotting respectively. Variable IV determines the coarseness of the display (distance between lines), and the MOVE command is used to hide some of the lines.

The hidden-line algorithm is straightforward. An array (YM) is initially set to the value corresponding to the bottom of the display. As data is drawn out to the screen, the vertical value (Y1) is compared with the contents of the array at the element corresponding to the horizontal position of the line. If the array value is less than the value of the current line's height, the array element is set to Y1 and the line is drawn.



```
10 PAGE
20 PI=3.14159
30 MX=6*PI\ZM=8
40 IV=PI/7
45 DF=0
50 DELETE YM
60 DIM YM(1023)
70 YM=-MX
80 ST=SIN(PI/4)\CT=COS(PI/4)
90 WINDOW -MX-.5,MX+2*CT*MX+.5,-MX*ST-.5,MX*ST+ZM+.5
100 FOR Y=-MX TO MX STEP IV
110 MOVE -MX+CT*(MX+Y),Y*ST
120 FOR X=-MX TO MX STEP IV
130 Z=SQR(X*X+Y*Y)
140 IF Z<>0 THEN 170
150 Z=ZM
160 GOTO 180
170 Z=ZM*SIN(Z)/Z
180 EN=(X+CT*(MX+Y)+MX+.5)/IV
185 Y1=Y*ST+Z
190 IF DF=0 THEN 240
200 IF Y1<YM(EN) THEN 290
210 YM(EN)=Y1
215 DF=0
220 MOVE X+CT*(MX+Y),Y1
230 GOTO 290
240 IF Y1>YM(EN) THEN 270
250 DF=1
260 GOTO 290
270 YM(EN)=Y1
280 DRAW X+CT*(MX+Y),Y1
290 NEXT X
300 NEXT Y
```

2737-31

Fig. 3-3. Sin x/x plot and the program that generates it.

When a situation is found where the array value is greater than or equal to the Y1 value, the line is not drawn. Instead, a MOVE to the specified position is performed, thus avoiding a line being drawn over a previously drawn line.

by Garey Fouts, HANDSHAKE Staff
program by John Shaw, Tektronix SPS Programmer

I. Three-dimensional Graphics (cont.)

Techniques Illustrated:

Hidden-line plotting, three-dimensional time-signature plots.

Optional TEK SPS BASIC Packages Required:

Graphics.

Abstract:

The following article describes a program that draws a three-dimensional time-signature plot. The plot is accompanied by three axes, each with its own scale and predetermined units. However, these parameters can easily be changed to conform to any given application.

Article:

Orthogonal Projection of a Time-Signature Plot

Spectrum analysis is a powerful technique for showing the dependence of signal amplitude on frequency, and is very useful in applications such as analyzing vibrations and resonances in rotating machinery. However, even more insight can be gained by noting the effect of a third parameter -- time. By noting how a spectrum changes with time, one can often detect patterns that appear only intermittently or that evolve with time. This article describes a TEK SPS BASIC program that displays such a time-signature plot. (This program does not include any code for data acquisition, since a similar acquisition routine was included in a previous article.) Given any number of traces (up to 100 anyway) the program will produce an orthographic projection of reasonable accuracy.

The program first asks for the name of the file in which the data is stored. It assumes the file is stored on the system device. It then asks for the number of traces to be projected. Next, some sage advice is given on how to select the proper angles for a nice picture; the user is then asked to specify these angles. The program does not pass judgement on the

inputs, and if the input angles would cause the drawing to go off screen, the program does not prevent it.

At this point, the input angles are converted to radians, and the transforms for the X and Y arrays are computed (tanØ and tanθ). The X, Y, and two hidden-line arrays are set up, and then the maximum and minimum Y values are found. In this program, X ranges from 0 to 511, because no data acquisition is done. The user can easily change the program to reflect the proper values of X.

A trace is read in and scaled to show up better (line 310), in case the Y values are much smaller or larger with respect to the X values. The X and Y arrays are then converted to screen points and sheared appropriately (lines 330 to 390). If this is the first or last trace, either the first or last points are saved to set up the axes later. The determination of which points are saved depends upon whether X is sheared to the right or left.

The display loop is then started. The points are checked against the hidden-line array to see if they should be displayed or not. The decision to display or not display a line depends upon which of the following test cases applies:

1) **Outside drawing to outside:** the line is drawn, and the hidden line is updated.

2) **Outside drawing to inside:** an intersecting point between the array and hidden-line array must be found and drawn to.

3) **Inside drawing to inside:** nothing is done.

4) **Inside drawing to outside:** the point of emergence is found and a line is drawn from the point of emergence to the current point.

These tests are repeated for all points in all traces to be drawn.

The last actions of the program are to plot the axes and print the accompanying units. Again, the actual units are not acquired by the program, so some default units are used (POINTS PER TRACE, TRACE, and AMPLITUDE). However, the user can easily change this. Figures 3-4 and 3-5 show two examples of running the program. Figure 3-4 shows a real data plot consisting of 10 traces of electrocardiogram data. Figure 3-5 shows a simulated data plot containing 100 traces.

To modify the program for acquisition of actual signal data from an instrument, simply follow the rules outlined below:

How to change this program for data acquisition

line 210 -- Get sampling interval from instrument. Multiply sampling interval by instrument's memory length (normally 512) to obtain total value of array X. Example: time/trace = 5 sec.; X(0) = 0, X(511) = 5.

line 1320 -- For variable DI, get elapsed time between acquisition of first and last trace; also get units.

line 1399 -- Instead of "TRACE", print the time (or other) units.

line 1401 -- DX = X(511)/5

line 1480 -- Instead of "POINTS PER TRACE", print the frequency (or other) units.

line 1590 -- Instead of "AMPLITUDE", print the magnitude (or other) units.

The actual data acquisition routine is up to you. For help on writing a data-acquisition routine, consult the instruction manual for your acquisition instrument.

by Lynne Axel, SPS Software Engineer,
and Cliff Morgan, SPS Documentation Group

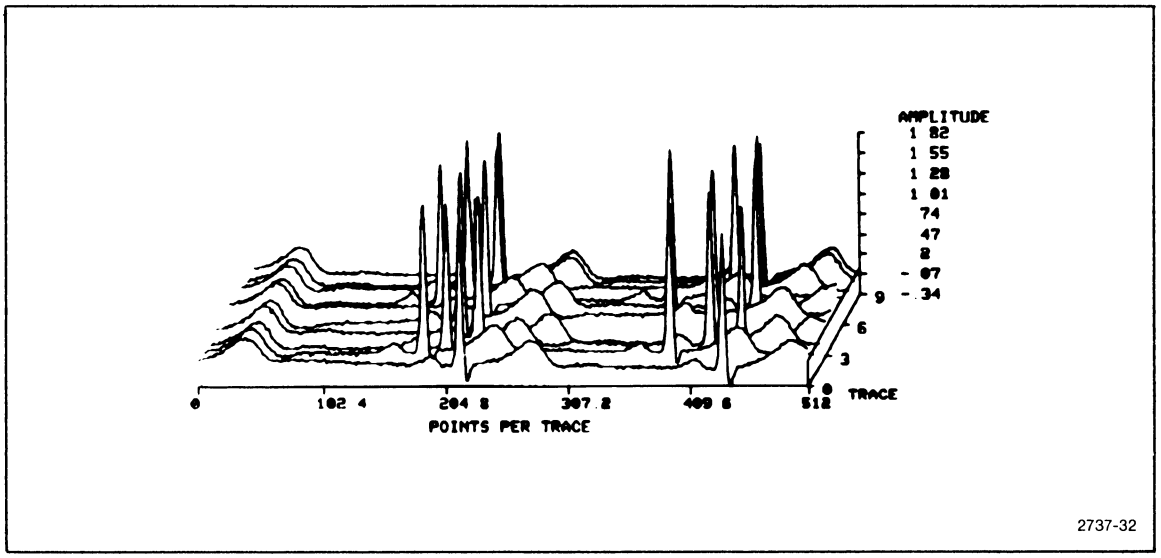


Fig. 3-4. 10 traces of real signal data.

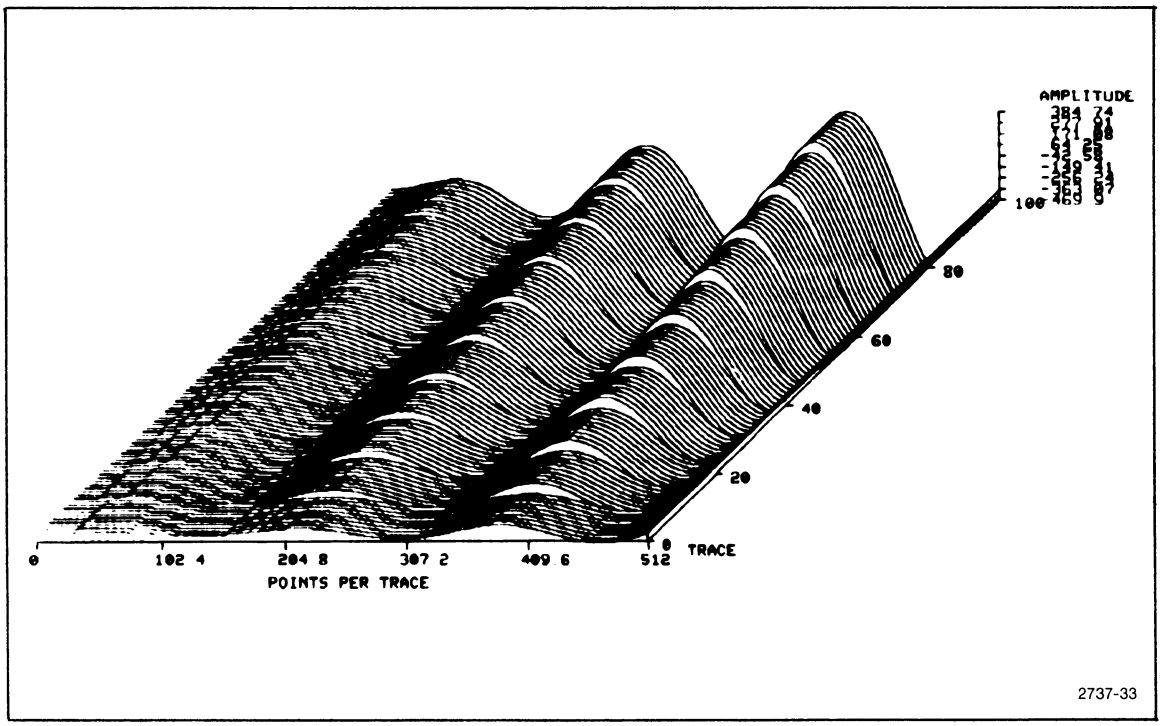


Fig. 3-5. 100 traces of simulated signal data.

```
1 REM          ****ORTHOG.PRJ****
3 REM  A PROGRAM IN TEK SPS BASIC TO DO ORTHOGRAPHIC PROJECTIONS
4 REM  OF UP TO 1ØØ 512 POINT ARRAYS OR WAVE TRACES.
6 REM  INPUTS:
7 REM      A$--NAME OF THE DATA FILE WHERE THE DATA TO BE GRAPHED
8 REM          IS STORED.  THERE SHOULD BE Z TRACES OF 512
9 REM          POINTS EACH.
1Ø REM     Z--NUMBER OF TRACES TO BE PROJECTED 1<=Z<=1ØØ
11 REM     PHI--Y SHEAR ANGLE IN DEGREES -89<=PHI<=89
12 REM     THETA--X SHEAR ANGLE IN DEGREES -89<=THETA<=89
14 REM OUTPUT:
15 REM     AN ORTHOGRAPHIC PROJECTION OF THE DATA FILE ON THE DISPLAY
16 REM         SCREEN WITH APPROPRIATE AXES AND LABELING.
19 CLOSE #1
2Ø PRINT "INPUT NAME OF DATA FILE"\INPUT A$
3Ø OPEN #1 AS A$ FOR READ
4Ø PRINT "INPUT NUMBER OF TRACES WANTED"
5Ø INPUT Z
51 PRINT " FOR A SMALL NUMBER OF TRACES; LIKE UNDER 4Ø , YOU"
52 PRINT " SHOULD SPACE THE LINES FURTHER APART BY INCREASING THE"
53 PRINT " Y SHEAR ANGLE. DO NOT; HOWEVER, LET THE X OR Y SHEAR ANGLES"
54 PRINT " EXCEED PLUS OR MINUS 89 DEGREES!!!"
6Ø PRINT " INPUT PHI (Y SHEAR ANGLE) AND THETA (X SHEAR ANGLE)"
7Ø INPUT PH,TH
75 REM CONVERT PHI AND THETA TO RADIANS
8Ø PH=PH*2*3.14159/36Ø
9Ø TH=TH*2*3.14159/36Ø
95 REM THE SHEAR ANGLE IS THE TANGENT OF PHI OR THETA, RESPECTIVELY
1ØØ TX=SIN(TH)/COS(TH)
11Ø TY=SIN(PH)/COS(PH)
12Ø REM SET UP HIDDEN LINE AND WORKING ARRAYS
13Ø DIM X(511),Y(511),WH(1Ø23),H(1Ø23)
15Ø REM  FIND DATA MAX AND MIN
17Ø GOSUB 82Ø
19Ø REM  INITIALIZE HIDDEN LINE ARRAYS
2ØØ WH=Ø
21Ø H=Ø
23Ø REM  START DATA READ IN AND CONVERSION
25Ø RESET #1
26Ø PAGE
27Ø FOR Z1=1 TO Z
28Ø Z2=Z1-1
29Ø X=1\INT X,X
3ØØ READ #1,Y
31Ø Y=Y*2ØØ/(YH-ZZ)
32Ø REM  CONVERT X AND Y TO SCREEN COORDINATES
33Ø IF TX<=Ø THEN GOTO 35Ø
34Ø X=X+1ØØ+TX*Z2*3\GOTO 36Ø
35Ø X=X+4ØØ+TX*Z2*3
36Ø Y=39Ø/512*Y+1ØØ+TY*Z2*3
386 REM  PULL OUT SCREEN POINTS FOR PROPER PLACEMENT OF AXES
39Ø IF Z1=1 THEN GOSUB 1Ø3Ø
4ØØ IF Z1=Z THEN GOSUB 111Ø
41Ø SMOVE X(Ø),Y(Ø)
42Ø REM  DISPLAY LOOP
43Ø WH=H
44Ø FOR I=1 TO 511
45Ø XM=I-1
46Ø IF Y(XM)<=H(X(XM)) THEN GOTO 49Ø
47Ø IF Y(I)>H(X(I)) THEN GOTO 54Ø
48Ø IF Y(I)<=H(X(I)) THEN GOTO 6ØØ
49Ø IF Y(I)<=H(X(I)) THEN GOTO 74Ø
5ØØ IF Y(I)>H(X(I)) THEN GOTO 7ØØ
52Ø REM  OUTSIDE DRAWING TO OUTSIDE
54Ø SDRAW X(I),Y(I)
55Ø WH(X(I))=Y(I)
56Ø GOTO 74Ø
58Ø REM  OUTSIDE DRAWING TO INSIDE
6ØØ GOSUB 94Ø
61Ø SDRAW XI,YI
62Ø GOTO 74Ø
64Ø REM  INSIDE DRAWING TO INSIDE
66Ø REM       SKIP IT
68Ø REM INSIDE DRAWING TO OUTSIDE
7ØØ GOSUB 94Ø
71Ø SMOVE XI,YI
72Ø SDRAW X(I),Y(I)
73Ø WH(X(I))=Y(I)
74Ø NEXT I
75Ø H=WH
76Ø NEXT Z1
766 REM PUT IN AXES AND LABELING
77Ø GOSUB 118Ø
78Ø END
8ØØ REM  FIND DATA MIN AND MAX
82Ø YH=-1ØØØØØØ
83Ø YL=1ØØØØØØ
```


@

```
840 FOR I=1 TO Z
850 READ #1,Y
856 REM SAVE A POINT FOR AXES LABELING
860 IF I=I THEN GOSUB 1620
870 IF YH<MAX(Y) THEN YH=MAX(Y)
880 IF YL>MIN(Y) THEN YL=MIN(Y)
890 NEXT I
900 RETURN
920 REM  FIND INTERMEDIATE POINT
940 S1=(H(X(I))-H(X(XM)))/(I-XM)
950 S2=(Y(I)-Y(XM))/(I-XM)
960 XI=(H(X(XM))-S1*XM-Y(XM)+S2*XM)/(S2-S1)
970 YI=S2*XI+Y(XM)-S2*XM
980 IF TX<=0 THEN GOTO 1000
990 XI=XI+100+TX*Z2*3\GOTO 1010
1000 XI=XI+400+TX*Z2*3
1010 RETURN
1020 REM SAVING SCREEN LOCATIONS FOR AXIS PLACEMENT
1030 DIM A(1),B(1)
1040 IF TX<=0 THEN GOTO 1060
1050 A(0)=X(511)\GOTO 1070
1060 A(0)=X(0)
1070 A(1)=Y(0)
1080 AA=A(1)-MIN(Y)
1090 RETURN
1100 REM ENTRY FOR LAST LOCATION SAVE IS HERE
1110 IF TX<=0 THEN GOTO 1130
1120 B(0)=X(511)\GOTO 1140
1130 B(0)=X(0)
1140 B(1)=Y(0)
1150 BB=MAX(Y)
1160 RETURN
1170 REM DRAWING THE AXES
1180 IF TX<=0 THEN GOTO 1210
1190 SMOVE A(0)-511,A(1)-AA
1200 GOTO 1220
1210 SMOVE A(0)+511,A(1)-AA
1220 SDRAW A(0),A(1)-AA
1230 SDRAW A(0),A(1)
1240 SMOVE A(0),A(1)-AA
1250 SDRAW B(0),B(1)-AA
1260 SDRAW B(0),BB
1270 SMOVE B(0),B(1)
1280 SDRAW A(0),A(1)
1290 REM ADDING IN UNITS
1310 REM Z AXIS--TIME
1320 DI=5
1330 IF Z<20 THEN DI=3
1340 DX=(B(0)-A(0))/DI
1350 BC=(B(1)-A(1))/DI
1360 Q=ITP(Z/DI)
1370 FOR I=0 TO DI
1380 SMOVE A(0)+I*DX,A(1)-(AA+4)+I*BC
1390 PRINT "-"
1391 SP=-30
1392 IF TX>0 THEN SP=5
1394 SMOVE A(0)+I*DX+SP,A(1)-(AA+7)+I*BC
1395 PRINT Q*I
1396 NEXT I
1397 SP=-70
1398 IF TX>0 THEN SP=35
1399 SMOVE A(0)+SP,A(1)-AA-12\PRINT "TRACE"
1401 REM X AXIS--FREQUENCY
1409 DX=102.4
1410 IF TX>0 THEN A(0)=A(0)-511
1420 FOR I=0 TO 5
1430 SMOVE A(0)+DX*I,A(1)-AA
1440 SDRAW A(0)+DX*I,A(1)-(AA+7)
1450 SMOVE A(0)+DX*I-14,A(1)-AA-20
1460 PRINT I*DX
1470 NEXT I
1480 SMOVE A(0)+2*DX-14,A(1)-AA-40\PRINT "POINTS PER TRACE"
1490 REM Y AXIS--AMPLITUDE
1500 Q=YH-ZZ\QQ=ITP(Q*100/8)/100
1510 ZZ=ITP(ZZ*100)/100
1520 BC=(BB-B(1)+AA)/8
1530 FOR P=0 TO 8
1540 SP=-80
1550 IF TX>0 THEN SP=35
1560 SMOVE B(0),BC*P+B(1)-(AA+4)\PRINT "-"
1570 SMOVE B(0)+SP,BC*P+B(1)-(AA+4)
1580 PRINT ZZ+QQ*P
1590 NEXT P
1591 SMOVE B(0)+SP,BB+10\PRINT "AMPLITUDE"
1600 RETURN
1610 REM GETTING VALUE FOR AXES
1620 ZZ=MIN(Y)
1630 RETURN
```

II. Using the 4662 Digital Plotter

Techniques Illustrated:

Directing computer graphics to the 4662; creating multiple color graphs on paper or overhead transparencies.

Optional TEK SPS BASIC Packages Required:

Graphics.

Abstract:

The following HANDSHAKE article (from Vol. 3 No. 2) describes how to interface a TEKTRONIX 4662 Interactive Digital Plotter to a signal processing system. A demonstration program is included illustrating how to obtain multiple-waveform plots via the 4662.

Article:

For a More Exciting Plot, Try the 4662

The TEK SPS BASIC Graphics Package has already proven its ability to make a variety of plots and graphs via a graphics terminal and hard copy unit. A recently announced interface card (Optional Data Communications Interface) now allows you to direct this output to the TEKTRONIX 4662 plotter, as well. The interface card fits inside the pedestal of a TEKTRONIX 4010-Series terminal. No special driver software is needed.

The 4662 Interactive Digital Plotter (Fig. 3-6) can significantly enhance the capability of a signal processing system. For example, multi-waveform graphs can be color coded by simply changing the plotter pen for each new array plotted. Thus a plot simultaneously displaying voltage, current, and power could be coded with green, blue, and red ink, or any of the nine colors of pens available for the 4662.

Making overhead transparencies with the 4662 is also a snap. Insert the transparency just as you would a sheet of paper, and then put the plotter through its paces. A transparency showing actual computer output can be a great aid in presenting research seminars or class lectures.

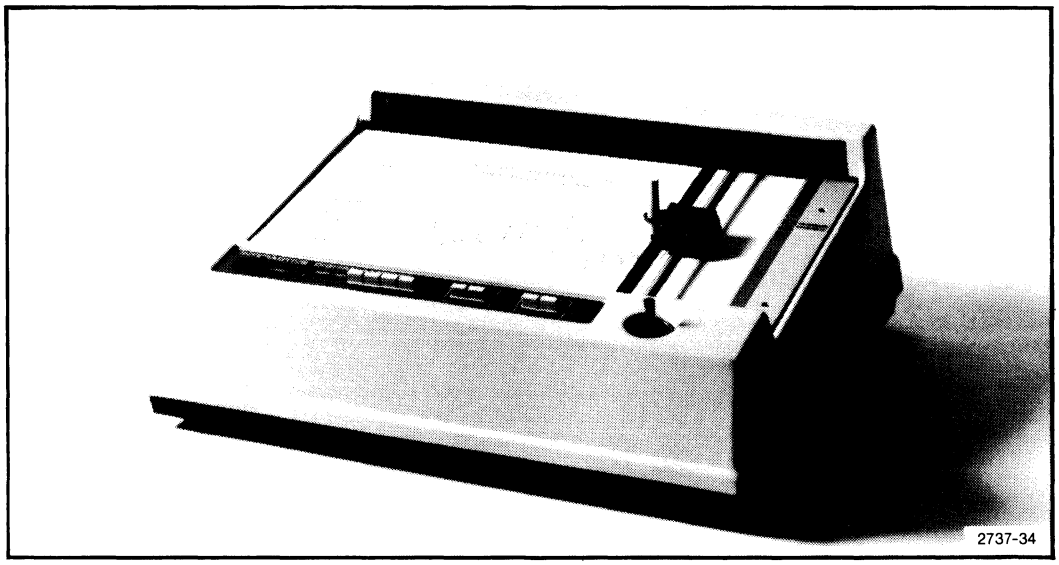


Fig. 3-6. The TEKTRONIX 4662 Interactive Digital Plotter.

Since the 4662 is an intelligent plotter (it incorporates a microprocessor) page scaling is easily accomplished with the plotter joystick and margin-set controls. This allows you to change the aspect ratio of a graph or to scale the plots to a different page size without ever touching the terminal. Alphanumeric characters are also scaled in the process. This same feature allows you to create inverted or mirror-image plots. This can be an advantage when plotting overhead transparencies, as explained later.

Getting to the Plot

To use the 4662 with a Tektronix signal processing system, the following installation and setup steps must be performed. (The actual installation should be performed only by qualified service personnel.) In the following steps, it is assumed that the terminal is cabled to the minicomputer and that the TTY port interface card (inside the terminal) is set for normal operation.

1) Set all straps on the Optional Data Communications Interface as per factory setting. (See the Interface instruction manual for the proper settings.)

2) Unplug the terminal and install the interface card inside the pedestal of the 4010-Series terminal. Instructions for installing the interface card are also found in the interface card manual.

3) Set the external switch panel of the interface card as follows:

a) TRANSMIT BAUD RATE to 1200
b) RECEIVE BAUD RATE to 1200
c) SUPR/NORM/FULL switch to FULL (Duplex)
d) INT/OFF/AUX switch to AUX

4) Connect the RS232 cable from the Optional Data Communications Interface to J104 on the back of 4662.

5) Set the four hexadecimal switches on the back panel of the 4662 to:

A:6,B:2,C:2,D:3

Switch D determines the baud rate, which in this case, has been set to 1200. This corresponds to the baud rate set in steps 3a and 3b above.

For some applications it may be necessary to set the baud rate slower than 1200 to prevent the plotter's internal buffer memory from overflowing. Overflow occurs when the computer is supplying data faster than the plotter can process it. Buffer overflow results in a loss of data and is signaled by continuous ringing of the plotter bell.

Overflow does not normally occur unless plotting a lot of alphanumerics preceding long data arrays, but if it does, the situation can usually be remedied by reducing the baud rate to 300 or 600. To set the baud rate to 300, set hexadecimal switch D (on the 4662 back panel) to position 1 and set the baud rate switches (on the optional Data Communications Interface) to 300 for both transmit and receive. A baud rate of 600 is selected by setting hexadecimal switch D to position 2 and the Interface baud rate switches to 600.

Once the 4662 has been correctly installed, you can power up the terminal, plotter, and computer system. At this time you may want to verify that the 4662 is working properly by exercising the plotter's self-diagnostic routine. This can be done as follows:

1) Latch the plotter LOAD button in the down position. This removes the static charge from the platen, allowing a new sheet of paper to be installed. It also moves the pen carriage to the upper right corner of the platen.

2) Install a clean sheet of paper at the desired position on the platen; then press and release the LOAD button, thereby reactivating the platen hold-down charge.

3) Install a pen of the desired color by inserting the tabs on the pen into the grooves in the pen holder and twisting the pen clockwise.

4) Set the page boundaries of the plotting area with the joystick and SET buttons. Set the lower left boundary by moving the pen to the desired position with the joystick. Then press and hold the SET: LOWER LEFT button until the plotter bell rings. Similarly, to set the upper-right boundary, move the pen to the appropriate position. Then press and hold the SET: UPPER RIGHT button until the plotter bell rings.

5) Call up the plotter self-diagnostics feature (stored in a Read-Only-Memory) by holding the CALL button down until the plotter bell rings twice. The plotter will immediately begin plotting a predetermined set of patterns; a rectangle in the upper-right corner, a sunburst in the lower-left corner and a message in the middle. When the plotter is finished you can remove the sheet by latching the LOAD button down. To install a new sheet of paper or a transparency, repeat steps 1 through 4.

Using the 4662 Under Computer Control

To use the 4662 under computer control, latch the plotter LOCAL button in the down position. All terminal printout will then be produced on the plotter as well as the terminal screen. To disable the plotter, simply unlatch the LOCAL button.

The following program demonstrates the use of the 4662 by plotting three waveforms on the same graticule. Lines 10 through 90 dimension three 512-element waveforms labeled A, B, and C, and fill them with simulated data representing current, voltage, and power respectively. Lines 100 through 240 are actually responsible for plotting the data. Since all three waveforms are plotted with respect to the same graticule, the horizontal and vertical scale factors apply to all three waveforms. (Lines 110 and 120 cause a graticule to be displayed which applies to all

three waveforms.) The DISPLAY commands at lines 150, 180, and 210 cause each waveform array to be plotted individually. a WAIT command is included after the graticule is generated and before each array is plotted so that a different color pen can be mounted. To exit the WAIT and plot the next waveform, simply press the return key on the terminal. After all waveforms have been plotted, the program ENDs at line 240. (The WAIT command at line 240 prevents READY from appearing on the plot.)

Here is a listing of the program, as plotted on the 4662:

```
LIST
10 WAVEFORM A IS WA(511),SA,HA$,VA$
20 WAVEFORM B IS WB(511),SB,HB$,VB$
30 WAVEFORM C IS WC(511),SC,HC$,VC$
40 SA=.1\SB=.1\SC=.1
50 HA$="MILLISECONDS"\HB$=HA$\HC$=HA$
60 WA=.6283\INT WA,WA
70 WA=SIN(WA*.05)\WA=WA*5
80 DIFF WA,WB\WB=WB*5
90 WC=ABS(WA*WB)
100 PAGE
110 SETGR NOPL
120 GRAPH A,B,C
130 SMOVE 300,750
140 PRINT "INSTANTANEOUS POWER PLOT"
150 WAIT\DISPLAY A
160 SMOVE 925,600
170 PRINT "VOLTS"
180 WAIT\DISPLAY B
190 SMOVE 925,500
200 PRINT "AMPS"
210 WAIT\DISPLAY C
220 SMOVE 925,400
230 PRINT "WATTS"
240 WAIT\END

READY
*
```

Before entering this program on the terminal, make sure that the plotter LOCAL button is in the up position. This ensures that the program lines will be plotted only on the terminal and not on the plotter. After the program has been entered, type RUN on the terminal, press the plotter LOCAL button, and press the terminal RETURN key. The output from the program is now directed to the 4662 as well as to the terminal screen.

If you want to plot the entire graph in one color, simply press the RETURN key to exit each WAIT command. The plotter output will then appear as in Fig. 3-7. To make the same graph in four colors, change plotter pens and press RETURN after each WAIT occurs

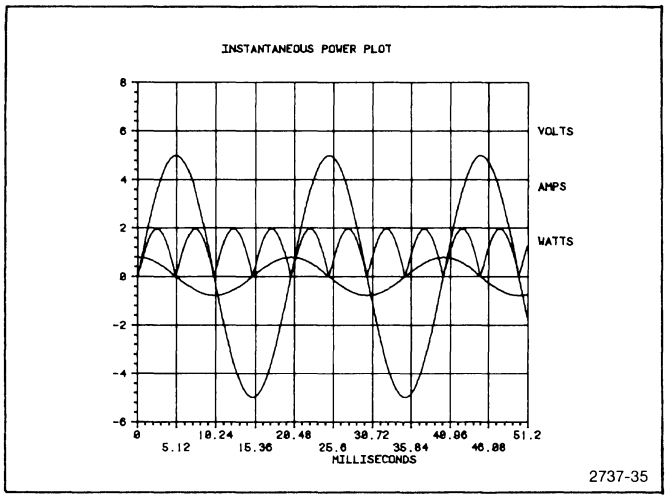


Fig. 3-7. Example of a multiple-waveform graph plotted on the 4662.

Variations on a Theme

We stated earlier that mirror images can be plotted with the 4662. This is a particular advantage when making overhead transparencies, since the transparency can be placed ink-down on the projection plate and the image projected normally. Thus you can use a marking pen to highlight or note items on top of the transparency during the presentation. These notes can be erased after each session without disturbing the graph or chart plotted on the other side.

A mirror image can be easily plotted using the 4662 front panel controls. Simply move the pen (via the joystick) to the lower right corner of the desired page area and press the SET: LOWER LEFT button until the plotter bell rings. Then move the pen to the upper left corner of the desired page area and press the SET: UPPER RIGHT button until the bell rings. All incoming data, including alphanumerics, is subsequently plotted in mirror-image format.

The preceding example illustrates some of the posibilities of using a digital plotter with a signal processing system. We have explored only a small part of the 4662's capabilities. Other possibilities include overlaying plots on Smith charts, log-log paper or semi-log paper --

tasks that would be difficult to perform by conventional hardcopy techniques.

More sophisticated plots can be generated under program control using a set of plotter control commands. These commands consist of ASCII character strings that allow the computer to control data transfers to and from the plotter, set the character size and font, print the characters at any specified angle, read or write plotter status, and control several other plotter functions. The commands can be addressed to any one of up to four plotters tied to the same computer.

If, for example, you wished to emphasize the heading INSTANTANEOUS POWER PLOT (Fig. 3-7) by making it larger, the program might include these statements:

```
135 PRINT "^KAI112,176"
14Ø PRINT "INSTANTANEOUS POWER PLOT"
145 PRINT "^KAV"
```

When the computer executes line 135, it sends the ASCII character string enclosed in the quotes to the plotter via the terminal. This string sets the character space (width) to 112 units (twice the default width) and the line space (height) to 176 units (twice the default height). Line 145 resets the alpha size to the default size.

Using similar command strings you can print special characters in the heading or rotate the heading to any specified angle.

by Cliff Morgan, HANDSHAKE Staff

III. Computer Games

Techniques Illustrated:

Interactive graphics; processing keyboard interrupts.

Optional TEK SPS BASIC Packages Required:

Graphics, High-Level Support.

Abstract:

The following HANDSHAKE article (from Vol. 2 No. 4) describes a computer game that demonstrates the concepts of interactive graphics by using keyboard interrupts to direct the course of a "guided missile." This "missile" is used to destroy the "alien spacecraft," which randomly fires "death rays" at your castle.

Article:

No "Aliens" Allowed

Games are an easy and fun way to learn about software tools. This issue's graphics program is a game that demonstrates some very powerful TEK SPS BASIC commands, and throws in some excitement too. The game is called ALIEN and, like many computer games, the object is to eliminate the "bad guy" before you are eliminated.

The ALIEN spacecraft swoops down on your castle while randomly firing "death rays." Your only defense is to launch a guided missile at the spacecraft before it scores a direct hit. To make the game more interesting, the ALIEN apparently dodges your missiles during descent.

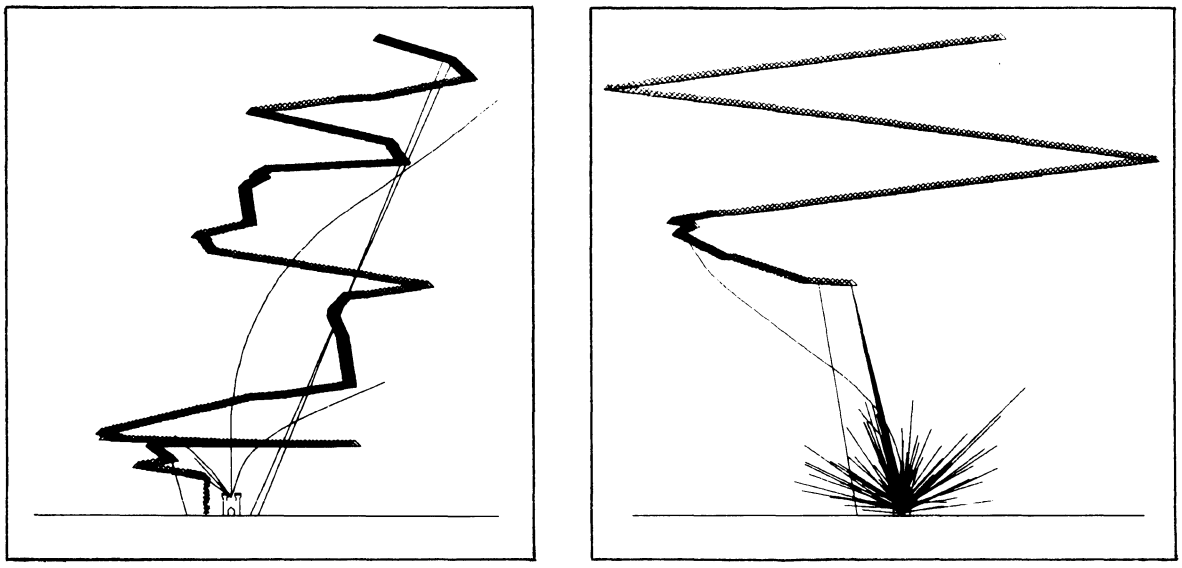
The game operates in real time. That is, the display is constantly being updated. There are no pauses for turns or input from the keyboard. Instead, the IV (Interrupt Vector) Driver in TEK SPS BASIC is used to capture interrupts. When a keyboard character is pressed, control automatically goes to line 2120. There the GETLOC command is used to get the seven bits of ASCII data at the terminal input buffer register. This

character is compared against the three missile control keys and Control P (used to terminate the game).

The keys used in the game are the digits 0 and 1, and the space bar. The space bar fires a missile if one is not already in flight. The other two keys control the direction of the flight; digit 1 turns the missile to the left, and digit 0 turns it to the right. Of course you must overcome forward momentum, so your control of the missile is not immediate or perfectly accurate. This makes the game more difficult, but also more rewarding when you finally destroy the enemy.

The game is rich in graphic detail (see Fig. 3-8). When your missile connects with the ALIEN, the spacecraft burns like a Fourth-of-July sparkler. However, if the enemy gets your castle first (perish the thought), the castle undergoes multiple explosions.

Of course the IV Driver was not designed just for games. With it and the High-Level Support Commands, TEK SPS BASIC can directly communicate with any peripheral or instrument that can be interfaced to your minicomputer. Interrupts can be handled at the BASIC command level, saving you the trouble of writing your own assembly-language drivers. The High-Level Support Package also allows you to read, write, and modify any location in your minicomputer, including status and data registers.



2737-39

Fig. 3-8. Example output of the "ALIEN" program.

```
1000 LOAD "IV"
1010 PAGE
1020 PRINT '"ALIEN.BAS" - A TEK SPS BASIC GAME'
1030 PRINT "ORIGINAL VERSION BY CLIFF MOULTON"
1040 PRINT "SPS COMPATIBILITY BY DAVID STUBBS"
1050 PRINT
1060 PRINT ,"CHARACTER","ACTION"
1070 PRINT ,"-----------------------------------"
1080 PRINT ,"SPACE BAR","- FIRES MISSILE"
1090 PRINT ,"1","- TURNS MISSILE TO LEFT"
1100 PRINT ,"0","- TURNS MISSILE TO RIGHT"
1110 PRINT ,"CTRL P","- STOPS THE GAME"
1120 PRINT
1130 PRINT "STRIKE ANY CHARACTER TO BEGIN GAME ";
1140 WAIT\DETACH #1
1150 KB=65394
1160 ATTACH #1 AS IV48:
1170 WHEN #1 GOSUB 2120
1180 GOTO 1200
1190 WAIT 1000
1200 PAGE
1210 REM -----SET UP VARIABLES-----
1220 MS=0\NS=0\DX=0\NX=511\NY=30
1230 REM -----DRAW BASE-----
1240 SMOVE 0,0
1250 SDRAW 1023,0
1260 SMOVE 498,0
1270 RSDRAW 0,25
1280 GOSUB 1340
1290 RSDRAW 15,0\GOSUB 1340
1300 RSDRAW 0,-25
1310 SMOVE 506,0
1320 RSDRAW 0,10,5,5,5,-5,0,-10
1330 GOTO 1390
1340 REM -----DRAW TOWER-----
1350 RSDRAW 0,5,-2,2,0,5,2,0,0,-2,2,0,0,2,2,0
1360 RSDRAW 0,-2,2,0,0,2,2,0,0,-5,-2,-2,0,-5
1370 RETURN
1380 REM -----START INTRUDER-----
1390 IX=RND(0)*720+50
1400 IY=780
1410 AX=RND(0)*4
1420 IF RND(0)<.5 THEN AX=-AX
1430 AY=-RND(0)*2
1440 REM -----LOOP-----
1450 SMOVE IX-10,IY
1460 RSDRAW 20,0,-10,10,-10,-10
1470 IX=IX+AX\IY=IY+AY
1480 REM -----ESTABLISH DESCENT TRIANGLE-----
1490 QL=(780-IY)*.45
1500 IX=IX+AX
1510 IF IX>QL+12 THEN IF IX<1011-QL THEN GOTO 1550
1520 IF IX<QL+12 THEN IX=QL+12
1530 IF IX>1011-QL THEN IX=1011-QL
1540 AX=-AX
1550 REM -----IF MISSILE, THEN MOVE IT-----
1560 IF MS=0 THEN GOTO 1780
1570 SMOVE NX,NY
1580 NY=NY+5+IY*5E-03\NX=NX+DX
1590 IF NX>0 THEN IF NX<1020 THEN GOTO 1610
1600 GOTO 1630
1610 SDRAW NX,NY
1620 IF NY<IY+10 THEN GOTO 1640
1630 MS=0\NX=511\NY=30
1640 REM -----SEE IF MISSILE HIT ALIEN-----
1650 IF NY<IY THEN GOTO 1800
1660 IF ABS(NX-IX)>10 THEN GOTO 1800
1670 REM -----HIT ALIEN, DO DISPLAY-----
1680 PRINT CHR(7)
1690 I=5
1700 I=I+.3
1710 IY=IY-.5
1720 SMOVE IX,IY
1730 RSDRAW I-RND(0)*(I+I),I-RND(0)*(1+I)
1740 IF IY<1 THEN GOTO 1190
1750 IF I>100 THEN GOTO 1190
1760 IF IX-I>5 THEN IF IX+I<1015 THEN GOTO 1710
1770 GOTO 1190
1780 REM -----POTENTIAL ALIEN COURSE CHANGE----
1790 IF NS=0 THEN GOTO 1810
1800 IF ITP(RND(0)*25)=1 THEN GOTO 1410
1810 REM -----POTENTIAL ALIEN ATTACK-----
1820 IF IY>40 THEN GOTO 1850
1830 TX=512
1840 GOTO 1880
1850 IF MS=0 THEN GOTO 1450
1860 IF ITP(RND(0)*30)<>1 THEN GOTO 1450
1870 TX=RND(0)*300+362
1880 SMOVE IX,IY
1890 IF ABS(512-TX)>15 THEN GOTO 1920
1900 SDRAW TX,25
1910 GOTO 1950
1920 SDRAW TX,1
1930 GOTO 1450
1940 REM -----SUCCESSFUL ALIEN ATTACK-----
1950 PRINT CHR(7);
1960 II=75
1970 FOR I=25 TO 1 STEP -.3
1980 II=II+2
1990 SMOVE IX,IY
2000 RSDRAW 10,0,-10,10,-10,-10,10,0
2010 SDRAW 507+RND(0)*10,I
2020 RSDRAW II-RND(0)*(II+II),RND(0)*II
2030 WAIT 35
2040 NEXT I
2050 WAIT 40
2060 SMOVE IX,IY
2070 FOR I=10 TO 0 STEP -.25
2080 RSDRAW I,0,-I,I,-I,-I,I,0
2090 NEXT I
2100 REM -----END LOOP-----
2110 GOTO 1190
2120 REM -----GET A KEYBOARD CHARACTER-----
2130 GETLOC KB,A,0,6
2140 IF A<>48 THEN GOTO 2170
2150 DX=DX+1+100/IY
2160 RETURN
2170 IF A<>49 THEN GOTO 2200
2180 DX=DX-1-100/IY
2190 RETURN
2200 IF MS=0 THEN DX=ITP(SGN(IX-512)*1/IY*500)
2210 MS=1
2220 IF MS=1 THEN IF NS=0 THEN NS=1
2230 IF A<>16 THEN RETURN
2240 DETACH #1\END
```

2737-36

Fig. 3-9. Listing of the "ALIEN" program.

EDITOR'S NOTE

Tektronix, Inc. assumes no responsibility, expressed or implied, for expenses incurred as a result of employees playing games on company time.

by Garey Fouts, HANDSHAKE Staff. The ALIEN game was originally developed by Cliff Moulton of Beaverton, Oregon, and was adapted to TEK SPS BASIC by David Stubbs of Tektronix.

III. Computer Games (cont.)

Techniques Illustrated:

Use of random vectors to create unusual patterns.

Optional TEK SPS BASIC Packages Required:

Graphics, Signal Processing.

Abstract:

The following was adapted from a HANDSHAKE article (from Vol. 2 No. 1) and explains a demonstration-game that generates snowflake-type patterns on the terminal screen.

Article:

Our Cover Story -- Snowflake Generator

A "snowflake generator" program was used to create most of the designs for the cover of this HANDSHAKE issue. The program is referred to as a "snowflake generator" because each pattern is unique and usually resembles a snowflake, as shown in Fig. 3-10.

Each iteration of the program produces a pattern that is a plot of an array generated from two random numbers. Here is the program:

```
1Ø VIEWPORT 15Ø,85Ø,Ø,7ØØ
2Ø WINDOW -2,2,-2,2
3Ø DIM A(511),B(256),C(256)
4Ø A=Ø
5Ø A(ITP(RND(Ø)*511)+1)=1
6Ø A(ITP(RND(Ø)*511)+1)=1
7Ø RFFT A,B,C,T
8Ø PAGE
9Ø MOVE B(Ø),C(Ø)
1ØØ FOR I = 1 TO 256
```

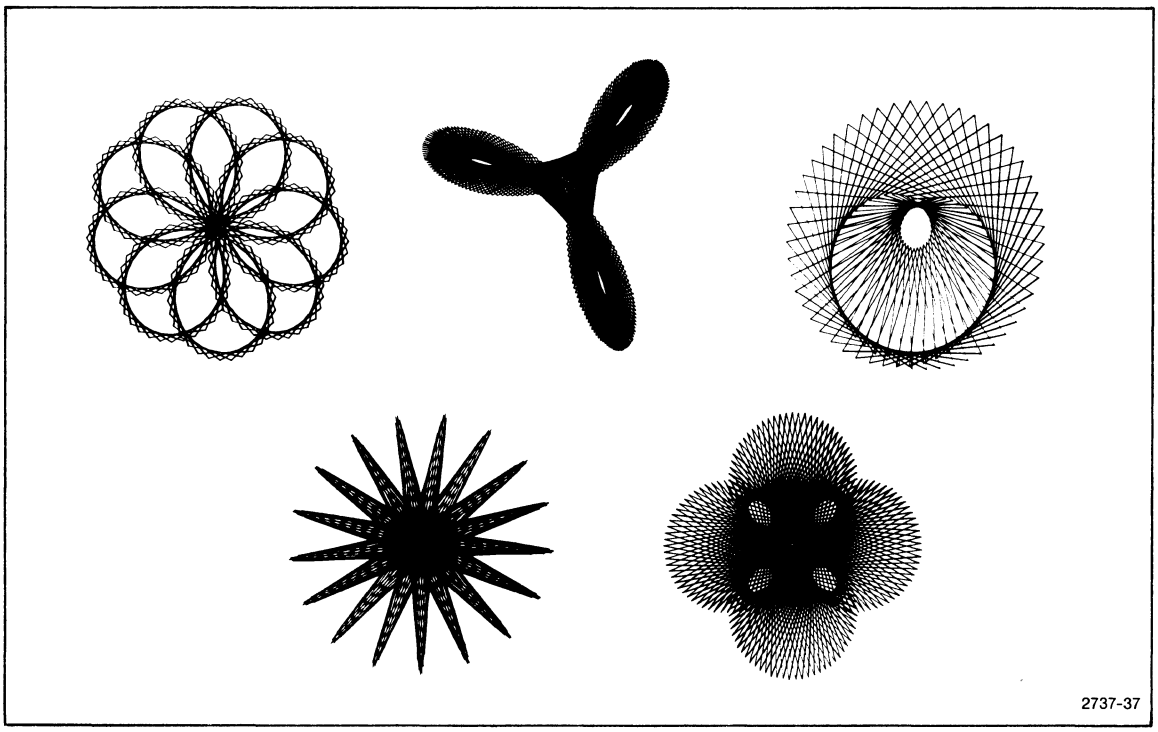


Fig. 3-10. Typical patterns produced by the "snowflake" program.

```
11Ø DRAW B(I),C(I)
12Ø NEXT I
13Ø GOTO 4Ø
```

Lines 50 and 60 generate the random numbers that are used as subscripts for two elements in array A. A value of 1 is placed in these two random elements, but all other elements of array A are left at zero. The two non-zero elements look like two spikes in the time domain to the Real Fast Fourier Transform -- computed at line 70. After line 70 has executed, arrays B and C will contain the real and imaginary components of the frequency spectrum, respectively. Arrays B and C are then used, element-by-element, as the X and Y values to be plotted. The actual plotting occurs in lines 100 through 120. After a particular pattern has been generated, line 130 causes a branch to line 40 where the entire process is repeated and a new pattern is generated.

The program continuously generates new patterns on the terminal screen until Control-P is typed on the keyboard.

IV. Creating Graphs and Charts

Techniques Illustrated:

Construction of bar graphs, shading techniques.

Optional TEK SPS BASIC Packages Required:

Graphics.

Abstract:

The following HANDSHAKE article (from Vol. 4 No. 1) describes a program for creating bar graphs with or without shading of the bars.

Article:

Build Bars - To See Better

It can be frustrating to gather information, arrange it in a meaningful manner for one of those inevitable reports that attempts to communicate what is being done and how well it's being done, and have the mechanics of the presentation -- the graphics -- hinder instead of enhance the message. Bars -- bar graphs, that is -- are tools you can use to add clarity to your reports.

Graphs, generally, are effective tools for communications. That's why you see them used so often. But, for certain information, some graphs work better than others. The line graph you get when you use the GRAPH command may not always be the best presentation of your data. Sometimes, a bar graph works better.

The program listing in Fig. 3-11 contains some easy-to-use routines that will give you the bar graphs shown in Figs. 3-13, 3-14, or 3-15. You pick the graph style that best suits your information, and include those routines that will do the job for you.

The graphs included as examples here use "month" as the horizontal units. We must first, then, dimension the data array to the number of periods (months) we want to show. Since array elements are numbered from zero, dimensioning the array to 11 gives us 12 periods, or 12 months on the graph. If you want a 24-month scale graphed, as in Fig. 3-15, dimension A to 23. Line 60 is the only program line you need alter to change the horizontal periods graphed.

In the SETGR statement (line 280) we make the number of major horizontal tic marks equal to the size of A. We thereby automatically scale the horizontal margin of the graticule to match the periods you chose in line 60. Also, the keyword NOPLOT is added to the SETGR command so the array data will not be displayed by the GRAPH command that follows. Omitting NOPLOT would result in the normal line graph of the data shown in Fig. 3-12.

The three routines included in Fig. 3-11 can be retained, deleted, or altered at your option.

The first routine draws the outline bar graph (Figs. 3-13 and 3-14). Line 340 moves the pointer to the left-hand corner of the graticule where we find the zero X value and the first Y value of the array data. MOVE was used instead of SMOVE because MOVE employs the user units per the current window; SMOVE uses screen coordinates. Line 360 draws the vertical lines between the months, connecting the varying Y values (Fig. 3-13). The vertical lines that drop to the bottom of the graph (Y=0) are created by the ",X+1,0" at the end of line 370 (Fig. 3-14). The first pair of coordinates in line 370 draw the horizontal lines (top of the bars). The bar graph in Fig. 3-15 requires routines 2 and 3 to draw the hatching within the bars.

The second routine draws the horizontal hatching, beginning with the bar for the first month and for every other month after that (line 440 sets up this loop). Line 420 determines the incremental spacing of the hatching. If you want the horizontal hatching lines closer together, change line 420 to FOR Y=1 TO 100.

The third routine draws the vertical hatching, beginning with the second month and for alternate months after that (line 520 sets up this loop). The horizontal space between the vertical hatching is determined by line 530. Again, if you want closer spacing, you could write this line as FOR I=.1 TO .9 STEP .1. We do not step through this loop beginning at zero or ending at 1 to avoid writing over a line that is already there.

```
10 REM
20 REM CREATE A 12-MONTH ARRAY OF RANDOM DATA
30 REM FOR THIS GRAPHICS DEMONSTRATION
40 REM
50 DELETE A
60 DIM A(11)
70 A=RND(A)*100
80 REM
90 REM USE WAVEFORM COMMAND TO HOLD ARRAY
100 REM LABELING INFORMATION
110 REM
120 REM     A - ARRAY WITH HORIZONTAL DATA TO PLOT
130 REM   HA$ - HORIZONTAL UNITS (MONTHS)
140 REM   VA$ - VERTICAL UNITS (PERCENT)
150 REM
160 WAVEFORM WA IS A,SA,HA$,VA$
170 SA=1
180 HA$="MONTH"
190 VA$="PERCENT"
200 GOSUB 250
210 END
220 REM
230 REM ERASE 4010/4014 SCREEN AND DRAW GRATICULE
240 REM
250 PAGE
260 REM
270 WINDOW 0,SIZ(A),0,100
280 SETGR GRAT 2,2.NOPLOT,WIND,TICS SIZ(A),5,2,2
290 GRAPH WA
300 REM
310 REM ROUTINE TO MOVE TO FIRST ARRAY LEVEL
320 REM AND DRAW OUTLINE BAR GRAPH
330 REM
340 MOVE 0,A(0)
350 FOR X=0 TO SIZ(A)-1
360 DRAW X,A(X)
370 DRAW X+1,A(X),X+1,0
380 NEXT X
390 REM
400 REM ROUTINE TO DRAW HORIZONTAL HATCHING
410 REM
420 FOR Y=2 TO 100 STEP 2
430 MOVE 0,Y
440 FOR X=0 TO SIZ(A)-1 STEP 2
450 IF A(X)>Y THEN DRAW X+1,Y
460 MOVE X+2,Y
470 NEXT X
480 NEXT Y
490 REM
500 REM ROUTINE TO DRAW VERTICAL HATCHING
510 REM
520 FOR X=1 TO SIZ(A) 1 STEP 2
530 FOR I=.2 TO .8 STEP .2
540 MOVE X+I,0
550 DRAW X+I,A(X)
560 NEXT I
570 NEXT X
580 RETURN
```

2737-38

Fig. 3-11. Bar graph program listing.

If, on the other hand, you would like the entire area within the bars hatched either horizontally or vertically instead of the alternate patterns in Fig. 3-15, this is easily done by first deleting the routine you do not want to use, then by changing the following program lines:

All vertical hatching: make line 520 FOR X=0 TO SIZ(A)-1

All horizontal hatching: make line 440 FOR X=0 TO SIZ(A)-1, and make line 460 MOVE X+1,Y

by Walt Robatzek, HANDSHAKE Staff
with significant contributions by David Stubbs, SPS Software Engineer
and by Joyce Ferriss, HANDSHAKE Staff

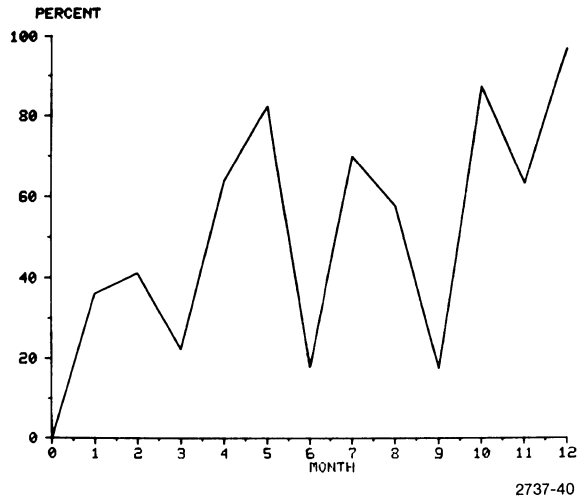


Fig. 3-12. Normal line graph.

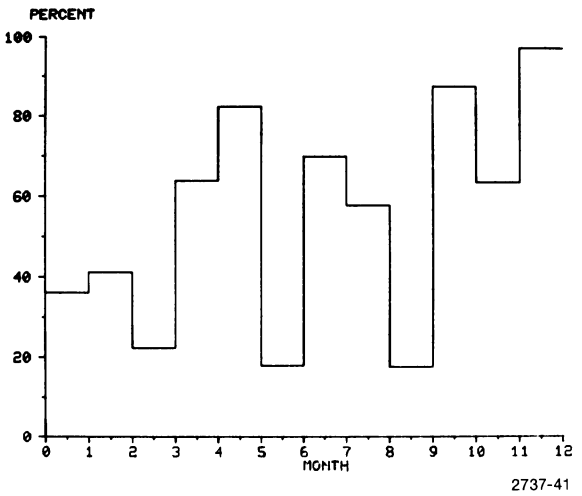


Fig. 3-13. Outline bar graph; one style.

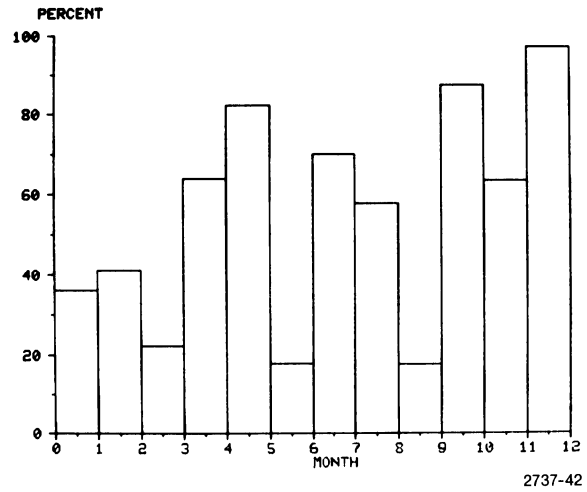


Fig. 3-14. Outline bar graph; another style.

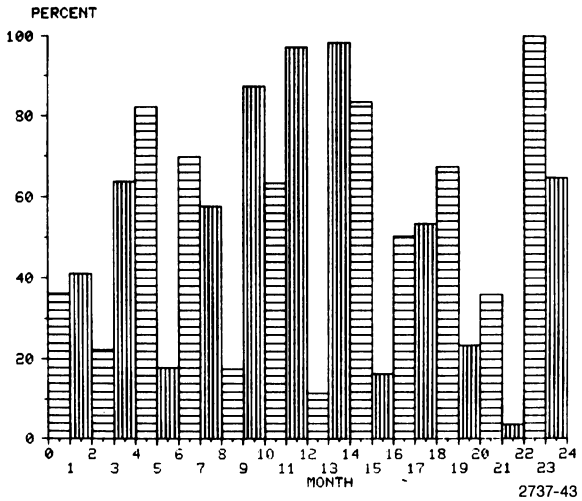


Fig. 3-15. Bar graph with contrasting hatching.

IV. Creating Graphs and Charts (cont.)

Techniques Illustrated:

Graphing multiple arrays on separate graticules.

Optional TEK SPS BASIC Packages Required:

Graphics, Signal Processing.

Abstract:

The following article provides a program for graphing four arrays or waveforms on the same page.

Article:

Graphing Four Arrays on Separate Graticules

The ability to place several graphs (be they waveforms, arrays, or charts) on a single sheet of paper can be of great importance when presenting summaries of acquired data. By means of the VIEWPORT command in TEK SPS BASIC, it is very simple to create such a multiple-array plot.

There is a trade-off, of course -- the loss of resolution. However, if the waveforms or arrays being plotted do not contain rapid variations, or if these variations are not critical in your documentation, the technique works quite well. The following program demonstrates the method:

```
1Ø DIM A(511),B(511),C(511),D(511)
2Ø A=.Ø6283
3Ø INT A,B
4Ø C=SIN(B)
5Ø D=COS(B)
6Ø VIEWPORT 1ØØ,4ØØ,5ØØ,7ØØ
7Ø SETGR VIEW
```

```
8Ø GRAPH A
9Ø VIEWPORT 6ØØ,9ØØ,5ØØ,7ØØ
1ØØ SETGR VIEW
11Ø GRAPH B
12Ø VIEWPORT 1ØØ,4ØØ,1ØØ,3ØØ
13Ø SETGR VIEW
14Ø GRAPH C
15Ø VIEWPORT 6ØØ,9ØØ,1ØØ,3ØØ
16Ø SETGR VIEW
17Ø GRAPH D
18Ø WAIT
```

Lines 10 through 50 are included only for demonstration purposes. They dimension the four arrays (A through D) and fill them with data representing a constant (A), a ramp (B), a sine wave (C), and a cosine wave (D).

Lines 60 through 180 perform the actual array plotting. At line 60 a viewport is defined which causes the next graph to appear in the upper-left quadrant of the terminal screen. Lines 70 and 80 then graph array A into the newly defined viewport. A similar process is repeated for arrays B, C, and D. At the conclusion of the program, array A will be shown in the upper-left quadrant, array B will appear in the upper-right quadrant, array C will be in the lower-left quadrant, and array D will be in the lower-right quadrant. (The WAIT at line 180 simply suppresses the printing of READY until the terminal screen is erased.) Figure 3-16 shows the results of running the program.

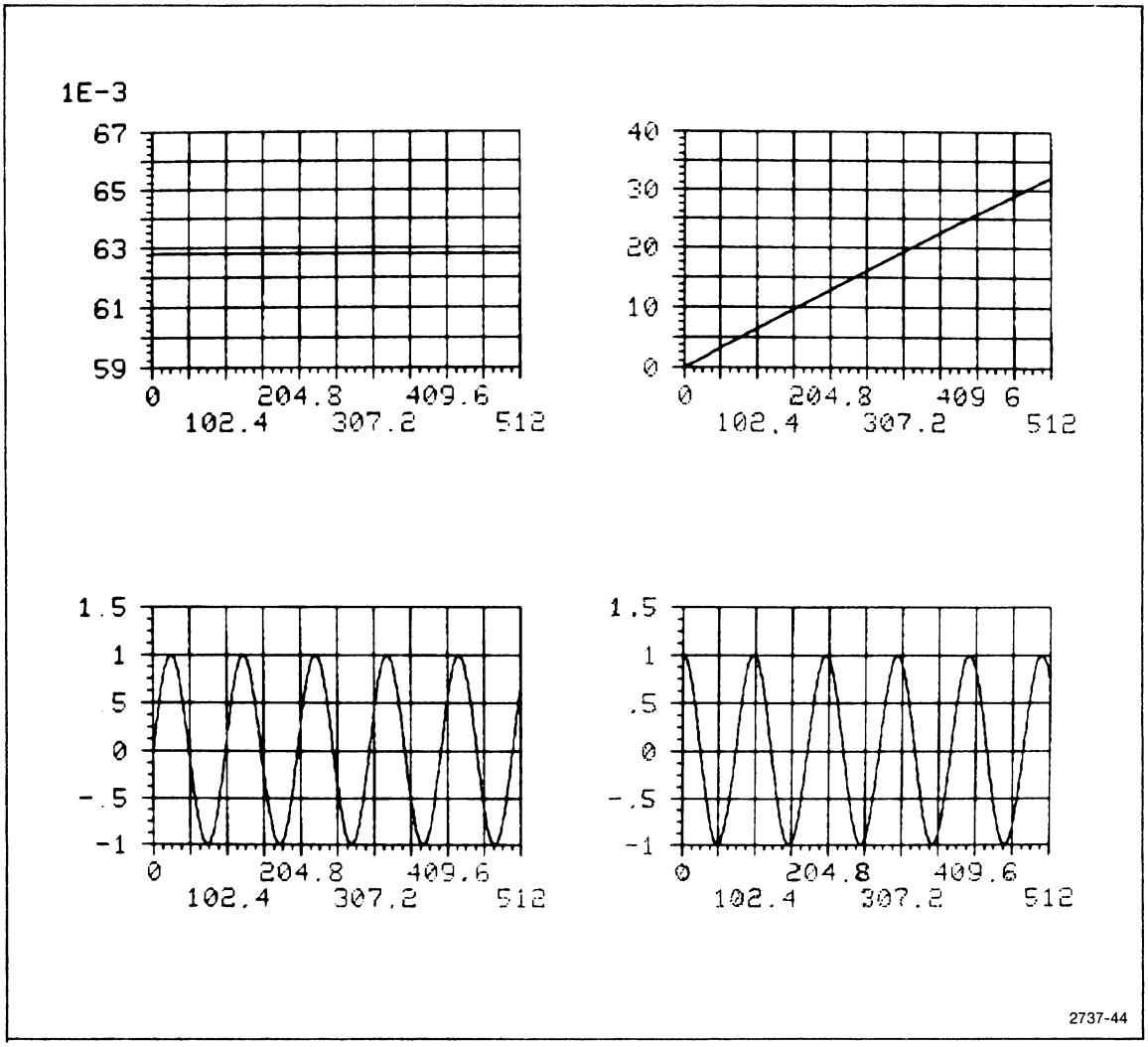


Fig. 3-16. Example of a four-array plot.

IV. Creating Graphs and Charts (cont.)

Techniques Illustrated:

Gray-scale plotting.

Optional TEK SPS BASIC Packages Required:

Graphics.

Abstract:

The following HANDSHAKE article (from Vol. 3 No. 3) demonstrates a type of three-dimensional graphics plot in which intensity (the darkness of a pixel) serves as the third dimension. Several demonstration programs are included.

Article:

Gray Scale Adds a New Dimension

Have you ever wanted to give new depth to your graphics or add another dimension to a plot? In this article we offer you a tool. It is an intensity or gray-scale plotting routine (Fig. 3-17).

The routine graphs a number between zero and nine inclusive as an eight-by-eight pixel (in graphic-screen units). The larger the number, the greater the intensity of the pixel. A zero causes no plotting; a nine produces a nearly filled square. The shading is done by drawing lines inside the eight-by-eight area rather than by making dots. This enables the routine to execute faster and allows greater resolution than with the larger nine-by-nine pixel of a dot gray scale. The shape, length, and number of lines for each of the shading scales, zero to nine, were developed empirically.

The results of using the routine with two programs that calculate their own data are shown here. The first program produces a simple bar graph illustrating the possible shading variations (Fig. 3-18). The second program demonstrates how shading can imply a third dimension (Fig.

```
1000 REM ROUTINE TO GRAPH GRAY SCALE VALUES 0-9 (BY R. O. DECKER)
1010 GOSUB N+1 OF 1030,1040,1060,1080,1100,1120,1140,1160 ,1180,1200
1020 RETURN
1030 RETURN
1040 RSDRAW 0,0
1050 RETURN
1060 RSDRAW 1,1
1070 RETURN
1080 RSDRAW 2,2
1090 RETURN
1100 RSDRAW 1,1,1,0,-1,0,-1,1
1110 RETURN
1120 RSDRAW 2,2,-1,-1,-1,1,2,-2
1130 RETURN
1140 RSDRAW 3,0,0,2,-3,0,0,-1
1150 RETURN
1160 RSDRAW 3,0,0,3,-3,0,0,-2
1170 RETURN
1180 RSDRAW 4,0,0,4,-4,0,0,-3,3,0,0,1,-1,0
1190 RETURN
1200 RSDRAW 5,0,0,5,-5,0,0,-4,4,0,0,3,-2,0,0,-1,1,0
1210 RETURN
1220 REM END OF GRAPHING ROUTINE
```

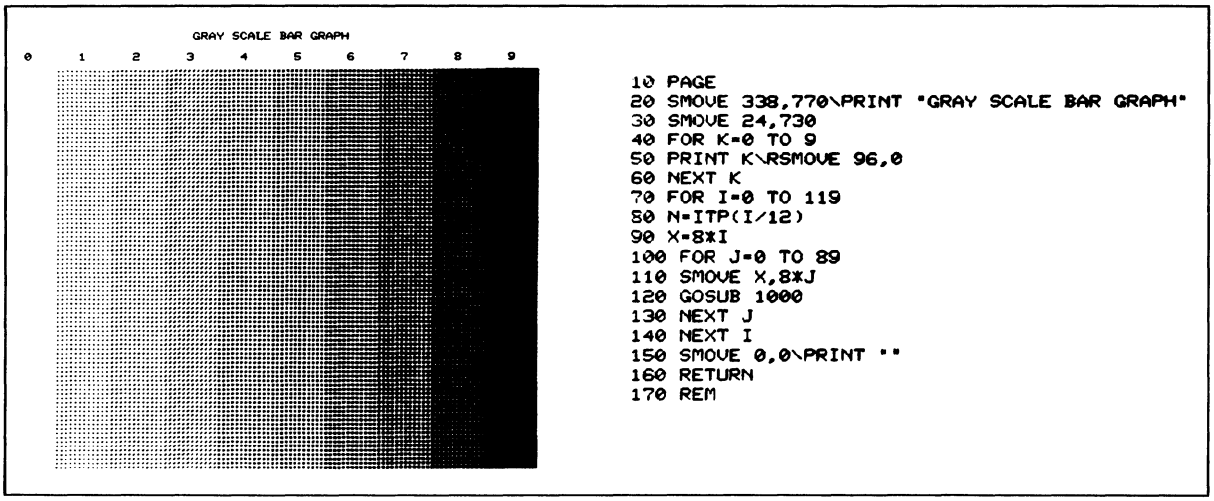
2737-45

Fig. 3-17. This subroutine draws the 8 X 8 gray-scale pixel when called by a main program supplying a number between 0 and 9.

3-19). In this case, intensity is a function of the distance from the center of the circle whose origin is the center of the graphics terminal screen (line 70). Notice that, in the second routine, the gray-scale value of one goes through a randomizing process (line 100) so that it is not always plotted. This makes a less abrupt change from zero to one, but at a loss of resolution.

We have included the partial program listed in Fig. 3-20 to suggest a practical application of the gray-scale routine. This program maps a set of 97 or fewer data arrays of equal dimension into rows of gray-scale pixels. Two application-dependent subroutines are left for the user to write. One is the routine called for in line 270 to return the size of the data arrays plus the minimum and maximum expected values. This could be a simple input request or an actual scan of the data. Similarly, the routine to acquire each array of data (line 440) is omitted. Most likely, the array will be read in from a storage peripheral. The minimum and maximum are used to calculate the correction factor (line 300) for the mapping routine.

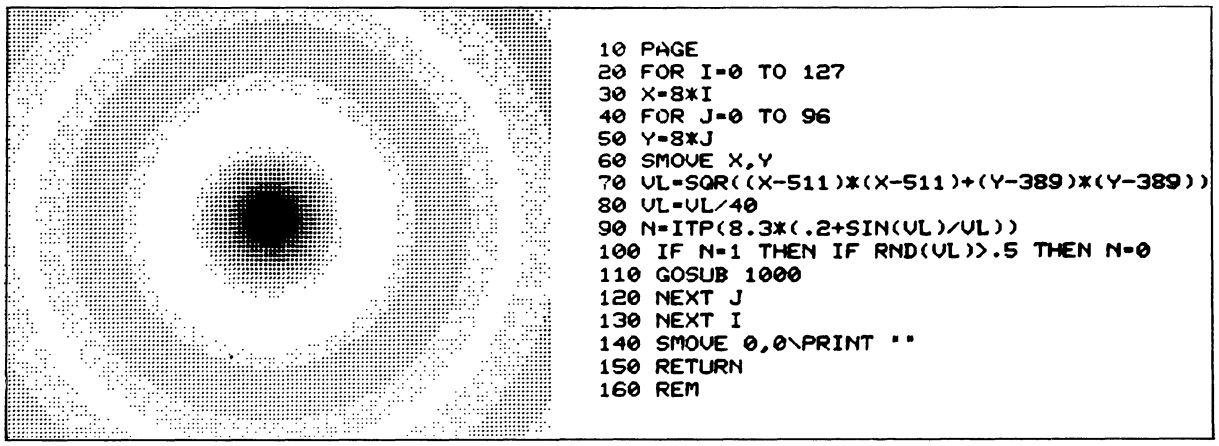
The arrays are divided into no more than 128 zones for the 128 possible horizontal pixels (lines 340 to 370). This is done by making the number of elements per zone equal the step size of the inner loop in line 470. If this step size isn't an even divisor of the array size, a flag is set to insure the graphing of the last partial zone (line 390).




```
10 PAGE
20 SMOVE 338,770\PRINT "GRAY SCALE BAR GRAPH"
30 SMOVE 24,730
40 FOR K=0 TO 9
50 PRINT K\RSMOVE 96,0
60 NEXT K
70 FOR I=0 TO 119
80 N=ITP(I/12)
90 X=8*I
100 FOR J=0 TO 89
110 SMOVE X,8*J
120 GOSUB 1000
130 NEXT J
140 NEXT I
150 SMOVE 0,0\PRINT ""
160 RETURN
170 REM
```

2737-46

Fig. 3-18. This program demonstrates the variations of intensity the gray-scale routine (called for in program line 120) is capable of drawing. When the value passed is zero, the pixel remains blank. When a one is passed, just a dot is drawn (the minimum line length). Finally, a nine results in the maximum number of lines being drawn. The bar graph illustrates the ten graduations possible with the gray-scale routine.



```
10 PAGE
20 FOR I=0 TO 127
30 X=8*I
40 FOR J=0 TO 96
50 Y=8*J
60 SMOVE X,Y
70 VL=SQR((X-511)*(X-511)+(Y-389)*(Y-389))
80 VL=VL/40
90 N=ITP(8.3*(.2+SIN(VL)/VL))
100 IF N=1 THEN IF RND(VL)>.5 THEN N=0
110 GOSUB 1000
120 NEXT J
130 NEXT I
140 SMOVE 0,0\PRINT ""
150 RETURN
160 REM
```

2737-47

Fig. 3-19. The purpose of this program is to generate data to demonstrate a use of the gray-scale routine (called for in program line 110). The intensity of the shading in the illustration is a function of the distance from the center. This rotated sin x/x plot is similar to the pattern produced by a drop of water falling into a pond: an intense splash in the center; then, waves of diminish -ing strength radiating outward.

```
100 REM GRAY SCALE DISPLAY PROGRAM
110 REM LIST OF VARIABLES USED
120 REM
130 REM A      THE ARRAY OF ACQUIRED DATA
140 REM SI     SIZE OF THE ARRAY
150 REM MI     MINIMUM VALUE EXPECTED IN ARRAY ELEMENTS
160 REM MA     MAXIMUM VALUE EXPECTED IN ARRAY ELEMENTS
170 REM C      CORRECTION FACTOR
180 REM FL     FLAG TO INSURE GRAPHING ALL DATA
190 REM ST     CALCULATED STEP VALUE OF INNER LOOP
200 REM J      LOOP COUNTER OF OUTER LOOP
210 REM X      DISPLACEMENT ALONG X AXIS
220 REM I      LOOP COUNTER OF INNER LOOP
230 REM M      MEAN VALUE OF ARRAY ZONE BEING MAPPED INTO GRAY SCALE
240 REM N      CALCULATED GRAY SCALE VALUE TO BE GRAPHED
250 REM
260 REM SUBROUTINE TO ACQUIRE SI, MI, MA
270 GOSUB 3000
280 DIM A(SI-1)
290 REM CALCULATE CORRECTION FACTOR
300 C=(MA-MI)/9
310 REM INITIALIZE FLAG
320 FL=0
330 REM CALCULATE STEP VALUE FOR INNER LOOP
340 ST=ITP(SI/128)
350 IF ST<1 THEN ST=1
360 REM ONLY HAVE ROOM FOR 128 PIXELS (HORIZONTIALLY)
370 IF SI/ST>128 THEN ST=ST+1
380 REM SET FLAG IF STEP NOT AN EVEN DIVISOR OF SIZE OF ARRAY
390 IF ITP(SI/ST)<>SI/ST THEN FL=1
400 PAGE
410 REM 2 LOOPS GRAPH DATA, LEFT TO RIGHT, BOTTOM TO TOP
420 FOR J=0 TO 768 STEP 8
430 REM ACQUIRE AN ARRAY OF DATA
440 GOSUB 2000
450 REM INITIALIZE X TO ZERO
460 X=0
470 FOR I=0 TO SI-1 STEP ST
480 REM CALCULATE MEAN OF ZONE
490 M=MEA(A(I:I+ST-1))
500 REM MAP AND GRAPH ONE ZONE
510 GOSUB 670
520 REM INCREMENT X
530 X=X+8
540 NEXT I
550 REM IF FLAG SET, GRAPH LAST FEW ARRAY VALUES
560 IF FL=0 THEN 590
570 M=MEAN (A(SI-ST:SI-1))
580 GOSUB 670
590 NEXT J
600 REM END LOOPS
610 REM PREVENT "READY" FROM BEING PRINTED
620 SMOVE 0,0\PRINT ""
630 RETURN
640 REM END OF MAIN ROUTINE
650 REM ROUTINE TO MAP MEAN INTO GRAY SCALE AND GRAPH
660 REM CHECK FOR UNEXPECTED DATA AND ADJUST IF FOUND
670 IF M<MI THEN M=MI
680 IF M>MA THEN M=MA
690 REM CALCULATE GRAY SCALE VALUE
700 N=(M-MI)/C
710 REM NOW GRAPH IT
720 SMOVE X,J
730 GOSUB 1000
740 RETURN
750 REM
```

2737-48

Fig. 3-20. This is a partial program for mapping array zones into gray-scale values. Application-dependent acquisition routines are omitted.

Two loops control the main routine. The outer loop controls the vertical displacement on the graph. The inner loop calculates the mean value of each zone (line 490) and calls the mapping routine.

If the mean is outside the expected range, the minimum or maximum is substituted (lines 650 to 740). Finally, in line 700, the mean value is mapped to a value between zero and nine before the gray-scale routine is called (line 730).

One application for this program could use the digitized data from the heat scan of a building. Each array would represent one left-to-right scan. Each scan would pass over the building at a greater height than the first. Such data, when graphed using this process and the gray-scale routine, would produce a picture of the building in terms of the heat emitted from it.

by Joyce Ferriss
Based on programs contributed by
Randall O. Decker of United Technologies
Research Center, East Hartford, CT

APPENDIX A

GLOSSARY

alpha cursor -- (Applies only to Alpha mode) A non-storing 5x7 dot matrix which flashes to indicate the writing position of the next character to be displayed on the terminal screen.

alpha mode -- An operating mode of the Tektronix Computer Display Terminal in which alphanumeric information (but no graphics data) can be presented. The terminal enters alpha mode when the PAGE key is pressed or when the PAGE command is executed.

alphanumeric characters -- Alphabetic and numeric characters, as well as special symbols.

array -- A collection of data, each having a common name, data-range, and data type. An array can be defined as either an integer or floating-point array. It can be further classified as one-dimensional or two-dimensional depending on how the array elements are indexed.

ASCII code -- A standardized code for alphanumeric characters, symbols, and special "control" characters. ASCII is an acronym for American Standard Code for Information Interchange.

axis -- (Two-dimensional definition) One of two perpendicular, gridded lines that define the location of a point in a coordinate system. By convention, the vertical axis is called the Y-axis, and the horizontal axis is called the X-axis.

Cartesian coordinate system -- A two-axis grid system that allows you to specify the location of a point in a plane.

cross-hair cursor -- (Applies only to graphics input mode) A pair of non-storing horizontal and vertical lines that appear on the terminal screen during execution of a GIN or SGIN command. The position of the cross hairs is controllable by two thumbwheels at the right side of the keyboard. These cross hairs can be used to locate and input coordinates from the terminal screen.

CRT -- (Cathode ray tube) A vacuum-type picture tube in which a focused electron beam from the cathode is projected onto a fluorescent screen. The TEKTRONIX Computer Display Terminals use storage-type CRTs for displaying alphanumeric and graphic data.

device coordinates -- (Also called "screen coordinates") Refers to X and Y coordinates that are directly related to the hardware of a display device. There is a one-to-one correspondence between points on the display screen and the corresponding device coordinates. (The TEKTRONIX 4010 terminal has a screen coordinate system that ranges from 0 to 1023 in the X-axis, and 0 to 779 in the Y-axis.)

DVST -- (Direct View Storage Tube) A type of cathode ray tube that can be used for storing information long after the writing beam has written the trace.

erase -- To remove the information on the terminal screen, permitting new information to be displayed. The TEKTRONIX Computer Display Terminal is erased by pressing the PAGE key, or by executing the PAGE command.

graphics input mode -- An operating mode of a TEKTRONIX Computer Display Termimal in which screen coordinates can be input to the computer via the cross-hair cursor.

graphics plot mode -- An operating mode of a TEKTRONIX Computer Display Terminal in which vectors (but no alphanumeric information) can be displayed on the terminal screen. The terminal enters graphics plot mode when a MOVE or similar command is executed.

graphics tablet --A graphics input device that allows you to digitize (input the graphic coordinates) from a previously drawn graph, chart, or photo.

hard copy -- A sheet containing a black and white replica of all alphanumeric and graphic data on the terminal screen. A hard copy can be produced by a hard copy unit such as the TEKTRONIX 4631, or by means of a digital plotter such as the TEKTRONIX 4662.

hard copy mode -- An operating mode of the TEKTRONIX Computer Display Terminal in which a hard copy can be generated using the TEKTRONIX 4631 Hard Copy Unit. This type of hard copy can be generated by pressing the terminal's MAKE COPY button, or by executing a PRINT "^[^W" (the characters in quotes are: CTRL-SHIFT-K, CTRL-W).

home position -- (Applies only to alpha mode) A location near the upper-left corner of the terminal screen. The alpha cursor returns to this position when a PAGE, RESETG, or INITG command is executed.

Local operation -- The mode of operation that the terminal is in when the LOCAL/LINE switch is in the LOCAL position. In this mode, no data is sent to the computer (or received from the computer); data entered at the keyboard is simply displayed on the terminal screen.

On-line operation -- The mode of operation that the computer terminal is in when the LOCAL/LINE switch is in the LINE position. In this mode, data can be sent to and received from the computer.

pixel -- A picture element or picture cell. This term is used to describe the smallest unit of information comprising a display.

platen -- The flat surface on a digital plotter or graphics tablet on which data is drawn or digitized.

plotter -- A graphics output device that allows you to plot alphanumeric and graphics data.

pointer -- (Applies only to graphics plot mode) The last position at which the terminal's writing beam stopped after drawing a hidden or actual vector.

tic mark -- One of several equally spaced dashes marked on an axis to denote a particular value. The size and spacing of these tic marks can be modified under program control with the SETGR command.

vector -- A line segment drawn on the terminal when it is in graphics plot mode. The term **dark vector** is used to describe a vector not visible on the screen (as produced by a MOVE command). The term **light vector** describes a vector that **is** visible on the screen (as produced by a DRAW command).

viewport -- An area of the terminal screen (or display device) bounded by a minimum and maximum horizontal screen coordinate and a minimum and maximum vertical screen coordinate. After the viewport has been defined, certain graphics commands will treat it as though it were the entire screen.

waveform -- An array, data-sampling variable, vertical-units string variable, and horizontal-units string variable which have been associated with a variable name for manipulation as a single entity.

window -- The minimum and maximum X and Y values that define the limits of the user data space.

X-coordinate -- A number that specifies the horizontal location of a point in a Cartesian coordinate system.

Y-coordinate -- A number that specifies the vertical location of a point in a Cartesian coordinate system.

APPENDIX B

GRAPHIC COMMAND SUMMARY

I. Initialization Commands

PAGE -- Erases screen of graphics terminal, sets terminal to alpha mode, and returns alpha cursor to home position.

RESETG -- Initializes window and viewport to default values, sets the terminal to alpha mode, and returns the alpha cursor to home position. (It does not erase the terminal screen.)

INITG -- Same as RESETG but the terminal screen is also erased.

II. Pointer-Control Commands

MOVE -- Moves pointer to position specified in user coordinates (no line is drawn).

RMOVE -- Moves pointer to position displaced from current position by specified number of user units.

SMOVE -- Moves pointer to position specified in screen (device) coordinates.

RSMOVE -- Moves pointer to position displaced from current position by specified number of screen units.

DRAW -- Draws a line from current pointer position to position specified in user coordinates.

RDRAW --Draws a line from current pointer position to a point displaced from current position by specified number of user units.

SDRAW -- Draws a line from current pointer position to position specified in screen (device) coordinates.

RSDRAW -- Draws a line from current pointer position to position displaced from current position by specified number of device units.

III. Window and Viewport Commands

WINDOW -- Specifies range of user data to be drawn on selected portion of graphics device.

VIEWPORT -- Specifies portion of graphics device space to be used for plotting data.

SEEWINDOW -- Obtains the minimum and maximum X and Y values that define the current graphics window.

SEEVIEW -- Obtains the minimum and maximum X and Y screen coordinates that define the current graphics viewport.

IV. Array and Waveform Plotting Commands

DISPLAY -- Plots arrays and waveforms into current graphics window without displaying any graticule, axes, or labels.

GRAPH -- Plots arrays or waveforms complete with graticule and axes labels.

XYPLOT -- Creates an X-Y plot of one array or waveform against a second one. Axes and labels accompany the plot.

SETGR -- Modifies the default plot produced by GRAPH or XYPLOT.

V. Graphic Input/Output Commands

GIN -- Inputs position of cross-hair cursor in user coordinates.

SGIN -- Inputs position of cross-hair cursor in screen (device) coordinates.

DRAWON -- Transfers graphic output (normally directed to terminal) to any device OPEN for WRITE.

APPENDIX C

BIBLIOGRAPHY

Newman, William M. and Robert F. Sproull. **Principles of Interactive Computer Graphics**. New York: McGraw-Hill Book Company, 1973.

Rogers, David F. and J. Alan Adams. **Mathematical Elements for Computer Graphics**. New York: McGraw-Hill Book Company, 1976.

TEKTRONIX 4010 Computer Display Terminal Users Manual. (070-1225-00). Beaverton, OR: Tektronix, Inc., 1972.

HANDSHAKE, Tektronix, Inc. (Newsletter of the Signal Processing Systems Users Group.): Vol. 2, Nos. 1-4; Vol. 3, Nos. 2 & 3; Vol. 4, No. 1.

YOUR COMMENTS COUNT

The Manual Writers at Tektronix, Inc. are interested in what you think about this manual, how you use it, and changes you might like to see in future manuals. Any queries regarding this manual will be answered personally.

What did you find that was:

interesting? _____

_____

frustrating? _____

_____

helpful? _____

_____

confusing? _____

_____

Is there anything you would like to see added to or deleted from this manual? _____

_____

_____

_____

What is your major application area for this product? _____

_____

_____

_____

Have you found any interesting applications, operating hints, or software routines which you would like to share with us? _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

* * * * *

Name: _____ Position: _____

Company: _____ Department: _____

Street: _____

City: _____ State: _____ Zip: _____

Fold on dotted lines and tape.
Postage will be paid by Tektronix, Inc. if mailed in U.S.A.

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1 BEAVERTON, OR

POSTAGE WILL BE PAID BY ADDRESSEE

TEKTRONIX, INC.
P.O. Box 500
Beaverton, Oregon 97005
Attn: Del. Sta. 94-384