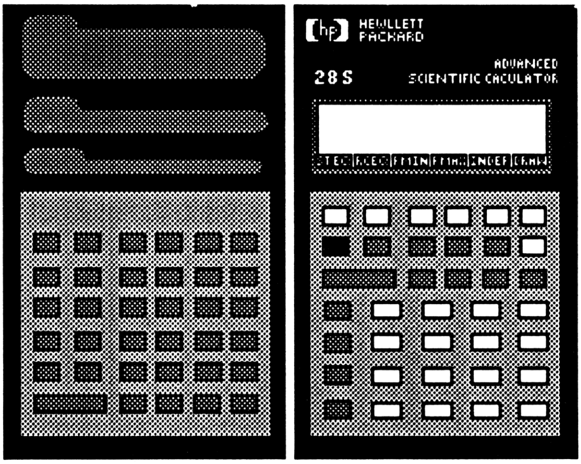
ADVANCED GRAPHING FOR THE HP 28S



Brian Maguire

EduCALC

27953 CABOT ROAD
LAGUNA NIGUEL, CA 92677 U.S.A.

(714) 582-2637

Advanced Graphing

for the HP 28S

Copyright ©1990 by Brian Maguire
All rights reserved

Acknowledgments: My extreme thanks to Judy Zavisa, whose time and patience helped make this book possible.

Copyright: Copyright © 1990. All rights reserved. No portions of this text may be copied, electronically or mechanically, nor stored in a retrieval system without written permission of the author.

Disclaimer: No warrantee, expressed or implied has been made with regard to the contents of this book, and keystroke procedures herein. They are presented on an "as is" basis with out guarantee of merchantability or use for any special purpose. The buyer, not the author nor the publisher, shall bear all cost of incidental or consequential damage due to any error in this text.

TABLE OF CONTENTS

PART TWO

ADVANCED GRAPHING PROGRAMS

CHAPTER ONE

INTRODUCTION

The 28S is a powerful hand-sized computer. It has more memory and can do much more than most home computers from 10 years ago. Although the owner's manuals are down to earth and do an excellent job of explaining this intricate machine, they only touch the surface of the 28S's broad range of applications. This book picks up where the owner's manuals leave off. It gives a few of the endless ways the 28S can be used as a graphics tool.

WHAT'S BETWEEN THE COVERS?

This book presents five different root level graphing applications (chapter two through five) and three advanced graphing applications (chapter six). A root level graphing application can be thought of as a basic building block. It can only draw one specific type of graph. For example, SDRW from chapter four only draws shapes, nothing else.

In contrast, an advanced graphing program can draw almost any type of graph, but only in one specific way. For example, DDRW from chapter six can draw shapes as well as functions, but always in three dimensions. It does this by calling on a root level graphing program to do the actual drawing for it. DDRW would call SDRW to draw a three dimensional shape and DRAW to plot a three dimensional function. The relationship between a root level and advanced graphing application can be thought of as a foundation and a house. The advanced graphing program needs the support of the root level program.

Both root level and advanced applications have been incorporated into a user friendly, custom menu similar to the built in 28S menus. If you are at all familiar with the built in PLOT menu you'll have no problem with these custom menus. Like the PLOT menu, each custom graphing menu has a specific drawing program, the name of an object containing the parameters used by that program, and supporting programs that allow you to alter these parameters. For example, the PLOT menu's drawing program, parameter name and several of the supporting programs are DRAW, PPAR, and PMIN, PMAX, and INDEP respectively. Because of each graphing menu's similarity, once you learn one menu you'll have no problem with the others.

Application menus aren't the only type of custom menus you'll encounter in this text. Before ever coming to an application menu you will use a selection menu. A selection menu does just what its name implies. It allows you to select from a number of application menus. The programs in this chapter set up the frame work for the starting selection menu, as well as the custom application menu for each chapter. You must key these programs in.

WHAT TO DO FIRST

The first thing you should do is read the introduction. I was hesitant to even use the heading "INTRODUCTION" for this chapter, in fear that some readers would flip right through it. The introduction in this text is slightly different than most. A common introduction would give a basic background on how the 28S works and how to enter a program. Most readers already have a working understanding of the 28S from reading the owner's manuals. That's why I have dedicated the introduction on how to operate this book. You must read the introduction to understand the remaining chapters. It gives the basic outline each chapter follows. It describes how programs are listed, where to put them, and which programs you'll need. For those readers that feel they need a refresher on how to use the 28S, read the owner's manuals or flip through appendix A.

Finally, at the end of this chapter, you'll come across a section containing program listings. The first program, CKNM, checks for typographical mistakes in any object you store in your calculator. It generates a check number that can be compared with the one in each program listing. CKNM is optional, but highly recommended. All of the remaining programs in this chapter take you in and out of the various custom menus and directories. They are needed by every chapter in this book so you must key them in.

After reading the introduction and entering the programs at the end of this chapter, you will be ready for any of the remaining chapters. You can follow them chapter by chapter, or skip to one you'd like to read first. Chapters two through six are all self-supporting. After reading the introduction you'll have the basic tools and knowledge to jump in to any of the remaining chapters.

A BASIC OUTLINE OF THE REMAINING CHAPTERS

The remaining chapters are divided into two sections. Section one contains the programming. It has a brief introduction and a number of program listings. Section two is the example section. It describes how to use each graphing program and gives a number of examples. The next several pages describe section one of every chapter.

INTRODUCTION

As you might expect, each chapter starts with an introduction. These first few paragraphs briefly explain what the programs in that chapter will do. By reading the introduction you can decide if that chapter is for you. You might also want to skim through the example section before keying in any programs.

A PROGRAM TABLE

A program table is included with every chapter. **Pay close attention to this table !** The first column lists all the programs that are needed in that chapter. Some of these programs may have been used in earlier chapters. If so, they won't be listed in that chapter. They will have an asterisk after their name to let you know you'll need to flip back to the page number indicated and check whether or not you've entered that program already.

The second column shows the page number for each program listing. This is very useful when you need to key in a program that is not listed it that chapter. It saves you the time of flipping to the index.

Finally, the last column gives the memory usage of each program. This number, in bytes, will give you an idea of how much memory is needed to run each program. The sum of this column will be the total memory needed to use that particular chapter. Remember though, some of these programs may have been used in earlier chapters and may be in your calculator already.

If the amount of memory you'll need is too large, you may decide to enter only the programs that are needed for that chapter. Most of the programs in each chapter make it easier for you to create, store, and alter different graphing parameters. These programs are extremely useful and highly recommended, but can be omitted. For example, the built in PLOT menu has commands like STEQ, PMAX and COLΣ to store and manipulate EQ, PPAR, and ΣPAR respectively. You would never want to trash all these commands, but there may be several that you could do without. Omitting these programs from a particular chapter can sometimes trim the required memory in half.

ABOUT EACH PROGRAM LISTING

Each object that you must store in your calculator has a listing. They are in alphabetical order in the first section of every chapter. If applicable, each program listing will have the following headings.

A SHORT PHRASE TO DESCRIBE THE PROGRAM

Each program has an underlined phrase just above its name. This phrase briefly describes that program. In most cases, the name of the program is just a mnemonic of this phrase.

PROGRAM NAME(CHECK NUMBER)

The name of the program will be printed in boxy characters resembling the characters displayed by the 28S. It will be followed by a check number in parenthesis. Each program or object stored in your calculator will have a unique check number generated by the program CKNM, on page 19.

After storing the program you can check for typographical errors by putting its name on level one and running CKNM. The number that is returned should be the same as the one in parenthesis. If not, just skim through you program, correct the mistake, and try again.

The use of CKNM is optional, but highly recommended. Even the best programmer can delete a character or accidently swap two letters. With the use of CKNM though, these mistakes can be easily found and corrected before running a program. Entering a program without using CKNM is like using a word processor without a spell checker.

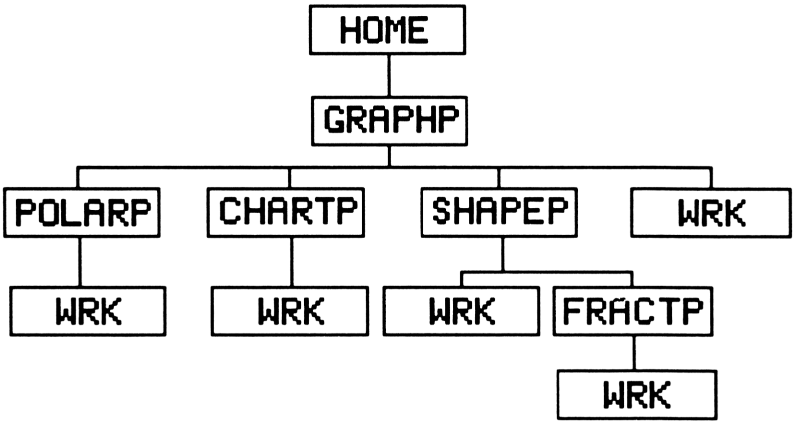
PROGRAM LISTING

The program is the most important part of each listing, so it is boxed and centered on the page. It is listed in the same way the 28S lists its programs when on the stack. This makes it much easier to find and correct any mistakes that are made when keying in a program. Simply use the view up and view down keys to compare the listing on your calculator with the one in the book.

PATH

Programs with a common interest should normally be stored in one directory. A tree of the directories used in this book are shown on the next page. It may be helpful to refer to it when keying in programs. POLARP, CHARTP, SHAPEP, and FRACTP store programs geared towards one type of graphing technique. GRAPHP, on the other hand, contains programs that are used by more than one graphing application. The directories named WRK below each program directory are work directories. You'll be creating and storing your own programs and variables here. Having a separate work space makes it easier to sort through the multitude of programs. More importantly, it protects the program directory from accidently being purged.

A list representing the suggested path is given for each program. The directory that the program should be stored in is the last one in this list. For example, the the path { HOME GRAPHP POLARP } says store the program in the polar graphing directory, POLARP, below the main graphing directory, GRAPHP, below the home directory, HOME.



SUMMARY

The opening few paragraphs summarize program operation. You'll find a brief explanation on how the program or object is used, possible errors, how the program operates, and other added tidbits.

INPUT

Any input received by the program such as objects on the stack, values of current variables, or input through the keyboard. Each input will be labeled under one of the following headings.

LCD	(liquid crystal display) Any input the program receives from the current display. A program receives display input using the 28S command LCD→.
KEYBOARD	What keys are active during program execution.
MEMORY	Memory locations, such as flags or variables, that the program uses.
LEVEL #	Objects required to be on specified level of the stack.

OUTPUT

Any program output will be listed under this heading. An example of common program output is an object on the stack. Other possibilities are a value stored in memory, information sent to the printer, or display output. The output headings are as follows.

LCD	(liquid crystal display) A brief description of any display output.
MEMORY	Any memory location such as a flag or an object that is changed or created by the program. Custom menus are listed under memory, not lcd, because the calculator stores the custom menu internally.
PRINTER	The type of output, if any, sent to the printer.
BEEP	Any audible tones and what they signify.
LEVEL #	Objects left on the stack by the program.

UTILITIES

All subroutines used by the program are listed under this heading. Any program that isn't a built in 28S command is considered a subroutine. If it already isn't in your calculator's memory you can easily flip to the indicated page number and type it in (their page numbers are listed in the beginning of each chapter).

Variables are not considered utilities. If a variable appears in a program it will be listed under MEMORY for either the input or output heading.

PROGRAM LISTINGS

The remaining part of this chapter contains listings for every program, directory, list, or other object that you must store in your calculator. The listings in this section are used in all of the remaining chapters, so you must key them in. They allow you to move from directory to directory and create the custom graphing menus used by each chapter.

Before storing your first program you should create the main graphing directory. This is the parent directory for all other directories. Everything is stored either in GRAPHP or in a directory below it (see the directory tree on page 14). Go to the home directory and create GRAPHP by entering HOME 'GRAPHP' CRDIR. Now you're ready to enter all the programs from this chapter. A table of these programs is given below.

PROGRAM TABLE

Program	Page#	Bytes
CKNM	19	127.5
End	20	16.0
GRAPH	21	37.5
GRAPHP	22	18.0
QUIT	23	45.0
Start	24	50.0
STLST	25	64.5
WRK	26	15.0

Many of the programs in this book are used in more than one chapter. A good example is Start. It creates each chapter's custom graphing menu. Since every chapter uses this same program, it is only listed once, in chapter one. Any latter chapter will refer to it in the table at the beginning of that chapter. If you are following this text chapter by chapter you won't have to worry about flipping back to a program listed in a previous chapter since you will have already stored it in your calculator. If you're jumping from chapter to chapter though, you might want to check your calculator's memory to see if every program from the table has been entered.

NOTE

There are several programs in this book with the same name. Just because two chapters contain the same name in their program tables doesn't necessarily mean they're the same program. They could be two very different programs stored in two different directories.

For example, chapters two through five all have listings for STLST. A different version of STLST is used to create each custom graphing menu. Don't pass over a listing because you remember entering a program with the same name. It could be a very different program.

Generate a Check Number

CKNM(822964)

```
« DUP IFERR RCL THEN
IF 31 FS? THEN DROP
END "" END →STR SWAP
→STR + 0 OVER SIZE 1
SWAP FOR i OVER i
DUP SUB NUM i * +
NEXT SWAP DROP »
```

SUMMARY

This program is designed to generate a unique number for every object. All the programs in this text will have their check number in parenthesis following the program name. To check whether a program has been keyed in properly, simply put the name of the program on level one of the stack and run CKNM. After a few seconds a number will be returned to level one.

This number is generated by changing the object and the object's name into a character string. It then generates a number which is the sum of each character's ASCII value times its position in the string. Note that the check number for this program is 822964.

PATH { HOME }

INPUT LEVEL ONE The name of the program you're checking.

OUTPUT LEVEL ONE An integer (The resultant check number).

UTILITIES None

End Specific Graphing Menu

End(5393)

'GRAPH'

SUMMARY

End does just what it sounds like. It leaves the menu you're in and returns you to the main graphing menu. It also puts you back in the work directory that is subordinate to GRAPHP. This is illustrated in the directory tree on page 22. Also see GRAPH for an explanation of the main graphing menu.

End can be found as the last entry in the following custom graphing menus.

Menu	Chapter
Polar Graphing Menu	2
Chart Graphing Menu	3
Shape Drawing Menu	4
Fractal Drawing Menu	5

PATH { HOME GRAPHP }

INPUT MEMORY The list stored in $TLST located in GRAPHP, the main graphing directory.

OUTPUT MEMORY Directory control is given to the work directory subordinate to GRAPHP and the main graphing menu is created.

UTILITIES GRAPH

Main Graphing Program Directory

GRAPH(20826)

```
« GRAPHP Start »
```

SUMMARY

GRAPH puts you into the work directory, WRK, just below the main graphing program directory and creates the main graphing menu. This is a selection menu as opposed to an application menu. Pressing any one of the keys in this menu will create another custom menu. The menu labels are described below.

Menu Key	Menu Created	Chapter	Control given to work directory below...
POLAR	Polar Graphing	2	POLARP
CHART	Chart Drawing	3	CHARTP
SHAPE	Shape Drawing	4	SHAPEP
FRACT	Fractal Drawing	5	FRACTP
ADVAN	Advanced Graphing	6	Same as before
QUIT	User Menu	1	HOME

PATH { HOME }

INPUT MEMORY The list stored in STLST located in GRAPHP, the main graphing directory.

OUTPUT MEMORY Directory control is given to the work directory subordinate to GRAPHP and the main graphing menu is created.

UTILITIES Start

Polar Graphing Program Directory

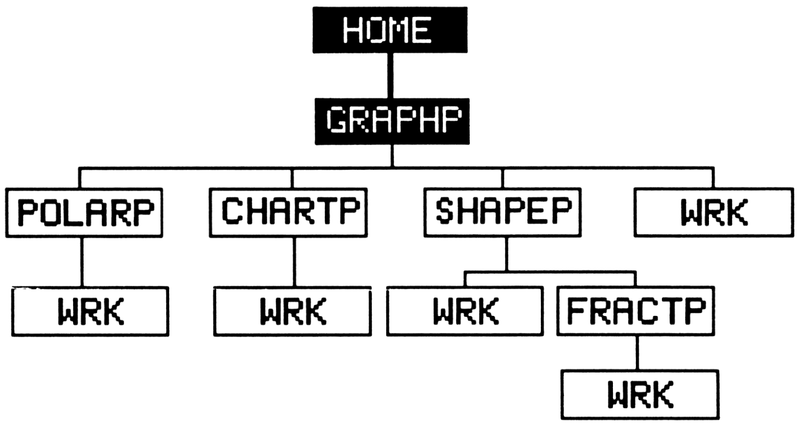
GRAPHP(2391)

Directory

SUMMARY

The graphing application directories and all the programs that are used in more than one graphing directory are stored in GRAPHP. It is the parent directory for all other graphing directories. All the programs stored here can be used by any one of the programs stored in lower directories (See the directory tree below).

In contrast, programs that are specific to one graphing technique should be stored in a separate directory. For example, programs that are only used for polar graphing should be stored in POLARP. If you classify your programs wisely you will never get lost in a web of directories.



PATH { HOME }

Quit the Main Graphing Menu

QUIT(43721)

```
« HOME 23 MENU »
```

SUMMARY

QUIT allows you to quit the main graphing menu and return to the home directory. It also displays the user menu. It is suggested that you put this program in the main graphing directory, GRAPHP. You could, however, store it in the home directory. This way you can use QUIT in your own custom menus. Then, no matter where you're at, QUIT will return you to the home directory and display the user menu.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	None
OUTPUT	MEMORY	Command is given to the home directory and the user menu is displayed.
UTILITIES		None

Start an Application Menu

Start(38113)

```
« 17 CF WRK STLST
MENU »
```

SUMMARY

Start is used to create the graphing menus for chapter one to five. First, it clears flag 17. This flag is tested by most of the graphing programs in this book. Then, it puts you in the work directory located just below the program directory you're in. Then, the list generated by STLST is used to create a custom graphing menu.

As you read through this book, you'll notice every chapter has a different listing for STLST. Sometimes STLST is simply a list while other times it is a program that creates a list. Although they all have the same name, they really are quite different. Each one is stored in a different program directory and returns a unique list. Thus depending on the directory you're in, Start will create different custom menus because it calls on different versions of STLST.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list stored in STLST is used to create a custom menu. (If it is a list)
OUTPUT	MEMORY	A custom menu is created.
UTILITIES		STLST (If it is a program)

Main Starting Menu List

STLST(71785)

```
{ POLAR CHART SHAPE
FRACT ADVAN QUIT }
```

SUMMARY

You'll find a different version of STLST in every program directory. Start, the program that creates the custom graphing menus, takes the list from STLST, or in some cases the list created by STLST, and forms a custom graphing menu.

This version of STLST contains the list that is used to create the main graphing selection menu. Each object is a program that creates a new custom menu and puts you in a different directory. The table on page 21 describes what each program does and which chapter it is located in.

PATH { HOME GRAPHP }

Main Graphing Work Directory

WRK(954)

Directory

SUMMARY

As you read through this text you'll notice every chapter has a listing for a directory named WRK. Don't let this confuse you. These are all different directories. Each program directory has its own work directory named WRK.

WRK is used to store variables or programs you create. The work directory listed here is located just below GRAPHP. This is shown on the directory tree on page 22. Working in WRK protects the programs in GRAPHP, the main graphing program directory, from accidentally being purged. It is also easier sorting through variables in a smaller work directory than sifting through all your variables and programs in one big directory.

PATH { HOME GRAPHP }

CHAPTER TWO

GRAPHING IN POLAR COORDINATES

SECTION ONE

REFERENCE SECTION

INTRODUCTION

This is the reference section for the polar graphing programs. These programs allow you to graph any equation in polar coordinates. Before starting this chapter, you should flip through section two, the example section. It will give you an idea of what to expect from these programs. Then, if you like what you see, you can jump right in. Be sure you've entered the programs from chapter one first.

Most of the programs in this chapter are stored in POLARP, the polar graphing program directory. You must create this directory before storing any programs (where would you put them?). Go to GRAPHP, the main graphing program directory, and create POLARP by entering, HOME GRAPHP ENTER POLARP CRDIR. See the directory tree on page 43. Now you're ready to enter the programs from this chapter.

A table of all the programs you'll need in this chapter is given on the next page. Of course, you must enter the programs from chapter one before starting this chapter.

PROGRAM TABLE

Program	Page#	Bytes	Program	Page#	Bytes
ADRW	30	275.0	PCHK	41	28.5
AMAX	33	41	POLAR	42	37.5
AMIN	34	41	POLARP	43	18.0
APAR	35	29	PUTA	44	54.5
CLCD?	36	52.5	PUTP	45	33.5
CTL$T	37	72.5	Rad	46	32.5
Deg	38	32.5	RAD•	47	18.5
DEG•	39	18.5	TLT	48	92.5
DGTZ?	40	47.5	WRK	49	15.0

NOTE

There are several programs in this book with the same name. Just because two chapters contain the same name in their program tables doesn't necessarily mean they are the same program. They could be two very different programs stored in two different directories.

For example, chapters two through five all have different listings for TLT. TLT is used to create a custom graphing menu. Since every chapter creates its own custom menu, each one will have its own version of TLT. Don't pass over a listing because you remember entering a program with the same name. It could be a very different program.

Angular (polar) Drawing Procedure

ADRW<2595534>

```
« CLCD? DRAX RCLF 1
*H APAR LIST→ DROP
EVAL 5 IF 60 FS?
THEN D→R END 'PPAR'
3 GETI 4 ROLLD GET *
→ t res « FOR i i t
STO EQ →NUM IF DUP
TYPE 0 == THEN DUP i
R→C P→R PIXEL ABS
.001 + INV .1 MAX 1
MIN ELSE DROP 1 END
res * STEP t PURGE
STOF » DGTZ? »
```

SUMMARY

Using the parameters from the list stored in APAR, ADRW graphs the polar equation stored in EQ. The parameter list reads as follows.

{ minimum angle (any real number),
maximum angle (any real number)
polar graphing angle mode (RAD or DEG) }

ADRW begins with the starting angle and continues to plot points until the ending angle is reached. Both these values can be set by using AMIN and AMAX. The third item in APAR "tells" ADRW what angle mode to graph in. This allows you to keep your calculator in radians, but graph a function using degrees.

You'll notice two keys labeled RAD and DEG in the polar graphing menu. One of these labels will have a box after its name. This is the current polar graphing angle mode. Pressing the menu key without a box will activate that mode, puts a box after its name, and clears the box from the name of the old angle mode.

NOTE

The two custom menu labels RAD and DEG are not the same as in the 28S mode menu. They do not reflect the angle mode your calculator is in. they do indicate the angle mode that ADRW will graph in. This means you can graph a polar function in radians while your calculator is set to degrees.

ADRW, like most other root level drawing programs, uses the utilities CLCD? and DGTZ?. When I say root level I mean it does the actual graphing. In contrast, the graphing programs in chapter six are advanced graphing programs. They call other procedures, like ADRW, to do the graphing for them.

Both CLCD? and DGTZ?check the status of flag 17 and, if set, clear the display before graphing and digitize after graphing is complete. DGTZ? also clears flag 17 after checking its status. These two subprograms allow you to use ADRW directly from the custom menu or as part of a larger program.

When using this program from the menu (it is assumed that flag 17 is cleared) the display will be cleared, a polar function will be graphed, and you will enter digitizing mode. On the other hand, if you want to use it as a subprogram, have the calling program set flag 17 before executing ADRW. This prevents the display from being cleared or having execution halted for digitizing. This is exactly what the advanced graphing programs from chapter six do.

You might notice the step size of the main program loop is inversely proportioned to the radius. The radius is taken as a control on the step size since a larger radius means ADRW has to travel a greater distance. The larger the radius is, the more points there are to be graphed and likewise, the longer it takes to graph the equation.

To add to this program's versatility, ADRW gets the dependent variable and resolution from the plot parameters, PPAR. This means you can change them using the built in 28S commands RES and INDEP found in the PLOT menu

NOTE

If the display doesn't clear when using ADRW from the custom menu, flag 17 is set. If this happens, you can stop the program and clear flag 17 or ADRW will clear it after it is done graphing. Either way, flag 17 will be cleared and you can try again.

PATH		{ HOME GRAPHP }
INPUT	<u>MEMORY</u>	The equation or program to be graphed must be stored in EQ. The list in APAR is used to define the starting angle, ending angle, and the angle mode to graph in. The fourth and fifth item in the plot parameters, PPAR, are used to define the dependent variable and resolution. Finally, the status of flag 17 is checked.
	<u>KEYBOARD</u>	If flag 17 is clear digitizing mode is activated. Refer to appendix A if you're unfamiliar with digitizing.
OUTPUT	<u>LCD</u>	The polar graph of the equation in EQ is drawn on the screen. If flag 17 is cleared the display will be cleared before graphing.
	<u>STACK</u>	Any objects left by digitizing. (if flag 17 was clear before running)
UTILITIES		CLCD?, DGTZ

Angle Maximum (ending angle)

AMAX(18408)

```
« 2 'APAR' PUTP »
```

SUMMARY

Starting at the minimum angle, ADRW graphs the equation stored in EQ until the maximum (ending) angle has been reached. The maximum angle can be set by putting it on level one and running AMAX. AMAX puts the object from level one into the second position in APAR, the polar graphing parameter. APAR reads as follows.

{ minimum angle (any real number),
maximum angle (any real number)
polar graphing angle mode (RAD or DEG) }

PATH { HOME GRAPHP POLARP }

INPUT LEVEL ONE Any real number

OUTPUT MEMORY The number from level one is put into the second position in APAR.

UTILITIES PUTP

Angle Minimum(starting angle)

AMIN(18353)

```
« 1 'APAR' PUTP »
```

SUMMARY

As its name implies, AMIN stores the minimum (starting) angle into APAR, the polar graphing parameters. It does this by putting the object from level one into the first position in APAR. ADRW starts graphing from this minimum angle and continues until the maximum, (ending) angle is reached. APAR reads as follows.

{ minimum angle (any real number),
maximum angle (any real number)
polar graphing angle mode (RAD or DEG) }

PATH		{ HOME GRAPHP POLARP }
INPUT	LEVEL ONE	Any real number
OUTPUT	MEMORY	The number from level one is put into the first position in the polar graphing parameters, APAR.
UTILITIES		PUTP

Angular (polar) Parameter

APAR(11845)

```
{ 0 360 DEG }
```

SUMMARY

APAR is a list of parameters used by ADRW when graphing a polar equation. It reads as follows.

{ minimum angle (any real number),
maximum angle (any real number)
polar graphing angle mode (RAD or DEG) }

Most of the programs in this chapter use or manipulate APAR. But what happens when APAR doesn't exist in the work directory? The polar graphing parameters defined in the main graphing directory, GRAPHP, will be used as a default.

As an example, lets say you're starting out with an empty work directory and want to set the minimum graphing angle to 90°. You would enter 90 AMIN. AMIN first creates APAR by recalling its default value stored in the main graphing directory and storing it in the work directory. Now AMIN is able to put 90 into APAR. Remember a variable in a different directory can be recalled, but can't be changed. After pressing AMIN APAR will suddenly appear in the work directory.

PATH { HOME GRAPHP }

Check if the Display Should be Cleared

CLCD?(63015)

```
« 1 *W IF 17 FC?
THEN CLLCD END »
```

SUMMARY

ADRW, as well as every graphing program in this book, can be used as an independent program or as a subprogram. When you press its menu key in the custom menu you want it to clear the display, draw a graph, and enter digitizing mode. In this instance, it acts as a free standing program.

What if you want to use it as a subprogram? You might not want the display cleared before graphing. This is where CLCD? comes in. It checks to see if flag 17 is clear. If so, it clears the display. If you don't want the display cleared simply have the calling program set flag 17 before calling ADRW.

This is exactly what the advanced graphing programs from chapter six do. DDRW, EDRW, and MDRW, the three dimensional, animated, and extended graphing programs need to call a root level graphing procedure to do the actual graphing for them. Before calling a root level program they set flag 17.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	Flag 17 is tested
OUTPUT	LCD	If flag 17 is clear the display is cleared
UTILITIES		None

Constant Starting List

CTL$T(83163)

```
{APAR AMIN AMAX Deg
Rad ADRW ADVAN End }
```

SUMMARY

CTL$T is used by TLT as a base for the polar graphing menu. TLT puts CTL$T on the stack and, depending on the last object in APAR, replaces either Deg with DEG• or Rad with RAD• . The resultant list is then used by $tart to create the polar graphing menu.

PATH { HOME GRAPHP POLARP }

Plot the Polar Graph in Degrees

Deg(16748)

```
« { DEG } PUTA »
```

SUMMARY

Deg stores DEG as the third object in APAR, the polar graphing parameters. This lets the calculator "know" that the minimum and maximum angles in APAR are in degrees. APAR reads as follows.

{ minimum (starting) angle, maximum (ending) angle, polar graphing angle mode }

Since lower case letters appear in all capitals in the user or custom menu, Deg is displayed as DEG. After pressing DEG **from the polar menu**, you'll see a box appear after its name. This is to show that degrees is now the active polar graphing angle mode.

PATH	{ HOME GRAPHP POLARP }
INPUT	None
OUTPUT MEMORY	DEG is stored as the third object in PPAR. A new custom menu is also created to display DEG as the new polar graphing angle mode.
UTILITIES	PUTA

An Empty Program

DEG■ (3525)

```
« »
```

SUMMARY

DEG■ is used for display purposes only. It is used in the custom polar graphing menu to show that the active polar graphing angle mode is degrees.

Since having DEG■ in the custom menu means degrees is the active polar graphing angle mode, pressing its menu key should not change anything. This is why an empty program is stored here.

PATH { HOME GRAPHP POLARP }

Check if Digitizing Mode Should be Entered

DGTZ?(55688)

```
« IF 17 FC?C THEN
DGTIZ END »
```

SUMMARY

ADRW, as well as every graphing program in this book, can be used as an independent program or used as a subprogram. When you press its menu key in the custom menu you want it to clear the display, draw a graph, and enter digitizing mode. In this instance it acts as a free standing program.

What if you want to use it as a subprogram? You might not want to digitize. This is where DGTZ? comes in. It checks flag 17 and, if it is clear, enters digitizing mode. If you don't want to digitize, simply have the calling program set flag 17.

This is exactly what DDRW, EDRW, and MDRW from chapter six do. They set flag 17 before calling a root level graphing procedure, such as ADRW, to do the graphing for them.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The status of flag 17
OUTPUT	KEYBOARD	If flag 17 is clear you will enter digitizing mode
	MEMORY	Flag 17 is cleared
	STACK	Any objects left from digitizing
UTILITIES		None

Parameter Checking Procedure

PCHK(23931)

```
« DUP RCL OVER STO »
```

SUMMARY

PCHK recalls the object on level one and stores it back into the same variable name. Nothing happens if the object already exists in the current directory. If it doesn't, its value from the closest directory in the current path is stored in the current directory. Now any program can manipulate that object, since it is in the current directory.

PATH		{ HOME GRAPHP }
INPUT	LEVEL ONE	An object name
	MEMORY	The contents of the object on level one.
OUTPUT	LEVEL ONE	An object name
	MEMORY	If it doesn't exist, the variable from level one is created in the work directory, WRK.
UTILITIES		None

Create Polar Graphing Menu

POLAR(21106)

```
« POLARP Start »
```

SUMMARY

POLAR puts you into the work directory, WRK, and creates the polar graphing menu. You'll probably notice, if you haven't already, that every graphing menu has its own program directory, the directory with the programing specific to that topic, its own work directory, titled WRK, its own graphing menu, and a program to enter that menu.

Except for the last letter, the commands to enter the graphing menus all have the same name as their program directories. For example, POLAR puts you in the work directory, WRK, just below POLARP. Likewise, SHAPE puts you into the shape drawing work directory, WRK, just below SHAPEP.

PATH		{ HOME GRAPHP POLARP }
INPUT	MEMORY	The list stored in CTLST in the directory POLARP and the last object in the list stored in APAR.
OUTPUT	MEMORY	Directory control is given to the work directory, WRK subordinate to POLARP. This is shown in the directory tree on the next page. The custom polar graphing menu is also created.
UTILITIES		Start

Polar Graphing Program Directory

POLARP(2429)

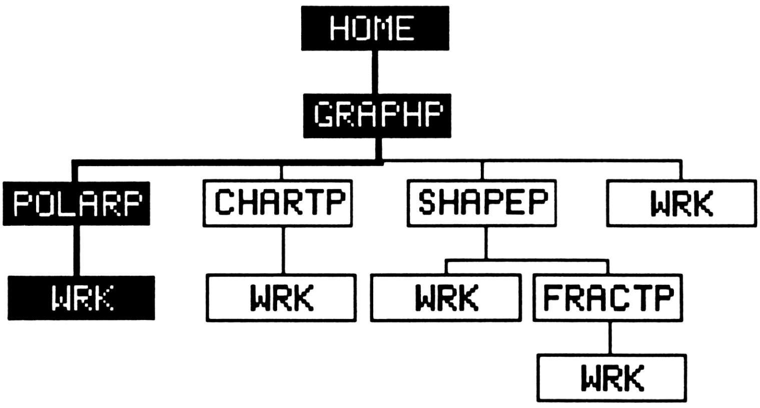
Directory

SUMMARY

This is the directory where all the programs that are specific to polar graphing are stored. Storing the polar graphing programs in their own directory creates a sense of order, like chapters in a book.

Some programs that are used for polar graphing are stored in GRAPHP. This is because they are used by graphing procedures in other directories. Putting these programs in GRAPHP allows access to any program in or subordinate to GRAPHP. Thus a program in CHARTP can use ADRW as well as a program in POLARP, since it is in stored in GRAPHP. A program in CHARTP can't use AMIN though, because it is stored in POLARP, which isn't in CHARTP's path. The directory tree below illustrates this.

PATH { HOME GRAPHP }



Put the Polar Graphing Angle Mode into APAR

PUTA(59773)

```
« LIST→ DROP 3
'APAR' PUTP Start »
```

SUMMARY

PUTA is called on by both Deg and Rad. It puts the object from the list on level one, either { DEG } or { RAD }, into the third position in APAR and creates a new custom menu. This "tells" ADRW whether to graph in degrees or radians. APAR reads as follows.

{ minimum angle (any real number),
maximum angle (any real number),
polar graphing angle mode (DEG or RAD) }

PATH { HOME GRAPHP POLARP }

INPUT LEVEL ONE A list (Either { DEG } or { RAD })

OUTPUT MEMORY The new graphing angle mode, either RAD or DEG is stored as the third object in APAR. A new polar graphing menu is also created.

UTILITIES PUTP, Start

Put an Object into a Parameter List

PUTP(26801)

```
« PCHK SWAP ROT PUT
»
```

SUMMARY

First, PUTP takes the name of the parameter list from level one and checks whether it exists in the work directory. If not, it is created using that parameter list's default value. The default value is always stored in the main graphing directory. GRAPHP. Then, it puts the object from level three into the parameter list. The integer from level two specifies the position to put it in.

PATH		{ HOME GRAPHP }
INPUT	LEVEL THREE	An object to be put into the parameter list
	LEVEL TWO	The position to put the object from level three
	LEVEL ONE	The name of a parameter list
	MEMORY	If it doesn't exist, the default value of the parameter list on level one.
OUTPUT	MEMORY	If it doesn't exist in the work directory, the parameter list from level one is created. Then, the object on level three is put into that parameter list in the position indicated by the integer on level two.
UTILITIES		PCHK

Plot the Polar Graph in Degrees

Rad(16889)

```
« { RAD } PUTA »
```

SUMMARY

Rad stores RAD as the third object in APAR, the polar graphing parameters. This lets the calculator "know" that the minimum and maximum angles in APAR are in radians. APAR reads as follows.

{ minimum angle (any real number),
maximum angle (any real number),
polar graphing angle mode (DEG or RAD) }

Since lower case letters appear in all capitals when in the user or custom menu Rad is displayed as RAD. After pressing RAD **from the polar menu**, you'll see a box appear after its name. This is to show that radians is now the active polar graphing angle mode.

PATH	{ HOME GRAPHP POLARP }
INPUT	None
OUTPUT MEMORY	RAD is stored as the third object in PPAR. A new custom menu is also created to display the new polar graphing angle mode.
UTILITIES	PUTA

An Empty Program

RAD■ (3550)

```
« »
```

SUMMARY

RAD■ is used in the custom graphing menu to show that the active polar graphing angle mode is radians.

Since having RAD■ in the custom menu means radians is the active polar graphing angle mode, pressing its menu key should not change anything. This is why an empty program is stored here. We don't want anything to happen when its menu key is pressed.

PATH	{ HOME GRAPHP POLARP }
INPUT	None
OUTPUT	None
UTILITIES	None

Polar Menu Starting List

STLST(233687)

```
« CTLST APAR 3 GET
→STR "" SWAP + "°"
+ STR→ 4 OVER 'RAD°'
SAME + SWAP PUT »
```

SUMMARY

You'll find a different version of STLST in every program directory. Start, the program that creates the custom graphing menu, takes the list in STLST, or in this case the list created by STLST, and forms a custom menu.

This version of STLST checks the active polar graphing angle mode indicated by the third object in APAR. It adds a box (°) to the name of the active mode and puts it in the polar menu list. This lets you know exactly which angle mode ADRW will graph in.

PATH		{ HOME GRAPHP POLARP }
INPUT	MEMORY	The third object in APAR specifies the current polar graphing angle mode.
OUTPUT	LEVEL ONE	The polar graphing menu list
UTILITIES		None

Polar Graphing Work Directory

WRK(954)

Directory

SUMMARY

As you read through this text you'll notice every chapter has a listing for a directory named WRK. These are all different directories. Each program directory has its own work directory named WRK.

WRK is used to store any variables or programs you create. The work directory listed here is located just below POLARP. This is shown on the directory tree on page 43. Working in WRK protects the programs in POLARP from accidentally being purged. It is also easier sorting through variables in a separate work directory than sifting through all your variables and programs in one big directory.

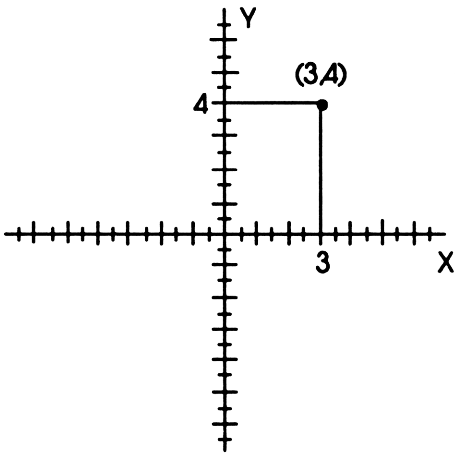
PATH { HOME GRAPHP POLARP}

SECTION TWO

EXAMPLES

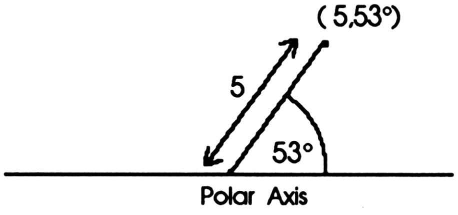
ABOUT POLAR COORDINATES

The most commonly used coordinate system is the Cartesian coordinate system. In two dimensions, it uses the vertical and horizontal components of a point along given axis. This is the easiest representation for most applications, but some equations can be simplified when represented in polar coordinates.



The Point (3, 4) in Cartesian Coordinates
Figure 2.1

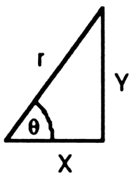
The two dimensional polar coordinate system represents points by their radial distance from the origin and the angle they make with the polar axis. The polar axis is normally the same as the X axis in the Cartesian coordinate system.



The Point (5, 53°) in Polar Coordinates
Figure 2.2

Figure 2.1 shows the point (3, 4) in Cartesian coordinates. The same point is (5, 53°) in polar coordinates. This is shown in figure 2.2.

The relationship between the Cartesian and Polar coordinates is as follows.



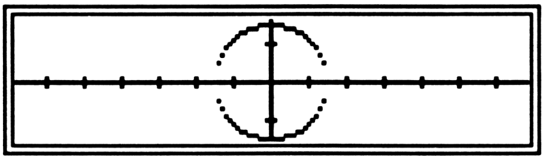
Polar to Cartesian	x=r*sin(θ)	Equation (2.1)
(r,θ)	y=r*cos(θ)	Equation (2.2)
Cartesian to Polar	r=√(x^2+y^2)	Equation (2.3)
(x,y)	$\theta=\tan^{-1}(x/y)$	Equation (2.4)

In general, equations with circular graphs are simpler in polar coordinates. For example, the equation of a circle in Cartesian coordinates is x^2+y^2=r^2. This describes all points that are the same distance, r, from the origin. In polar coordinate the equation is simply r=c, where c is any real number.

DRAWING A CIRCLE USING THE PLOT MENU

Try graphing a circle with a radius of 1.5 in Cartesian coordinates using the built in PLOT menu. First, you have to isolate y in the equation x^2+y^2=1.5^2. This gives you two roots,-√(2.25-x^2) and +√(2.25-x^2). If you recall, when an equation with an equal sign is stored in EQ, DRAW graphs both sides of the equation. If we want to graph both roots, we need to set them equal to each other and store the resultant equation in EQ. Remember, this is not a valid equation but only a trick we can use to graph two roots of an equation. First, purge PPAR if it exists. Then store -√(2.25-x^2)=+√(2.25-x^2) in EQ and draw its graph.

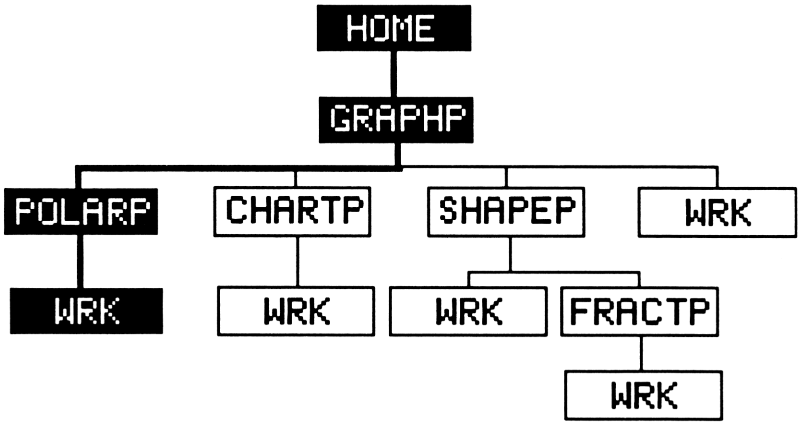
```
'PPAR [PURGE] '√(2.25-X^2) [ENTER] [ENTER] [CHS]
[=] [ENTER] [STEQ] [DRAW]
```



Graph of a circle using DRAW
Figure 2.3

DRAWING A CIRCLE USING THE POLAR GRAPHING MENU

Now lets try graphing the same circle using the polar graphing menu. To get to the polar graphing menu you have to start in the main graphing menu. You can get here by typing HOME [ENTER] GRAPH [ENTER] or pressing [END] if you are in a different graphing menu. Now enter the polar graphing menu by pressing the menu key labeled POLAR. This will put you in the work directory just below POLARP, the polar graphing program directory (See the directory tree below). You should also see the polar graphing menu. A description of its menu keys is given at the end of this chapter. If you leave this menu to use a 28S menu such as TRIG, you can always recall it by pressing CUSTOM.



This directory tree shows which directory you should be in.
Figure 2.4

You'll notice two menu labels titled DEG, for degrees, and RAD, for radians. These are the two types of angles that can be stored in APAR, the polar graphing parameters. One of these labels will have a box after its name, indicating it is the active polar graphing angle mode. The beginning and ending angles stored in APAR will be evaluated as degrees or radians depending on this angle mode. You can change modes by pressing the label without the box. A box will then appear after that menu label to show it is the new mode. APAR, the polar graphing parameters, are defined as follows.

{ starting angle(any real number), ending angle (any real number), angle mode (RAD or DEG)}

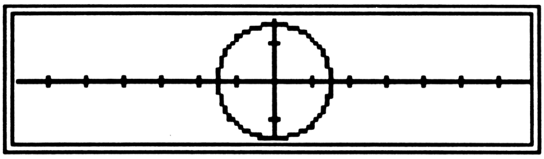
The starting and ending angles define the range for graphing the equation stored in EQ, while the angle mode indicates whether the starting and ending angles are in degrees or radians. The contents of APAR can be recalled by pressing APAR in the custom menu.

When you set the starting angle, ending angle, or polar graphing angle mode, you change the contents of APAR. But what will happen if APAR doesn't exist. If you haven't defined APAR in the work directory the default value,{ 0 360 DEG }, stored in the main graphing directory, GRAPHP, will be used. For example, if you wanted to graph a function in radians you would press RAD. A box will appear after its name and the new polar graphing mode, RAD, will be stored in APAR. If APAR doesn't exist in the work directory, the default starting and ending angles will be used. Its new value,{ 0 360 RAD }, will be stored in APAR.

Getting back to our example, a circle with a radius of 1.5 is the set of all points that are 1.5 measurements away from its center. The polar equation for this circle is simply r=1.5. Since there are 360 degrees in a circle, we need to graph our polar equation from 0 to 360 degrees. Store these values in EQ and APAR and press the menu key for ADRW.

1.5 STEQ 0 AMIN 360 AMAX
DEG (From the custom Menu) ADRW

The display will be cleared and a circle will be drawn, after which, the digitizing keyboard is activated. Appendix A explains how to digitize points on the display.



Graph of a circle using ADRW
Figure 2.5

INDEP and RES perform the same function for polar graphing as they do in the PLOT menu. RES sets the resolution, or step size for the main program loop, and INDEP sets the independent variable.

NOTE

There is a chance that ADRW will not clear the display or enter digitizing mode. This means flag 17 is set. Normally, flag 17 is clear so ADRW can be used as a menu driven program. The only time you would want it set is if you were using ADRW as a subprogram (See page 59). If this happens, you can stop the program and manually clear flag 17 or wait for ADRW to finish running, where it will clear flag 17 itself. Then try again.

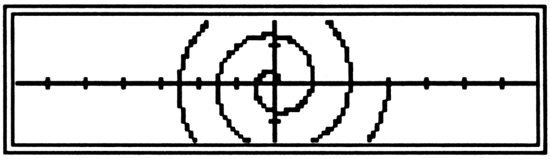
MORE EXAMPLES

Now that you've become familiar with the polar graphing menu you can graph more complicated functions. For example, the graph for the equation 2.5 is an endless spiral. Try graphing this function from 0 to 1080 degrees.

r=T/360 Equation (2.5)

Store T in 'EQ' and set T as the independent variable. Then, from the polar graphing menu, set the polar graphing angle mode to degrees, the starting angle at 0, and the ending angle at 1080.

'T' ENTER ENTER 360 ÷ STEQ INDEP DEG
0 AMIN 1080 AMAX ADRW



Graph of r=θ/360
Figure 2.6

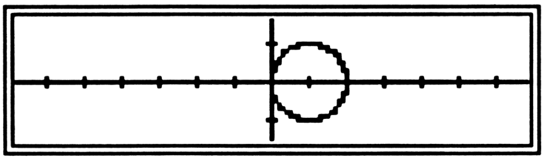
The sine and cosine functions can create a wide variety of different polar graphs. For instance, when graphed from 0 to 360 degrees, equation 2.6 will draw a circle centered at (1, 0) while equation 2.7 graphs two circles, one centered at (1, 0) and the other centered at (-1, 0). Can you see why? The graphs of both these functions and the keystrokes you need to enter are given in figures 2.7 and 2.8.

r=2*COS(T)	Equation (2.6)
r=2*ABS(COS(T))	Equation (2.7)

'2*COS(T)' STEQ

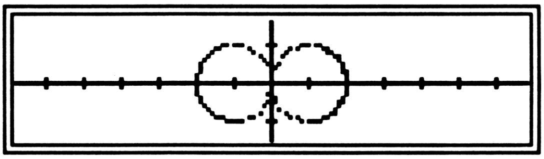
0 XMIN 2 ENTER π × →NUM XMAX

RAD (From the custom menu) DRAW



Graph of 2*COS(T)
Figure 2.7

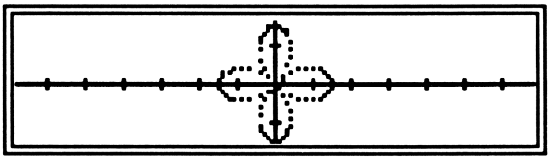
'2*ABS(COS(T))' STEQ DRAW



Graph of 2*ABS(COS(T))
Figure 2.8

Changing the period of equation 2.7 (multiplying θ by any real number) will change the number of petals on the graph. Try graphing r=1.5*ABS(COS(2*θ)) from 0 to 2π radians.

'1.5*ABS(COS(2*T))' STEQ NORM

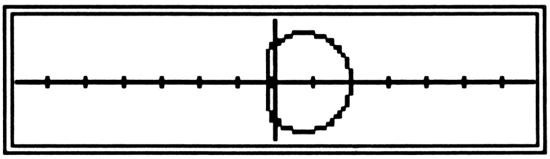


Graph of 1.5*ABS(COS(2*T))
Figure 2.9

The family of graphs for equation 2.5 are called limacons. The shape of the graph depends on the value of a. If a is equal to 1 the graph is a cardioid. The name seem quite appropriate since it looks like a heart. Try graphing 1+COS(T) from 0 to 2π radians. Its graph is shown in figure 2.10

r=1+a*COS(T) Equation (2.5)

'1+COS(T)' STEQ NORM



Graph of 1+COS(T)
Figure 2.10

Keeping PPAR the same, try graphing equation 2.5 for a <1, a>1.

ADRW AS A SUBPROGRAM

If flag 17 is clear, ADRW clears the display, draws the axis, graphs the polar equation in EQ, and activates the digitizing keyboard. It acts as a menu driven graphing program. But what if you want to incorporate ADRW as a subroutine in a larger program? Chances are, you won't want the display cleared and you probably won't want the digitizing keyboard activated.

When used as a subprogram, you need to set flag 17 before calling ADRW. Then ADRW will draw a polar graph without clearing the display or entering digitizing mode. It also resets flag 17 so it will always be ready when you want to use it as a menu driven program.

THE POLAR GRAPHING MENU

Menu Key	Operation
APAR	The value of APAR, the polar graphing parameters, are returned to the stack. They read { starting angle, ending angle, angle mode}.
AMIN	The real number from level one is stored in the first position of APAR. This number represents the angle ADRW starts graphing from.
AMAX	The real number from level one is stored as the second object in APAR. This number represents the angle ADRW stops graphing from.
DEG DEG■	The polar graphing parameters, APAR, are in degrees when a box appears after this label. If it doesn't have a box after its name you can activate it just by pressing its menu key.
RAD RAD■	The polar graphing parameters, APAR, are in radians when a box appears after this label. If it doesn't have a box after its name you can activate it just by pressing its menu key.
ADRW	This is the menu driven polar graphing program. Using the parameters in APAR, it graphs the polar equation in EQ.
ADVAN	This will create the advanced graphing menu from chapter six.
END	END will leave the polar graphing menu and put you in the main graphing menu described on page 21.

CHAPTER THREE

CREATING PIE CHARTS

SECTION ONE

REFERENCE SECTION

INTRODUCTION

This is the reference section for the chart drawing programs. These programs allow you to draw a pie chart for the data stored in the statistical array ΣDAT. You also have the option of labeling each section with small or large print. The small print is the size used in the menu labels while the large is the size the 28S uses for its display.

Before starting this chapter you should flip through section two, the example section. It will give you an idea of what to expect from these programs. Then, if you like what you see, you can jump right in, but be sure you've entered the programs from chapter one first.

Most of the programs in this chapter are stored in CHARTP, the chart drawing program directory. You must create this directory before storing any programs (where would you put them?). Go to GRAPHP, the main graphing directory, and create CHARTP by entering, HOME GRAPHP ENTER CHARTP CRDIR. See the directory tree on page 65. Now you're ready to enter all the programs from this chapter.

A table of all the programs you'll need for this chapter is given below. An asterisk (*) after the page number indicates it is listed in a different chapter. Flip to that page number and check whether you've already keyed that program in.

PROGRAM TABLE

Program	Page	Bytes	Program	Page	Bytes
ADRW*	30	275.0	PLCE	75	61.0
CHART	64	37.5	PLCU	76	294.5
CHARTP	65	18.0	PUTP*	45	33.5
CLCD?	36	52.5	PUTπ	77	72.0
CONT	66	252.5	RADIUS	78	45.5
CRSM	67	61.5	SMP	79	147.0
CTLST	68	74.5	SM	80	14.0
DGTZ?	40	47.5	SPOT	81	126.5
ENd	69	16.0	STLST	82	74.5
LGP	70	59.0	WRK	83	15.0
LARGE	71	17.0	πDRW	84	251.0
LINE	72	178.5	πPAR	87	26.5
NONE	73	16.0	πVAL	88	120.5
NONP	74	18.5			

NOTE

There are several programs in this book with the same name. Just because two chapters contain the same name in their program tables doesn't necessarily mean they are the same program. They could be two very different programs stored in two different directories.

For example, chapters two through five all have listings for STLST. STLST is used to create a custom graphing menu. Since every chapter creates its own custom menu, each one will have its own unique version.

Create the Chart Drawing Menu

CHART(20914)

```
« CHARTP Start »
```

SUMMARY

CHART puts you into the work directory, WRK, just below CHARTP and creates the custom pie chart menu. See the directory tree on the next page.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list stored in CTLST and the second object in πPAR are used to create the pie chart menu.
OUTPUT	MEMORY	Directory control is given to the work directory, WRK subordinate to CHARTP. This is shown in the directory tree on the next page. The custom chart menu is also created.
UTILITIES		Start

Chart Drawing Program Directory

CHARTP(2435)

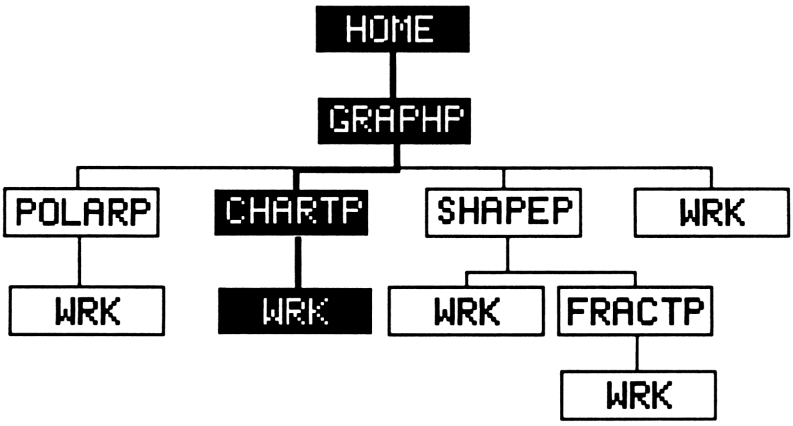
Directory

SUMMARY

This is the directory where all the programs that are specific to drawing pie charts are stored. Storing the pie chart programs in their own directory creates a sense of order, like chapters in a book.

Some programs used in this chapter are stored in GRAPHP rather than CHARTP. This is because they are used by graphing procedures in other directories. The programs in GRAPHP can be accessed by any program in or subordinate to GRAPHP. Thus a program in CHARTP can use LINE as well as a program in SHAPEP, since it is in stored in GRAPHP. A program in SHAPEP can't use πDRW though, because it is stored in CHARTP, which isn't in SHAPEP's path. The directory tree below illustrates this.

PATH { HOME GRAPHP }



Continue Creating List of Small Print Graphics

CONT(850225)

```
« LCD→ 411 548 SUB
NOT 1 11 FOR I DUP {
2 6 10 14 18 25 29
33 37 41 56 } I GET
DUP 3 + SUB 1 CHR
DUP + DUP + XOR SWAP
NEXT DROP 11 →LIST
'SMPR' STO Start »
```

SUMMARY

CONT is part of the custom menu created by CRSM. Pressing the menu key under CONT extracts the graphic strings for the letter O, numbers 1 through 9 and the percent character. These graphics are then put into a list and stored in SMPR. This list will be called on by πDRW when using small print in a pie chart, i.e. SM is displayed in the chart menu. See CRSM for more details.

PATH	{ HOME GRAPHP CHARTP }
INPUT LCD	The graphic strings for the letter O, numbers 1 to 9, and % in the custom menu are used.
OUTPUT MEMORY	A list of graphic characters are stored in 'SMPR' and the pie chart menu is created.
UTILITIES	Start

Create the List of Small Print Graphics

CRSM(56571)

```
« {O1234 56789 %
CONT ENd } MENU »
```

NOTE:The third character is the letter O, not the number 0.

SUMMARY

CRSM creates a custom menu that is used to create the graphics for small print in your pie charts. This custom menu contains the letter O, numerals 1 through 9, the character %, CONT, and END. The letter O is used rather than the number 0 because it is narrower. In the 28S menu, it takes three graphic characters to create the letter O and the numbers 1 through 9. In contrast, it takes five graphic characters to create the number 0.

Pressing CONT will create the graphics for small print and store them in 'SMPR' while pressing END returns you to the chart graphing menu. Pressing the menu keys under 01234 or 56789 will return that object to the stack. Pressing % will execute the percent function as if you had pressed the percent key.

PATH	{ HOME GRAPHP CHARTP }
INPUT	None
OUTPUT MEMORY	A custom menu is created.
UTILITIES	None

Constant Chart Drawing Menu List

CTLST(98589)

```
{ πPAR NONE RADIUS
∏VAL CRSM ∏DRW ADVAN
End }
```

SUMMARY

CTLST is used by STLST as a base for the pie chart menu. STLST puts CTLST on the stack and, depending on the last number in πPAR, replaces the second object with NONE, SM, or LARGE. Each represent a different sized print used when labeling each section in a pie chart. Finally, the resultant list is used by Start to create the pie chart menu.

PATH { HOME GRAPHP CHARTP }

Return to Graphing Menu

ENd(5761)

```
'Start'
```

SUMMARY

ENd can be found in the small print creation menu, the fractal editing menu, and the advanced graphing selection menu from chapter six. It returns you to the graphing menu you are in, in this case, the pie chart menu.

PATH		{HOME GRAPHP }
INPUT	MEMORY	The list stored in STLST
OUTPUT	MEMORY	The graphing menu that you were in is recreated.
UTILITIES		None

Increment Print Size to Large

LARGE (6101)

'PUTπ'

SUMMARY

LARGE is one of the three possible labels you'll see in the second position of the pie chart menu. Pressing this menu key causes the second object in πPAR, which is 3, to rap around to 1. This number "tells" πDRW what size print to use when labeling the percentage of each piece of the pie. 1 is for no print, 2 is for small print (the same size print as the menu labels), and 3 is for large print (the normal display print). In effect, this program causes the print size to rap around from large to none. A new custom menu is also created to display the new print size.

NONE, SM and LARGE are really the same programs. They all increment the print size, rapping around to NONE after reaching LARGE. See PUTπ for more details.

PATH	{ HOME GRAPHP CHARTP }
INPUT MEMORY	The number in the second position in πPAR.
OUTPUT MEMORY	The number in the second position in πPAR raps around to 1 and a new chart drawing menu is created.
UTILITIES	PUTπ

Large Printing Program

LGP(44494)

```
« SPOT 100 * IP →STR
"%" + PLCE »
```

SUMMARY

LGP is used by πDRW to put large labels on each section of a pie chart. The large print is the standard size the 28S uses for normal display. This size print is normally too large to use unless you're drawing a large pie chart with the extended drawing program, EDRW, from chapter six.

LGP first finds the appropriate spot to place the percentage of each section and then converts the real number on level one to a character string, which it places on the display.

PATH		{ HOME GRAPHP CHARTP }
INPUT	LEVEL TWO	A real number (This is the percentage of a given section that you'll be putting on the display).
	LEVEL ONE	A real number
	MEMORY	The second item in πPAR and the real number stored in EQ. EQ is created by πDRW.
	LCD	After the string from level one is displayed, the graphic string representing the LCD is used.
OUTPUT	LCD	The percentage of the current section is printed on that section.
UTILITIES		SPOT, PLCE

Draw a Line Between Two Points

LINE(1305046)

```
« OVER - PPAR LIST→
4 ROLL 3 DROPN 5
ROLLD - C→R 31 /
SWAP 136 / 3 PICK
C→R 4 ROLL / ABS
SWAP ROT / ABS MAX 1
MAX SWAP OVER / 4
PICK * → i « 1 + ROT
/ 1 SWAP START DUP
PIXEL i + NEXT DROP
» »
```

SUMMARY

LINE draws a line between the two points from level one and two. Be careful when using LINE. It will connect the two points no matter how far apart they are. If they are too far apart the program will get caught in what seems to be an infinite loop. LINE is actually drawing a line that is way off the screen. Imagine how long it would take to connect (0, 0) and (9.9E50, 9.9E50)!

PATH		{ HOME GRAPHP }
INPUT	LEVEL TWO	A complex number
	LEVEL ONE	A complex number
OUTPUT	LCD	A line is drawn on the display.
UTILITIES		None

Rap Print Size Back to None

NONE(5314)

$\boxed{\text{'PUT}\pi\text{'}}$

SUMMARY

NONE is one of the three possible labels you'll see in the second position of the pie chart menu. Pressing this menu key causes the second object in πPAR, which is 1, to increment to 2. This number "tells" πDRW what size print to use when labeling the percentage of each piece of the pie. 1 is for no print, 2 is for small print (the same size print as the menu labels), and 3 is for large print (the normal display print). In effect, this program causes the print size to increment from none to small. Finally, a new pie chart menu is created to display the new print size.

NONE, SM and LARGE are really the same programs. They all increment the print size, rapping around to NONE after reaching LARGE. See PUTπ for more details.

PATH		{ HOME GRAPHP CHARTP }
INPUT	MEMORY	The number in the second position in πPAR.
OUTPUT	MEMORY	The number in the second position in πPAR increments to 2 and a new chart drawing menu is created.
UTILITIES		PUTπ

Don't Print Any Percentages

NONP(3164)

```
« »
```

SUMMARY

NONP is simply an empty program. Every time πDRW draws a new section in a pie chart it evaluates one of the three programs, NONP SMP or LGP. These programs print the percentage of each section in the chart.

When you don't want any printing on a pie chart (1 is the second object in πPAR), πDRW will execute NONP when labeling each section. Because NONP is an empty program, none of the sections will be labeled with their percentage.

PATH	{ HOME GRAPHP CHARTP }
INPUT	None
OUTPUT	None
UTILITIES	None

Place Text on the Display

PLCE(157208)

```
« DUP SIZE 6 * LCD→
ROT 1 DISP LCD→ SWAP
→LCD 1 ROT SUB PLCU
»
```

SUMMARY

PLCE takes the string from level one and places it, pixel by pixel, on the display. The point from level two defines the bottom left corner of the string being printed. This allows you to put text anywhere on the display by specifying the value of the pixel you want to start as defined by the 28S's plot parameters PPAR.

PLCE first displays the string from level one on the top right corner of the display via the built in command DISP. It then returns the graphic string representing the display to level one and takes only the portion of this string that contains the graphics for what you're trying to place on the display. Finally, PLCU places the string on the display starting at the point that was input on level two.

PATH		{ HOME GRAPHP CHARTP }
INPUT	LEVEL TWO	A complex number
	LEVEL ONE	A string
	LCD	After the string from level one is displayed, the string representing the LCD is used.
OUTPUT	LCD	The string from level one is plotted on the display.
UTILITIES		PLCU

Place Utility

PLCU(2907379)

```
« 1 *W PPAR LIST→ 4
DROPN - C→R -31 /
SWAP -136 / 4 ROLL
C→R 4 PICK 8 * OVER
+ ROT 6 PICK SIZE 5
PICK * OVER + → ys
xs y1 y2 x1 x2 « x1
x2 FOR x DUP 2 137
SUB SWAP NUM R→B y1
y2 FOR y IF DUP
# 80h AND B→R THEN x
y R→C PIXEL END RL
ys STEP DROP xs STEP
DROP » »
```

SUMMARY

This program plots a graphic string on the display. Simply put a point on level two and the graphic string on level one. Starting at the point from level two, PLCU takes each character of the graphic string and draws it, pixel by pixel, on the display. It continues plotting each character until it reaches the end of the graphic string.

PATH		{ HOME GRAPHP CHARTP }
INPUT	LEVEL TWO	A complex number
	LEVEL ONE	A graphic string
OUTPUT	LCD	The graphic string from level one is drawn on the display, bottom to top, left to right.
UTILITIES		None

Put Size Number into ΠPAR

PUTπ(110563)

```
« 'πPAR' PCHK 2 DUP2
GET DUP 3 ≠ * 1 +
PUT Start »
```

SUMMARY

PUTπ increments the print size number (The second object in πPAR). If this number is greater than 3 it raps around to 1. πDRW uses this number to decide what size print should be used to label each section of the pie chart. Finally, PUTπ calls Start to create a new custom menu. The second label of this new menu will be either NONE, SM, or LARGE, depending on the number in the second position of πPAR.

PATH { HOME GRAPHP CHARTP }

INPUT MEMORY The number in the second position in πPAR.

OUTPUT MEMORY The number in the second position in πPAR is incremented. If it is greater than 3, it raps around to 1. A new chart drawing menu is also created.

UTILITIES Start

Define Radius of Pie Chart

RADIUS(28067)

```
« RE 1 'πPAR' PUTP »
```

SUMMARY

RADIUS puts the real number from level one into the first position in the pie chart parameters, πPAR. This number "tells" πDRW what radius to use when drawing a pie chart.

RADIUS will only use the real part of the number on level one. This number must be positive for πDRW, the pie chart drawing program, to work properly.

PATH		{ HOME GRAPHP CHARTP }
INPUT	LEVEL ONE	A real number
OUTPUT	MEMORY	The number from level one is put in the first position in the pie chart parameters, πPAR.
UTILITIES		PUTP

Increment Print Size to Large

SM(3719)

```
'PUTπ'
```

SUMMARY

SM is located in the chart drawing menu. Pressing this menu key causes the second object in πPAR, which is 2, to increment to 3. This number "tells" πDRW what size print to label the percentage of each piece of the pie. 1 is for no printing, 2 is for small printing (the same size print as the menu labels), and 3 is for large print (the normal display print). In effect, this program causes the print size to increment from small to large. A new custom menu is also create to display the new print size.

NONE, SM and LARGE are really the same programs. They all increment the print size. The size raps around to NONE after reaching LARGE. See PUTπ for more details on how this is done.

PATH		{ HOME GRAPHP CHARTP }
INPUT	<u>MEMORY</u>	The number in the second position in πPAR.
OUTPUT	<u>MEMORY</u>	The number in the second position in πPAR is incremented to 3. A new chart drawing menu is also created.
UTILITIES		PUTπ

Label each Section with Small Print

SMP(289021)

```
« SPOT 'SMPR' SWAP
DUP2 10 * IP 1 + GET
3 ROLLD 10 * FP 10 *
IP 1 + GET + 'SMPR'
11 GET + PLCU »
```

SUMMARY

Using SPOT, SMP Finds the appropriate spot to place the percentage of each section. It then converts the percentage to a graphic string and places it, in small print, on the pie chart. This small print is the same size the 28S uses for its menu keys. This size print can be used by many charts. In some cases though, it may be too large to use unless you're drawing an enlarged pie chart with the drawing program EDRW. This is illustrated in chapter 7.

PATH { HOME GRAPHP CHARTP }

INPUT <u>LEVEL TWO</u> A real number (This is the percentage you'll be putting on the display.
<u>LEVEL ONE</u> A real number
<u>MEMORY</u> The second item in the list πPAR and the real number stored in EQ (this is the radius of the pie chart).

OUTPUT <u>LCD</u> The percentage of the current section is plotted on the display.

UTILITIES SPOT, PLCU

Calculate the Spot to Place the Percentage

SPOT(260672)

```
« DUP2 OVER 2 / +
360 * EQ 2 / SWAP
R→C P→R (.45,.3)
(.45, 0) πPAR 2 GET 3
== * + - SWAP »
```

SUMMARY

SPOT finds the appropriate spot to place the percentage of each section. It is called on by both SMP, the smaller printing program, and LGP, the larger printing program. Because of the difference in size, the two types of print should be placed in different location. SPOT accommodates for the larger print size by adding (.45, 0) to the position the percentage will be printed at.

PATH		{ HOME GRAPHP CHARTP }
INPUT	LEVEL TWO	A real number
	LEVEL ONE	A real number
	MEMORY	The second item in the list πPAR and the real number stored in EQ. (This is the radius of the pie chart.)
OUTPUT	LEVEL THREE	The real number input on level one
	LEVEL TWO	A complex number representing the location that the percentage should be drawn.
	LEVEL ONE	The real number input on level two
UTILITIES		None

Create Starting Chart Graphing Menu List

STLST(104721)

```
« CTLST 2 { NONE SM
LARGE } πPAR 2 GET
GET PUT »
```

SUMMARY

The number in the second position in πPAR indicates which item from the list {NONE SM LARGE} STLST will use. STLST take this object and puts it into the second position in the list stored in CTLST. CTLST is always the same and serves as a base to build the custom chart drawing menu on. The final list will be used by Start to create a custom menu having either NONE, SM, or LARGE in the second position. This tells you, the user, which size print πDRW will use.

PATH		{ HOME GRAPHP CHARTP }
INPUT	MEMORY	The real number stored in the second position of πPAR.
OUTPUT	LEVEL ONE	A list that will be used to create a custom menu.
UTILITIES		None

Chart Drawing Work Directory

WRK(954)

Directory

SUMMARY

As you read through this text you'll notice that every chapter has its own work directory named WRK. This version is used to store any variables or programs you create while using the chart drawing menu. It is located just below CHARTP. This is shown on the directory tree on page 65. Working in WRK protects the programs in CHARTP, the chart drawing program directory, from accidentally being purged. It is also easier sorting through variables in a smaller work directory than sifting through all your variables and programs in one big directory.

PATH { HOME GRAPHP CHARTP}

Pie Chart Drawing Program

πDRW(1760699)

```
« 1.E50 DUP R→C AXES
 CLCD? RCLF STD DEG
πPAR LIST→ DROP {
NONP  SMP LGP } SWAP
GET → p « STEQ 17 SF
ADRW ΣDAT TOT /
ARRY→ 1  GET 0 1 ROT
START p EVAL + EQ
OVER 360 * R→C P→R
(0, 0) LINE  NEXT DROP
'EQ' PURGE  STOF
DGTZ? » »
```

SUMMARY

This is the program that does all the work. Using the statistical array in ΣDAT, πDRW draws a pie chart. It portions the pie among the entries in this array. For example, if ΣDAT equals

```
[ [4]
  [2]
  [14]]
```

Then πDRW will draw a pie chart with three sections. The first would be 20% of the chart. Likewise, the second and third section would be 10% and 70% of the chart.

The pie chart parameter list, πPAR, contains two objects. The first is the radius of the pie chart. The second, an integer from 1 to 3, indicates the current print size used to print the percentages of each section.

There are three possible print sizes. Size 1 will omit any printing, size 2 will use small print (the print used in the menu labels) and size 3 will use large print (the same print that the 28S uses for normal display). The current print size is displayed as the second menu key in the custom chart menu as either NONE, SM or LARGE.

πDRW, like most other root level drawing programs, uses the two utilities CLCD? and DGTZ?. When I say root level I mean it does the actual graphing. In contrast, the graphing programs in chapter six are advanced graphing programs. They call other procedures, like πDRW, to do the actual graphing for them.

Both CLCD? and DGTZ?check the status of flag 17 and, if it is clear, clears the display before graphing and digitize after graphing is complete. DGTZ? also clears flag 17 after checking its status. These two subprograms allow you to use πDRW directly from the custom menu or as part of a larger program.

When using this program from the menu (it is assumed that flag 17 is cleared) the display will be cleared, a pie chart will be drawn, and you will enter digitizing mode. On the other hand, if you want to use it as a subprogram, set flag 17 before executing πDRW. This prevents the display from being cleared or having execution halted for digitizing. This is exactly what the graphing programs from chapter six do.

> NOTE: If πDRW doesn't clear the display or allow you to digitize, flag 17 is set. If this happens you can stop the program and clear flag 17 or wait until it is done running. Either way, flag 17 will be cleared. Then just try it again.

PATH		{ HOME GRAPHP CHARTP }
INPUT	<u>MEMORY</u>	The list in πPAR and the array stored in ΣDAT as well as flag 17 are used.

	KEYBOARD	If flag 17 is clear the digitizing keyboard is activated. See appendix A for an explanation on digitizing.
OUTPUT	MEMORY	The location of the axis defined be 'PPAR' is shifted off the display, preventing them from being drawn.
	LCD	If flag 17 is clear the display is cleared. The the pie chart defined by πPAR and ΣDAT is drawn.
	STACK	If flag 17 was clear before running the program any objects left by digitizing will be on the stack.

UTILITIES ADRW, CLCD?, DGTZ?, LINE, NONP, SMP, LGP

Default Pie Chart Parameters

πPAR(8112)

```
{ 1.5 1 }
```

SUMMARY

The default pie chart parameters are defined in GRAPHP. This version will be used as a default when it isn't already defined in the work directory. Of course any version of πPAR created in the work directory will take precedence over this one.

The second object from this list, 1, is used as the default print size. This should become obvious when you first enter the chart graphing menu. **NINE** will be displayed as the second menu label. It was πPAR that "told" TLT, the program that created the custom menu, which size to label this key.

PATH { HOME GRAPHP }

Calculate the Value of any Section

πVAL(304537)

```
« R→P IM 360 MOD
SWAP R→P IM 360 MOD
- IF DUP SIGN 1≠
THEN 360 + END 360 /
RCLΣ CNRM * »
```

SUMMARY

This program allows you to find the area of any section in a pie chart. While running πDRW, digitize the two lines that make up the section or combination of sections counter clockwise. Then, running πVAL with both points on the stack will return the area of that section.

πVAL uses ΣDAT in computing this value, so changing this array after digitizing your points will cause πVAL to give you an incorrect answer.

PATH		{ HOME GRAPHP CHARTP }
INPUT	LEVEL ONE	A complex number
	LEVEL TWO	A complex number
OUTPUT	LEVEL ONE	A real number
UTILITIES		None

SECTION TWO

INTRODUCTION

The pie chart menu allows you to represent data stored in the statistical array ΣDAT as a pie chart. You also have the option of labeling each section with small or large print. The small print is the same size used in the menu labels while the large is the same print the 28S uses.

πDRW, the pie chart drawing program, and its supporting programs are incorporated into an easy to use custom menu. To get to the pie chart menu you have to start in the main graphing menu. Enter HOME ENTER GRAPH ENTER or pressing END if you are in a different graphing menu. Now enter the pie chart menu by pressing the menu key labeled CHART. This will put you in the work directory just below CHARTP (See the directory tree below). You should also see the pie chart menu. A description of each menu key is given at the end of this chapter. If you leave this menu to use a 28S menu such as TRIG, you can always recall it by pressing CUSTOM.

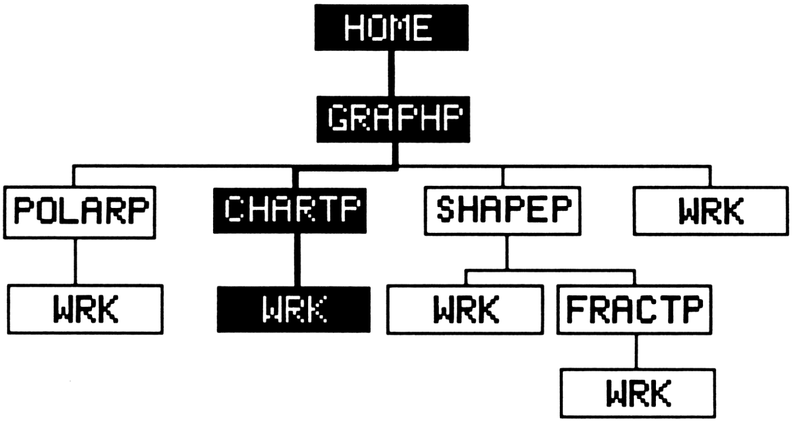


Figure 3.1

The next few pages will demonstrate the basic outline to follow when drawing a pie chart. From storing the data to finding the area of any section in the chart, each step is described with an example.

STORING DATA

Before you can draw a pie chart you need to store your data in an array. πDRW portions one section for each number in the one dimensional statistical array stored in 'ΣDAT'. You can enter your data using the commands from the 28S statistical menu, STAT . Σ+ adds the number from level one to ΣDAT while Σ- subtracts the last number from ΣDAT and puts it on level one.

The data we'll use for our example is [[195][447][362][60]]. Try storing this array into ΣDAT. This can be done several ways. You can enter the array on the stack and store it directly into 'ΣDAT' or you can clear any existing ΣDAT and use Σ+ to add each element individually. I'll use the second method.

```
CLΣ 195 Σ+ 447 Σ+ 362 Σ+ 60 Σ+
```

CHOOSING A PRINT SIZE

Now we must decide whether or not we want to label the percentage of each section and, if so, the size print we would like. There is the large print that the 28S uses for normal display, or the smaller print that is used for menu key labels. Most charts can only handle small printing. Unless the chart is drawn on several display screens, the large print is so big that the label from one section will overlap others. Only an extended chart (a chart drawn on more than one display screen) created using EDRW from chapter six should have large printing. Since this is the first example, we will omit any labels on our pie chart.

The second menu key displays the current print size. It will read either NONE, SM, or LARGE. The default value, NONE, should be displayed. If not, press the second menu key until it reads NONE. Pressing this key causes the print size to increment and changes the second menu key label to display the new print size. Pressing the menu label when it reads LARGE will cause the print size to rap around to NONE .

CHOOSING A RADIUS

Next we must choose a radius for our chart. An extended pie chart drawn on several display screens might need a radius of 4.5 or 7.5 while a chart without any print can normally be drawn on one display screen using a radius of 1.5 or less. To enter a radius, put a real number on level one and press the menu key under RADIU . For our example, set the radius to 1.5. After pressing RADIU the first number in the list stored in πPAR should be 1.5. This list is used by πDRW when drawing a pie chart.

1.5 RADIU

THE PIE CHART PARAMETERS

πPAR, the pie chart parameter list, reads { radius, print size }. The radius is just the distance from the center of the chart to any point on the circle. It can be any real number. The print size is the size characters used when labeling each section of the pie. It will be either 1 for no print, 2 for small print, or 3 for large print.

πPAR can be recalled by pressing its menu key in the pie chart menu. If πPAR doesn't exist in the work directory, the default value stored in the main graphing directory will be returned. This default value is { 1.5 1 }. It is use by the pie chart programs when πPAR doesn't exist in the pie chart work directory.

DRAWING A PIE CHART

With ΣDAT and πPAR defined you're ready to draw a pie chart with a radius of 1.5 where the percentage of each section is not printed. ΣDAT and πPAR should contain

```
ΣDAT        [[195][447][362][60]]
πPAR        {1.5, 1}
```

Pressing ADRW will draw the pie chart for the data you stored in ΣDAT. A section is drawn for each element, from last to first.

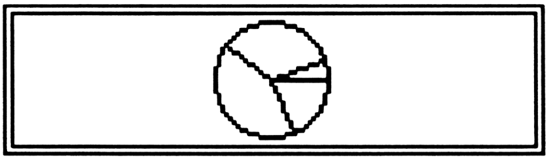


Figure 3.1

NOTE

There is a chance that ADRW will not clear the display or enter digitizing mode. This means flag 17 is set. Normally, flag 17 is clear so ADRW can be used as a menu driven program. The only time you would want it set is if you were using ADRW as a subprogram (See page 95). If this happens, you can stop the program and manually clear flag 17 or wait for ADRW to finish running, at which time it will clear flag 17 itself. Then just try again.

FINDING THE AREA OF ANY SECTION

After drawing the pie chart, the 28S enters digitizing mode. There is an added feature in the pie chart menu. You can find the value of any section or combination of sections in the pie by digitizing its boundaries clockwise, pressing ON to exit digitizing mode, and running πVAL. The approximate value of the area is returned to level one.

As an example, try finding the combined value the first two segments of the chart we drew earlier. If you already left digitizing mode you will have to draw the pie chart again.

Digitize any point on the lower border of the first section and the opposite border of the second. Any point along the line will work, but points further from the center will give a more accurate estimate. Press ON to exit digitizing mode and press the custom menu key labeled πVAL. The estimate of 423.03 is very close to the actual value of (60+362)=422. You may get a slightly different value depending on the points you digitized.

LABELING THE PERCENTAGE OF EACH SECTION

You probably won't label the percentage of each section when drawing a pie chart on one display screen. But what if you're drawing a chart on several displays. This can be done using EDRW, the extended drawing program from chapter six.

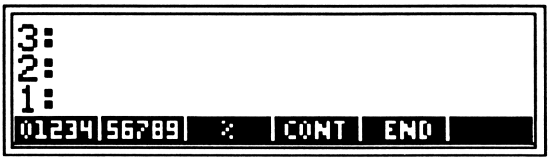
The 28S displays data using characters that are eight pixels high and five pixels wide. It displays these characters on a four line by 23 column display screen. This gives you 92 different position in which you can display a character. This is fine for most uses, but what if you want to display a character anywhere on the screen?

Each section in a pie chart normally has very little area. With such little space to work in and only 92 different possible positions, it would be impossible to label each section with its percentage of the pie. If, however, we could break the four line, 23 column restriction and put characters anywhere on the screen, we might be able to squeeze everything in.

The display screen is made up of 137 by 32 pixels. We could have 4384 different possible positions if we were able to specify which pixel we want a character to start at. This is exactly what the subprogram PLCU does. It allows us to places a graphic string anywhere on the display screen by plotting it, pixel by pixel. By first converting a number to a graphic string and using PLCU to draw it on the display, we can easily label the sections of a pie chart.

We have two different sizes to choose from. We can label the sections with small (the same size as the menu labels) or large (the same size the 28S uses on its display) print. The small print is fine when using two display screens while the large print can be used for three or more displays.

Before using the small print on your chart you have to create the graphics for each number. Pressing the menu key labeled CRSM will display the small print creation menu.



The first three menu labels, O1234, 6789, and % will be used to create the graphics. The fourth, CONT, will extract the graphics for each character in the first three labels of the custom menu and put you back in the pie chart menu. END gives you the option to exit this menu and recall the pie chart menu without creating any graphics.

The first character in this menu is the letter O not the number 0. The letter O is used rather than the number 0 because it is narrower when in the 28S menu. It takes three graphic characters to create the letter O and the numbers 1 through 9. In contrast, it takes five graphic characters to create the number 0.

You must press CONT to create the small print graphics. This extracts the graphics and stores them as a list in 'SMPR'. This list is used when putting small labels on a pie chart. If you decide you do not want to create this list, END will return you to the pie chart menu. Pressing the menu keys under O1234 or 56789 will return that object to the stack. While pressing % will execute the percent function as if you had directly pressed the percent key.

If you are using the large print you will not have to define the graphics before hand. Instead, πDRW will periodically flash the percentage on the display, where it extracts the graphics it needs. Large print is easier to use and doesn't consume memory space storing graphics, but is limited by its awkward size.

A PIE CHART USING SMALL PRINT

Lets draw a pie chart labeling each section with small print. Using the same data as the last example, subtract the last point from ΣDAT. The statistical array should now be [[195][447][362]]. Next, create the graphics for the small printing. Finally, press the second menu key until it reads SM and run πDRW. Notice how the labels just barely fit on each section. As a rule, pie charts drawn on one display should not use labels. Only chart drawn on several screen using EDRW from chapter six have the room needed to label the percent of each section. An example of an extended pie chart is given in chapter seven.

Σ- DROP CRSM CONT NONE πDRW

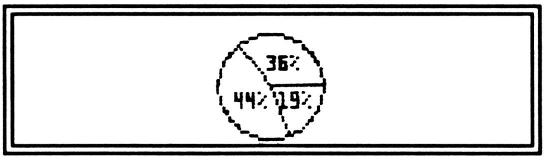


Figure 3.2

ΠDRW AS A SUBPROGRAM

If flag 17 is clear, πDRW clears the display, draws a pie chart, and activates the digitizing keyboard. It acts as a menu driven program. But what if you want to incorporate πDRW as a subroutine in a larger program? Chances are, you won't want the display cleared and you probably won't want the digitizing keyboard activated.

When used as a subprogram, you need to set flag 17 before calling πDRW. Then, πDRW will draw a pie chart without clearing the display or entering digitizing mode. It will also reset flag 17 so it will always be ready when you want to use it as a menu driven program.

PIE CHART MENU

Menu Key	Operation
πPAR	The pie chart parameters are { radius, print size }. The radius is just the radial distance from the center to the edge of the pie chart. The print size is the size print used to label each section of the pie chart. 1 omits any labeling, 2 uses small printing, and 3 uses larger printing.
NONE SM LARGE	The second menu key tells you, the user, the current print size. Pressing this key causes the size to increment, rapping around to NONE after reaching LARGE. The number in the second position of πPAR will also change to 1, 2, or 3 to reflect the new print size.
RADIU	You can specify the size of the pie chart using this program. Pressing this menu key puts the number from level one into the first position in πPAR.
πVAL	This handy program allows you to find the value of any number of sections in a pie chart. Simply digitize the boundaries clockwise and press ON. Then, running πVAL with the two points on the stack will return the value between the digitized boundaries.
CRSM	Pressing this menu key will display the menu used to create the graphics for small labels. Be sure you do this before trying to graph a pie chart whose labels are in small print.

πDRAW	This is the pie chart drawing program. It draws a circle whose radius is given as the first object in πPAR. It then sections the pie among the elements in the one dimensional statistical array stored in ΣDAT. Finally, using the second object in πPAR, each piece of the pie is labeled with small print, large print, or left unlabeled.
ADVAN	As in most other menus from this text, this key will create the advanced graphing menu. The programs in this menu are in chapter six.
END	The last menu key will exit the pie chart menu and return you to the main graphing menu.

SMALL PRINT CREATION MENU

Menu Key	Operation
01234	Pressing this menu key returns the string "01234" to the stack. This menu label is used by CONT to define 5 by 3 pixel characters.
56789	Pressing this menu key returns the number 56789 to the stack. This menu label is used by CONT to define 5 by 3 pixel characters.
%	Pressing this key will preform the percent function as if you have pressed the percent key.
CONT	CONT extracts the graphics for the characters in the first three menu labels and puts them in a list stored in 'SMPR'.
END	Pressing END returns you to the pie chart menu.

CHAPTER FOUR

DRAWING SHAPES

SECTION ONE

REFERENCE SECTION

INTRODUCTION

This is the reference section for the shape drawing programs. These programs allow you to draw a variety of polygons and stars. Before starting this chapter you should flip through section two, the example section. It will give you an idea of what to expect. Then, if you like what you see, you can jump right in, but be sure you've entered the programs from chapter one first.

Most of the programs in this chapter are stored in SHAPEP, the shape drawing program directory. You must create this directory before storing any programs (where would you put them?). Go to GRAPHP, the main graphing directory, and create SHAPEP by entering, HOME GRAPHP ENTER SHAPEP CRDIR. See the directory tree on page 110. Now you're ready to enter all the programs from this chapter.

A table of all the programs you'll need in this chapter is given on the next page. An asterisk (*) after its name indicates it is listed in a different chapter. Flip to the indicated page number and check whether you've already keyed that program in.

PROGRAM TABLE

Program	Page	Bytes	Program	Page	Bytes
CLCD?*	36	52.5	$DRW	106	51.5
CNCT	102	87.5	$HAPE	109	37.5
DGTZ?*	40	47.5	$HAPEP	110	18.0
LINE*	72	178.5	$HAPL	111	143.5
POINT$	103	43.0	$PAR	112	37.0
PUTP*	45	33.5	TLT	113	78.5
RADIU$	104	65.5	THETA	114	59.5
REV	105	40.0	WRK	115	15.0

NOTE

There are several programs in this book with the same name. Just because two chapters contain the same name in their program tables doesn't necessarily mean they are the same program. They could be two very different programs stored in two different directories.

For example, chapters one through five all have listings for TLT. TLT is used to create each custom graphing menu. Since every chapter creates its own custom menu, each one will have a different version of TLT.

Connect a List of Points

CNCT(251388)

```
« 1 *W 1 DO GETI →
p1« GETI p1 SWAP
LINE » 1 - UNTIL 46
FS? END DROP2 »
```

SUMMARY

It is often easier to represent a figure as a series of end points. Connecting these points trace out the figure in a dot to dot fashion. This is exactly what CNCT does.

CNCT will connect a list of points on the display. Simply put any list of complex numbers on level one and CNCT will draw a line from one point to the next. Be sure the points aren't too far out of the range of the display screen. If the display range is (-6.8,-1.5) to (6.8, 1.6), connecting the points (9.9E50, 9.9E50) and (0, 0) would put you in a near endless loop.

PATH		{ HOME GRAPHP SHAPEP}
INPUT	LEVEL ONE	A list containing at least one complex number.
OUTPUT	LCD	Each point from the list input on level one is connected in order on the display
UTILITIES		LINE

Number of Points in Shape

POINTS(22791)

```
« 2 'SPAR' PUTP »
```

SUMMARY

POINTS puts the integer from level one into the second position of the shape parameters, SPAR. This integer specifies the number of end points on the shape you're drawing. For instance, a triangle has three end points while a five pointed star has five points. Likewise, a pentagon also has five points.

This program will accept any object as input. Be careful not to put anything other than an integer on level one. If you do, you'll get an error when running SDRW.

PATH		{ HOME GRAPHP SHAPEP }
INPUT	LEVEL ONE	Any integer
OUTPUT	MEMORY	The integer from level one is put into the second position of SPAR.
UTILITIES		PUTP

Radius of Shape

RADIUS(126795)

```
« ABS 'SPAR' PCHK 1
DUP2 GET IM 4 ROLL
SWAP R→C PUT »
```

SUMMARY

RADIUS puts the real number from level one as the real part of the complex number in the first position of SPAR, the shape parameters. This number "tells" SDRW, the shape grapher, what radius to use when creating a shape. Each time it makes a revolution, SDRW will connect a number of points that are a distance R away from the origin. It keeps on connecting points until it completes the total number of revolutions SPAR "tells" it. In a nutshell, this program controls the size of your shape. The larger the radius, the larger the shape.

PATH		{ HOME GRAPHP SHAPEP }
INPUT	LEVEL ONE	Any real number
OUTPUT	MEMORY	The real number from level one is put into the real part of the complex number in the first position of SPAR.
UTILITIES		PCHK

Number of Revolution in Shape

REV(17148)

```
« 3 'SPAR' PUTP »
```

SUMMARY

REV puts the integer from level one into the third position in SPAR, the shape parameter list. This integer "tells" SDRW the number of revolutions to make. Each time it makes a revolution SDRW will connect a number of points. It keeps on connecting points until it's completed the total number of revolutions.

The number of revolutions can drastically change the shape you're drawing. For instance, if you want to create a shape that has five points and you do it in one revolution, you'll end up with a pentagon. On the other hand, if you decided you wanted to try drawing your shape in two revolutions, you'd end up with a five pointed star.

Be careful when using REV. It doesn't check your input. This means you can input any object. You're sure to get an error when running SDRW if, by accident, you input anything other than an integer.

PATH { HOME GRAPHP SHAPEP }

INPUT LEVEL ONE Any integer

OUTPUT MEMORY The integer from level one is put into the third position in SPAR.

UTILITIES PUTP

Shape Drawing Program

SDRW(35934)

```
« CLCD? SHAPL CNCT
DGTZ? »
```

SUMMARY

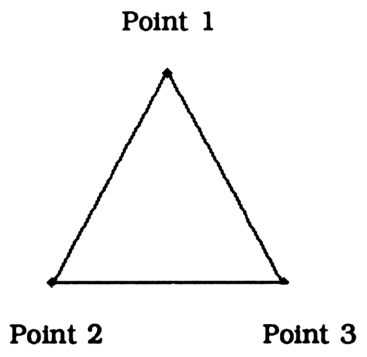
SDRW draws a shape that is defined by SPAR, the shape parameters. SPAR read as follows.

{ (radius, starting angle), number of points, number of revolutions }

The first number in SPAR is complex. The real part represents the radial distance of every endpoint from the origin. The complex part is the angle, in degrees, that the first point makes with the X axis. It is, more or less, a polar coordinate. This is the spot where the first point is plotted. All other points differ only by the angle (the complex part of this number).

SDRW starts at the first point and circles the origin. The third number in SPAR is the number of revolutions SDRW will make. While revolving around the center, it plots a number of equally spaced points around the circle. The total number of points are specified by the second number in SPAR. Finally, these points are connected together to draw a shape.

As an example, if you stored {(1.5, 90),3, 1} in SPAR, SDRW would connect three points that have a radial distance of 1.5 from the center and are equally spaced apart. The first point would be rotated 90° from the X axis. This is the description of the equilateral triangle shown on the next page.



SDRW, like most other root level drawing programs uses the two utilities CLCD? and DGTZ?. A root level graphing program does the actual graphing. In contrast, the graphing programs in chapter six are advanced graphing programs. They call other procedures, like SDRW, to do the graphing for them.

Both CLCD? and DGTZ?check the status of flag 17 and, if set, clear the display before graphing and digitize after graphing is complete. DGTZ? also clears flag 17 after checking its status. These two subprograms allow you to use SDRW directly from the custom menu or as part of a larger program.

When using this program from the menu (it is assumed that flag 17 is cleared) the display will be cleared, a shape will be drawn, and you will enter digitizing mode. On the other hand, if you want to use it as a subprogram, have the calling program set flag 17 before executing SDRW. This prevents the display from being cleared or having execution halted for digitizing. This is exactly what the advanced graphing programs from chapter six do.

NOTE

If SDRW doesn't clear the display or allow you to digitize, flag 17 is set. You can stop the program and clear flag 17 or wait until SDRW is done running. Either way, flag 17 will be cleared. Then just try again.

PATH		{ HOME GRAPHP SHAPEP }
INPUT	MEMORY	The list in SPAR and PPAR as well as flag 17 are used.
	KEYBOARD	If flag 17 is clear the digitizing keyboard is activated. See appendix A for an explanation on digitizing.
OUTPUT	LCD	If flag 17 is clear the display is cleared. The shape defined by SPAR is drawn.
	STACK	If flag 17 is clear before running, any objects left from digitizing.
UTILITIES		CLCD?, SHAPL, CNCT, DGTZ?

Create Shape Drawing Menu

SHAPE(20761)

```
« SHAPEP Start »
```

SUMMARY

SHAPE puts you into the work directory, WRK, and creates the shape drawing menu. You'll probably notice, if you haven't already, that every graphing menu has its own program directory, the directory with the programing specific to that topic, its own work directory, titled WRK, its own graphing menu, and a command to enter that menu.

Except for the last letter, the commands to enter the graphing menus all have the same name as their program directories. For example, SHAPE puts you in the work directory, WRK, just below SHAPEP. Likewise, FRACT puts you into the shape drawing work directory, WRK, just below FRACTP.

PATH		{ HOME GRAPHP SHAPEP }
INPUT	MEMORY	STLST, the list in the SHAPEP directory
OUTPUT	MEMORY	Directory control is given to the work directory, WRK subordinate to SHAPEP. This is shown in the directory tree on the next page. The custom shape graphing menu is also created.
UTILITIES		Start

Shape Program Directory

SHAPEP(2367)

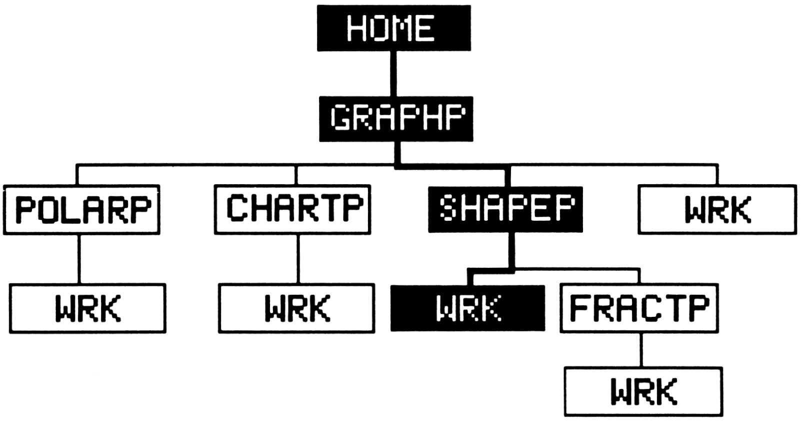
Directory

SUMMARY

This is the directory where all the programs that are specific to drawing shapes are stored. Storing the shape drawing programs in their own directory creates a sense of order, like chapters in a book.

Some programs that are used for drawing shapes are stored in GRAPHP rather than SHAPEP. This is because they are used by graphing procedures in other directories. Putting these programs in GRAPHP allows access to any program in or subordinate to it. Thus a program in CHARTP can use LINE as well as a program in SHAPEP, since it is in stored in GRAPHP. A program in CHARTP can't use SDRW though, because it is stored in SHAPEP, which isn't in CHARTP's path. The directory tree below illustrates this.

PATH { HOME GRAPHP }



Create a Shape List

SHAPL(765293)

```
«SPAR LIST→ DROP 1
*W 360 * OVER / RCLF
DEG → d a fg « DUP
P→R SWAP 1 d START 0
a R→C - DUP P→R SWAP
NEXT DROP d 1 +
→LIST fg STOF » »
```

SUMMARY

Using the list stored in SPAR, SHAPL creates a list of points that are equally spaced around a circle. Connecting these points will draw a symmetric shape. The list in SPAR is used as follows.

{ (radius, starting angle), number of points, number of revolutions}

SHAPL starts at the polar coordinates of the complex number from SPAR. While revolving around the origin, it continually adds equally spaced points to a list. The second object in SPAR specifies the total number of points. The third object is the number of revolutions made while generating points.

PATH		{ HOME GRAPHP SHAPEP }
INPUT	MEMORY	The shape parameter list stored in SPAR
OUTPUT	LEVEL ONE	A list of points which are equally spaced around a circle.
UTILITIES		None

Shape Parameters (Default)

SPAR(14760)

```
{ (1.5, 90) 3 1 }
```

SUMMARY

SPAR is a list of parameters used by SDRW to draw a shape. It reads as follows.

{ (radius, starting angle), number of points, number of revolutions}

Most of the programs in this chapter use or manipulate SPAR. What happens when SPAR doesn't exist in the work directory? The default shape parameters defined in the main graphing directory, GRAPHP, will be used as a default when it isn't defined in the work directory, WRK.

As an example, if you're starting out with an empty work directory and want to set the minimum radius to 2, you would enter 2 RADIU. RADIUS first creates SPAR by recalling its default value stored in the main graphing directory and storing it in the work directory. Remember a variable in a different directory can be recalled, but can't be changed. Now it is able to put 2 into SPAR.

PATH { HOME GRAPHP }

Shape Start List

STLST(104681)

```
{ SPAR RADIUS THETA
POINTS REV SDRW
ADVAN End }
```

SUMMARY

You'll find a different version of STLST in every program directory. Start, the program that creates the custom graphing menu, takes the list in STLST or the list created by STLST, and forms a custom graphing menu.

This version of STLST is used to create the shape graphing menu. A description of each menu key is given at the end of this chapter.

PATH { HOME GRAPHP SHAPEP }

Store Starting Angle

THETA(87602)

```
« 'SPAR' PCHK 1 DUP2
GET RE 4 ROLL R→C
PUT »
```

SUMMARY

Using the parameters stored in SPAR, SDRW will plot and connect a number of points that are a distance R away from the origin. THETA puts the real number from level one into the imaginary part of the complex number in the first position of SPAR, the shape parameters. This "tells" SDRW, what angle to start drawing from. This angle, measured in degrees, will be the angle the starting point makes with the X axis.

> IMPORTANT
> The angle put on level one must be in degrees.

PATH		{ HOME GRAPHP SHAPEP }
INPUT	LEVEL ONE	Any real number
OUTPUT	MEMORY	The real number from level one is put into the imaginary part of the complex number in the first position of SPAR.
UTILITIES		PCHK

Shape Work Directory

WRK(954)

Directory

SUMMARY

As you read through this text you'll notice every chapter has a listing for a directory named WRK. Don't let this confuse you. These are all different directories. Each program directory has its own work directory named WRK. See the directory tree on page 110.

WRK is used to store any objects or programs you create. The work directory listed here is located just below SHAPEP. Working in WRK protects the programs in SHAPEP, the shape drawing program directory, from accidentally being purged. It is also easier sorting through variables in a smaller work directory than sifting through all your variables and programs in one big directory.

PATH { HOME GRAPHP SHAPEP }

SECTION TWO

EXAMPLES

INTRODUCTION

The shape graphing menu allows you to create a variety of shapes, from a simple triangle to a multi-pointed star. Just specify the radius, starting angle, number of end points and number of revolutions to use when creating a shape. Because of it's simplicity, a triangle will be used when describing each key in the shape menu.

DEFINING A BASIC POLYGON

A basic polygon can be described as a number of points that are equally spaced around a circle. Connecting these points in order traces out the polygon. The number of points determines its type.

Figure 4.1 shows three points, P1, P2, and P3, that are equally spaced around a circle whose radius is r. The fact that the points are equally spaced means that they are 360°/3=120° apart. Connecting these points draws an equilateral triangle. Likewise, four points define a square, five a pentagon, six a hexagon, and so on.

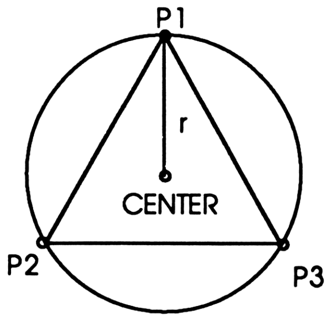


Figure 4.1

THE SHAPE PARAMETER LIST (SPAR)

All the information needed to draw a shape is stored in SPAR, the shape parameter list, which reads { (radius, starting angle), number of points, number of revolutions}. The first object is the starting point for the shape. The real part is the radial distance from the center while the imaginary part is the angle a ray connecting that point and the center makes with the X axis. The starting point for figure 4.1 is (r, 90°). The second object is the number of points that define the shape. Finally, the last object is the number of revolutions made when creating the shape. This is one for polygons. Any other number will create a star. The parameters can be set by putting a real number on level one and pressing the key described below.

Position in SPAR	Represents	Menu Key
1 (real part)	Radial distance from each point to the center	RADIU
1 (imaginary part)	Angle first point makes with the X axis	THETA
2	Number of points defining shape	POINT
3	Number of revolutions made when connecting points	REV

As with every other parameter list in this book, SPAR has a default value in the main graphing directory, GRAPHP. The default value { (1.5, 90), 3, 1 } is used when SPAR isn't defined in the work directory.

DRAWING POLYGONS

Try drawing a triangle with a radius of 1.5 whose first point is at the top of the display.

As we saw earlier, a triangle is defined by three points. If the starting point is at the top of the display, a ray connecting this point to the center makes a 90° angle with the X axis. Finally, the number of revolutions for any polygon is always one. The shape parameters should be {(1.5, 90), 3, 1 }. This just happens to be the default value for $PAR. Since this list will be used when $PAR doesn't exist, you can define a triangle by purging $PAR. Of course you can always define $PAR using the shape menu. Then, running $DRW will create the shape shown in figure 4.3.

```
1.5 RADUI 90 THETA 3 POINTS 1 REV $DRW
```

Seven different polygons are shown below. The only parameter that differs between them is the number of points. Notice as the number of points increase the polygon converges to a circle.

```
1.5 RADUI 90 THETA 2 POINTS 1 REV $DRW
```

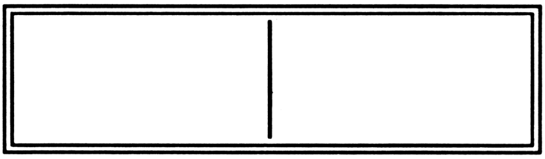


Figure 4.2 (2 points)

```
3 POINTS $DRW
```

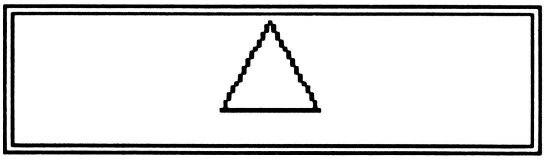


Figure 4.3 (3 points)

4 POINTS DRAW

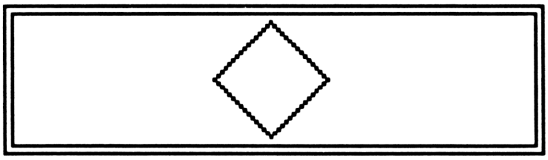


Figure 4.4 (4 points)

5 POINTS DRAW

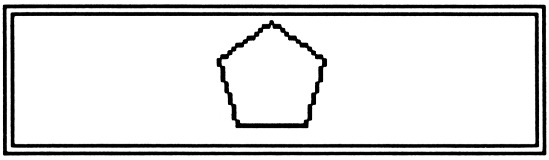


Figure 4.5 (5 points)

6 POINTS DRAW

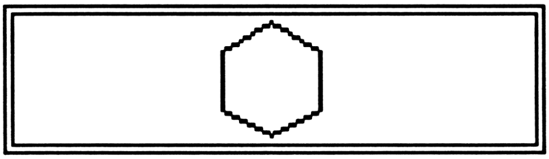


Figure 4.6 (6 points)

7 POINTS SDRW

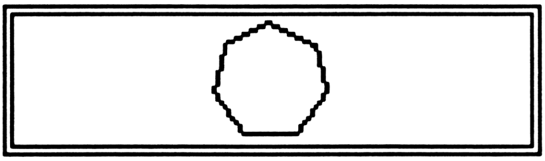


Figure 4.7 (7 points)

8 POINTS SDRW

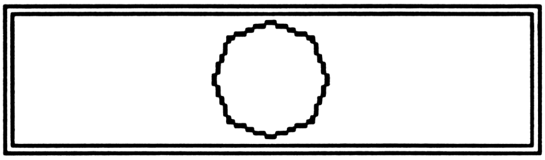


Figure 4.8 (8 points)

DRAWING STARS

Until now we have restricted SDRW to one revolution. What happens when we increase the number of revolutions SDRW makes when connecting points.

As an example, try drawing a shape with 5 points in two revolutions whose starting point is (1.5, 90) and compare it to figure 4.5.

1.5 RADUI 90 THETA 5 POINTS 2 REV SDRW

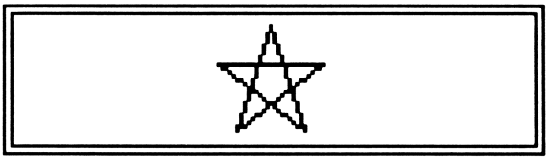


Figure 4.9

The points are in the same position, but they were connected in a different order. Instead of connecting the points one after another as in figure 4.5, SDRW connected every other point and took two revolutions to connect all of them. If we had used 3 for the number of revolutions, every third point would have been connected.

A series of stars are shown in figures 4.10 through 4.15. The all have 17 points and their starting point is (1.5, 90). They differ only in the number of revolutions made while connecting the points. The first stars can be defined by entering

1.5 RADUI 90 THETA 13 POINTS 1 REV SDRW

All others can be defined by substituting the 1 in the example above with the appropriate number of revolutions.

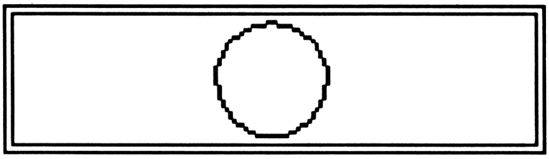


Figure 4.10 (1 revolution)

2 REV SDRAW

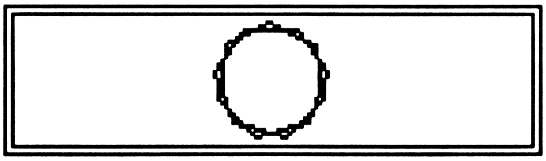


Figure 4.11 (2 revolutions)

3 REV SDRAW

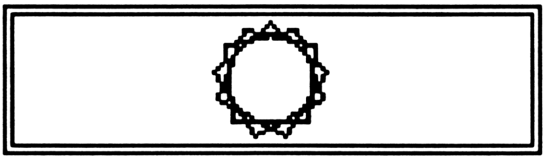


Figure 4.12 (3 revolutions)

4 REV SDRAW

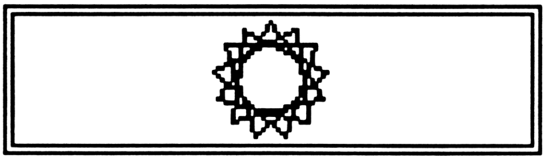


Figure 4.13 (4 revolutions)

5 REV SDRW

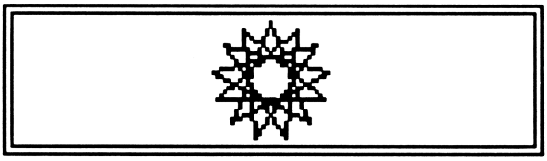


Figure 4.14 (5 revolutions)

6 REV SDRW

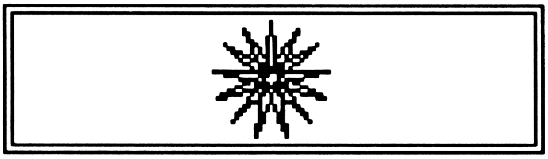


Figure 4.15 (6 revolutions)

Having a radius larger than one doesn't guarantee SDRW will draw a star. Try drawing the shape defined by the parameter list {(1.5, 90), 12, 4}. You might expect to get a 12 pointed star, but instead, a triangle will be drawn. Connecting every fourth point only traced out the same triangle four times.

This is because the number of points defining the shape (12) is divisible by the number of revolutions (4). Carrying out the division gives us the parameters {(1.5, 90), 3, 1} which, to no surprise, is the parameters for a triangle.

SDRW AS A SUBPROGRAM

If flag 17 is clear, SDRW clears the display, draws a shape, and activates the digitizing keyboard. It acts as a menu driven graphing program. But what if you want to incorporate SDRW as a subroutine in a larger program? Chances are, you won't want the display cleared and you probably won't want the digitizing keyboard activated.

When used as a subprogram, you need to set flag 17 before calling `$DRW`. Then `$DRW` will draw a shape without clearing the display or entering digitizing mode. It also resets flag 17, so it will always be ready when you want to use it as a menu driven program.

SHAPE DRAWING MENU

Menu Key	Operation
SPAR	Pressing this menu key will return the shape parameters. They read { (radius, starting angle), number of points, number of revolutions }
RADIU	This program puts the number from level one into the real part of the complex number in SPAR. The number on level one must be a real number.
THETA	This program puts the number from level one into the imaginary part of the complex number in SPAR. The number on level one must be a real number.
POINT	POINT takes the real number from level one and puts it into the second position in SPAR. This number should be a positive integer.
REV	This program puts the real number from level one into the second position in SPAR. This number should be a positive integer.
SDRW	SDRW draws a shape using the shape parameter list stored in SPAR. Beginning with the starting point, SDRW divides the number of points evenly around a circle. It then connects these points in the number of revolutions specified.
ADVAN	As in most menus in this book, this key will create the advanced graphing menu described in chapter six.
END	The last menu key returns you to the main graphing menu and put in the work directory just below GRAPHP.

CHAPTER FIVE

DRAWING SIMPLE FRACTALS

SECTION ONE

REFERENCE SECTION

INTRODUCTION

This is the reference section for the fractal drawing menu. The programming in this section allow you to define and draw simple fractals. Before starting this chapter you should flip through section two, the example section. It will give you an idea of what to expect from these programs. Then, if you like what you see, you can jump right in.

Most of the programs in this chapter are stored in FRACTP, the fractal drawing program directory. You must create this directory before storing any programs. Go to GRAPHP, the main graphing directory, and create FRACTP by entering, HOME GRAPHP ENTER . This will put you in the main graphing directory. Looking at the directory tree on page 140 you can see that we need to create the fractal program directory in the shape program directory, SHAPEP. If SHAPEP is isn't already created type SHAPEP CRDIR . Now enter the shape directory and create FRACTP by typing SHAPEP FRACTP CRDIR . Now you're ready to enter all the programs from this chapter.

A table of all the programs you'll need in this chapter is given on the next page. An asterisk (*) after a name indicates it is listed in a different chapter. Flip to that page number and check whether you've already keyed that program in.

PROGRAM TABLE

Program	Page #	Bytes	Program	Page #	Bytes
CLCD?•	36	52.5	LNE	144	42.5
CNCT•	102	87.5	MAKE	145	42.0
CREATC	130	58.0	NMST	146	135.0
CREATM	132	178.5	SEG2	147	58.5
DGTZ?•	40	47.5	SEG4	148	82.0
DRAWC	134	37.0	Shape	149	82.5
DRAWM	134	45.5	SHAPL•	111	143.5
EDIT	135	129.5	STAT	150	103.5
ENd•	69	16.0	STLST	151	52.0
FDRW	136	312.0	STOC	152	33.5
FRACT	139	47.0	STOK	153	41.5
FRACTP	140	18.0	STOM	154	33.5
LCONV	141	104.5	WRK	155	14.0
LDGU	142	279.5			

NOTE

There are several programs in this book with the same name. Just because two chapters contain the same name in their program tables doesn't necessarily mean they are the same program. They could be two very different programs stored in two different directories.

For example, chapters two through five all have listings for STLST. STLST is used to create a custom graphing menu. Since every chapter creates its own custom menu, each one will have a different version of STLST.

Create an Initial Curve

CREATC(86207)

```
« 'PPAR' PURGE CLLCD
LCD→ LDGU STOC CLMF
»
```

SUMMARY

CREATC is located in the fractal editing menu. It allows you to draw your own freehand initial curve. First, it clears the display and enters digitizing mode. You can define an initial curve by digitizing points and pressing ON . These points will be connected on the display and stored as a list. Adding to the curve is as easy as digitizing more points and pressing ON again. The new points will be appended to the curve.

If you make a mistake while drawing your curve, just press DEL and then ON . This will drop the points that were digitized since the last time you pressed ON . If you make a big error you can start on a new curve by pressing DEL , digitizing at least one point, and pressing ON . The old curve will still be displayed, but its list of points will be dropped. You can use the image of the old curve as a guide for your new one.

Finally, when you've finished drawing your curve, press ON again to store the list of points representing the curve in 'CURV'. A summary is given on the next page.

PATH { HOME GRAPHP SHAPEP FRACTP }

INPUT (KEYBOARD)	**OUTPUT**
digitize points by pressing INS and then pressing ON .	The points you digitized are connected from the last point on the current curve.
digitizing points by pressing INS , pressing DEL and immediately pressing ON	The points you just digitized are cleared. You can continue with the same curve.
pressing DEL , digitizing at least one point, and then pressing ON .	You'll start a new curve. The old curve will still be displayed, but the old list of points representing that curve will be dropped.
pressing ON	You'll exit CREATC and the list of points representing the current curve will be stored in 'CURV'

UTILITIES LDGU, STOC

Create a Model Construction

CREATM(1547988)

```
« RCLF DEG 'PPAR'
PURGE 5 INV DUP *H
*W CLLCD 2 INV 0 R→C
DUP NEG PIXEL PIXEL
LCD→ LDGU 1 DO GETI
3 ROLLD DUP2 GET 4
ROLL - R→P 3 ROLLD
UNTIL 46 FS? END ROT
DROP2 SIZE 1 - →LIST
STOM 'PPAR' PURGE
STOF CLMF »
```

SUMMARY

CREATM allows you to draw a freehand model construction. First, it clears the display and draws two dots. These dots are a guide for the beginning and ending points on your model. You can draw your model anywhere, but positioning it at these suggested points will create a continuous fractal. Positioning your model elsewhere will cause the fractal to be segmented.

You can define a model by digitizing points and pressing ON . These points will be connected on the display and stored as a list. Adding to it is as easy as digitizing more points and pressing ON again. The new points will be appended to the model. If you make a mistake while drawing your model just press DEL and then ON . This will drop the points that were digitized since the last time you pressed ON . If you make a big error you can start on a new model by pressing DEL , digitizing at least one point, and pressing ON . The old curve will still be displayed, but its list of points will be dropped. When you've finished, press ON again to store the list of points representing the current model in 'MODL'.

PATH { HOME GRAPHP SHAPEP FRACTP }

INPUT (KEYBOARD)	OUTPUT
digitize points by pressing INS and then pressing ON .	The points you digitized are connected. If you already started a model the points will be added to the last point on the current model.
digitizing points by pressing INS , pressing DEL and immediately pressing ON	The points you just digitized are cleared. You can continue with the current model.
pressing DEL , digitizing at least one point, and then pressing ON	You'll start on a new model. The old model will be displayed, but the old list representing this model will be dropped.
pressing ON	You'll exit CREATM and the list of points representing the current model are converted to polar coordinates and stored in 'MODL'

UTILITIES LDGU, STOM

Draw Initial Curve

DRAWC(23897)

```
« CLLCD CURV CNCT »
```

DRAWC draws the initial curve stored in CURV on the display.

PATH	{ HOME GRAPHP SHAPEP FRACTP }
INPUT <u>MEMORY</u>	The list in 'CURV'
OUTPUT <u>LCD</u>	A figure is drawn on the display.
UTILITIES	CNCT

Draw Model Construction

DRAWM(35760)

```
« CLLCD MODL LCONV CNCT »
```

DRAWM draws the model construction stored in MODL.

PATH	{ HOME GRAPHP SHAPEP FRACTP }
INPUT <u>MEMORY</u>	The list in 'MODL'
OUTPUT <u>LCD</u>	A figure is drawn on the display.
UTILITIES	LCONV, CNCT

Create the Fractal Edit Menu

EDIT(270695)

```
    « { STOK CURV CREATC
    STOC Shape DRAWC
    MODL CREATM STOM
    SEG2 SEG4 DRAWM
ENd
    } MENU »
```

SUMMARY

EDIT creates the fractal editing menu. In this custom menu you will be able to edit the initial curve and model construction, and set the number of replacements, K.

PATH	{ HOME GRAPHP SHAPEP FRACTP }
INPUT	None
OUTPUT MEMORY	A custom menu is created.
UTILITIES	None

Fractal Drawing Program

FDRW(3466957)

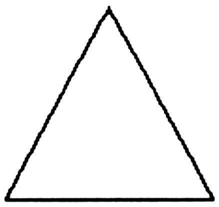
```
« CLCD? RCLF DEG 1
CURV SIZE 1 - FOR i
'CURV' i GETI 3
ROLLD GET DUP2 - NEG
R→P SWAP DROP MODL
SIZE → d m n « d 0 n
K ^ 1 - FOR j m j K
1 - 0 FOR k n k ^
DUP2 / IP 'MODL'
OVER 1 + GET 5 ROLL
P→R OVER RE * R→P
SWAP DUP RE NEG + +
4 ROLLD * - -1 STEP
DROP P→R OVER + LNE
NEXT » DROP NEXT
STOF DGTZ? »
```

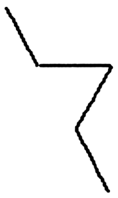
SUMMARY

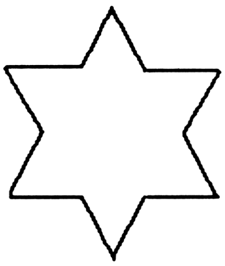
FDRW is the program that does all the work. It uses the list from CURV as an initial curve and the list from MODL as a model construction. FDRW replaces each line segment of the initial curve with the model construction. This creates a new curve. The same process is repeated K times. An integer must be stored in K.

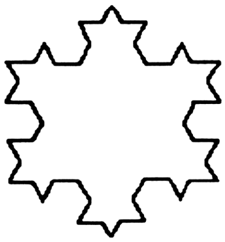
The initial curve, CURV and the model construction, MODL, are simply a series of line segments. They can be a shape, such as a triangle, or even a design, like the letter Z. The only requirement is that they be stored as a list. This list must be a series of points that, when connected in order, form the initial curve or the model construction. You can create these lists manually, or use any one of the fractal editing tools.

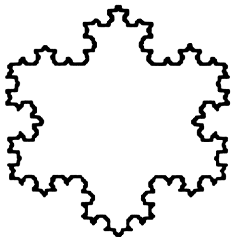
A sample initial curve and model construction together with the fractal that would be drawn for different values of K are shown below.


Initial Curve


Model Construction


One replacement (K=1)


Two Replacements (K=2)


Three Replacements (K=3)

NOTE

If the display doesn't clear when using FDRW from the custom menu, flag 17 is set. If this happens you can stop the program and clear flag 17 or FDRW will clear it after it is done graphing. Either way, flag 17 will be cleared and you can try again.

PATH		{ HOME GRAPHP SHAPEP FRACTP }
INPUT	MEMORY	The lists stored in CURV and MODL, as well as the integer in K.
	KEYBOARD	If flag 17 is clear, the digitizing keyboard will be activated.
OUTPUT	LCD	A fractal is drawn on the display screen. If flag 17 is clear the display will be cleared before drawing the fractal.
	STACK	If flag 17 was clear before running, any digitized objects will be on the stack.
UTILITIES		CLCD?, DGTZ?, LNE

Create Fractal Drawing Menu

FRACT(34466)

```
« SHAPEP FRACTP
Start »
```

SUMMARY

FRACT puts you in the work directory just below FRACTP and creates the fractal drawing menu. If you haven't already noticed, every graphing menu has its own program directory, the directory with the programing specific to that topic, its own work directory, titled WRK, its own graphing menu, and a command to enter that menu.

Except for the last letter, the commands to enter the graphing menus all have the same name as their program directories. For example, FRACT puts you in the work directory, WRK, just below FRACTP. Likewise, SHAPE puts you into the shape drawing work directory, WRK, just below SHAPEP. See the directory tree on the next page.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	STLST, the starting list, stored in FRACTP
OUTPUT	MEMORY	Directory control is given to the work directory, WRK subordinate to FRACTP. This is shown in the directory tree on the next page. The custom fractal drawing menu is also created.
UTILITIES		Start

Fractal Drawing Program Directory

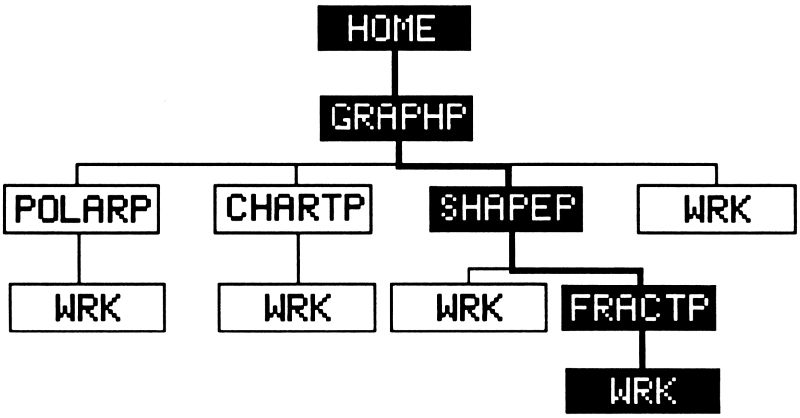
FRACTP(2396)

Directory

SUMMARY

This directory contains all the programs used specifically for drawing fractals. Storing the fractal drawing programs in their own directory creates a sense of order, like chapters in a book.

Some programs that are used for drawing fractals are stored in directories other than FRACTP. This is because they are used by graphing procedures in other directories. Putting these programs in other directories allows any program in or subordinate to it access. Thus a program in CHARTP can use LINE as well as a program in FRACTP, since it is in stored in GRAPHP. A program in CHARTP can't use SEG2 though, because it is stored in FRACTP which isn't in CHARTP's path. The directory tree below illustrates this.



Convert List of Points from Polar to Rectangular Coordinates

LCONV(400594)

```
« RCLF DEG (-1.5, 0)
ROT + 2 OVER SIZE
FOR i i DUP2 1 -
GETI 3 ROLLD GET P→R
3 * + PUT NEXT SWAP
STOF »
```

SUMMARY

The list of complex numbers stored in MODL represent line segments. The real part is the length (or radius) and the complex part is the angle made with the positive X axis. In contrast, the list in CURV and any other list used by CNCT represents lines by the rectangular coordinates of their end points.

LCONV converts the list of points in MODL from polar to rectangular coordinates. The new list can then be drawn using CNCT.

PATH		{ HOME GRAPHP SHAPEP FRACTP }
INPUT	LEVEL ONE	A list of complex numbers representing line segments. The complex numbers are in polar form, i.e. (radius, angle).
OUTPUT	LEVEL ONE	A list of complex numbers representing the end points of the line segments in the Cartesian coordinate system (XY plane).
UTILITIES		None

Line Drawing Utility

LDGU(3997276)

```
« { } DEPTH 2 - →
lcd lst d « WHILE
DGTIZ lcd →LCD DEPTH
d - DUP REPEAT IF
OVER TYPE 1 ≠ THEN
DROPN ELSE lst { }
ROT 2 + 3 FOR i i
ROLL IF DUP TYPE 1 -
THEN 3 DROPN { } { }
ELSE + END -1 STEP
OVER DUP SIZE DUP
SUB OVER + CNCT +
'lst' STO LCD→ 'lcd'
STO END END DROP lst
» »
```

SUMMARY

LDGU is used by both CREATC and CREATM when creating a freehand initial curve or model construction. Just digitize one or more points and press ON . LDGU will connect these points on the display and return to digitizing mode. You can add new points by digitizing them and pressing ON again. The new points will be added to the current drawing. This program also keeps track of all the points you entered by storing them as a list.

If you make a mistake while drawing just press DEL and then ON . If you had just digitized points, they will be drop, allowing you to continue on the same drawing. If you hadn't digitized points since ON was last pressed, you will start out on a new drawing. The old sketch will still be displayed, but its list of points will be dropped.

You can use the image of the old drawing as a guide for your new one.

Finally, when you've finished, press ON to return the list of points representing the sketch to level one. If you are creating an initial curve this list will be stored in 'CURV', else if you are creating a model construction the list will be converted to polar coordinates and stored in 'MODL'. See CREATC and CREATM for more details.

A summary of different keyboard input and its related output is given below.

PATH { HOME GRAPHP SHAPEP FRACTP }

SEQUENCE	WHAT HAPPENS
Digitize points by pressing INS and then pressing ON	The points you digitized are connected from the last point on the drawing.
pressing DEL digitizing at least one point, and then pressing ON	You'll start on a new drawing. The old sketch will still be displayed, but the list of points representing the drawing will be dropped.
Digitizing points by pressing INS, pressing DEL, and immediately pressing ON	The points you just digitized are cleared. You can continue with the same drawing.
Pressing ON	You'll exit LDGU and the list of points representing the current drawing are left on level one.

UTILITIES CNCT

Line Drawing Utility

LNE(89634)

```
« DUP IFERR ROT THEN
DROP ELSE LINE END »
```

SUMMARY

LNE is used by FDRW to connect the end points it generates on the fractal. If there are two points on levels one and two, it will connect them and drop the point from level one. If only one point is on the stack it does nothing.

LNE has a wider range of application. Most drawing programs simply plot a series of points. Often, the points are far apart, giving the graph a disconnected appearance. You can substitute this program for PIXEL in most any graphing programs. The modified program will trace out a graph instead of plotting individual points.

NOTE

This program assumes LAST■ is set in the MODE menu. You will get an error if it isn't.

PATH		{ GRAPHP SHAPEP FRACTP }
INPUT	LEVEL TWO	A complex number (Optional)
	LEVEL ONE	A complex number
OUTPUT	LEVEL ONE	If an object was on level two it is dropped to level one.
	LCD	If two complex numbers were on level one and two they are connected on the display.
UTILITIES		LINE

Make the Shape Defined by SPAR and Store in CURV

MAKE(21754)

```
« SHAPL STOC EDIT »
```

SUMMARY

MAKE creates a list of points that, when connected, create the shape described by the parameter list stored in SPAR. This list is stored in CURV, where it is used as an initial curve by FDRW.

After this list is stored, MAKE puts you back in the fractal editing menu. You can view the curve you just created by pressing DRAW from the first set of menu keys. This connects all the points from the list in CURV on the display.

PATH		{ HOME GRAPHP SHAPEP FRACTP }
INPUT	MEMORY	The list in SPAR, the shape parameters.
OUTPUT	MEMORY	A list of points is stored in 'CURV'.
UTILITIES		SHAPL, STOC, EDIT

Normalize a List

NMST(903584)

```
« RCLF SWAP DEG DUP
LIST→ (0, 0) SWAP 1
SWAP START SWAP P→R
+ NEXT R→P RE INV →
L « 1 OVER SIZE FOR
 i i DUP2 GET P→R L *
R→P PUT NEXT SWAP »
STOF »
```

SUMMARY

NMST is used to normalize the list of polar coordinates that represent the model construction. The model must be normalized so that the total distance between its starting and ending point equal one. The end points of a normalized model will properly match the end points of each segment in the initial curve it is replaces. If the list isn't normalized, the end points will not match up and the resultant fractal will be broken.

PATH		{HOME GRAPHP SHAPEP FRACTP }
INPUT	LEVEL ONE	A list of points
OUTPUT	MEMORY	A normalized list of points.
UTILITIES		None

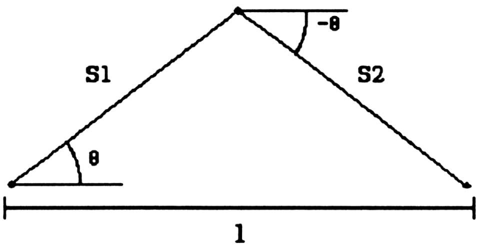
Create a Model Construction with 2 Segments

SEG2(120908)

```
« 1 SWAP DUP2 R→C 3
ROLLD NEG R→C 2
→LIST NMST STOM »
```

SUMMARY

SEG2 can be found in the fractal editing menu. It takes the angle from level one (in degrees) and creates a symmetric, two segment model construction. The first segment will make a positive angle with the horizontal axis while the second segment makes a negative angle. (See the figure below) The list representing this model construction is then normalized and stored into 'MODL'.



Segment P1P2 make an angle θ with the horizontal axis while segment P2P3 make a angle -θ.

PATH { HOME GRAPHP SHAPEP FRACTP }

INPUT LEVEL ONE Any angle (in degrees)

OUTPUT MEMORY A list is stored in 'MODL'

UTILITIES NMST, STOM

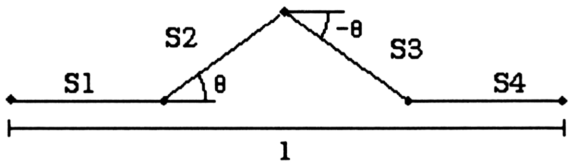
Create a 4 Segment Model Construction

SEG4(169966)

```
« (1, 0) 1 ROT DUP2
R→C 3 ROLLD NEG R→C
3 PICK 4 →LIST NMST
STOM »
```

SUMMARY

SEG4 can be found in the fractal editing menu. It takes the angle from level one (in degrees) and creates a symmetric, four segment model construction. The first and fourth segment are parallel to the horizontal axis while, using the angle from level one, the second and third segment make a positive and negative angle with the horizontal axis.



Segments P1P2 and P4P5 are parallel with the horizontal axis while segments P2P3 and P3P4 make a positive and negative angle θ with the horizon respectively.

PATH { HOME GRAPHP SHAPEP FRACTP }

INPUT LEVEL ONE Any angle (in degrees)

OUTPUT MEMORY A list representing the model construction is stored in 'MODL'.

UTILITIES NMST, STOM

<u>Menu to Create a Shape</u>
<u>for an Initial Curve</u>

Shape(120189)

```
« { SPAR RADIUS THETA
POINTS REV MAKE ENd
} MENU »
```

SUMMARY

Shape displays a menu that allows you to create an initial curve using many of the tools from the shape drawing directory. This menu is almost identical to the shape drawing menu from chapter four. The main difference is the new key labeled MAKE.

Rather than drawing the shape defined by SPAR, MAKE creates a list representing that shape and stores it in 'CURV'. This list will be used by FDRW as a initial curve. You can view this initial curve by running DRAW from the first set of menu labels in the fractal editing menu.

PATH	{ HOME GRAPHP SHAPEP FRACTP }
INPUT	None
OUTPUT <u>MEMORY</u>	A custom menu is created.
UTILITIES	None

Show Status of Fractal

STAT(271768)

```
« CURV SIZE 1 - →STR
" IN CURVE▪" + MODL
SIZE →STR +
" IN MODEL▪ K= "+ K
→STR + 1 DISP »
```

SUMMARY

STAT displays the number of segments in the initial curve ('CURV') and the model construction ('MODL'), and the number of times FDRW will replace the segments of the initial curve with the model construction (K).

The character▪ in the program listing is character 10, the new line character. When entering the program press SHIFT SPACE . You won't see the new line character (▪) while in edit mode, but the cursor will go to the next line.

PATH		{ HOME GRAPHP SHAPEP FRACTP }
INPUT	MEMORY	The list in 'CURV' and 'MODL', and the integer in 'K'.
OUTPUT	LCD	The number of segments of the model construction and initial curve, and the number of replacements are displayed.
UTILITIES		None

Starting Fractal Menu List

STLST(44777)

```
{ EDIT STAT FDRW
ADVAN End }
```

SUMMARY

You'll find a different version of STLST in every program directory. Start, the program that creates the custom graphing menu, takes the list in STLST or, in some cases, the list created by STLST, and forms a custom graphing menu.

This version of STLST is used to create the fractal menu. A description of each menu key is given at the end of this chapter.

PATH { HOME GRAPHP SHAPEP FRACTP }

Store List Into the Initial Curve Variable (CURV)

STOC(14420)

```
« 'CURV' STO »
```

SUMMARY

STOC can be found in the fractal editing menu. It simply stores the list on level one into 'CURV'., the variable that contains the list representing the initial curve.

After you've created an initial curve using the tools in the edit menu, you may want to store in under a different name. Then, just use STOC whenever you want to use that initial curve again. This can be very useful with freehand initial curves that were created using CREATC.

PATH		{ HOME GRAPHP SHAPEP FRACTP }
INPUT	LEVEL ONE	A list that represents the initial curve.
OUTPUT	MEMORY	The list from level one is stored in 'CURV'.
UTILITIES		None

Store Value into K

STOK(26358)

```
« IP 'K' STO Start »
```

SUMMARY

STOK stores the integer part of the number from level one in K. This number "tells" FDRW how many times it should substitute the model construction for each line segment of the initial curve. The curve created by each substitution is used as the curve for the next substitution.

STOK also exits the editing menu and returns you to the main fractal menu.

PATH		{ HOME GRAPHP SHAPEP FRACTP }
INPUT	LEVEL ONE	A real number
OUTPUT	MEMORY	The integer part of the real number from level one is stored in K.
UTILITIES		Start

Store List into the Model Construction Variable (MODL)

STOM(14466)

```
« 'MODL' STO »
```

SUMMARY

This program simply stores the list on level one into 'MODL', the variable that contains the list representing the model construction.

After you've created a model using the tools in the edit menu, you may want to store in under a different name. Then, just use STOM whenever you want to use that model again. This can be very useful for freehand models that were created using CREATM.

PATH		{ HOME GRAPHP SHAPEP FRACTP }
INPUT	LEVEL ONE	A list that represents the model construction.
OUTPUT	MEMORY	The list from level one is stored in 'MODL'.
UTILITIES		None

Fractal Drawing Work Directory

WRK(954)

Directory

SUMMARY

As you read through this text you'll notice every chapter has a listing for a directory named WRK. Don't let this confuse you. These are all different directories.

The work directory listed here is located just below FRACTP. This is shown on the directory tree on page 140. It is used to store any objects you create. Working in WRK protects the programs in FRACTP, the fractal drawing program directory, from accidentally being purged. It is also easier sorting through variables in a smaller work directory than sifting through all your variables and programs in one big directory.

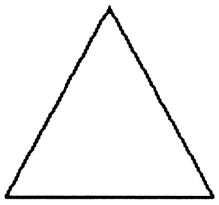
PATH { HOME GRAPHP SHAPEP FRACTP }

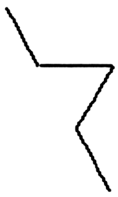
SECTION TWO

EXAMPLES

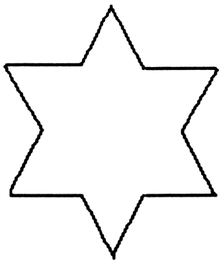
A SIMPLE FRACTAL

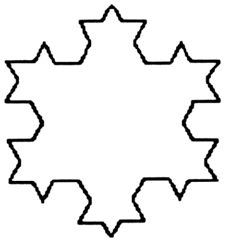
Many types of fractals exist, but we will limit this book to simple fractals. Don't be mislead by the name. A simple fractal is hardly simple. Its curves can be extremely complex and winding. A simple fractal consists of an initial curve and a model construction. Each is just a series of connected line segments. Examples of an initial curve and model construction are given below.

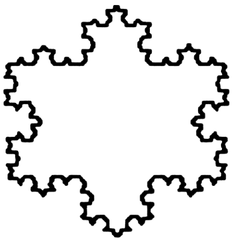

Initial Curve
Figure 5.1


Model Construction
Figure 5.2

An initial curve can be any shape or design. The initial curve shown above is a triangle. A triangle consists of three segments. To create a simple fractal we need only replace each segment in the initial curve with the model construction. We end up with a new curve which we will call C1.


One Replacement C1
Figure 5.3


Two Replacements C2
Figure 5.4


Three Replacements C3
Figure 5.5

The new curve, C1, has twelve segments. If we again replace each of these twelve segments with the same model construction we will end up with a new curve, C2. Continuing, a third replacement will transform C2 into yet a new curve, C3.

One through three replacements are show in figures 5.3 to 5.5. You'll notice that each successive curve has a greater number of segments that are shorter in length. Each new curve is more defined than the last. If we continue the replacement process our fractal will converge towards a smooth curve. The length of each segment will tend towards 0 while the number of segments tend towards infinity. Because of the lower resolution of the display, a fractal drawn by the 28S will converge quickly. In many fractals drawn on the 28S, there is no visual difference between successive curves with replacements higher than 4. Of course this number will vary depending on the initial curve and model construction.

GETTING STARTED

The first step in designing a fractal is creating an initial curve and model construction. This can be done in the fractal editing menu. First, go to the fractal menu by entering HOME GRAPH ENTER or press END in the graphing menu you're in. This will put you in the main graphing selection menu. Pressing FRACT and then EDIT from the custom menus will put you in the fractal editing menu.

CREATING AN INITIAL CURVE

The initial curve is stored, as a list of points, in 'CURV'. These points represent the end points of the segments that make up the initial curve. Connecting them will draw the initial curve like a dot to dot drawing.

You can create or edit an initial curve from the fractal editing menu. The second label in this menu, CURV, is the variable where the initial curve is stored. Pressing it will return it's value. It's name, 'CURV', will be returned if an initial curve hasn't been defined.

There are three ways to define an initial curve. You can draw one on the screen using CREAT, define a shape using the SHAPE sub-menu, or store a list that was previously defined using STOC.

CREATING A FREEHAND INITIAL CURVE

CREAT, the simplest of the three, allows you to draw your own freehand initial curve. First, it clears the display and enters digitizing mode. You can define an initial curve by digitizing points and pressing ON. These points will be connected on the display and stored as a list. Adding to the curve is as easy as digitizing more points and pressing ON again. The new points will be appended to the curve.

If you make a mistake while drawing your curve just press DEL and then ON . This will drop the points that were digitized since the last time you pressed ON . If you make a big error you can start on a new curve by pressing DEL , digitizing at least one point, and pressing ON . The old curve will still be displayed, but its list of points will be dropped. You can use the image of the old curve as a guide for your new one. Finally, when you've finished drawing your curve, press ON again to store the list of points representing the curve in 'CURV'.

CREATM, A similar program in the model construction half of the editing menu, allows you to create a freehand model . The table below describes both programs. For generalization, the word sketch has been used in place of initial curve or model construction.

INPUT (KEYBOARD)	OUTPUT
digitize at least one point by pressing INS and then pressing ON .	The points you digitized are connected. If you've already started a sketch the points will be appended to the last point on the current sketch.
Digitizing points by pressing INS, pressing DEL and immediately pressing ON	The points you just digitized are cleared. You can continue with the same sketch.
pressing DEL, digitizing at least one point, and then pressing ON .	You'll start a new sketch. The old sketch will still be displayed, but the list of points representing it will be dropped.
pressing ON	You'll exit the program and the list of points representing the current sketch will be stored.

A SHAPE AS AN INITIAL CURVE

The next easiest way to create an initial curve is to use the initial curve shape menu. Pressing SHAPE in the editing menu will create a menu similar to the shape drawing menu from chapter four. A summary of this menu is given at the end of this chapter. Because the two menus are almost identical, the reader should refer to chapter four for a full description of the shape drawing menu.

The main difference between the two menus is SDRAW, the shape drawing program, has been replaced with MAKE. Rather than drawing a shape on the screen, MAKE generates a list of points representing the shape. This list is stored in CURV and will be used as an initial curve. After pressing MAKE you will return to the editing menu. Here, you can press DRAW from the first set of menu labels to have the initial curve drawn on the display.

STORING A PREDEFINED INITIAL CURVE

The third and final way to store an initial curve is to enter a list of points on level one and press STOC. The list from level one will be stored directly into 'CURV'. This means you only need to create an initial curve once. If you plan to use it later, recall the list and store it into any name. Anytime you want to use this initial curve, simply recall it and press STOC. The previously defined list will be stored back into 'CURV'.

THE MODEL CONSTRUCTION

Creating an initial curve is only half of the job. A model construction must also be defined in order to draw a simple fractal. Unlike the initial curve, which was defined as a list of rectangular points, a model construction is defined as a list of line segments and stored in the variable 'MODL'. Each line segment is represented as a complex number. The real part is its length and the complex part is the angle it makes with the X axis. When connected in series, these segments define the model construction.

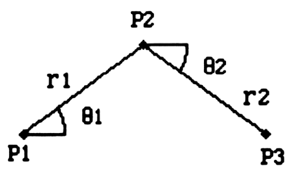


Figure 5.6

As an example, the model construction shown above has two line segments. Line P1P2 has length r1 and make and angle θ1 with the horizontal axis. Likewise, line P2P3 has length r2 and makes an angle θ2. The list { (r1, θ1) (r2, θ2) } completely describes this model.

THE MODEL CONSTRUCTION EDITING MENU

A model construction can be created or edited using the second set of menu keys in the fractal editing menu (pressing NEXT).

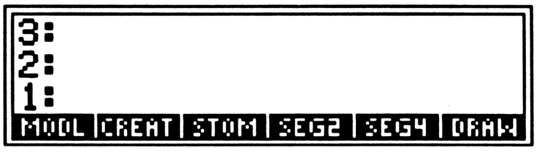


Figure 5.7

The first label, MODL, is the variable where the model construction is stored. Pressing it will return it's value. Its name, 'MODL', will be returned if a model construction hasn't been defined. CREAT STOM SEG2 , and SEG4 are all programs that allow you to create a model construction. The last key, DRAW, will draw the model construction stored in 'MODL'.

CREATING A FREEHAND MODEL CONSTRUCTION

The second menu key, CREAT, functions like the initial curve editing version. Simply digitize the end points of the segments in the model construction and press ON . A full description is given for the initial curve editing menu. Since this program is identical you should read the table on page 160.

There is one difference between this program and the one used to create an initial curve . Before entering digitizing mode, two dots are drawn on the display. These dots are a guide for the beginning and ending points on your model (P1 and P3 in figure 5.6). You can draw your model anywhere, but positioning it at these suggested points will create a continuous fractal. Positioning your model elsewhere will cause the fractal to be segmented. You can check whether you've lined up the cursor with each dot by pressing the cursor key ◂⬍▸.

USING A PREDEFINED FORMAT

SEG2 and SEG4 allow you to create a model construction using a predefined format. SEG2 uses a two segment model while SEG4 uses four. All the segments have the same length. The only variable you need to define is the angle θ. Simply put an angle (in degrees) on level one and press either SEG2 or SEG4. Figure 5.8 shows the format used by SEG2 while figure 5.9 shows SEG4's format.

Notice how both models are normalized (the total distance traveled equals). The model must be normalized to create a fluent fractal. If it wasn't, the resultant fractal would be broken. Using these two programs, you can create complicated, symmetric fractals that would be extremely difficult to obtain using CREATM.

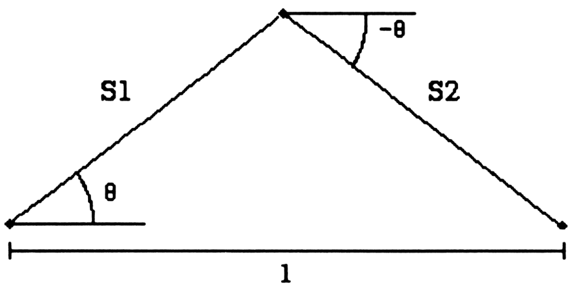


Figure 5.8

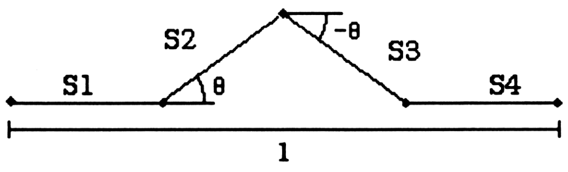


Figure 5.9

STORING A PREDEFINED MODEL

Still another way to store a model construction is to enter a list of points on level one and press STOM. The list from level one will be stored directly into 'MODL'. This means you only need to create a model once. If you plan to use it later, recall the model list and store it under a different name. Any time you want to use it again, simply recall it and run STOM. The previously defined list will be stored back into MODL.

VIEWING A MODEL

To parallel the initial curve half of the editing menu, DRMD, draws the model construction stored in MODL. If the construction drawn on the display isn't what you want you can easily return to the editing menu and try again. You will get an error though, if you haven't already defined a model construction using one of the programs in this menu.

SETTING THE NUMBER OF REPLACEMENTS

Now that you've created an initial curve and model construction, the last thing you need to define is the number of replacements stored in 'K'. K can be set by entering any real number and pressing the menu key labeled STOK in the editing menu. Figures 5.3 through 5.5 show drawings for several different numbers of replacements.

You'll notice that as K gets larger, the fractal becomes more and more defined. Eventually, the fractal will converge, meaning there will is no visible difference between drawings of successive values of K. Due to the lower resolution of the 28S, most fractals can be drawn using a value of 3 or 4 for K. Values higher take much longer to draw and usually show very little improvement. If you're unsure of what value to use start with a value of 1 or 2 and see what the fractal looks like. Then you can decide if you want to increase K.

THE FRACTAL DRAWING PROGRAM (FDRW)

Once you've defined the initial curve, model construction, and number of replacements, you are ready to draw a fractal. Return to the fractal drawing menu and press FDRW . The 28S does the rest. Segment by segment, the fractal is slowly drawn on the display. If the value of K is set at one or two, it should only take a few minutes. After completing the drawing you are put in digitizing mode.

THE STATUS OF THE FRACTAL PARAMETERS

From time to time you might get a drawing you didn't expect. Maybe you changed the initial curve and forgot to change the model construction, or maybe you forgot to define K. You can quickly check the status of all the parameters by pressing STAT in the fractal menu. The number of segments in the initial curve and model curve, as well as the number of replacements, K, are displayed on the screen.

EXAMPLE 5.1

Now that we have the basics lets try drawing the fractal on page 158. Starting out in the fractal menu, press EDIT to enter the editing menu. Next press SHAPE so we can create a triangle for the initial curve (Chapter four describes how a shape is created). The shape parameters for the triangle shown in figure 5.1 are { (1.5, 90), 3 1}. It just so happens this is the default parameter list so we won't have to define it. Purge SPAR if it exists in the fractal work directory and press MAKE to create the initial curve and store it in the CURV. You'll find yourself back in the editing menu. You can view the initial curve at any time by pressing DRAW from the first set of menu labels.

Creating the model construction is much easier. The model has four segments whose second and third segment make an angle of 60° and -60° with the X axis. Enter the first angle, 60, and press SEGM. To view the model construction simply press DRAW from the second set of menu labels.

Next, you must specify the number of replacements made to create the fractal. This is done by putting a real number on level one and pressing STOK in the editing menu. You might want to start out with a value of 1 or 2. This gives you a chance to see the basic outline of the fractal. If you like what you see you can increase the value of K to get better definition. To draw the fractal press FORM in the fractal drawing menu. The time it takes will depend on the value of K.

EDIT SHAPE 'SPAR' PURGE MAKE 60 SEGM 1 STOK FORM

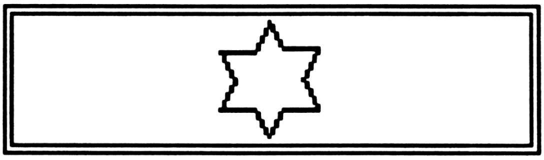


Figure 5.10

EDIT 2 STOK FORM

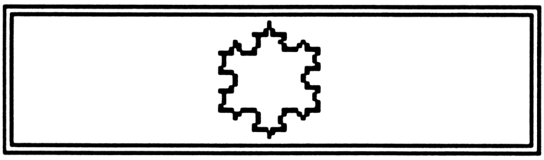


Figure 5.11

EDIT 3 STOK FDRW

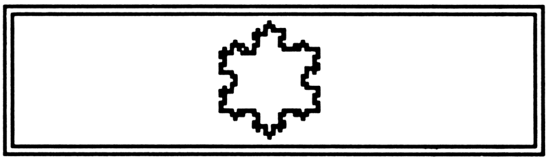


Figure 5.12

Several fractals, their parameters, and the suggested keystrokes you should enter are listed below. These fractals are drawn on only one frame. You can combine FDRW with the extended graph program in chapter six for larger, more defined fractals. See the examples in chapter 7.

EXAMPLE 5.2

Create a five pointed star with a radius of 1.5 for the initial curve. Then, create a four segment model construction whose second and third segments make an angle of -36 and 36 degrees with the horizontal axis. Draw fractals for K=1,2

SOLUTION

Go to the shape menu under the fractal editing menu, define 'SPAR' to be { (1.5,90) 5 2 } and press MAKE. Then, create a four segment model construction by entering -36 SEG4. You can view the initial curve and model by pressing DRAW from the first and second sets on keys in the edit menu.

Finally, define the number of replacements, K. Running FDRW will create figures 5.15 and 5.16.

EDIT SHAPE 1.5 RADIU 90 THETA 5 POINTS 2 REV MAKE
DRAW (from the first set of menu labels)

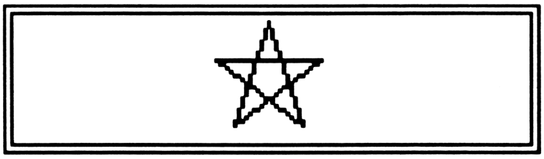


Figure 5.13 (initial curve)

-36 SEG4
DRAW (from the first set of menu labels)

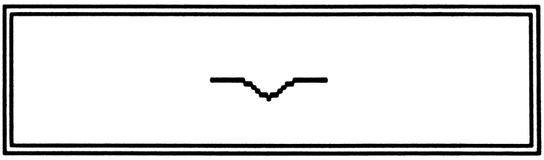


Figure 5.14 (model construction)

1 STOK FORW

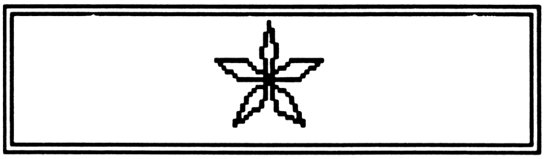


Figure 5.15 (K=1)

EDIT 2 STOK FORW

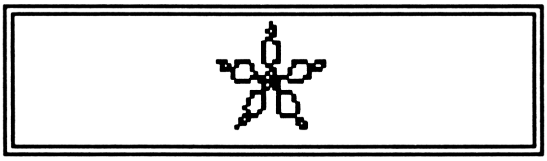


Figure 5.16 (K=2)

EXAMPLE 5.3

Using the same initial curve as example 5.2, draw a fractal whose four segment model is created by entering -72 SEG4. Next try +72.

EDIT -72 SEG4 1 STOK FORW

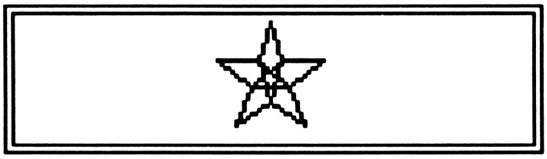


Figure 5.17 (K=1)

EDIT 2 STOK FORW

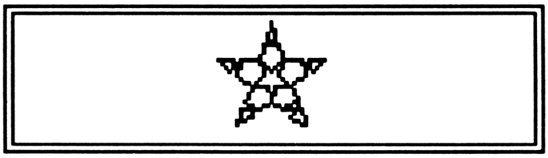


Figure 5.18 (K=2)

EDIT 72 SEG4 1 STOK FDRW

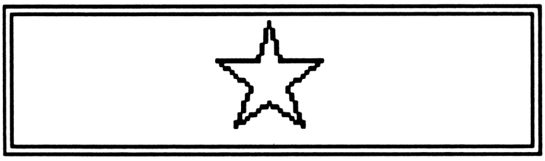


Figure 5.19 (K=1)

EDIT 2 STOK FDRW

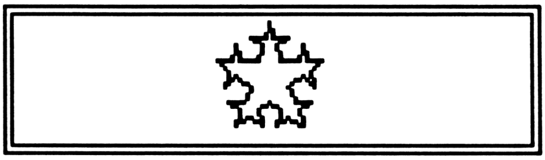


Figure 5.20 (K=2)

EXAMPLE 5.4

Using the same initial curve as example 5.1, draw a fractal whose four segment model is created using -60 as an argument for SEG4. Try a two segment model using -30 and -60 as an argument for SEG2.

SOLUTION

Go to the shape menu under the fractal editing menu, purge 'SPAR' if it exists, and press MAKE to create the initial curve. Then, create the model construction by entering -60 SEG4. Finally, define the number of replacements, K. Running FDRW will create figure 5.21. Try drawing different values of K.

EDIT SHAPE 'SPAR' PURGE MAKE 60 CHS SEG4 2 STOK FORW

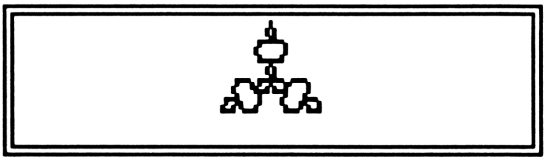


Figure 5.21 (K=2)

Keep the same initial curve and create a two segment model making a angle of -30 and 30 degrees with the horizontal axis by entering -30 SEG2 . Try drawing fractals for 1,2,3, and 4 replacements (Only the fractal for K=4 is shown).

EDIT -30 SEG2 4 STOK FORW

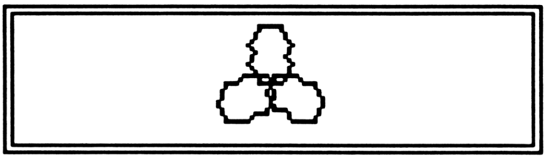


Figure 5.22 (K=4)

Create a -60 degree two segment model by entering -60 SEG2. Draw fractals for K=3,4,5. You may have to expand the range of the display for larger values of K. If the entire fractal doesn't fit on the display extend the range using *H and *W and try again. You may have to repeat this process several times.

NOTE

If you change the dimensions of the display, i.e. change PPAR, be sure to purge the plot parameters before creating a different fractal.

EDIT -60 SEG2 3 STOK 2.25 ENTER ENTER *W *H FORW

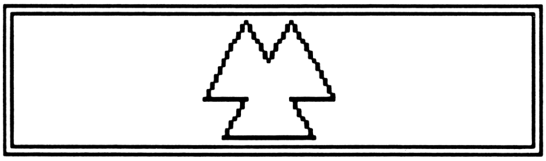


Figure 5.23 (K=3)

EDIT 4 STOK 1.5 ENTER ENTER *W *H FORW

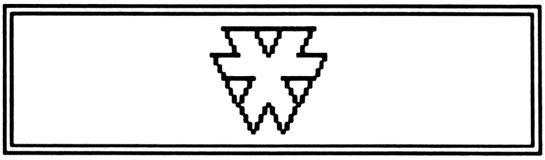


Figure 5.24 (K=4)

EDIT 5 STOK 1.25 ENTER ENTER *W *H FORW

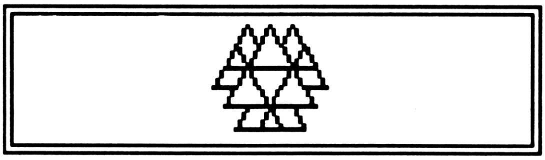


Figure 5.25 (K=5)

EXAMPLE 5.5

Create a fractal whose initial curve list is { (3.2,0) (-3.2,0) (3.2,0) } and model construction has four segments. The first and forth segments are parallel with the horizontal axis while the second and third make an angle of 45 and -45 degrees. Draw the fractal for K=3.

SOLUTION

The initial curve can be created several ways. You could use CREATC by locating each point (pressing the cursor key), digitize them (pressing INS), and pressing ON . Then press ON agian. You could also define SPAR to be { (3.2,0) 2 1} and run MAKE , or you could enter the list on level one and store it directly into 'CURV'. The model construction can be defined by entering 45 SEG4 . The resultant fractal is shown in figure 5.26.

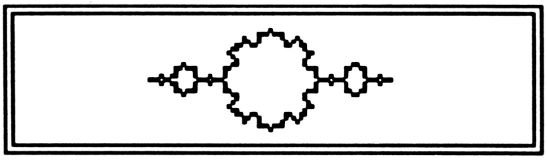


Figure 5.26 (K=3)

Fractal Drawing Menu

Menu Key	Operation
EDIT	This program will create the fractal editing menu, where you can edit the initial curve, model construction, or the number of replacements, K.
STAT	This is a quick way to check the status of the fractal parameters. Pressing this key will display the number of segments in the initial curve and model construction, and the number stored in K.
FDRW	Pressing this menu key will draw a fractal according to the parameters. An initial curve must be stored in CURV, a model construction in MODL, and the number of replacements you want to make in K.
ADVAN	This menu key creates the advanced graphing menu described in chapter six.
END	Pressing END in the fractal menu will return you to the main graphing menu.

Fractal Editing Menu

Menu Key	Operation
STOK	STOK stores the number from level one into the variable K.
CURV	This is where the initial curve list is stored. Pressing this menu key will return it to the stack.
CREAT	Using this program, you can create a freehand initial curve . A detailed description is given on page 159 and 160.
STOC	This key stores the list on level one into 'CURV'.
SHAPE	SHAPE creates the initial curve shape drawing menu. This menu lets you create a shape for an initial curve. Most all the keys and their functions are the same as the shape menu in chapter four.
DRAW	Pressing this menu key will draw the initial curve on the display.
MODL	This key will return the value of MODL, the variable containing the model curve. Its name will be returned if it isn't defined yet.
CREAT	Here's a handy program that lets you create a freehand model construction. A complete description is given on page 160 and 162.
STOM	STOM stores the list from level one into 'MODL'.
SEG2	Put an angle, in degrees, on level one and this program creates a two segment model construction using the outline shown in figure 5.8.
SEG4	Put an angle, in degrees, on level one and SEG4 creates a four segment model construction using the outline shown in figure 5.9.
DRAW	Pressing this menu key will draw the model construction on the display.
END	Pressing END in the editing menu will return you to the fractal drawing menu.

Initial Curve (Shape) Editing Menu

Menu Key	Operation
SPAR	Pressing this menu key will return the shape parameters. They read { (radius, starting angle), number of points, number of revolutions }
RADUI	This program puts the number from level one into the real part of the complex number in SPAR. The number on level one must be a real number.
THETA	This program puts the number from level one into the imaginary part of the complex number in SPAR. The number on level one must be a real number.
POINT	POINT puts the real number from level one into the second position in SPAR. This number must be a positive integer.
REV	This program puts the positive real integer from level one into the second position in SPAR.
MAKE	This program creates the shape defined by SPAR and stores it, as a list of points, in CURV.
END	This program returns you to the main graphing menu.

CHAPTER SIX

ADVANCED GRAPHING

SECTION ONE

REFERENCE SECTION

INTRODUCTION

This is the reference section for the advanced graphing programs. These programs enhance the graphing programs from chapters two through five, the built in program DRAW, or your own graphing programs. Before starting this chapter you should flip through section two, the example section. It will give you an idea of what to expect. Then, if you like what you see, you can jump right in, but be sure you've entered the programs from chapter one first.

There are three different advanced graphing programs with their own graphing menus. DDRW transforms most any graphing program into three dimensions. MDRW, the animated grapher, brings your drawings to life. Finally, EDRW lets you create any size graph.

An advanced graphing program doesn't actually do any graphing. It simply drives a root level program a given number of times, manipulating the input and output for the root level program each time. For example, If you wanted to create an animated graph of the function 'Z=X^2*T' you would tell MDRW, the animated grapher, to use DRAW as a root level program and vary T over a given range. MDRW would then input a different value for T, run DRAW, and store the graphics for the display a number of times. What you end up with is a list of graphic strings that, when displayed in series, show the function as it changed with time. DRAW did the actual graphing while MDRW manipulated the data.

A table of all the programs you'll need in this chapter is given below. All these programs should be stored in the main graphing directory, GRAPHP (see page 22). an asterisk after a program name indicates it is listed in a different chapter. You should flip to the indicated page and check if that program is in your calculator's memory.

PROGRAM TABLE

Program	Page	Bytes	Program	Page	Bytes
ADD	181	85.0	MCHK	207	142.5
ADVAN	182	56.0	MDRW	208	263.5
CHL$T	183	32.5	ML$T	210	219.5
CTR?	184	76.5	MMENU	211	39.5
DDRW	185	274.5	MOVE	212	37.0
DEL	188	77.5	MPAR	213	23.5
Dim	189	36.0	MRCL	214	106.5
DPAR	190	23.5	MRKR	215	39.0
EDRW	191	398.0	MTYP	216	66.0
end	194	16.0	N→FG	217	67.0
ENd*	69	16.0	PBAK	218	83.5
EPAR	195	21.0	PCHK*	41	28.5
EXIT	196	34.5	PGET	219	58.5
EXTND	197	38.0	Prog	220	36.0
FILE	198	68.5	PROG	221	36.0
FRAM	199	38.5	PUTL	222	48.5
GFIL	200	16.0	PUTP*	45	33.5
GGFL	201	53.0	$AVE	223	92.0
GLIDE	202	73.0	$HAKE	224	108.5
KEYW	203	28.5	$tep	225	87.0
LGET	204	52.0	VIEW	226	210.0
LINE$	205	17.0	XFRM	227	38.5
LOAD	206	38.5	YFRM	228	38.5

A table of the programs needed for each graphing menu is given on the next page. If you don't plan on using all three menus you can store only the programs you need using these tables. Because many of the programs are in more that one list though, adding a graphing menu may require only a few more programs.

PROGRAMS USED FOR THE THREE DIMENSIONAL MENU

ADD	CTR?	DDRW	DEL	Dim
DPAR	end	ENd	FRAM	LINES
MLST	MMENU	MRCL	MRKR	MTYP
N→FG	PCHK	PGET	PROG	PUTL

PROGRAMS USED FOR THE EXTENDED MENU

ADVAN	CHLST	CTR?	EDRW	end
ENd	EPAR	EXIT	EXTND	FILE
GFIL	GGFL	KEYW	LGET	LOAD
MCHK	MLST	MMENU	MRKR	N→FG
PCHK	PGET	Prog	PUTP	SAVE
VIEW	XFRM	YFRM		

PROGRAMS USED FOR THE ANIMATED MENU

ADD	ADVAN	CHLST	DEL	end
ENd	EXIT	FILE	FRAM	GFIL
GGFL	GLIDE	KEYW	LGET	LOAD
MCHK	MDRW	MLST	MMENU	MOVE
MPAR	MRCL	MRKR	MTYP	N→FG
PBAK	PCHK	PGET	PROG	PUTL
SAVE	SHAKE	Step		

Add a Variable List

ADD(251496)

```
« IF DUP TYPE 5 ≠
THEN 3 →LIST END 1
→LIST PGET PCHK DUP
RCL ROT + SWAP STO »
```

SUMMARY

ADD will accept a list, or three objects as arguments. Each case is shown under the heading input . Depending on the value of the first four user flags, this program adds a variable list to either MPAR or DPAR. A variable list contains a variable and two real numbers. The two real numbers specify the range for the variable. The listings for MDRW and DDRW explain how a variable list is evaluated in each case.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The first four flags are used to detect which advanced graphing menu the user is in.
CASE A	LEVEL ONE	A list
CASE B	LEVEL THREE	A variable name
	LEVEL TWO	A real number
	LEVEL ONE	A real number
OUTPUT	MEMORY	If flags one through three equal binary 2, #0010 b, a list containing a variable and two real numbers will be added to DPAR, otherwise, the list will be added to MPAR.
UTILITIES		PGET, PCHK

Create the Advanced Graphing Menu

ADVAN(51250)

```
« { Dim EXTND MOVE
ENd } MENU »
```

SUMMARY

This program creates the advanced graphing menu. This menu, like the main graphing menu, is a selection menu. You have the option of selecting one of the three advanced graphing sub-menus. Pressing the key labeled DIM EXTND or MOVE creates the three dimensional, extended, or animated graphing menu respectively. Pressing END puts you back in the calling graphing menu.

PATH	{ HOME GRAPHP }
INPUT	None
OUTPUT MEMORY	A custom menu is created.
UTILITIES	None

Mode Character List

CHLST(15147)

```
{ "E" "M" "D" }
```

SUMMARY

CHLST is simply a list of characters. By adding one of these characters to a variable name, a program can classify its data. Many of the programs in this chapter look for this added character to distinguish the type of data stored in a variable.

One of the programs that uses CHLST is PGET. PGET uses it to get the quoted name of a specific graphing parameter. It converts the first four user flags from a binary number to a real number. It then adds the character in CHLST pointed to by this real number and PAR. For example, if the first four user flags equaled binary one, PGET would return the name 'EPAR'. Likewise, if the first four flags equaled binary three PGET would return 'DPAR'.

You'll notice CHLST only has three character strings but the binary value of the first four user flags can be as high as fifteen. This means you can incorporate many of the programs in this chapter into your own programs by adding your own characters to CHLST. The listings for MRKR, N→FG, PGET, and LGET might give you some ideas.

PATH { HOME GRAPHP }

Calculate the Center of the Display

CTR?(113108)

```
« 1 *W PPAR LIST→ 4
DROPN DUP2 - IM 31 /
0 SWAP R→C + + 2 / »
```

SUMMARY

DDRW and EDRW call CTR? to compute the center point on the display. This point is used as a reference point when shifting the display and restoring the plot parameters (PPAR) to it's original value.

First, if PPAR doesn't exist, CTR? creates it. Then, the minimum and maximum points in PPAR are used to determine the center of the display. Finally, this point is returned to level one.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	If it doesn't exist, 'PPAR' is created. Then the first and second objects from this list are used to determine the center point on the display.
OUTPUT	LEVEL ONE	The center point of the display (a complex number).
UTILITIES		None

Three Dimensional Drawing Program

DDRW(2733704)

```
« MAXR →NUM DUP R→C
AXES CLLCD CTR? DPAR
1 GET LIST→ DROP
1 - → ctr p s « 0 s
FOR j DPAR 2 DO GETI
LIST→ DROP OVER - s
/ j * + DUP .7 * 30
IF 60 FS? THEN D→R
END R→C P→R ctr +
CENTR SWAP MTYP
UNTIL 46 FS? END
DROP2 17 SF p EVAL
NEXT ctr CENTR » »
```

SUMMARY

DDRW takes the parameters from DPAR and creates a three dimensional graph. DPAR read as follows.

> { { any graphing procedure, a real number}
> { first variable or program , first upper
> limit, first lower limit}. . . . . { Nth variable
> or program, Nth upper limit, Nth lower
> limit }}

DDRW uses the first item of the first list as its graphing procedure. This is the program that will be used to draw the three dimensional graph. The second number in the first list is the number of times to evaluate this procedure. A cross section is drawn each time the procedure is evaluated. All other lists are variable lists.

A variable lists should have a variable or a program as the first object and two real numbers as the second and third objects. The two real numbers define the range that the variable will vary over or the program will be evaluated over. The range of the last list is used as the range for the axis coming out of the display. Although you would normally only have one variable list, DPAR can have as many lists as you want. Before drawing a cross section, DDRW takes each variable list and adds

> [(the upper limit - the lower limit)/
> the number of cross sections being
> drawn]

to the lower limit and puts this number on level one of the stack. It then takes the first object of that variable list and, if it is a variable, stores the number from level one in it, or, if it is a program, evaluates it.

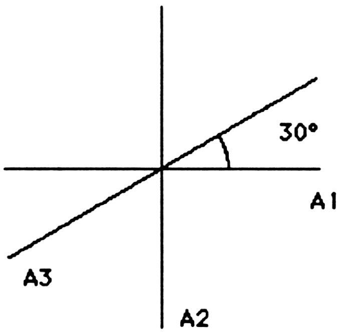
As an example, A three dimensional sine wave will be graphed if the following objects are stored in the listed locations

```
DPAR     {{«DRAW» 6}{Z -1.5 1.5 }}

EQ       'Z*SIN(X)'

PPAR     {(-6.8,-1.5)
         (6.8,1.6) X 1 (0,0) }
```

In two dimensions, the axis coming out of the display (A3) is about .7 times the length of the other two axis and makes a 30 degree angle with the positive A1 axis. By shifting the display screen along this line each time a cross section is drawn DDRW creates a three dimensional graph.



If you stop the program while its running you'll find that the plot parameters, PPAR, have been changed. This is because DDRW actually shifts the origin every time a cross section is drawn. This is what creates the 3-D effect. Normally, PPAR is restored to its original value, but if the program is stopped during a run you'll have to check these parameters and change them or purge them.

NOTE

DDRW sets flag 17 before it runs a drawing program. All of the drawing programs in chapters two through five check the status of flag 17. Having the flag set causes these programs to do nothing more than draw on the display. If flag 17 isn't set, the display won't be cleared and you'll enter digitizing mode each time a cross section is drawn.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The lists stored in PPAR and DPAR
OUTPUT	LCD	A three dimensional graph is drawn on the display screen.
UTILITIES		CTR?, MTYP

Delete a Variable List

DEL(286479)

```
« PGET DUP RCL DUP 1
OVER SIZE 1 - 1 MAX
SUB ROT STO DUP SIZE
2 MAX DUP SUB LIST→
DROP »
```

SUMMARY

DEL does just the opposite of ADD. It subtracts the last list from either MPAR or DPAR, depending on what menu you're in. If MPAR or DPAR doesn't contain a variable list nothing happens. MPAR or DPAR's default value will be stored in the current directory if it didn't exist.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	Flags one through four equal binary 2, MPAR is used, otherwise, DPAR is used.
OUTPUT	LEVEL ONE	If the specified parameter list contained at least one variable list it is returned to level one.
	MEMORY	Depending on the value of the first four user flags, the last variable list from either MPAR or DPAR will be subtracted. If it doesn't exist in the current directory, it is created using the default value.
UTILITIES		PGET

Create Three Dimensional Graphing Menu

Dim(17105)

```
« 3 N→FG MMENU »
```

SUMMARY

Dim stores binary three (# 0011 b) in the first four user flags. This tells other programs, such as FILE, LOAD, and EXIT, that you are working in the three dimensional menu. These flags also tell MMENU which advanced graphing menu to create, in this case the three dimensional menu.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list stored in MLST
OUTPUT	MEMORY	The user flags one through four are changed to binary three, # 0011 b. A custom menu is also created.
UTILITIES		N→FG, MMENU

Default DDRW Parameters

DPAR(14084)

```
{ { DRAW 9 } }
```

SUMMARY

DPAR is a list of parameters used by DDRW when drawing in three dimensions. It reads as follows.

> { { any graphing procedure, a real number}
> { first variable or program , first upper
> limit, first lower limit}. . . . . { Nth variable
> or program, Nth upper limit, Nth lower
> limit }}

DDRW uses the first item of the first list as its graphing procedure. This is the program that will be used for the three dimensional graph. The second number in the first list is the number of times it evaluates this program. A cross section parallel to the XY plane is drawn each time the program is evaluated. All other lists are variable lists.

A variable lists has a variable or a program as the first object and two real numbers as the second and third objects. The two real numbers define the range that the variable will vary over or program will be evaluated over. Although you would normally have only one variable list, DPAR can have as many lists as you want. Only the range of the last list though, is used as the range for the three dimensional (coming out of the display) axis.

This version of DPAR is stored in the main graphing directory, GRAPHP. It will be used as a default value when it isn't defined in the directory that you're in.

PATH { HOME GRAPHP }

Extended Draw Procedure
(Menu Driven)

EDRW(5456263)

```
« EPAR LIST→ DROP
ABS 1 - 2 / 18 FS?
.845 1 IFTE *W CTR?
PPAR LIST→ 4 DROPN -
C→R OVER 136 / ROT +
OVER 31 / ROT + R→C
NEG 4 ROLL ABS 1  -  2
/ 0 OVER  NEG ROT FOR
i DROP "" CR 4 PICK
DUP NEG FOR j  OVER
C→R j  * SWAP  i  *
SWAP R→C 4 PICK +
CENTR CLLCD 17  SF  5
PICK EVAL IF 18 FS?
THEN PRLCD ELSE LCD→
+  END -1  STEP NEXT
ROT CENTR IF 18 FS?
THEN DROP .845 INV
ELSE GGFL  LOAD 1 END
*W  3 DROPN »
```

SUMMARY

EDRW uses the list stored in EPAR to draw an extended graph. This list contains three objects. The first is a drawing procedure while the second and third are the number of horizontal and vertical frames you want graphed. You can change these parameters using Prog, XFRM and YFRM.

For example, if { DRAW 2 2 } is stored in 'EPAR' the equation in 'EQ' would be graphed on four display screens, two horizontal by two vertical.

Each time EDRW evaluated the program specified by EPAR a section of the graph is drawn and, if flag 18 is set, the graph is sent to the printer. Because the HP thermal printer doesn't print a line between graphics, a vertical series of frames will be printed. After printing an entire vertical strip EDRW prints two blank lines and shifts over one horizontal frame. It then starts graphing the next vertical strip. You can tape or paste these vertical strips together to get any sized graph.

If flag 18 is cleared EDRW will not send output to the printer. Instead, the graphic string for the last vertical strip is stored as the variable 'GSTRE' in the graphic file directory, GFIL. Since only the last vertical strip is saved, the second number in EPAR should be 1. Having a number larger than 1 will graph more vertical strips and just waste time.

Storing graphics in the the graphic file directory helps you keep track of memory hogging graphic strings. It also allows you to view a graph from any directory, no matter which directory the graphic was created in. See FILE, LOAD and SAVE for more information on the graphic file.

NOTE

If flag 18 is set (the graph is being sent to the printer) the plot parameters are changed before and restored after graphing. This is because the resolution of the printer isn't the same as the 28S display. The dots on the printer are tall rectangles while the pixels on the 28S are squares. Because of this you may have to edit or purge PPAR if you interrupt EDRW while running .

PATH		{ HOME GRAPHP }
INPUT	<u>LCD</u>	The graphic string representing each frame of the graph.
	<u>MEMORY</u>	The list stored in EPAR and PPAR, flag 18, and any input used by the drawing procedure in EPAR

OUTPUT	LCD	A number of graphs are drawn on the display.
	MEMORY	If flag 18 is set the graphic string representing the graph is stored in G$TRE in the graphic file directory.
	Printer	If flag 18 is cleared the graph will be sent to the printer.

UTILITIES CTR?, GGFL, LOAD

Return to the Advanced Graphing Menu

end(5672)

'ADVAN'

SUMMARY

end can be found in the extended, animated, and three dimensional graphing menu. It returns you to the advanced graphing select menu. Because lower case letters appear as upper case in the user or custom menu, it will be displayed as END.

PATH	{ HOME GRAPHP }
INPUT	None
OUTPUT MEMORY	The advanced graphing selection menu is created.
UTILITIES	None

Default EDRW Parameters

EPAR(10631)

```
{ DRAW 1 2 }
```

SUMMARY

EPAR is the list of parameters used by EDRW to draw an extended graph. It reads as follows.

{ any graphing procedure, a positive integer, a positive integer}

The first object is the drawing program used to create an extended graph while the second and third objects are the number of horizontal and vertical frames to be graphed.

This version of EPAR is stored in the main graphing directory, GRAPHP. It will be used as a default value when it isn't defined in the directory that you're in.

PATH { HOME GRAPHP }

Exit the Graphic File Directory and Return to Previous Directory

EXIT(14653)

```
« PBAK MMENU »
```

SUMMARY

EXIT can be found in the graphic file menu. It returns you to your previous directory and recreates the previous custom.

Whenever you enter the graphic file directory your current path is stored in 'Path' in the main graphing directory, GRAPHP. EXIT uses the list stored in Path to return you to the previous directory and recreate the calling menu.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The lists stored in Path, and MLST.
OUTPUT	MEMORY	The list stored in Path is made the current path and then purged. A custom menu is also created.
UTILITIES		PBAK, MMENU

Create Extended Graphing Menu

EXTND(19506)

```
« 1 N→FG MMENU »
```

SUMMARY

EXTND stores binary one (# 0001 b) as the first four user flags. This tells other programs, such as FILE, LOAD, and EXIT, that you are working in the extended graphing menu. The extended graphing menu is also created.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list stored in MLST
OUTPUT	MEMORY	User flags one through four are changed to binary one (# 0001 b) and the extended graphing menu is created.
UTILITIES		N→FG, MMENU

Create Graphic String File Menu

FILE(83107)

```
« GGFL { LOAD SAVE
EXIT } VARS MCHK +
MENU »
```

SUMMARY

FILE is used to load, save, delete, or manipulate graphics created by EDRW and MDRW. It stores your current path in 'Path' in the main graphic directory, GRAPHP, and moves to the graphic file directory. Then, it sorts through all the variable names in the graphic file and puts the ones created by the calling menu in a list. This is done by checking the last character of each name. Names ending with the letter E are selected when you're in the extended graphing menu while names ending in M as selected when in the animated(motion) menu. Finally, this list is added to { LOAD SAVE EXIT } and used to create the graphic file menu.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The first four user flags determine which character from the list stored in CHLST is used as a test character. Any variable stored in the graphic file directory that ends with this test character will be used when creating the custom menu.
OUTPUT	MEMORY	The path before running FILE is stored in 'Path' in the main graphic directory. A custom menu with all the names in GFIL ending with the test character is also created.
UTILITIES		GGFL, MCHK

Input Number of Frames

FRAM(20232)

```
« IP 2 PGET PUTL »
```

SUMMARY

FRAM puts the integer part of the real number from level one into the second position in the first list of the current advanced graphing parameter list. This command only appears in the animated menu, but is used under the name LINES in the three dimensional menu.

PATH		{ HOME GRAPHP }
INPUT	LEVEL ONE	A real number
OUTPUT	MEMORY	If flags one through four equal binary two or three the real number from level one is put into either MPAR, or DPAR.
UTILITIES		PGET, PUTL

<u>Graphic File Directory</u>

GFIL(1297)

DIRECTORY

SUMMARY

Graphics created by both EDRW and MDRW are stored in the graphic file directory, GFIL. You can distinguish what programs created which graphics by the last letter in their name. The character E is added to the name of any graphic string that was created or stored while in the extended graph menu EXTND. Similarly, you'll find the letter M appended to the name of lists that were stored or created by any program from the animated graph menu MOVE.

The menu driven program that puts you in GFIL is FILE. It also creates the graphic file menu that allows to to save or load graphics. Refer to the program listings for FILE, LOAD, SAVE, and EXIT for further explanation on the graphic file.

PATH { HOME GRAPHP }

Store Current Path and go to Graphic File Directory

GGFL(47471)

```
« PATH GRAPHP 'Path'
STO GFIL »
```

SUMMARY

GGFL recalls the current path and stores it in 'Path' in the main graphing directory. Then, it gives control to the graphic file directory, GFIL. Later, when you exit out of the graphic file, the program PBAK will use the list stored in 'Path' to return control to the previous directory.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The current path as returned via the built in command PATH
OUTPUT	MEMORY	The variable 'Path' is created and stored in the main graphing directory and command is transferred to the graphic file directory.
UTILITIES		None

Glide View an Animated Graph

GLIDE(183200)

```
« GGFL 'GSTRM' 1 DO
GETI EVAL →LCD UNTIL
KEY END 3 DROPN PBAK »
```

SUMMARY

GLIDE is one of several programs that view an animated graph created by MDRW. It goes to the graphic file directory, GFIL, takes the list stored in 'GSTRM' and displays each graphic string from first to last. It keeps doing this until any key is pressed. The effect, in most cases, is a sort of gliding motion. Once a key is pressed the program returns control to the calling directory and stops. You'll get an error if a list of graphic strings isn't stored in 'GSTRM'. If this happens you will have to return to the main graphing menu and reselect the application menu you were working in by entering EXIT END END.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list stored in 'GSTRM'
	KEYBOARD	Pressing any key will end the program.
OUTPUT	LCD	The graphic strings stored in 'GSTRM' are displayed on the screen. The images continually cycle from first to last.
UTILITIES		GGFL, PBAK

Wait for a Key to be Pressed

KEYW(33289)

```
« DO UNTIL KEY END
»
```

SUMMARY

KEYW simply suspends a program while waiting for a key to be pressed. Pressing any key will return its string to level one. Because this program is so useful, you may decide to store it in the home directory instead of the main graphing directory, where it can be used by all your programs.

Be careful when using this program. If you leave your calculator unattended it will continue to run until the batteries go dead. This may reset your memory!

PATH		{ HOME GRAPHP }
INPUT	KEYBOARD	The program waits for you to press any key.
OUTPUT	LEVEL ONE	The string for the key that was pressed is returned to level one.
UTILITIES		None

Get graphic String Variable

LGET(53291)

```
« "'GSTR" CHLST MRKR
GET + STR→ »
```

SUMMARY

Both EDRW and MDRW store graphics in the graphic file directory under the name 'GSTRE' or 'GSTRM'. LGET creates the name where the graphic string or list will be stored. If you were in the animated graphing menu 'GSTRM' will be returned. Likewise, 'GSTRE' will be returned if you were in the extended graphing menu.

It does this by adding the strings "'GSTR", and the *mode character*. LGET generates the *mode character* by converting the first four user flags from binary to a decimal number (this is done by the subprogram MRKR) and getting the character in that position from the list in CHLST. Finally, the resultant string is converted to a name.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list in CHLST and the first four user flags are used to generate the *mode character*.
OUTPUT	LEVEL ONE	Either 'GSTRE' or 'GSTRM' will be returned depending on whether you're in the extended or animated graphing menu.
UTILITIES		MRKR

Input the Number of Lines into DPAR

LINES(5884)

'FRAM'

SUMMARY

LINES puts the object from level one into the second position in the first list in DPAR. This program is in the three dimensional menu. The number put into DPAR represents the number of cross sections, or lines, that will be drawn in a three dimensional graph. All this program really does is give FRAM, the program doing all the work, a second name. This second name does a better job describing what is really happening when it is used in the three dimensional graph menu.

PATH		{ HOME GRAPHP }
INPUT	LEVEL ONE	A real number
OUTPUT	MEMORY	The second number in the first list in DPAR is changed. If it doesn't exist in the directory you are in, DPAR is created.
UTILITIES		FRAM

Load a Graphic String or List

LOAD(27545)

```
« EVAL LGET STO EXIT »
```

SUMMARY

LOAD stores the object on level one into a specific graphic variable name. The names of the graphic variable for the animated and extended graphing menus are 'G$TRM' and 'G$TRE' respectively. Once a graph is loaded into the appropriate variable it can be viewed using programs like VIEW, GLIDE, or $HAKE. Finally, control is returned to the directory and menu you were previously working in.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The mode character list, CHL$T, as well as the first four user flags are used to get the last character in the graphic variable name.
OUTPUT	MEMORY	A variable whose name is a combination of G$TR and one of the characters from CHL$T. For example, if the first four user flags equaled binary two, then the second character from CHL$T would be added to G$TR to create the name 'G$TRM'.
UTILITIES		LGET, EXIT

Check for Mode Character

MCHK(877220)

```
« 0 + { } CHLST MRKR
GET ROT 1 DO GETI IF
DUP →STR DUP SIZE 1
- DUP SUB 5 PICK
SAME THEN 5 ROLL + 4
ROLLD ELSE DROP END
UNTIL 46 FS? END 3
DROPN »
```

SUMMARY

When you enter an advanced graphing menu a distinct binary number is stored in the first four user flags. This binary number is used to select the *mode character* from CHLST. The mode character for a given advanced graphing menu is added to any name loaded (stored) into the graphic file directory. When you enter the graphic file menu MCHK check the last character of every variable in the list on level one. If the last character matches the mode character it is added to a new list, otherwise, it is dropped. The resulting list is returned back to level one and will be used to create a custom menu.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The first four user flags, as well as the list stored in CHLST are used to select the *mode character.*
	LEVEL ONE	A list of all the user variables in GFIL
OUTPUT	LEVEL ONE	A list containing names with the proper *mode character* in the last position.
UTILITIES		MRKR

Animated (Move) Graphing Program

MDRW(2557821)

```
« MPAR 1 GET LIST→
DROP 1 - → p s « 0 s
FOR j CLLCD MPAR 2
DO GETI LIST→ DROP
 OVER - s / j * +
SWAP MTYP UNTIL DUP
1 == END DROP2 17 SF
p EVAL IF 18 FS?
THEN CR PRLCD CR
ELSE LCD→ END NEXT
IF 18 FC? THEN s 1 +
→LIST GGFL LOAD END
» »
```

SUMMARY

MDRW creates an animated graph using the parameter list stored in MPAR. An animated graph is nothing more than a series of frames in which one or more variable is changed in each graph. Then, if flag 18 is set, each graph is printed on the thermal printer. Otherwise, the graphic string for each graph is stored in a list that can be viewed later. The parameters in MPAR are as follows.

{ { graphing procedure, number of frames} { first variable or program , first upper limit, first lower limit}. . . . . { Nth variable or program, Nth upper limit, Nth lower limit }}

MDRW uses the first item of the first list as its graphing procedure. The second number in the first list is the number of times it evaluate this procedure. All other lists should have a variable or a program as the first entry, its upper limit as the second entry, and the lower limit as the third entry. You can have as many lists as you want, but MDRW require you to have at least two (The first with a graphing procedure and a positive integer and the second containing the vary parameters). MDRW adds

[(the upper limit - the lower limit)/ the number of graphs being drawn]

to the lower limit of each variable list (any list other than the first) and puts this number on level one of the stack. It then takes the first object in that variable list and, if it is a variable, stores the number, or if it is a program, evaluates it. As an example, assume the following objects are stored in the listed locations.

```
MPAR    {{«DRAW» 6}{Z -1.5 1.5 }}
EQ      'Z*SIN(X)'
PPAR    {(-6.8,-1.5)
        (6.8,1.6) X 1 (0,0) }
```

MDRW will graph the sine function six times. Assuming that 'Z' is a variable and not the name of a program, Z is changed each time a new graph is drawn. Viewing the graph using GLIDE will display a sine wave whose amplitude varies from -1.5 to 1.5.

MDRW can either store each frame of a graph in a list to be viewed using one of the programs in the animated graphing menu, or it can send each frame to the printer. If flag 18 is cleared, MDRW will store a list of graphic strings in 'GSTRM' in the graphic file directory. If flag 18 is set, each frame will be printed on the thermal printer.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list stored in MPAR and PPAR and flag 18
OUTPUT	LCD	A series of graphs are plotted.
	MEMORY	If flag 18 is cleared a list of graphic strings is stored in 'GSTRM' in the graphic file directory.
	PRINTER	If flag 18 is set each graph is sent to the printer.
UTILITIES		MTYP, GGFL, LOAD

<u>Mode List for EXTND, MOVE and Dim Menus</u>

MLST(753499)

```
{ { EPAR Prog XFRM
YFRM VIEW EDRW FILE
end}{ MPAR PROG
FRAM ADD DEL MDRW
FILE GLIDE SHAKE
Step end } { DPAR
PROG LINES ADD DEL
DDRW end } }
```

PATH { HOME GRAPHP }

SUMMARY

MLST is used by MMENU when creating an advanced graphing menu. Depending on the binary value of the first four user flags, one of the lists in MLST is used to create a custom menu. Refer to MMENU for more information.

Create an Advanced Graphing Menu

MMENU(30197)

```
« MLST MRKR GET MENU
»
```

SUMMARY

Dim, MOVE and EXTND are the programs that create the three advanced graphing menus. These programs all change the value of the first four user flags and call MMENU. Depending on the value of these flags, MMENU selects one of the three lists from MLST to create the appropriate custom menu.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The first four user flags and the list in MLST
OUTPUT	MEMORY	A custom menu is created.
UTILITIES		MRKR

Create the Animated (Motion) Graphing Menu

MOVE(17798)

```
« 2 N→FG MMENU »
```

SUMMARY

MOVE stores binary two (# 0010 b) in the first four user flags. This tells programs such as FILE, LOAD, and EXIT that you are working in the animated graphing menu. These flags also tell MMENU which custom menu to create.

PATH	{ HOME GRAPHP }
INPUT MEMORY	The list stored in MLST
OUTPUT MEMORY	The first four user flags are changed to binary two, # 0010 b and a custom menu is created.
UTILITIES	N→FG, MMENU

Default MDRW Parameters

MPAR(14188)

```
{ { DRAW 5 } }
```

SUMMARY

MPAR, The animated drawing parameter list, is used by MDRW. It reads as follows.

{ { graphing procedure, number of frames} { first variable or program , first upper limit, first lower limit}. . . . . { Nth variable or program, Nth upper limit, Nth lower limit }}

These parameters can be changed using PROG, FRAM, ADD, or DEL.

This version of MPAR is stored in the main graphing directory where it is used as a default when it isn't defined in the work directory.

PATH { HOME GRAPHP }

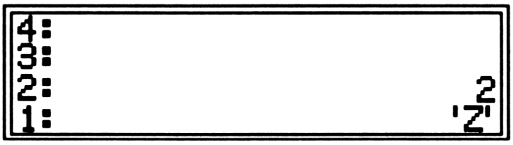
Check If Object is Program

MRCL(515072)

```
« IFERR DUP RCL THEN
IF 31 FS? THEN DROP
END STO ELSE IF TYPE
8 == THEN EVAL ELSE
STO END END »
```

SUMMARY

MRCL is called on by MTYP. It checks the object on level one and, if it is a program or the name of a program, runs it. Otherwise, it stores the real number on level two in that name. For example, if « LN 'X' $TO » is stored in 'Z' and the stack is as shown below, MRCL will store .693147 in X. If instead, 'Z' from level one is replaced with 'X' and a program isn't stored in X, 2 will be stored directly into X.



```
4:
3:
2:                  2
1:                'Z'
```

PATH		{ HOME GRAPHP }
INPUT	LEVEL TWO	A real number
	LEVEL ONE	A name or a program
OUTPUT	MEMORY	If a program or the name of a program is on level one it is evaluated. Otherwise, the number from level two is stored in the name on level one.
UTILITIES		None

Convert the First Four User Flags to a Real Number

MRKR(26445)

```
« # Fh RCLF AND B→R »
```

SUMMARY

This program converts the binary value of the first four user flags to a decimal number. This number represents the current mode set by N→FG. If 1 is returned, the current graphing menu is the extended menu. Likewise, two and three will be returned for the animated and three dimension menus.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The first four user flags
OUTPUT	LEVEL ONE	A real number
UTILITIES		None

Check Object Type for Level One

MTYP(137332)

```
« { DROP EVAL MRCL }
{ 8 6 } 3 PICK TYPE
POS 1 + GET EVAL »
```

SUMMARY

This program is called on by both MDRW and DDRW. It checks what type of object is on level one. If it is a program or the name of a program MTYP evaluates it. Otherwise it should be a variable name, so the number from level two is stored into it. See MRCL for more details.

PATH		{ HOME GRAPHP }
INPUT	LEVEL TWO	A real number
	LEVEL ONE	A variable name or a procedure
OUTPUT	MEMORY	If a name is on level one and that name doesn't contain a procedure, the value on level two is stored in that name . If the name of a procedure or a procedure is on level one, that procedure is run.
UTILITIES		MRCL

Convert a Decimal Number to a Binary Number and Store in the First Four User Flags

N→FG(137134)

```
« RCWS 64 STWS SWAP
15 MIN R→B RCLF SRB
SLB + STOF STWS »
```

SUMMARY

This program converts the real number from level one to binary and stores it as the first four user flags. By doing this, the first four user flags act as a universal marker. This marker tells any program which advanced graphing menu you are working and can be recalled from any directory.

PATH		{ HOME GRAPHP }
INPUT	LEVEL ONE	Any real number
OUTPUT	MEMORY	Depending on the real number input on level one, the first four user flags are set or cleared.
UTILITIES		None

Return to Previous Directory

PBAK(215067)

```
« GRAPHP Path 1
'Path' PURGE DO GETI
EVAL UNTIL 46 FS?
END DROP2 CLMF »
```

SUMMARY

This program passes directory control along the list stored in 'Path', which should have been created by GGFL. PBAK is called whenever you are in the graphic file directory and want to return to the directory you were previously working in. After recalling the list in 'Path' it is purged.

You normally wouldn't call this program directly. The programs, LOAD, SAVE, and EXIT all use PBAK to return you to your active directory.

PATH	{ HOME GRAPHP }
INPUT MEMORY	The list stored in 'Path' in the GRAPHP directory
OUTPUT MEMORY	Control is passed to the last directory in 'Path' and 'Path' is purged.
UTILITIES	None

Get Parameter Name

PGET(61296)

```
« "'" CHLST MRKR GET
+ "PAR" + STR→ »
```

SUMMARY

PGET gets the name of the active advanced graphing parameter list. If you are in the animated menu MPAR will be returned. Likewise, EPAR or DPAR will be returned if you are in the extended or three dimensional menu.

It does this by adding a single quote ("'"), the *mode character*, and "PAR". The resultant string is then converted to a name and returned to level one. The mode character is the character in CHLST pointed to by the decimal value of the first four insert flags.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list in CHLST and the first four user flags are used to get the *mode character*.
OUTPUT	LEVEL ONE	The name that contains the calling menu's graphing parameters.
UTILITIES		MRKR

Define Drawing Program for EPAR

Prog(17906)

```
«1 PGET PUTP »
```

SUMMARY

Prog takes the object from level one and puts it into the first position in the extended graphing parameter list, **EPAR**.

Prog doesn't check to see if the object on level one is a program or the name of a program. You'll end up with an error when you try to create a graph if you accidentally put any other object into the graphing parameters.

PATH		{ HOME GRAPHP }
INPUT	LEVEL ONE	A program or name of a program
OUTPUT	MEMORY	The object on level one is put into the first position in **EPAR**.
UTILITIES		PGET, PUTP

Define Drawing Program for MPAR or DPAR

PROG(16030)

```
«1 PGET PUTL »
```

SUMMARY

PROG takes the object from level one and puts it into the current advanced graphing parameter list. This program can be found in both the three dimensional and animated graphing menus. Simply put a graphing procedure or the name of a graphing program on level one and PROG will insert it into the graphing parameters for that menu. The first four user flags indicate which parameter list, MPAR or DPAR, it should be put in.

PROG doesn't check to see if the object on level one is a program or the name of a program. You'll end up with an error when you try to create a graph if you accidentally put any other object into the graphing parameters.

PATH		{ HOME GRAPHP }
INPUT	LEVEL ONE	A program or name of a program
	MEMORY	The first four user flags determine which list, MPAR or DPAR, will be changed.
OUTPUT	MEMORY	The object on level one is put in the first position in the first list in MPAR or DPAR. The the decimal value of the first four user flags determine which list will be changed.
UTILITIES		PGET, PUTL

Put an Object in a List Within a List

PUTL(75229)

```
« PCHK 1 DUP2 GET 4
ROLL 5 ROLL PUT PUT
»
```

SUMMARY

Both DPAR and MPAR contain a list of lists. The first list is the most important. It contains a drawing program and a real number. PUTL allows you to change either object in this list by putting the object from level three into the first list on level one. The real number on level two specifies the position it should be put in.

PATH		{ HOME GRAPHP }
INPUT	LEVEL THREE	A real number, program, or the name of a program
	LEVEL TWO	A real number
	LEVEL ONE	A parameter name. (Either DPAR or MPAR)
OUTPUT	MEMORY	The object input on level three is put into the first list from the graphing parameters on level one.
UTILITIES		PCHK

Save a Graphic String or List

$AVE(325108)

```
« CHLST MRKR GET
SWAP →STR 1 OVER
SIZE 1 - SUB SWAP +
STR→ LGET DUP RCL
ROT STO PURGE EXIT »
```

SUMMARY

$AVE allows you to save graphics created by EDRW or MDRW. Both EDRW and MDRW store their graphs in the graphic file directory in 'G$TRE' and 'G$TRM' respectively. If you would like to save one of these graphs under a different variable name, simply put the new name on level one and run $AVE.

For example, if you create a graph while in the extended graph menu and want to store that graph in 'FCN', you would enter 'FCN' on level one and press $AVE. G$TRE would be purged while the graphic string that was stored there will be stored in 'FCNE'. ($AVE adds the E to indicate it is an extended graph.)

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list in CHL$T, the first four user flags, and the graphics in either 'G$TRE' or 'G$TRM' are used.
	LEVEL ONE	Any name
OUTPUT	MEMORY	Either 'G$TRM' or 'G$TRE' is purged while its graphics are stored in a variable whose name is a combination of the name input on level one and either M or E.
UTILITIES		MRKR, Exit

View a Graphic List Created By Move (Back and Forth)

SHAKE(719111)

```
« GGFL GSTRM DO DUP
LIST→ 1 SWAP START
EVAL →LCD NEXT 1 DO
GETI  EVAL →LCD UNTIL
46  FS? END DROP
UNTIL KEY END DROP2
PBAK »
```

SUMMARY

SHAKE is one of several programs that view an animated graph created by MDRW. It goes to the graphic file directory, takes the list of graphic strings stored in GSTRM and displays each string from first to last. It then displays the images from last to first. It keeps doing this until any key is pressed. The effect is a shaking motion.

You'll get an error if a list of graphic strings wasn't created first by MDRW. If this happens you'll have to return to the main graphing menu and reselect the graphing menus you were working in.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list stored in 'GSTRM' in the graphic file directory.
	KEYBOARD	Pressing any key will end the program.
OUTPUT	LCD	The list of graphic strings stored in 'GSTRM' is displayed on the screen.
UTILITIES		GGFL, PBAK

View a Graphic List Created By Move
(One Step at a Time)

Step(249516)

```
« GGFL 'GSTRM' 1 DO
GETI EVAL →LCD UNTIL
KEYW "ENTER" SAME
END DROP2 PBAK »
```

SUMMARY

Step is one of several programs that let you view an animated graph created by MDRW. It goes to the graphic file directory, takes the list stored in 'GSTRM' and displays the first graphic string in that list. Pressing any key other than ENTER will display the next string. When the last string has been reached, Step cycles back to the first. This lets you view your graph one frame at a time. Pressing ENTER at any time will exit the program.

You'll get an error if a list of graphic strings wasn't first created by MDRW. If this happens you'll have to return to the main graphing menu and reselect the graphing menus you were working in.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The list of graphic strings in GSTRM stored in the graphic file directory, GFIL.
	KEYBOARD	Pressing ENTER will end the program. Any other key displays the next graphic string in GSTRM .
OUTPUT	LCD	The graphic strings stored in GSTRM are displayed, in series, on the screen.
UTILITIES		GGFL, PBAK, KEYW

View a Graphic String Created By EDRW

VIEW(1323245)

```
« GGFL GSTRE DUP DUP
→LCD SIZE 547 - SWAP
1 DO { -137 137 } {
"UP" "DOWN"} KEYW
POS IF DUP THEN GET
+ 1 MAX 3 PICK MIN
DUP2 DUP 548 + SUB
→LCD 0 ELSE 5 DROPN
1 END UNTIL END CLMF
PBAK »
```

SUMMARY

This program allows you to view any graphic string created by EDRW. The program initially displays the first four lines of the graph stored in 'GSTRE' in the graphic file directory. To view down a line simply press the down cursor key (▼). Likewise, the up cursor key (▲) shifts the display back up a line. Pressing any other key will exit the program.

PATH		{ HOME GRAPHP }
INPUT	MEMORY	The string stored in 'GSTRE' located in the graphic string directory, GFIL.
	KEYBOARD	The up and down keys view the extended graph up and down one line respectively.
OUTPUT	LCD	Four lines of the graphic string in GSTRE are displayed.
UTILITIES		GGFL, KEYW, PBAK

Set the Number of Horizontal Frames

XFRM(20778)

```
« IP 2 PGET PUTP »
```

SUMMARY

This program puts the real number from level one into the second position in EPAR. EPAR will be created in the work directory if it didn't exist before running the program.

PATH		{ HOME GRAPHP }
INPUT	LEVEL ONE	A real number
	MEMORY	The list stored in EPAR
OUTPUT	MEMORY	The number from level one is put in the second position in EPAR. EPAR is created if it didn't already exist in the work directory.
UTILITIES		PGET, PUTP

Set the number of Vertical Frames

YFRM(20804)

```
« IP 3 PGET PUTP »
```

SUMMARY

This program puts the real number from level one into the third position in EPAR. EPAR will be created in the work directory if it didn't exist before running the program.

PATH		{ HOME GRAPHP }
INPUT	LEVEL ONE	A real number
	MEMORY	The list stored in EPAR
OUTPUT	MEMORY	The number from level one is put in the third position in EPAR. EPAR is created if it didn't already exist in the work directory.
UTILITIES		PGET, PUTP

SECTION TWO

EXAMPLES

INTRODUCTION

Advanced graphic techniques can add detail and style to almost any graphing program. Using the three dimensional graphing menu, you can transform most normal graphing programs into three dimensional graphers. Team it up with the extended graphing menu and you are no longer limited to a four line display. Finally, with the animated graphing menu, you can watch a graph as different parameters vary before your eyes, or simply enhance the presentation of a drawing.

ABOUT ADVANCED GRAPHING PROCEDURES

Up till now this book has used root level graphing programs. A root level graphing program does the actual plotting on the display. It will always contain the command PIXEL or a subprogram that contains this command.

In contrast, the advanced graphing programs don't do any graphing themselves. They must call on a root level graphing program to do the plotting for them. They only manipulate the input and output of the program doing the graphing. For instance, if you wanted to draw a three dimensional polar graph you would tell DDRW, to evaluate ADRW, the polar graphing program, in three dimensions. Likewise, if you wanted to draw a three dimensional shape you would tell DDRW to evaluate SDRW in three dimensions.

A root level graphing program may call on a subprogram to draw for it, but this doesn't make it an advanced graphing program. The drawing program used by an advanced graphing program must be input via its parameter list. In contrast, a root level program must have the drawing subprogram imbedded in its code.

THE ADVANCED GRAPHING SELECTION MENU

You can access any of the advanced graphing menus by pressing **ADVAN** from the main graphing menu as well as the polar, shape, fractal, and pie chart drawing menus. This creates the advanced graphing selection menu where you have the option to enter the three dimensional graphing menu, **DIM**, the extended graphing menu, **EXTND**, the animated graphing menu, **MOVIE**, or exit back to the previous menu, **EXIT**. Remember that all advanced graphing menus are subordinate to the calling menu. If you enter the extended graphing menu from the fractal drawing menu the 28S will still be in the fractal work directory. Likewise, if you enter the animated graphing menu from the shape graphing menu, you will still be in the shape work directory.

THREE DIMENSIONAL GRAPHS

A THREE DIMENSIONAL GRAPH

Almost any drawing program can become a three dimensional graphing procedure with DDRW. DDRW doesn't actually do any graphing. It positions the screen in different spots in three dimensional space and runs the graphing program you give it. A cross section is drawn each time the graphing program is run.

Drawing cross sections is one of the easiest ways to plot a function in three dimensions. A cross section of a function is drawn by setting one of the three variables equal to a constant and graphing the resultant two dimensional equation. For example, the cross section of the equation

X^2+Y^2+Z^2=4 Equation 6.1

passing through the plane Z=0 (this is the XY plane) can be drawn by substituting 0 for Z and plotting the simpler two dimensional equation

X^2+Y^2=4 Equation 6.2

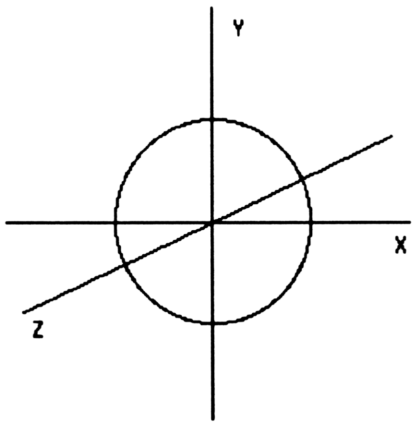


Figure 6.1
Cross Section of Equation 6.1
at Z=0

Plotting a series of cross sections for different values of Z will give us a three dimensional graph. The cross section of our example equation for the plane Z=0 is shown in figure 6.1. Drawing cross sections for different values of Z in equation 6.1 traces a three dimensional graph. Although there are only nine cross sections, the image of a sphere can easily be seen.

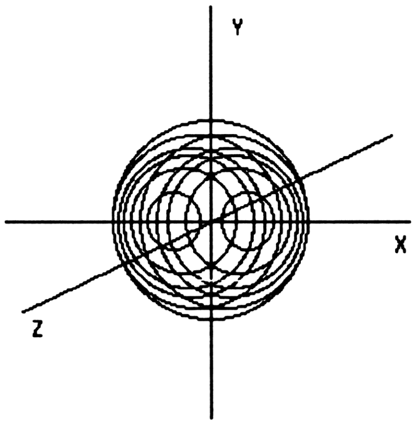


Figure 6.2
Nine Cross Sections of Equation 6.1

USING THE THREE DIMENSIONAL MENU

Exit any advanced graphing menu or press **ADVAN** from the menu you are in to go to the advanced graphing selection menu. Now press **DIM** in the selection menu to go to the three dimensional graphing menu. Simply specify the drawing program, number of cross sections, variable tied to the axis coming out of the display and the range of this variable and DDRW will create a three dimensional graph.

THE BASICS BEHIND DDRW

DDRW draws cross sections along the axis coming out of the display screen. In order to do this, it must some how represent this axis in two dimensions. DDRW assumes the image of this axis appears .7 times the length of the other two axis and makes a 30 degree angle with the positive A1 axis (See figure 6.3). By shifting the origin along this line each time a cross section is drawn, DDRW creates a three dimensional graph. It shifts the origin by redefining the maximum and minimum points in the plot parameters, PPAR.

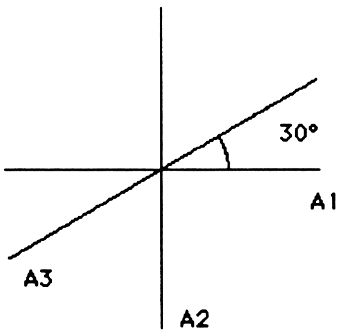


Figure 6.3

Normally, DDRW restores PPAR to its original value, but if the program is stopped during a run you'll have to check these parameters and change them or purge them.

THE THREE DIMENSIONAL PARAMETER LIST

The first key in the three dimensional menu is DPAR. This is the variable containing the three dimensional parameter list. DPAR reads as follows.

{ { any graphing procedure, a real number}
{ first variable or program, first upper
limit, first lower limit}. . . . . { Nth variable
or program, Nth upper limit, Nth lower
limit }}

Pressing DPAR when it isn't defined in the directory you're in will return { { DRAW 9 } } to the stack. This is its default value. It is used when DPAR hasn't been defined, as in this case.

You'll notice that DPAR contains lists within a list. The first list contains the graphing program to be evaluated in three dimensions and the number of times it should be evaluated. Each time the graphing program is evaluated a cross section is drawn.

The graphing program DDRW uses can easily be changed by putting either a program or the quoted name of a program on level one and pressing PROG. Likewise, the number of cross sections can be changed by entering a positive integer and pressing LINES.

VARIABLE LISTS

All lists other than the first are *variable lists* . They contain a variable name or a program and two real numbers. The two real numbers specify a range.

If the first object in the list is the name of a variable, DDRW divides the range up between the number of cross sections being drawn and stores the appropriate value in that variable. For example, the list { { DRAW 9 } { Z -2 2 } } tells DDRW to run the drawing program DRAW 9 times and vary Z from -2 to 2. Each time it runs DRAW a new value is stored in 'Z' and a new cross section is drawn -2 will be stored in Z for the first cross section, -1.5 for the second, -1 for the third, and so on, all the way up to 2.

If the first object in the variable list is a program or the name of a program, DDRW still divides the range up between the number of cross sections being drawn. But, rather than storing the appropriate value directly into a variable, it puts the specified program on the stack and runs it. For example, { { DRAW 9 } { « 2 / 'Z' STO » -2 2 } } tells DDRW to divide the range { -2 2 } up into 9 sections, -2, -1.5, -1, -.5, 0, .5, 1, 1.5, and 2. Before drawing each new cross section, the current value for that variable list is put on level two and the program « 2 / 'Z' STO » is evaluated. Only half the current value is stored in Z. This has the effect of stretching the Z axis. DDRW moves from -2 to 2 along the Z axis but stores half this in Z.

ADDING A VARIABLE LIST

There are two ways to add a variable list to DPAR. The easiest is to put the variable or program on level three, the upper limit on level two, the lower limit on level one and press ADD. You could also enter the entire variable list, { variable or program, upper limit, lower limit

}, on level one and press ADD. Either way, a new variable list will be added to DPAR.

DEL does just the opposite. It subtracts the last list from DPAR and returns it to level one. If {{ DRAW 9 }{ Z -1 1 }{ W -3 3 }} is stored in DPAR, pressing DEL will return { W -3 3 } to level one and {{ DRAW 9 }{ Z -1 1 }} will be stored in DPAR. If DPAR doesn't contain a variable list nothing will happen.

MULTIPLE VARIABLE LISTS

DDRW has the ability to accept more than one variable list. It will independently compute the appropriate value in the given range for each list. If the first object in that list is the name of a variable, the value will be stored in that variable. If the first object is a program or the name of a program, the appropriate value will be put on level one and that program will be run. Although you can have as many variable lists as you want, only the variable or program in the last list will be assigned to the axis coming out of the display.

DEFINING THE INDEPENDENT VARIABLES

When you draw a two dimensional graph using the built-in command DRAW, you must define an independent variable. This is the variable that is tied to the horizontal axis. For example, if you were to graph A=sin(B) using DRAW, you would first store 'SIN(B)' into 'EQ'. When graphed, B would be considered the independent variable, while A would be the dependant variable, since its value depends on B. Similarly, a three dimensional graph must have an independent variable assigned to the axis coming out of the display, which I will call the Z axis.

The variable or program in the last list in DPAR will be tied to the Z axis. For example, if the equation A^2+2*B+C-SIN(D) is stored in EQ and { { DRAW 9 } { B -2 2 } { C 0 5 } { D -1 1 } } is stored in DPAR, DDRW will graph 9 cross sections. For each cross section, the appropriate values in the given ranges will be computed and stored in B, C, and D, but only the range for the last list will be tied to the Z axis. Using the range for D, DDRW will draw six cross sections while shifting the display screen from -1 to 1 along the Z axis. If you change the positions of the three variable lists so a different list is in the last position, the graph will be drawn using a different range for the Z axis and will look different.

DRAWING A SPHERE

Try drawing a sphere with a radius of 1.5 in 9 cross sections. Its equation is

$$X\hat{}2+Y\hat{}2+Z\hat{}2=1.5\hat{}2 \qquad (6.3)$$

As in two dimensional graphing, we must first isolate the dependent variable (The one tied to the horizontal axis). Isolating X we get X=±√(-Y^2-Z^2+2.25) . This equation has two roots. We can make use of a trick that lets DRAW graph both roots by setting each root equal to each other and storing the equation in 'EQ'. When DRAW sees an equal sign, it graphs both sides independently, so both roots will be graphed. Equate both roots and store them in 'EQ'

```
'√(-Y^2-Z^2+2.25)' [ENTER] [ENTER]
[CHS] [=] [ENTER] STEQ
```

Exit any advanced graphing menus and press DIM from the advanced graphing selection menu. You should always purge any existing parameters before you start defining new ones. Purge both DPAR and PPAR if they exist. The drawing program we need to use is DRAW and the number of cross sections is 9. These happen to be the default values. Pressing DPAR will return { { DRAW 9 } }, the values we want.

```
{ DPAR PPAR } [PURGE]
```

There are two independent variables, Y and Z. We could choose either one to be tied to the axis coming out of the screen, but Z will be used in this example. Because of the symmetry of X, Y, and Z in equation 6.3, the same graph will be drawn regardless of what variable you assign to each axis. Add the appropriate variable list for the dependent variable Z and the range -1.45 to 1.45 to DPAR by typing Z, -1.45, 1.45 DPAR . DPAR should now contain { { DRAW 9 } { Z -1.45,1.45 } }. This tells DDRW to evaluate the graphing program DRAW 9 times and to vary Z from -1.45 to 1.45. DDRW will divide the range into nine sections. Each time a new cross section is drawn a new value is stored into Z.

Now we must set the independent variable tied to the horizontal axis. This variable will be used by DRAW, not DDRW. The only variable left is Y. Set Y as an independent variable by entering 'Y' INDEP from the PLOT menu. A summary of what we've stored and how they relate to the different axis is given below.

'EQ'	'√(-Y^2-Z^2+2.25)=-√(-Y^2-Z^2 +2.25)'	Because X was isolated from equation 6.3 it is assigned to the vertical axis
'DPAR'	{ { DRAW 9 } { Z -1.45,1.45 } }	The second list assigns Z to the axis coming out of the display
'PPAR'	{ (-6.8, -1.5) (6.8, 1.6) Y 1 (0, 0) }	Y is assigned to the horizontal axis

Pressing DDRW , the three dimensional graphing program, with the equation for a sphere stored in 'EQ' and the the correct parameters stored in 'DPAR' will draw the graph shown on the next page.

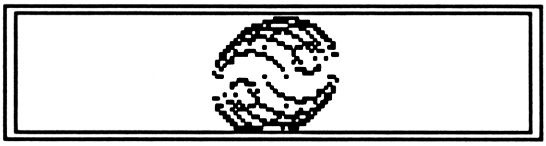


Figure 6.4

DRAWING A THREE DIMENSIONAL SINE WAVE

The cover of the 28S owner's manual, as well as most advertisements for the 28S, shows the graph of a sine wave on the display. This is because, without programming, the sine function is the most attractive graph. Lets transform this now ordinary plot into a three dimensional drawing.

Using the built-in command DRAW and the three dimensional grapher DDRW, draw the function Y=SIN(X) in 9 cross sections. The range of the three dimensional axis should be from -2 to 2. First, purge any old parameters and define new ones. Then, set your calculator to radians, store 'SIN(X)' into 'EQ' and run DDRW to create a 3-D sine wave.

{PPAR DPAR} PURGE MODE RAD 'SIN(X)' STEQ
'Z',-2,2 ADD DDRW

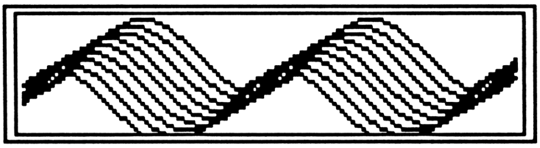


Figure 6.5

Notice how Z has been assigned to the axis coming out of the display but doesn't appear in the equation. We still needed to define a range for this axis.

ANOTHER SINE WAVE EXAMPLE

Graph the function Y=SIN(2*X)*Z in six cross sections. Assign Z to the three dimensional axis and vary it from -.2 to -1.5.

SOLUTION

This function has a period of two and the amplitude varies with Z. The following key sequence will create the graph shown in figure 6.6

```
{PPAR DPAR} PURGE MODE RAD 'SIN(2*X)*Z' STEQ
6 LINES 'Z',-.2,-1.5 HOO DDRW
```

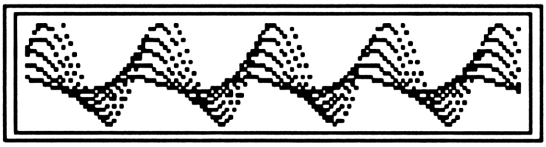


Figure 6.6

DIFFERENT AXIS ASSIGNMENTS

Any point in three dimensional space can be define by its relationship to the three perpendicular axis shown in figure 6.7.

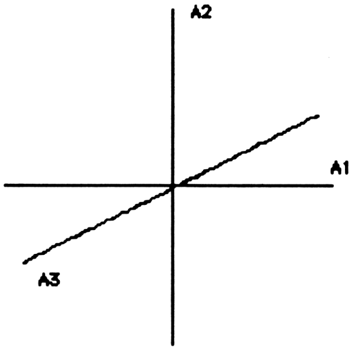


Figure 6.7
Three perpendicular axis

All the examples up till now have assumed that the X, Y and Z are assigned to A1, A2 and A3 respectively. Using this assignment, DDRW graphs cross sections parallel to the XY plane. In fact, most graphs you'll draw use this assignment. Quite often though, you may want to graph a function using cross sections parallel to other planes. This can easily be done by reassigning the variables to different axis. The six possibilities are show in figures 6.8 through 6.13.

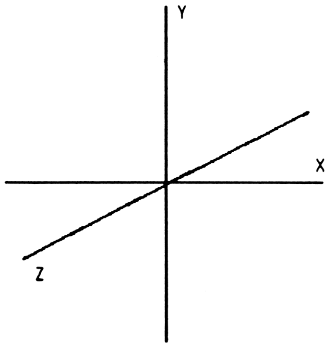


Figure 6.8
Cross section parallel
to XY plane

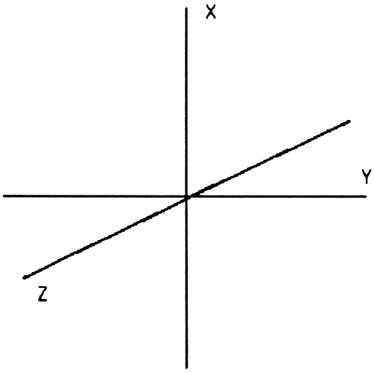


Figure 6.9
Cross sections parallel
to YX plane

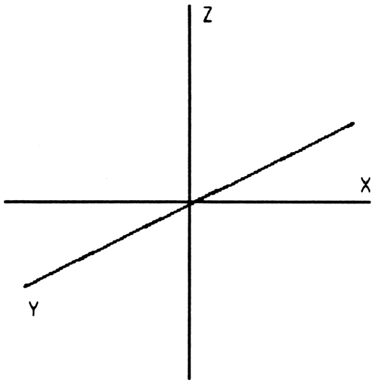


Figure 6.10
Cross section parallel
to XZ plane

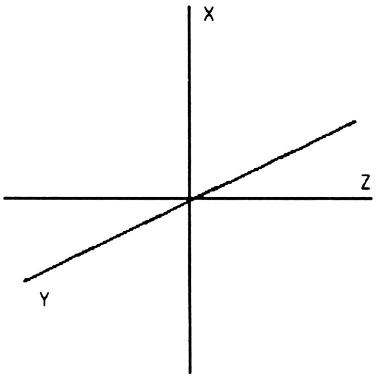


Figure 6.11
Cross sections parallel
to ZX plane

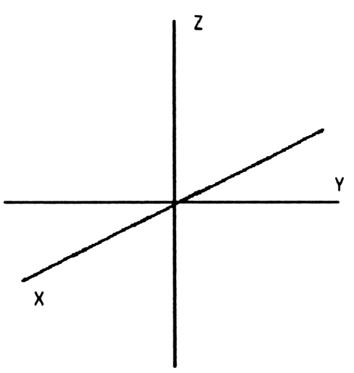


Figure 6.12
Cross section parallel
to YZ plane

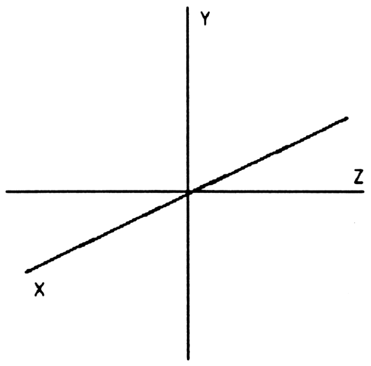


Figure 6.13
Cross sections parallel
to ZY plane

Although there are 6 different ways to assign 3 variables to the 3 axis (this doesn't take the direction of the axis into account), most functions are symmetric to one or more axis. This cuts down on the number of different views of a graph.

Equation 6.4 is an excellent example of a function that can be graphed using cross sections along different planes. Before doing anything, first purge old values in any variable by entering { EQ PPAR DPAR } PURGE . Isolate each of the three variable in equation 6.4 we obtain equations 6.5 through 6.7.

X^2+Y^2-Z^2=0 (6.4)

Y=±√(-X^2+Z^2) (6.5)
X=±√(-Y^2+Z^2) (6.6)
Z=±√(X^2+Y^2) (6.7)

Notice that, due to symmetry, switching the position of X and Y in equation 6.4 doesn't change the equation. Because of this symmetry, equations 6.5 and 6.6 will yield the same graphs. This means we can eliminate one and cut down the number of possible graphs.

Lets start by graphing equation 6.5 using the built in graphing program DRAW in 9 cross sections. To graph both roots we must set them equal to each other and store them in 'EQ'. DRAW will plot each side of this equality. This defines Y as the dependent variable tied to the A2 axis in figure 6.7.

'√(-X^2+Z^2)' ENTER ENTER CHS = ENTER STEQ

Now we must decide which of the two remaining independent variables, X or Z, will be tied to the A3 axis and the range it should be graphed from. Due to the limited display size, -2 2 is a good range to graph. Since switching X and Z in equation 6.2 will yield a different equation, assigning X to the A3 axis will give a different graph than having Z assigned to the A3 axis.

For our first graph lets assign Z to the A3 axis by adding the variable list { Z -2 2 } to DPAR. Finally assign X to the A1 axis by entering 'X' INDEP. This stores X as the third object in PPAR.

'Z',-2,2 ADD 'X' INDEP

The following should be stored in the specified variables.

AXIS	VARIABLE	VALUE
Y is A2 axis	'EQ'	√(-X^2+Z^2)=-√(-X^2+Z^2)
Z is A3 axis	'DPAR'	{{ «DRAW» 9 } { Z -2 2 } }
X is A1 axis	'PPAR'	{ (-6.8,-1.5) (6.8, 1.6) X 1 (0, 0) }

With the right values stored in the appropriate variables, DRAW will draw the graph shown in figure 6.14. Using this axis assignment, it is difficult to see what the graph really looks like. Lets try a different assignment and hope the resultant graph gives a better view.

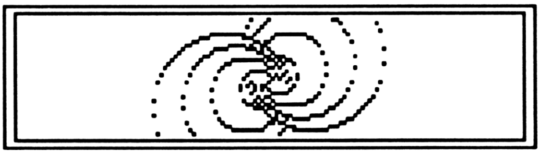


Figure 6.14

Using the same equation stored in 'EQ' (keeping Y as the dependent variable), we can draw a different graph by changing the axis assignments of X and Z. This means that we must switch Z in DPAR to X and X in PPAR to Z. Change the appropriate variables so they have the values shown below.

DEL DROP 'X',-2,2 ADD 'Z' INDEP

AXIS	VARIABLE	VALUE
Y is A2 axis	'EQ'	√(-X^2+Z^2)=-√(-X^2+Z^2)
X is A3 axis	'DPAR'	{{ «DRAW» 9 } { X -2 2 } }
Z is A1 axis	'PPAR'	{ (-6.8,-1.5) (6.8, 1.6) Z 1 (0, 0) }

Now press DRAW to draw the graph shown in figure 6.15. Although it is much easier to see what is going on, we can still try a different assignment in hopes that we will get yet a better view of the graph.

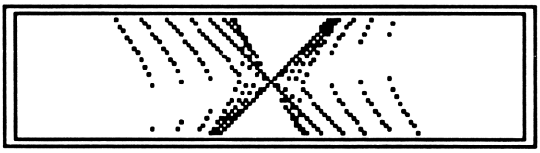


Figure 6.15

We still have equations 6.6 and 6.7 left to consider. For convenience, equations 6.4 through 6.7 are shown again.

X^2+Y^2-Z^2=0 (6.4)

X=±√(-Y^2+Z^2) (6.5)
Y=±√(-X^2+Z^2) (6.6)
Z=±√(X^2+Y^2) (6.7)

As stated earlier, due to symmetry, using equation 6.6 would give us the same two graphs we already have. This leaves us with equation 6.7. Following as we did earlier, equate both roots of this equation and store it into 'EQ'. The dependent variable Z, is now assigned to A2.

```
'√(X^2+Y^2)' [ENTER] [ENTER] [CHS] [=] [ENTER] STEQ
```

Now we need to decide which of the two independent variables, X or Y will be tied to the A1 axis and which will be tied to the A3 axis. But as we saw earlier, due to symmetry, switching X and Y will have no effect on the resultant graph. It doesn't matter how we assign X and Y to the two remaining axis. I have chosen Y for the A2 axis and X for the A3 axis. Again, the appropriate values for the specified variables are shown below. The resultant three dimensional graph created by DRAW is shown in figure 6.16.

```
'X',-2,2 ADD 'Y' INDEP DRAW
```

AXIS	**VARIABLE**	**VALUE**
Z is A2 axis	'EQ'	√(X^2+Y^2)=-√(X^2+Y^2)
X is A3 axis	'DPAR'	{{ «DRAW» 9 } { X -2 2 } }
Y is A1 axis	'PPAR'	{ (-6.8,-1.5) (6.8, 1.6) Y 1 (0, 0) }

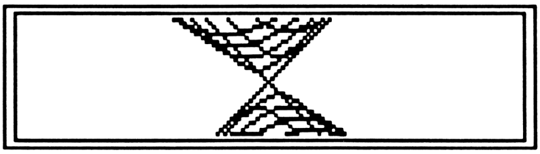


Figure 6.16

The combination of all three views gives us a good idea of what the graph of X^2+Y^2-Z^2=0 looks like. It is an hour glass. The two cones open towards the axis coming out of the display, in figure 6.14, the horizontal axis in figure 6.15, and the vertical axis in figure 6.16

SUMMARY OF AXIS ASSIGNMENTS

1) Check for symmetry and eliminate any possible axis assignments.

2) Graph the function for the remaining axis assignments.

When assigning variables to the three axis, A1, A2, and A3.

i) Isolate the equation with respect to the variable you want assigned to A2 (the vertical axis) and store the resultant equation in 'EQ'.

ii) Assign one of the two remaining variables to A1 (the horizontal axis) using **INDEP** in the PLOT menu.

iii) The remaining variable and two real numbers define the range for the remaining axis. They should be put in a list and added to DPAR. The variable will then be assigned to A3 (the axis coming out of the calculator).

A 3-D SADDLE

The three dimensional equation Z=-X^2+Y^2 looks much like a saddle. It flares up at the front and back while sloping down at the sides.

We can assign X,Y, and Z to any of the three axis, but assigning them to the horizontal, vertical, and protruding axis respectively gives us the best view. Isolating Y in the equation gives us two roots, Y=±√(X^2+Z). DRAW will graph both roots if we equate them and store the resulting equation in 'EQ'. The objects shown on the next page should have the listed values. You can set them using tools such as **STEP** and **LINE**, enter each value on the stack and store it directly into each name, or edit each object using VISIT. Finally, running DDRW will draw the graph shown in figure 6.17.

'EQ'	'√(SQ(X)+Z)=-√(SQ(X)+Z)'
'PPAR'	{ (-6.8,-1.5) (6.8,1.6) X 1 (0,0) }
'DPAR'	{ { DRAW 9 } { Z 2 -2 } }

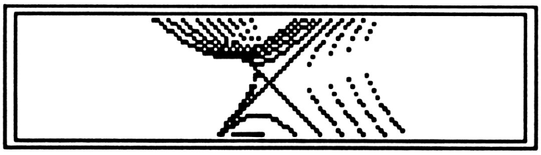


Figure 6.17

Try drawing the same equation using different axis assignments.

EXTENDED GRAPHS

INTRODUCTION

The 28S is a marvel to the calculator world. No other calculator can come close to its graphing ability. It laughs at undefined points as it graphs almost any equation. It accepts user defined functions and even programs as arguments. In all its glory though, it has one small drawback. It happily graphs anything you throw at it, but can only display 4 lines.

Your only solution to this dilemma is to shrink your graph to fit the display. A dwarfed plot can still give you a truckload of information, but in most cases the stumped resolution falls short. A larger screen would solve the problem, but alas we are stuck with a 4 line by 23 character display.

Don't go by a new calculator yet. There is a simple solution that can stretch your screen. Using the extended graphing program, EDRW, you can increase the size of any graph. Just sit back and watch EDRW plot any graph you tell it on more than one screen. The resultant graph can be stored in the calculators memory and viewed using VIEW, or printed on an optional thermal printer.

EDRW, like all the other advanced graphing programs, doesn't actually do any graphing. You must give it a graphing program and tell it how many horizontal and vertical display screens you want graphed. Starting with the top left screen, EDRW evaluates the graphing program you give it and either store the graphic strings representing each display or prints the displays to the thermal printer. It continues doing this for each display screen.

THE EXTENDED GRAPH PARAMETERS

The extended graph parameter list, EPAR, contains a program and two positive integers. The program is the graphing procedure you want used for plotting while the two integers indicate how many horizontal and vertical frames you want graphed. The graphing program, horizontal range, and vertical range are set by putting the new value on level one and running the programs PROG, XFRM, and YFRM respectively.

Pressing EPAR will return its value to level one. If EPAR hasn't been defined yet its default value, { DRAW 1 2 }, will be returned. This value is used when you haven't defined EPAR. For example, purge EPAR if it exists in the work directory and define ADRW as the graphing program by entering 'ADRW' PROG. Pressing EPAR will return { ADRW 1 2 }. The default values for the horizontal and vertical frames (1 and 2) were used to fill in the missing pieces.

PRINTING AN EXTENDED GRAPH

Flag 18 decides how an extended graph will be output. If it is set, each frame of the graph will be sent to the thermal printer. A gap is printed between each column of the graph. For example, graphing an equation with { DRAW 2 3 } stored in EPAR and flag 18 set will print 2 vertical strips. Each represents 3 display screens. By pasting the two strips together you'll get a graph that is two screens wide and three screens tall.

DISPLAY ADJUSTMENTS FOR PRINTED GRAPHS

Look closely at the pixels on your 28S display screen. Notice how each pixel is a perfect square. Now, if you own a thermal printer, look at each dot it prints. These taller dots form rectangles. Because of this slight discrepancy a graph plotted on the display will appear to be stretched vertically when printed. You might not notice its effects when printing only one display screen, but an extended graph composed of several screens will appear distorted.

As an example, graph a circle with a radius of 1.5.

'PPAR' PURGE '√(2.25-SQ(X))' ENTER ENTER CHS = ENTER STEQ DRAW

Next, print two copies of the graph by pressing L twice while holding the ON button. Now take one copy, turn it 90 degrees, and compare it to the other. If a true circle had been printed they would both look the same. Instead, two ovals will have been printed.

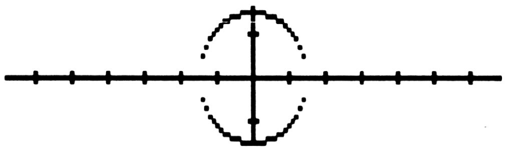


Figure 6.18

To accommodate for the extra height the second and third lines in EDRW's program listing test flag 18. If it is set, the horizontal components of the plot parameters are stretched. This compensates for any extra height that the printer adds. Note that the adjusted graph will look wider on the display, but will print out correctly.

SAVING AN EXTENDED GRAPH INTERNALLY

If flag 18 is cleared, the resultant graphic strings representing each display screen are added together and stored in 'G$TRE' in the graphic file directory GFIL. Although EDRW will graph any number of vertical strips, only the last one will be stored in 'G$TRE'. This means the number of horizontal displays (the second object in EPAR) should be set at 1 when storing an extended graph internally. Having a number larger than 1 will graph more than one vertical strip and only waste time.

VIEWING AND EXTENDED GRAPH

If an extended graph is drawn with flag 18 set it is stored in the graphic file. You can view this graph by pressing VIEW in the extended graph menu. This program initially displays the first four lines of the graphic string stored in 'G$TRE' in the graphic file directory. To view down a line simply press the down cursor key ▼. Likewise, the up cursor key ▲ shifts the display back up a line. Pressing any other key will exit the program.

You will get an error and become stranded in the graphic file directory if you press VIEW without storing an extended graph in the graphic file first. If this happens you'll have to return to the main graphing menu and reselect the graphing directory and menu you were working in.

AN EXAMPLE

As an example of how EDRW operates, let's see what would happen if the following values are stored in the specified locations. The keystrokes that will define them are also given.

```
{EPAR PPAR} [PURGE] 2 HFRM 2 VFRM
   '3*SIN(X)' STEQ 18 SF EDRW
```

LOCATION	**VALUE**
'EPAR'	{ DRAW 2 2}
'EQ'	'3*SIN(X)'
'PPAR'	{ (-6.8,-1.5)(6.8, 1.6) X 1 (0, 0) }
FLAG 18	SET

This tells EDRW to graph '3*SIN(X)' using the built in graphing procedure DRAW in four frames, two vertical and two horizontal. The range of the horizontal axis doubles, to -13.6 through 13.6. The range on the vertical axis also doubles to -3.1 through 3.2. The dimensions of each pixel (.1 by .1) are maintained, as they should be. The resultant graph is shown on the next page.

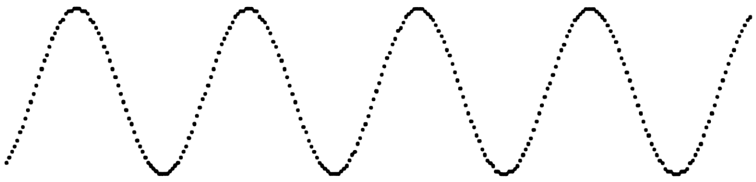


Figure 6.19

If you don't own a printer you will have to clear flag 18 before running EDRW and view the graph using VIEW . Unfortunately, only the last two frames will be saved.

ANIMATED GRAPHS

INTRODUCTION

The 28S was designed with many great advantages over its predecessor, the 28C. The most obvious new feature is its 32K of RAM. This stock pile of storage allows you to retain all your programs (although the amount of programing in this book may put that theory to the test). The additional memory allowed the designers to add several commands that were not practical on the 28C, which only had 4K worth of RAM. Two of these new tools deal with the display screen. LCD→ returns a graphic string representing the current display screen while →LCD displays the graphic string from level one. These two powerful commands allow you to store the display for any graph or picture and instantaneously recall it at any time.

An animated graph is simply a graph that has been drawn a number of times. Each time it is drawn some aspect of the graph is changed slightly. If the graphic strings for these graphs are stored and later played back, we end up with a small motion picture of the graph as it changes.

THE ANIMATED GRAPH MENU

The animated graphing menu can be found by pressing MOVE from the advanced graphing selection menu. The main program in this menu, MDRW, transforms any drawing procedure into an animated grapher. It takes full advantage of the 28S's ability to store and recall a display screen. Simply tell it which program you want to animate, the number of frames (pictures), and which variable or parameter you want varied with time.

ABOUT THE ANIMATED GRAPHING PROGRAM (MDRW)

MDRW uses the graphing program you specify when drawing a series of graphs. Each time it finishes a graph it stores the graphic string representing the display, or prints it on the thermal printer, and clears the screen for the next graph. After it is done you can playback the animated graph with any one of the viewing programs. Like a short motion picture, a series of graphs that took minutes to create can be played back in a few second. Or, if it was printed on the thermal printer, you'll have a series of snapshots of the graph.

Because storing graphic strings can quickly eat up valuable memory, MDRW incorporates a print option. Like EDRW, each graph can be printed on a thermal printer instead of stored in the graphic file. This feature also allows you use the extended graphing program EDRW as MDRW's drawing program, thus creating an animated extended graph.

THE ANIMATED GRAPH PARAMETERS (MPAR)

Before you can create moving graphs you must first understand the move parameters. The first menu key in the animated(move) menu is MPAR. This is the variable containing the animation parameter list. MPAR reads as follows.

{ { any graphing procedure, a real number}
{ first variable or program, first upper limit, first lower limit}. . . . . { Nth variable or program, Nth upper limit, Nth lower limit }}

You'll notice that MPAR is lists within a list. It can contain any number of lists, but must have a minimum of two to avoid having errors. Like the three dimensional parameters DPAR, the first list in MPAR contains the graphing program to be animated and the number of times to evaluate it. Each time the graphing program is evaluated a new frame(picture) is drawn.

Pressing MPAR when it isn't defined in the work directory will return { { DRAW 5 } }. This is the default value stored in the main graphing program directory GRAPHP. It is used to fill in any undefined values when MPAR doesn't exist in the current directory.

THE DRAWING PROGRAM

The program that is used by MDRW can easily be changed by putting either a program or the quoted name of a program on level one and pressing PROG. Any valid drawing procedure can be used. This versatility lets you animate any graphing program. The built-in graphing program DRAW, any program in this book, or one you've created yourself will all work equally well.

THE NUMBER OF FRAMES

Likewise, the number frames can be changed by entering a positive integer and pressing FRAM. Graphing more frames will give smoother motion when the graph is played back, but will take longer and consume a larger amount of memory when stored in the graphic file. Using fewer frames will cause the playback of the graph to be choppier, but is much quicker and drains less memory. A high resolution graph using 25 frames can easily devour any remaining memory. On the other hand, a much less modest graph with 6 to 10 frames should leave ample working and storage space.

VARIABLE LISTS

All lists following the first are considered variable lists. The name is some what deceiving since it can contain a variable or a program. The first object of every list must be a variable name or a program. The remaining two objects are real numbers and define the range that the variable or program will cover.

MDRW divides the range of each variable list among the number of frames being drawn. For example, if { { DRAW 5 } { Q -2 2 } } was stored in MPAR, MDRW would divide the range (-2, 2) into 5 parts, -2, -1, 0, 1 and 2. Each time a frame is drawn MDRW puts the appropriate number on level one. If a program is stored in Q, it will be run, otherwise, the current value on level one will be stored into Q.

Having a program or the name of a program as the first object in a variable list lets you preform complicated calculations and variable manipulation or even alter parameters within lists while running MDRW.

As an example in the shape drawing menu, the starting angle of a shape can only be set by the program THETA. THETA takes the number from level one and puts it into the shape parameter list SPAR. The variable list { THETA 0 45 } allows you to change the starting angle of a shape with time thus creating and animated graph of a spinning shape. When you play it back you'll see it rotate from 0 to 45 degrees. A similar graph is drawn in chapter seven.

ADDING AND DELETING VARIABLE LISTS

There are two ways to add a variable list to MPAR. The easiest is to put the name of the variable or program on level three, the upper limit on level two, the lower limit on level one and press ADD. You can also enter the entire variable list { variable or program, lower limit, ending range } on level one and press ADD. Either way, a new variable list will be added to MPAR .

To delete the last variable list from MPAR simply press DEL. It subtracts the last list and returns it to level one. Nothing happens if only one list is stored in MPAR i.e. it doesn't contain any variable lists.

USING MDRW

MDRW will draw an animated graph as defined by MPAR. Simply specify the drawing program to be animated, number of frames, and an object and range to vary with time. Then, MDRW will draw a series of graphs and, if flag 18 is set, print each one on the thermal printer. Otherwise, the display for each graph will be converted into a graphic string and stored, as a list, in 'GSTRM' in the graphic file directory. This list can then be viewed using the programs GLIDE, SHAKE, or Step.

PRINTING AN ANIMATED GRAPH

Setting flag 18 will cause MDRW to send an animated graph to the thermal printer. It is often more desirable to print a graph, as opposed to storing it in the graphic file directory. Printing not only gives you a hard copy, but doesn't tie up additional memory for storage. In fact, you won't even be able to store graphs having a large number of frames and will be forced to print them out.

As an example, print eight frames of the time domain function Y=SIN(X+tπ). Have t vary from 0 to 2 (one period). This function can easily be graphed using MDRW and the built-in program DRAW. DRAW will control the variable 'X' while MDRW will control 't'. First, store the equation in 'EQ'. Then set 8 as the number of frames and add the variable list { t 0 2 } to MPAR. Finally, set flag 18, turn your thermal printer on, and run MDRW. Figures 6.20 through 6.27 will be printed on the thermal printer. If you don't own a printer you can still graph the function, as the next section demonstrates.

```
{PPAR MPAR } PURGE 'SIN(X+t*π)' STEQ 8 FRAM
t, 0, 2 ADD 18 SF MDRW
```

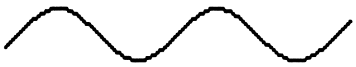


Figure 6.20

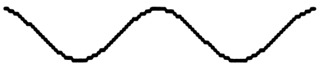


Figure 6.21

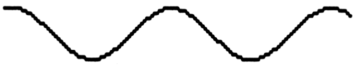


Figure 6.22

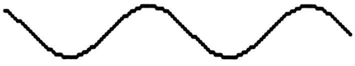


Figure 6.23

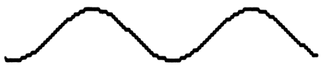


Figure 6.24

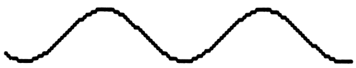


Figure 6.25

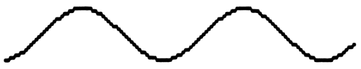


Figure 6.26

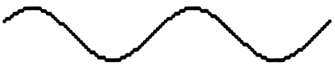


Figure 6.27

STORING AN ANIMATED GRAPH

Although printing an animated graph doesn't require any additional memory, the only way a true animated graph can be created is by storing the graphic strings for each frame. Then you can play back each frame in series, like a short motion picture.

Flag 18 controls how MDRW stores an animated graph. Clearing it will store the list of graphic strings for each frame 'GSTRM' in the graphic file directory.

As an example, try graphing the same example used to demonstrate printed animated graphs on page 258, but this time clear flag 18.

```
'SIN(X+t*π)' STEQ 8 FRAM t,0,2 ADD 18 CF MDRW
```

The same series of frames for the same graph will be drawn, but this time the printer isn't activated. Instead, a list of graphic strings has been stored in 'GSTRM' in the graphic file directory. You can see the list by pressing FILE. This will put you in the graphic file directory and create the graphic file menu. Page 261 describes this directory in greater detail.

VIEWING AN ANIMATED GRAPH

Now that you've created an animated graph you need to view it. GLIDE, SHAKE and Step allow you to look at the graphic list stored in 'GSTRM' in three different ways. GLIDE continually flashes each frame on the display in series,. SHAKE displays the frame from first to last and then last to first. Finally, Step show one frame and pauses. Pressing a key will advance to the next frame.

GLIDE, the first viewing program in the animated menu, goes to the graphic file directory, GFIL, takes the list in 'G$TRM' and displays each string from first to last. It keeps doing this until you press any key. If the graph has a period or cycle, like the sin function, the effect is a gliding motion. Once a key is pressed the program returns control to the calling directory and stops.

SHAKE displays the graphic strings in 'G$TRM' from first to last and then from last to first. It keeps doing this until you press any key. In most cases, this viewing program give a back and forth, or shaking motion.

The last viewing program, Step, operates much like GLIDE. It displays the graphic strings in 'G$TRM' from first to last. Unlike GLIDE, Step displays a graphic and waits for a key to be pressed. Pressing any key other that ENTER will display the next string in the list. This lets you view your graph one frame at a time. Pressing ENTER will exit the program and return you to the calling directory.

NOTE

You'll get an error if a list of graphic strings wasn't created first by MDRW. If this happens, you'll find yourself stranded in the directory GFIL. This can be easily solved by returning to the main graphing menu and reselecting the menu and any sub-menus you were in.

THE GRAPHIC FILE MENU

As your catalog of data grows, you may be forced to purge old programs, graphic strings, or other data. With so many directories it can become impossible to sift through hundreds of different variables in every different directory to find which ones should be purged. This is why all graphic strings created by MDRW and EDRW are stored in GFIL, the graphic file directory. Having all these graphics in one directory makes it easier to find these memory hoarding objects and, if need be, discard them. It also allows you to view a graph from any directory, no matter which directory the original graph was created in.

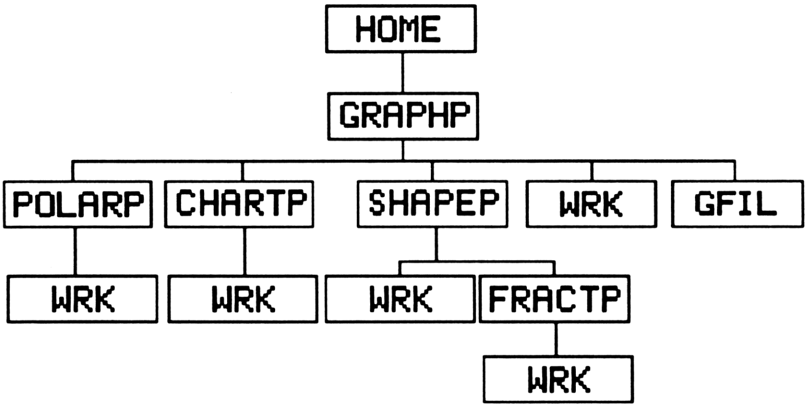


Figure 6.28

Directory Tree Showing GFIL's Relationship to the Other Directories

Any program that retrieves, views, or manipulates graphic strings must have some way of distinguishing between a graphic string created by EDRW and a list made by MDRW. To differentiate between the two, all the programs that store graphics in GFIL first categorize it by adding a special character to the end of its name. Any variable created by EDRW will end with an E while those create by MDRW will end with an M. This make it easier for you to sort through the different types of graphics. More importantly, other programs can find the graphics it needs simply by looking at the last character of each variable stored in the graphic file directory.

You can enter the graphic file from either the extended or animated graph menu by pressing FILE. This will put you in the graphic file directory and create the graphic file menu. The first three keys in this menu are labeled LOAD SAVE and EXIT. The remaining keys will be labeled with the names of graphics stored in the graphic file that were created by the calling (extended or animated) menu.

The extended and animated graphing menus store their graphs in 'G$TRE' and 'G$TRM' respectively. Creating a new graph will replace any old one. $AVE, allows you to save a graph under a different name. For example, you might want to save the sin function graph from page 259. Simply enter the name you want to store it under and press SAVE. Depending on the calling menu, either G$TRE or G$TRM will be purged while its contents will be stored under the new name.

Loading a saved graph is just as easy. Just put the appropriate graphics or the name of a variable containing the graphics on level one and press LOAD. These graphics will be stored (loaded) in either 'G$TRE' or 'G$TRM' and you'll exit the graphic file menu. The new graphics are now ready to be viewed.

SUMMARY OF MENUS

ADVANCED GRAPHING SELECTION MENU

Menu Key	Operation
DIM	Puts you in the three dimensional graphing menu.
EXTND	Put you in the extended graphing menu.
MOVE	Puts you into the animated graphing menu.
END	Returns you to the calling menu.

THREE DIMENSIONAL GRAPH MENU

Menu Key	Operation
DPAR	This variable contains the three dimensional parameter list which reads as follows. { { graphing program, number of cross sections } { first variable or program, real number, real number } ....{ last variable or program, real number, real number } }.
PROG	The program or program name from level one will be put into the first position in the first list in DPAR. This is the drawing program that will be graphed in three dimensions.
LINES	The integer from level one will be put into the second position in the first list in DPAR. This specifies the number of cross sections to be drawn.
ADD	This program adds a variable list to DPAR. Just put the name of the variable or program you want varied on level three, the beginning value on level two, and the ending value on level one. ADD will put all three objects into a list and add it to DPAR. You can also put the three items in a list on level one and run this program. The list will be added to DPAR.
DEL	This program deletes the last variable list from DPAR. DPAR will be unaffected if it doesn't contain a variable list.
DDRAW	This program draws the graphing program specified by DPAR in three dimensions.
END	The last label in the three dimensional graphing menu, END sends you back to the advanced graphing selection menu.

EXTENDED GRAPH MENU

Menu Key	Operation
EPAR	EPAR contains a list of extended graph parameters. The list reads as follows. { graphing program, number of horizontal frames, number of vertical frames}.
PROG	The program that will be used to create an extended graph can be set by putting its name on level one and running this program. The name from level one will be put into the first position in EPAR.
HFRM	The number of horizontal display screens (frames) you want your graph to be plotted on is set using this program. Pressing this menu key will put the real number from level one into the second position in EPAR.
VFRM	The number of vertical display screens (frames) you want your graph to be plotted on is set using this program. Pressing this menu key will put the real number from level one into the third position in EPAR.
FILE	This handy program puts you in the graphic file directory and creates a graphic file menu.
VIEW	If an extended graph is stored in 'G$TRE' in the graphic file this program will let you view it. After pressing this menu key, you'll see the top of the graph. Pressing the up and down keys, ▲▼, will move the graph up and down a line respectively. Press any other key to exit.
EDRW	The extended graphing program will plot the graphing program on a number of vertical and horizontal frames specified by EPAR. Depending on the status of flag 18, the resultant graph will either be printed or stored under 'G$TRE' in the graphic file.

END	The last label in the three dimensional graphing menu, END sends you back to the advanced graphing selection menu.

ANIMATED (MOTION) GRAPH MENU

Menu Key	Operation
MPAR	This variable contains the animated graphing parameter list, which reads as follows. { { graphing program, number of frames} { first variable or program, real number, real number } ....{ last variable or program, real number, real number } }.
PROG	The program or program name from level one will be stored as the first object in the first list in MPAR. This is the drawing program to be animated.
FRAM	The number of frames decides how many graphs will be drawn. It can be set by putting a real number on level one and pressing this menu key. The real number from level one will be put into the second position of the first list in MPAR.
ADD	This program adds a variable list to MPAR. Just put the object you want varied on level three, the beginning value on level two, and the ending value on level one. ADD will put all three objects into a list and add it to MPAR. You can also put the three items in a list on level one and run this program. The list will be added to MPAR.
DEL	This program deletes the last variable list (any list other than the first) from MPAR. MPAR will be unaffected if it doesn't contain any variable lists.

MDRW	Using the graphing program, number of graphs to draw, and variable lists stored in MPAR, MDRW draws an animated graph. This is nothing more that a series of graphs that change slightly from frame to frame. If flag 18 is set the resultant graph is printed. Otherwise, it is stored as a list under GSTRM in the graphic file.
FILE	This handy program puts you in the graphic file directory and creates the graphic file menu.
GLIDE	GLIDE continually displays each string stored in GSTRM in the graphic file directory from first to last. Pressing any key will return you to the calling directory and exit the program.
SHAKE	SHAKE continually displays each string stored in GSTRM in the graphic file directory from first to last and then last to first. Pressing any key will return you to the calling directory and exit the program.
STEP	An animated graph can be viewed one frame at a time using this program. It displays a graphic string from the list stored in GSTRM in the graphic file directory and waits for a key to be pressed. Pressing any key other than ENTER will display the next string. Pressing ENTER will return you to the calling directory and exit the program.
END	The last label in the three dimensional graphing menu, END sends you back to the advanced graphing selection menu.

THE GRAPHIC FILE MENU

Menu Key	Operation
LOAD	Depending on the calling menu (the extended or animated graphing menu), this program will store the graphics on level one or the graphics stored in the name on level one into either GSTRE or GSTRM and return to the calling directory and menu.
SAVE	A graphic string or list of graphic strings created by EDRW or MDRW can be saved by putting a quoted name on level one and pressing this menu key. Depending on the calling menu (the extended or animated graphing menu), the graphics stored in either GSTRE or GSTRM will be stored in that name. The original copy, in GSTRE or GSTRM, will be purged.
EXIT	This program exits the graphic file menu and directory and returns you to the calling menu and directory.
All other keys	Whenever a graphic string created by EDRW or list of strings created by MDRW is stored in the graphic file, the letter E or M respectively is added to the end of its name. This lets any program distinguish between the two. Depending on the calling menu, the names of all the variables in the graphic file directory ending with either E or M will be listed in the graphic file menu.

CHAPTER SEVEN

MORE EXAMPLES OF ADVANCED GRAPHS

INTRODUCTION

Chapter presented three different advanced graphing programs. Each one can drive any drawing program. The examples in chapter six used the built-in program DRAW, but there is no reason why an advanced graphing program can't drive root level programs from chapters two through five, or even another advanced graphing program. In fact, you can link all three advanced graphing programs and one root level program to create an extended, four dimensional graph.

ROOT LEVEL AND ADVANCED COMBINED

EXAMPLE 7.1

DRAWING A CONE IN THE POLAR MENU

There are several ways to draw a cone with a base of 1.5. This example uses the polar graphing menu. A cone can be graphed by drawing circles whose radius equate their distance along the Z axis. Thus the tip of the cone will be at Z=0 and the base will be at Z=2. In polar coordinates, the equation for a circle is just it's radius. In this example we want the radius to vary from 0 to 2.

The coordinates we are given in the polar and three dimensional menus are r, θ, and z. This happens to be cylindrical coordinates. The cylindrical equation of a cone is just r=z

Starting off in the polar graphing menu, purge any old parameter lists and set APAR to { 0 360 DEG }. Now go to the advanced graphing selection menu and enter the three dimensional menu. Set the drawing program to ADRW and the number of cross sections to 6.

```
{ PPAR APAR DPAR } [PURGE] 0 AMIN 360 AMAX  DEG
        ADVAN DIM 'ADRW' PROG 6 LINES
```

Now we need to decide how to vary the radius of each circle along the Z axis. If the value of Z is stored into 'EQ' for each cross section ADRW will draw a circle with that radius. Add the variable list { EQ .1 2 } to DPAR and draw the cone.

'EQ', .1, 2 ADD DRAW

You could have added { EQ -.1 -2 } as the variable list. The negative sign won't effect the radius of the circles, only the direction of the cones along the Z axis. Try it and see, but be sure you've deleted the old variable list first.

NOTE:

.1 is used as a lower limit instead of 0. because ADRW can't draw a circle with a radius of 0. ADRW uses the inverse of the radius as a control on the step size. You can see how it might not like a step size of INV(0), which is undefined.

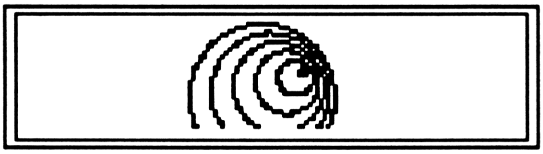


Figure 7.1

EXAMPLE 7.2

A THREE DIMENSIONAL POLAR EQUATION

Try drawing the equation 'ABS(2.5*COS(T))' in cylindrical coordinates. Vary the Z axis from -1 to 1 in five cross sections.

SOLUTION

While in the polar graphing menu, purge any old parameter lists, store the equation in 'EQ' and set the polar graphing parameters to

{0 6.283 RAD }. This tells ADRW to graph the polar function from 0 to 2π. Then, call the three dimensional menu and set the drawing program and number of cross sections to 'ADRW' and 5 respectively. Finally, add the parameter list for the Z axis and run DDRW. Figure 7.2 will be drawn on the display.

```
{APAR PPAR DPAR } PURGE 'ABS(2.5*COS(T))' STEQ
0 AMIN 2,π * SHIFT EVAL AMAX RAD ADVAN DIM
'ADRW' PROG 5 LINES 'Z',-1,1 ADD DDRW
```

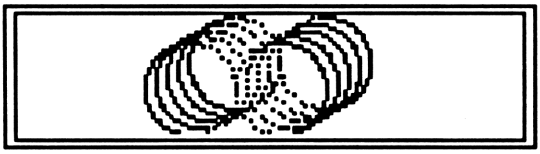


Figure 7.2

EXAMPLE 7.3

AN EXTENDED PIE CHART

In chapter five we create a pie chart, but there wasn't enough room to label each section with large print. If we create a chart with a radius of 4.5 (on three display screens), we will be able to use large printing.

Go to the pie chart menu, purge any old parameters, and store the following values in the specified names. The keystrokes are listed below.

ΣDAT	[[195][447][362][60]]
πPAR	{4.5,3}

```
195 Σ+ 447 Σ+ 362 Σ+ 2 60 Σ+ 4.5 RADIU
```

The second menu label should be LARGE. If it isn't, keep pressing it until it is.

Next, go to the extended menu and set the drawing program and the number of horizontal and vertical frames at πDRW, 1, and 3 respectively. If you want the graph to be printed set flag 18, else clear it to store the graph in memory. Finally, pressing EDRW will create the graph shown in figure 7.3.

ADVAN EXTND 'πDRW' PROG 1 HFRM 3 VFRM 18 SF EDRW

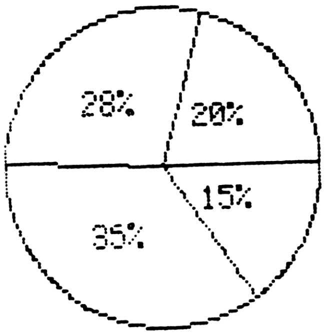


Figure 7.3

EXAMPLE 7.4

ANIMATED SHAPES

Any shape can easily be animated using the shape and animation menus. Simply define any shape and add the parameter list { THETA, θ1, θ2 } to MPAR. The angles θ1 and θ2 depend on the number of points in your shape and the number of frames used in your graph.

As an example, let's animate a five pointed star. First, define the shape in the shape menu. Remember, we don't need to define the starting angle of the shape because the animated graphing program, MDRW, will do it for us. Next call the animation menu and define $DRW as the drawing program.

```
5 POINT 2 REV 1.5 RADIU ADVAN MOVE '$DRW' PROG
```

Now, we need to decide how many frames to use. We will get better resolution using more frames, but it will take longer creating the graph and, if stored as a list in the graphic file, will take up more memory. For this example, 6 frames will give good resolution and only take a few minute to complete.

```
6 FRAM
```

Now we need to set the starting and ending angle in the variable list. The staring angle can be set at any number, while the ending angle, θ2, depends on the number of frames and number of points on the shape. The relationship is given in equation 7.1, where p is the number of points set by POINT and f is the number of frames set by FRAM.

$\theta 2=360/p*(1-1/f)+\theta 1$ Equation 7.1

The ending angle for a starting angle of 0°, five points on the shape, and six frames in the graph, is 60°. Put these values in a variable list and add it to MPAR. Finally, decide wether you want the graph stored in the graphic file or printed by clearing or setting flag 18. Running MDRW will create the graph show in figures 7.4 to 7.9.

```
'THETA',0,60 ADD MDRW
```

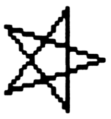


Figure 7.4

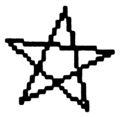


Figure 7.5

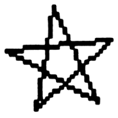


Figure 7.6

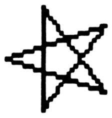


Figure 7.7

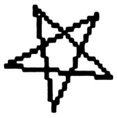


Figure 7.8

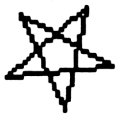


Figure 7.9

See if you can't create a three dimensional, rotating pentagon by storing the following values in the given objects and running MDRW.

'PPAR'	{ (-6.8, -1.5) (6.8, 1.6) constant 1 (0,0) }
'SPAR'	{ (1.5, 0) 5 1 }
'DPAR'	{{ SDRW 9 } { RADIU .1 1 }}
'MPAR'	{{ DDRW 6 } { THETA 0 60 }}
FLAG 18	Set for printing, clear for storing in graphic file

EXTENDED GRAPHS OF FRACTALS

The fractals drawn in chapter five were all drawn on one display screen. This limited the resolution you could get. Now that we have the extended graphing program, we can draw any size fractal. Several fractals from the example section in chapter five have been redrawn on two vertical displays using EDRW. Try redrawing the others on your own.

EXAMPLE 7.5 (example 5.1 redrawn)

First, go to the fractal menu, select the editing menu, and define the fractal. The fractal has a triangle for an initial curve and a four segment, 60° model construction. The radius of the initial curve should be larger than in chapter five because we are drawing the fractal on two displays now. Then, while in the fractal selection menu, go to the extended graphing menu and define EPAR. If you want the graph sent to the printer set flag 18, otherwise, the graph will be stored in the graphic file.

EDIT 'SPAR' PURGE SHAPE 3 RADIU MAKE 60 SEG4 3 STOK
ADVAN EXTND 'FDRW' PROG 1 XFRM 2 YFRM EDRW

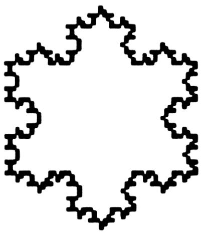


Figure 7.10

EXAMPLE 7.6 (example 5.3 redrawn)

The initial curve is a five pointed star with a radius of 3, since it will be drawn on two display screens. The four segment model construction's second and third segments make an angle of -72 and 72 degrees with the horizontal axis.

As in chapter five, go to the shape menu under the fractal editing menu, define 'SPAR', and press MAKE. Then, create the four segment model construction and define the number of replacements, K. Finally, go to the extended graph menu EPAR should still have the same value as in example 7.5 so all you need to do is run EDRW.

EDIT SHAPE 3 RADIU 90 THETA 5 POINT 2 REV MAKE
-72 SEG4 3 STOK ADVAN EXTND EDRW

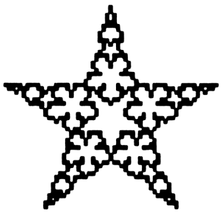


Figure 7.11

EXAMPLE 7.7

Create a fractal whose initial curve is a square with a radius of 2.5 and model construction has two segments making an angle -45° and 45° with the horizontal axis. Draw several different values of 'K' and use two vertical displays.

SOLUTION

While in the fractal editing menu, press SHAPE. Then, define a square and press MAKE. Now, define the model construction by entering -45 SEG2. Next, set the number of replacements at 3. Finally, go to the extended graph menu and, using the same parameters as the previous two examples, run EDRW. Return to the fractal editing menu, store different values it K and redraw the fractal.

EDIT SHAPE 2.5 RADIU 45 THETA 4 POINTS 1 REV MAKE
-45 SEG2 3 STOK ADVAN EXTND EDRW

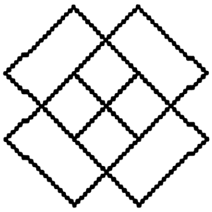


Figure 7.12

END END EDIT 4 STOK ADVAN EXTND EDRAW

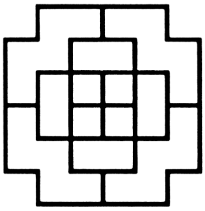


Figure 7.13

END END EDIT 5 STOK ADVAN EXTND EDRAW

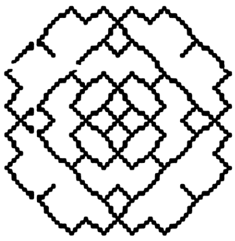


Figure 7.14

END END EDIT 6 STOK ADVAN EXTND EDRW

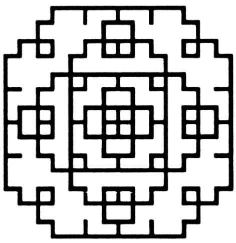
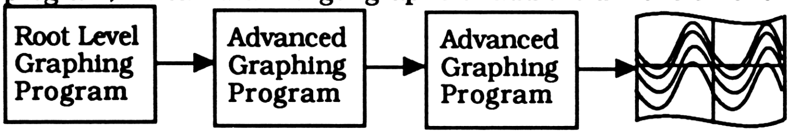


Figure 7.15

COMBINING SEVERAL ADVANCED GRAPHING PROGRAMS

We drew several three dimensional shaped in the example section in chapter six. Now, with more than one advanced graphing program, we can draw larger graphs or add the dimension of time.


Root Level Graphing Program
Advanced Graphing Program
Advanced Graphing Program

Several advanced graphing programs can be cascaded to produced large, multidimensional graphs. For example, by using DDRW as an argument for EDRW you can get an extended three dimensional graph. Or, by cascading DDRW with MDRW you can get an animated three dimensional graph. Several examples are given below.

EXAMPLE 7.8

DRAWING AN EXTENDED PYRAMID

There are times when the item you want to vary is not directly tied to a variable. One good example is drawing a pyramid with a base of 3.5. A pyramid is just a triangle that is wide at the base, gets smaller towards the top, and eventually comes to a point. 10 cross sections should be good enough to outline the pyramid.

Obviously, we must start in the shape menu. Go to the shape graphing menu and define a triangle by inputting 3 POINT 1 REV 0 THETA. We don't care about the radius now because DDRW will define it for us. Then call the three dimensional graphing menu and define the drawing program and number of cross sections as 'SDRW' and 10.

ADVAN DIM 'SDRW' PROG 10 LINES

Now we need to tell DDRW to vary the radius of the triangle along the axis coming out of the display. The radius of a shape is set by putting the radius we want on level one and running the program RADIUS. This stores the radius into the shape parameter list SPAR. Add 'RADIUS' as the program to be evaluated and the range .1 to 3.5 to DPAR.

'RADIUS',.1,3.5 ADD

In order to plot the entire pyramid we'll have to use two display screens. Because the pyramid drops towards the base, we also have to shift the center down 1.1 units. Exit the three dimensional graphing menu and enter the extended menu. Define 'DDRW' as the drawing program and set the vertical and horizontal frames at 1 and 2. Then, shift the center down 1.1 units Finally, if you want the graph printed set flag 18, otherwise, clear it.

END EXTND 'DDRW' PROG 1 HFRM 2 VFRM (0,-1.1) CENTR

$$\begin{tikzcd}
\boxed{\text{SDRW}} \arrow[r] & \boxed{\text{DDRW}} \arrow[r] & \boxed{\text{EDRW}}
\end{tikzcd}$$

SDRW is now the input program for DDRW and DDRW is the input program for EDRW. Running EDRW will plot the pyramid in figure 7.16 on several displays.

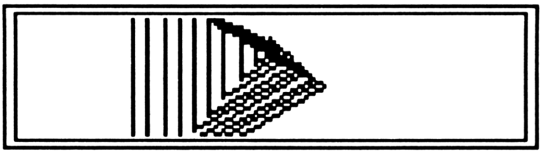


Figure 7.16

EXAMPLE 7.9

A LARGE SPHERE

A sphere was used as an example when describing the three dimensional drawing program DDRW. Because of the limited resolution of the display, we were limited to no more than a few cross section. With EDRW though, the same sphere can be drawn at any size. Lets graph a sphere with a radius of 9.5 in 16 cross sections. The entire graph will fit on 7 vertical and 2 horizontal display screens. Because this graph is so big, it can't be saved in the graphic file. It must be printed. It will also take about an hour, so be prepared.

Store the equation √(90-SQ(X)-SQ(Y))=-√(90-SQ(X)-SQ(Y)) into EQ. Then go to the three dimensional menu and set the three dimensional parameter list to { { DRAW 16 } { Y -9.3 9.3 } }. Next, go to the extended graph menu and set DDRW ,7, and 2 as the drawing program, vertical, and horizontal frames. The graph can only be drawn using the thermal printer, so you must set flag 18. Running EDRW will print 2 vertical strips. Cutting and pasting them together will give you the graph shown on the next page.

Purge any old parameter lists

{ PPAR DPAR EPAR } [PURGE]

Store the equation in 'EQ'

'√(90-SQ(X)-SQ(Y)) [ENTER] [ENTER] [CHS] [=] [ENTER] STEQ

Define the three dimensional parameters (DPAR)

ADVAN DIM 35 LINES 'Y' , -9.3,9.3 ADD

Finally, define EPAR, set flag 18, and run EDRW.

END EXTND 'DDRW' PROG 2 XFRM 7 YFRM 18 SF EDRW

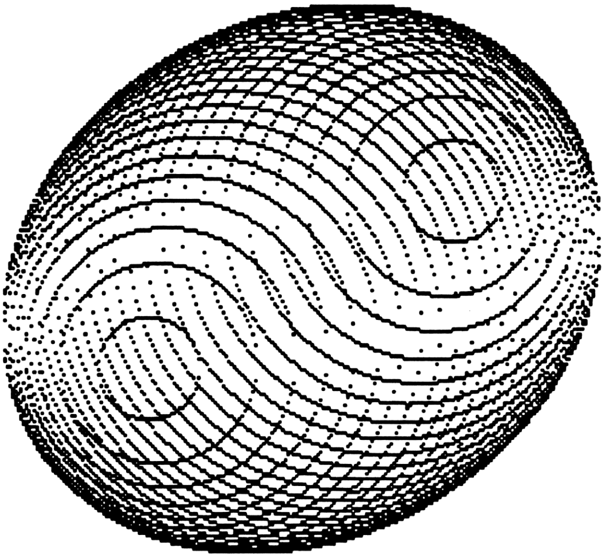


Figure 7.17

EXAMPLE 7.10

A LARGE SADDLE

In chapter six the equation Z=√(X^2+Y^2) gave us a saddle shaped graph (page 247). If we cascade DDRW with EDRW we can redraw this same graph on several display screens. This will give us a better picture than on one display. The following objects had the listed values in the example from chapter six. Running DDRW with these values drew the graph shown in figure 6.18. Lets try graphing this same function on 1 horizontal and 4 vertical display screens.

```
'EQ'        '√(SQ(X)+Z)=-√(SQ(X)+Z)'
'PPAR'      { (-6.8,-1.5) (6.8,1.6) X 1 (0,0) }
'DPAR'      { { DRAW 9 } { Z 2 -2 } }
```

DPAR is the only object we have to change. Because the range of the X and Y axis will increase we must increase the range of the Z axis and the number of cross sections. We must also define the extended graph parameters, EPAR. The new values are shown below.

```
'EQ'      '√(SQ(X)+Z)=-√(SQ(X)+Z)'
'PPAR'    { (-6.8,-1.5) (6.8,1.6) X 1 (0,0) }
'DPAR'    { { DRAW 15 } { Z 6 -6 } }
'EPAR'    { DDRW 1 4 }
```

If you want the graph printed set flag 18, otherwise, clear it. Running EDRW will draw the graph show on the next page.

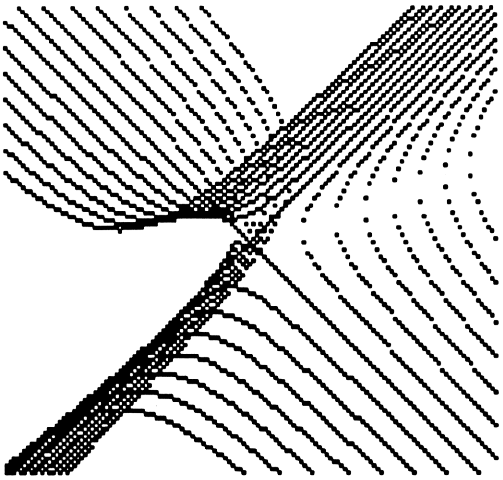


Figure 7.18

EXAMPLE 7.11

A LARGE SINE FUNCTION

The sine function is probably the most attractive. Try graphing the function Y=-SIN(X)*(Z+4)*(X/6) on three horizontal and six vertical display screens. Draw 15 cross sections and use the range { -.3, -4 } for Z.

First, store the equation in 'EQ' and set your calculator to radians. Then, purge any old parameters. Next, define DPAR from the 3-D menu and EPAR from the extended graph menu. Finally, set flag 18 and run EDRW to graph the function in figure 7.19.

```
'-SIN(X)*(Z+4)*(X/6)' STEQ
{PPAR DPAR EPAR } PURGE MODE RAD
ADVAN DIM 15 LINES 'Z',-.3,-4 ADD
END EXTND 'DDRW' PROG 3 XFRM 12 YFRM 18 SF EDRW
```

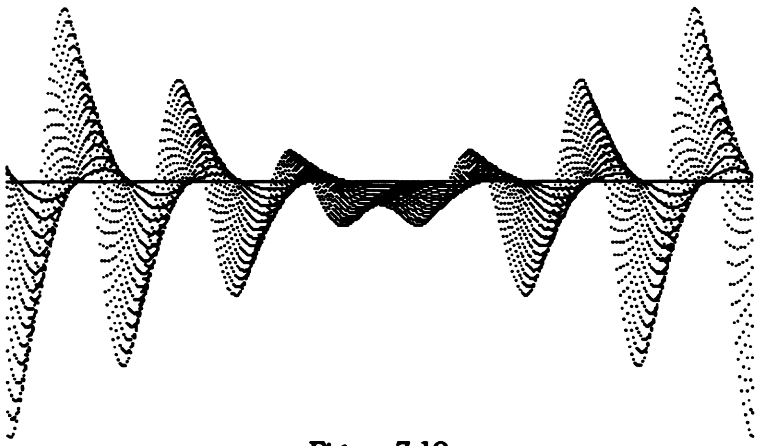


Figure 7.19

EXAMPLE 7.12

AN ANIMATED 3-D SINE FUNCTION

Try graphing the time domain, three dimensional function shown in equation 7.1. Because there are four different variables, X, Y, Z, and t, we must use DDRW and MDRW.

Z=SIN(X+tπ)*COS(Y+tπ) Equation 7.1

First, purge any old parameters, store the equation in 'EQ', and set your calculator to radians. Then, go to the three dimensional menu and set up the parameter list for DDRW. Drawing cross sections from -π/2 to π/2 will draw one entire period of the graph. and because the range for the X and Y axis is one display screen (we aren't using EDRW) we will draw 9 cross section.

```
{PPAR DPAR MPAR } PURGE
'SIN(X+t*π)*COS(Y+t*π)' STEQ
MODE RAD
DIM 'Y' ENTER π SHIFT EVAL 2 ÷ ENTER CHS ADD
```

Next, go to the animated graph menu and set the parameters for MDRW. The drawing program it uses is DDRW. The number of frames you use to draw the graph depends on whether you're printing the graph or how much memory you have available. It also depends on how long you're willing to wait or the resolution you want. If you have the memory try 10 frames. If not, use fewer frames or print the output (set flag 18).

Because the sine function is periodic, defining the range from 0 to 1 will cause MDRW to graph one period. But the graph for t=0 and one for t=1 are the same. This means if we graph 10 frames ranging from 0 to 1 the first and last frames will be identical. This gives us only 9 different pictures. If we found the step size for drawing 11 frames, subtracted it from 1 and used it as the upper limit, MDRW will draw 10 different frames. The equation to follow when determining the upper limit to use for a periodic function is

LL=lower limit in variable list
UL=upper limit in variable list
BP=beginning range for periodic function
EP=ending range for periodic function
N=number of frames

LL=BP UL=EP - [(EP - BP)/N]

For this example it is .9. If you are using a different number of frames you'll have to calculate the upper limit and substitute it for .9. Go to the animated menu, define MPAR and run MDRW to graph the function shown in figures 7.20 to 7.29.

```
MOVE 'DDRW' PROG 10 FRAM 't',0,.9 ADD MDRW
```

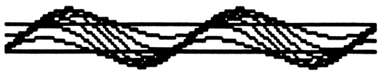


Figure 7.20

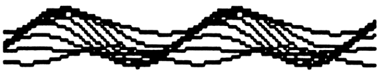


Figure 7.21

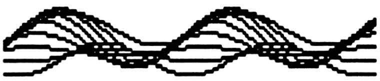


Figure 7.22

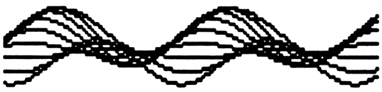


Figure 7.23

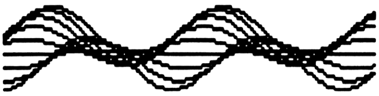


Figure 7.24

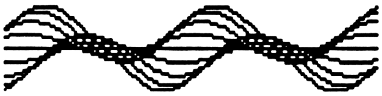


Figure 7.25

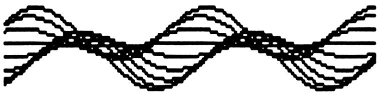


Figure 7.26

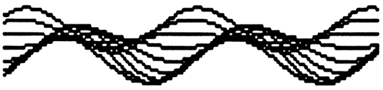


Figure 7.27

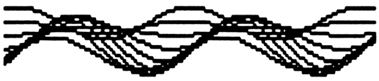


Figure 7.28

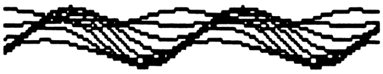


Figure 7.29

APPENDIX A

ENTERING DATA

There are several different ways to enter the programs from this book in your 28S. You could type the programs letter by letter. Not only will this take a long time, but you can easily make a typographical mistake. A simpler way is to press the menu key for any commands in the program listing.

As an example, both key sequences for entering « LCD→ NOT →LCD » are show below. This program inverts the display screen. It should be obvious that the second is a better method.

CHARACTER BY CHARACTER

« SPACE L C D SHIFT U SPACE N O T SPACE SHIFT U L C D ENTER

USING MENU KEYS

« STRING LCD→ BINARY NOT STRING →LCD ENTER

Although both key sequences are correct, the second is much easier. You should take the time to familiarize yourself with the different menus so you can take advantage of them.

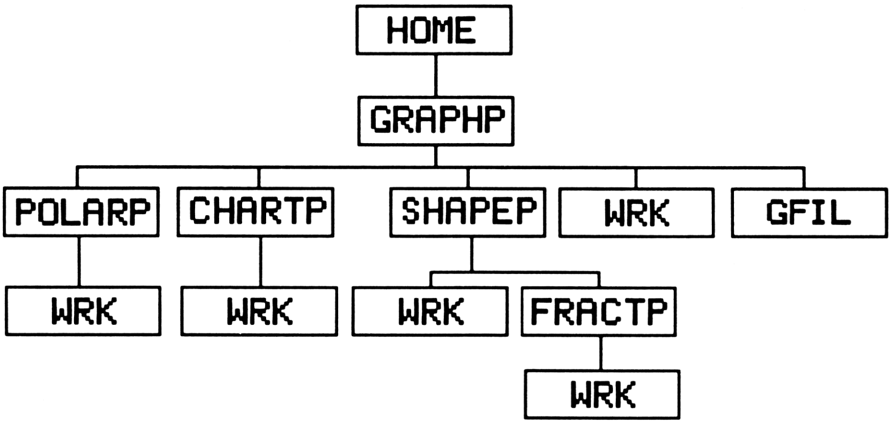
STORING DATA

After you've enter a program on the stack you'll have to store it in your calculator's memory. Simply enter the quoted name you want the program stored under and press STO. This will store the program in the user menu.

DIRECTORIES

The 28S can store hundreds of programs and data in any number of different directories. Anything stored in one directory is "hidden" from the others. Thus programs that share a common interest should be classified together in one directory.

The directory tree below shows all the directories used in this book. The parent directory for all other directories is HOME. GRAPHP, the directory below HOME, contains the programs and sub-directories used for graphing. The directories get more specific as you move down the line. POLARP only contains programs used for polar graphing.



Directory Tree

You can recall or run any program in your path. For example, when in FRACTP, you can run any program in FRACTP, SHAPEP, GRAPHP, or HOME. However, you can't access programs in POLARP, since it isn't in your path.

Notice all the directories named WRK below each program directory. These are the work directories. You'll be creating and storing your own programs and variables here. Having a separate work space makes it easier to sort through the multitude of programs. More importantly, it protects the programs in the main directory from accidently being purged.

MOVING FROM DIRECTORY TO DIRECTORY

Each custom graphing menu has keys that take you from menu to menu. If fact, the menus are so user friendly you don't even have to know which menu your in or which one you're going to. But when your entering or editing programs, you won't be able to use the menus. They put you in work directory while programs are stored in program directories.

There are several ways to move between directories. You can go to any directory in your path by entering its unquoted name. For example, you can go to SHAPEP from FRACTP by typing

S H A P E P ENTER

(See directory tree on the previous page). If the directory is not in your path, you should go to the home directory and enter the unquoted name of the directory that is next the path of the directory you want to get to. Keep doing this until you've reach your destination directory. For example, you could enter POLARP from FRACTP by typing

HOME G R A P H P ENTER P O L A R P ENTER

Even though you can also press the menu key for each directory that is next in the path, you should never do it for the directories in this book. Because the menu keys can only fit a set number of characters, for the labels for each program directory is the same as the labels of each programs that enters that directory's graphing menu. For example, GRAPH and GRAPHP will both read GRAPH in the user menu.

DIGITIZING

Digitizing mode allows you to return display points to the stack. Press DGTIZ from the [PLOT] menu. You will see a cross in the center of the display. The arrow keys ▲▼◀▶ will move the cross up, down, left, and right. Pressing the shift key and then an arrow key will move the cross to the edge of the display in that direction.

Pressing the cursor key ◂⬍▸ will display the point where the cross is at. Pressing [INS] will put the point on level one of the stack. Finally, pressing [DEL] will return the graphic string representing the display screen.

Program	**Uses the utilities. . . .**	**Used by the utilities . .**
ADD	PGET, PCHK	
ADRW	CLCD?, DGTZ	πDRW
ADVAN		
AMAX	PUTP	
AMIN	PUTP	
APAR		ADRW, AMAX, AMIN, PUTA, STLST (CH 2)
CHART	Start	
CHARTP		CHART
CHLST		MCHK
CKNM		
CLCD?		ADRW, πDRW, SDRW, FDRW
CNCT	LINE	SDRW, LDGU, DRAWC, DRAWM
CONT	Start	
CREATC	LDGU, STOC	
CREATM	LDGU, STOM	
CRSM		
CTLST		
CTR?		DDRW, EDRW
DDRW	CTR?, MTYP	
Deg	PUTA	
DEG°		
DEL	PGET	
DGTZ?		ADRW, πDRW, SDRW, FDRW
Dim	N→FG, MMENU	
DPAR		DDRW, PGET
DRAWC	CNCT	
DRAWM	LCONV, CNCT	
EDIT		MAKE
EDRW	CTR?, GGFL, LOAD	
ENd	Start	
End	GRAPH	

end		
EPAR		EDRW, PGET
EXIT	PBAK, MMENU	LOAD, SAVE
EXTND	N→FG, MMENU	
FDRW	CLCD?, DGTZ?, LNE	
FILE	GGFL, MCHK	
FRACT	Start	
FRACTP		FRACT
FRAM	PGET, PUTL	LINES
GFIL		
GGFL		EDRW, FILE, GLIDE, MDRW, SHAKE, Step
GLIDE	GGFL, PBAK	
GRAPH	Start	End
GRAPHP		
KEYW		Step, VIEW
LARGE	PUTπ	
LCONV		DRAWM
LDGU	CNCT	CREATC, CREATM
LGET	MRKR	LOAD
LGP	SPOT, PLCE	πDRW
LINE		πDRW, CNCT, LNE
LINES	FRAM	
LNE	LINE	FDRW
LOAD	LGET, EXIT	EDRW, MDRW
MAKE	SHAPL, STOC, EDIT	
MCHK	MRKR	FILE
MDRW	MTYP, GGFL, LOAD	
MLST		
MMENU	MRKR	Dim, EXIT, MOVE, EXTND
MOVE	N→FG, MMENU	
MPAR		MDRW, PGET
MRCL		MTYP
MRKR		LGET, MCHK, MMENU, PGET, SAVE
MTYP	MRCL	DDRW, MDRW
N→FG		Dim, EXTND, MOVE

NMST		SEG2, SEG4
NONE	PUTπ	
NONP		πDRW
PBAK		EXIT, GLIDE, Step, VIEW
PCHK		PUTP, ADD, THETA, PUTL, RADIUS (Ch.3)
PGET	MRKR	ADD, CHLST, DEL, FRAM, Prog, PROG, XFRM, YFRM
PLCE	PLCU	LGP
PLCU		PLCE, LGP
POINTS	PUTP	
POLAR	Start	
POLARP		POLAR
PROG	PGET, PUTL	
Prog	PGET, PUTP	
PUTA	PUTP, Start	Deg, Rad
PUTL	PCHK	FRAM, PROG
PUTP	PCHK	AMAX, AMIN, PUTA, POINTS, REV, Prog, XFRM, YFRM
PUTπ	Start	LARGE, NONE, SM
QUIT		
Rad	PUTA	
RADIUS (CH 3)	PUTP	
(CH 4)	PCHK	
RAD°		
REV	PUTP	
SAVE	MRKR, EXIT	
SDRW	CLCD?, SHAPL, CNCT, DGTZ?	
SEG2	NMST, STOM	
SEG4	NMST, STOM	
SHAKE	GGFL, PBAK	
SHAPE	Start	
Shape		

SHAPEP		SHAPE
SHAPL		SDRW, MAKE
SM	PUTπ	
SMP	SPOT, PLCU	πDRW
SPAR		POINTS, RADIUS, REV, SHAPL, THETA
SPOT		LGP, SMP
Start	STLST	GRAPH, POLAR
STAT		
Step	GGFL, PBAK, KEYW	
STLST (CH 1)		Start
(CH 2)		Start
(CH 3)		Start
(CH 4)		Start
(CH 5)		Start
STOC		CREATC, MAKE
STOK	Start	
STOM		CREATM, SEG2, SEG4
THETA	PCHK	
VIEW	GGFL, KEYW, PBAK	
WRK (CH 1)		Start
(CH 2)		POLAR
(CH 3)		CHART
(CH 4)		SHAPE
(CH 5)		FRACT
XFRM	PGET, PUTP	
YFRM	PGET, PUTP	
πDRW	ADRW, CLCD?, DGTZ?, LINE, NONP, SMP, LGP	
πPAR		PUTπ, πDRW, SPOT, RADIUS (CH 3), STLST(CH 3)
πVAL		

Chapter	TOPIC	DESCRIPTION
1	Introduction	The introduction describes the layout of each chapter and the programs used to create each custom menu
2	Polar Graphing	The polar graphing menu plots the equation stored in 'EQ' in polar coordinates.
3	Chart Drawing	This menu creates a pie chart for the data stored in the statistical array, 'ΣDAT'. You also have the option of labeling the percentage of each section with two different sized print.
4	Shape Drawing	The shape drawing menu creates a wide variety of polygons and stars.
5	Fractal Drawing	Chapter five introduces you to simple fractals. An endless assortment of fractals can be define in the fractal editing menu. Use a predefined format, or draw a freehand model construction and initial curve on the display.
6	Advanced Graphing	Chapter six contains three different advanced graphing programs. DDRW, MDRW, and EDRW transforms any procedure into a three dimensional, animated, or extended graphing program.
7	More Examples	Chapter seven shows you how the graphing menus from part one and part two can work together. Detailed examples demonstrate only a few of the infinite graphs that the 28S can create.

"ADVANCED GRAPHING FOR THE 28S" presents seven different graphing menus. Chapters two through five each contain one unique graphing program while chapter six comprises three. In all, over 120 programs and utilities are incorporated into user friendly menus similar to those built in to the 28S. These menus are described in detail with over 100 different graphs and illustrations. There is even a menu summary at the end of each chapter.

The book is divided into two parts. The first, chapters two through five, presents four different root level graphing menus. These menus allow you to graph polar equations, plot pie charts, draw polygons and stars, and draw fractals.

Chapter six, the second part, contains three different advanced graphing menus. In contrast to the menus in the first part, they can plot many different types of graphs, but always in one specify way. For example, DDRW can plot polar equations, pie charts, draw shapes, or any other graph, but always in three dimensions. Likewise, MDRW animates any graph while EDRW plots large graphs on any number of horizontal and vertical display screens. And best of all, they can be cascaded with any graphing program. You can combine the programs from part one with one or more of the programs in part two to create an endless variety of multi-dimensional graphs.

Finally, the last chapter shows you how the graphing menus from part one and part two can work together. Detailed examples demonstrate only a few of the infinite graphs you never imagined the 28S could create.