ALTOS

UNIX™ System V

Graphics Guide

UNIX™ System V Graphics Guide

ACKNOWLEDGEMENTS

The Altos logo, as it appears in this manual, is a registered trademark of Altos Computer Systems.

DOCUMENTER'S WORKBENCH™ is a trademark of AT&T Technologies.

HP® is a registered trademark of Hewlett-Packard Company.

TEKTRONIX® is a registered trademark of Tektronics, Inc.

UNIX™ is a trademark of AT&T Bell Laboratories.

Versatec® is a registered trademark of Versatec Corporation.

3B™ is a trademark of AT&T Technologies.

CONTENTS

Chapter 1

INTRODUCTION

The *UNIX* System Graphics Guide* provides numerical and graphical commands used to construct and edit numerical data plots and hierarchy charts. This guide is designed for individuals experienced in using the UNIX system, in a variety of ways, within the office environment (electronic mail, document preparation, data analysis, and so on). These individuals are not expected to know programming languages to use the *UNIX System Graphics Guide*, but may write shell procedures for general purposes. This guide also assumes the user is familiar with the *UNIX System User Reference Manual.*

The chapter " OVERVIEW" provides a general description of and an introduction to the UNIX system graphic facility.

The chapter " STATISTICAL NETWORK" (**stat**) describes a collection of routines that can be interconnected using the UNIX operating system shell to form numerical processing networks.

The chapter " GRAPHICS EDITOR" (**ged**) describes an interactive editor used to display, edit, and construct drawings on TEKTRONIX† 4010 series display terminals.

The chapter " ADMINISTRATIVE INFORMATION" is a reference guide for system administrators. Specific information is contained about directory structure, installation, makefiles, hardware requirements, and miscellaneous facilities of the graphics package.

Throughout this volume, each reference of the form **name**(1M), **name**(7), or **name**(8) refers to entries in the *UNIX System Administrator Reference Manual.* All other references to entries of the form **name**(N), where "N" is a number (1 and 6) possibly

---

* Trademark of AT&T Bell Laboratories.

† Registered trademark of Tektronix, Inc.

followed by a letter, refer to entry **name** in section **N** of the *UNIX System User Reference Manual.*

Chapter 2

OVERVIEW

PAGE

Chapter 2

OVERVIEW

1. Chapter Introduction

The UNIX System Graphics, or **graphics**, is the name given to a collection of numerical and graphical commands available as part of the UNIX operating system. In the current release, **graphics** includes commands to construct and edit numerical data plots and hierarchy charts. This chapter will help a user get started using **graphics** and show where to find more information. The examples below assume that the user is familiar with the UNIX operating system shell.

2. Basic Concepts

The basic approach taken with **graphics** is to generate a drawing by describing it rather than by drafting it. Any drawing is seen as having two fundamental attributes-its underlying logic and its visual layout. The layout encompasses one representation of the logic. For example, consider the attributes of a drawing that consists of a plot $y=x^2$ for x between 0 and 10:

- The logic of the plot is the description as just given, namely $y=x^2$, for x between 0 and 10.
- The layout consists of an x-y grid, axes labeled perhaps 0 to 10 and 0 to 100, and lines drawn connecting the x-y pairs 0,0 to 1,1 to 2,4 etc.

The way to generate a picture in **graphics** is:

gather data | transform the data | generate a layout | display the layout

The command to generate the plot, $y=x^2$, for x between 0 and 10 and display it on a TEKTRONIX display terminal would be

gas −s0,t10 | af " x^2" | plot | td

where:

- The **gas** command generates sequences of numbers, in this case starting at 0 and terminating at 10.
- The **af** command performs general arithmetic transformations.
- The **plot** command builds x-y plots.
- The **td** command displays drawings on TEKTRONIX terminals.

The resulting drawing is shown in Figure 2-1.

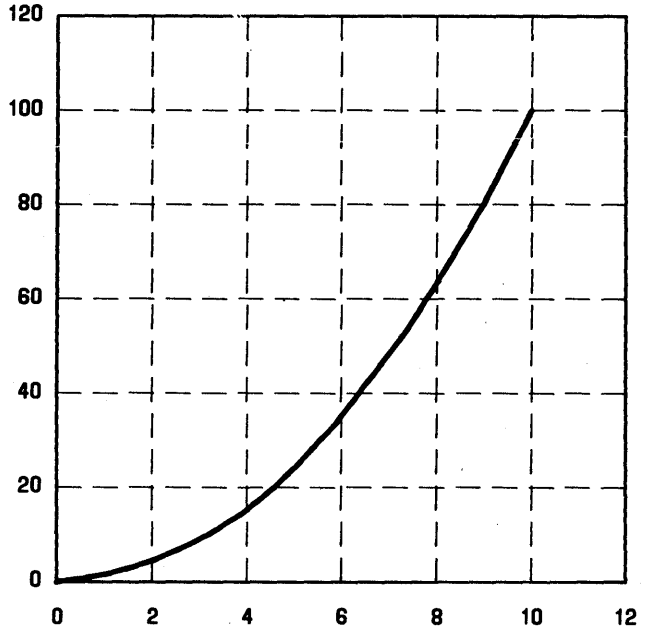

Figure 2-1. Plot of gas −s0,t10 | af " x^2" | plot | td

The layout generated by a **graphics** program may not always be precisely what is wanted. There are two ways to influence the layout. Each drawing program accepts options to direct certain layout features. For instance, in the previous example, it may be desired to have the x-axis labels indicate each of the numbers plotted and not have any y-axis labels at all. To achieve this the **plot** command would be changed to:

```
gas -s0,t10 | af " x^2" | plot -xil,ya | td
```

producing the drawing of Figure 2-2.

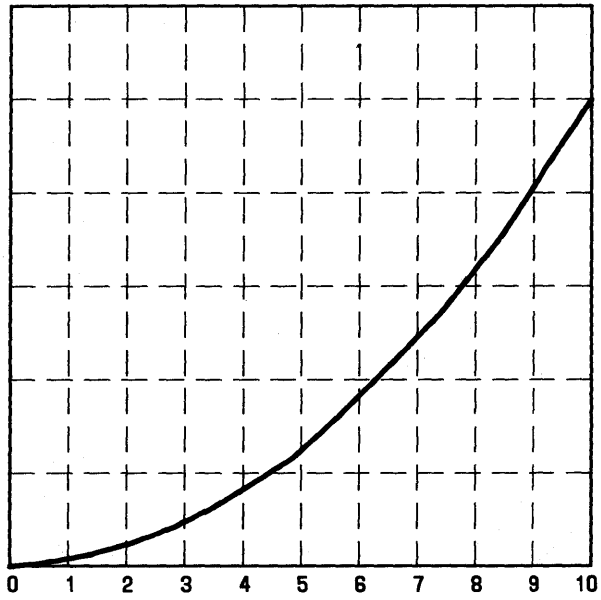

Figure 2-2. Plot of gas -s0,t10 | af " x^2" | plot -xil,ya | td

The output from any drawing command can also be affected by editing it directly at a display terminal using the graphical editor, **ged**. To edit a drawing really means to edit the computer representation of the drawing. In the case of **graphics** the representation is called a graphical primitive string, or GPS. All of the drawing commands (e.g., **plot**) write GPS, and all of the device filters (e.g., **td**) read GPS. **Ged** allows manipulation of GPS at a display terminal by interacting with the drawing the GPS describes.

The GPS describes graphical objects drawn within a Cartesian plane, 65,534 units on each axis. The plane, known as the *universe*, is partitioned into 25 equal-sized square regions. Multidrawing displays can be produced by placing drawings into adjacent regions and then displaying each region.

3. Getting Started

To access the **graphics** commands when logged in on a UNIX system, type **graphics**. The shell variable *PATH* will be altered to include the **graphics** commands and the shell primary prompt will be changed to ^. Any command accessible before typing **graphics** will still be accessible; **graphics** only adds commands, it does not take any away. Exception, the 3B*20 computers **list** command cannot be accessed in the **graphics** mode. Once in **graphics**, a user can find out about any of the **graphics** commands using **whatis**. Typing **whatis** by itself on a command line will generate a list of all the commands in **graphics** along with instructions on how to find out more about any of them.

All of the **graphics** commands accept the same command line format:

- A *command* is a *command-name* followed by *argument*(s).
- A *command-name* is the name of any of the **graphics** commands.

* Trademark of AT&T Technologies.

- An *argument* is a *file-name* or an *option-string.*
- A *file-name* is any file name not beginning with –, or a – by itself to reference the standard input.
- An *option-string* is a – followed by *option*(s).
- An *option* is a letter(s) followed by an optional value. Options may be separated by commas.

The **graphics** commands will produce the best results when used with a display terminal such as the TEKTRONIX display terminal. **Tplot**(1G) filters can be used in conjunction with **gtop** (see **gutil**(1G)) to get somewhat degraded drawings on Versatec printers and Dasi-type terminals. Since GPS can be stored in a file, it can be created from any terminal for later display on a graphical device.

The **graphics** commands can be removed from user's *PATH* shell variable by typing an end-of-file indication (control-d on most terminals). To log off the UNIX operating system from **graphics**, type **quit**.

4. Examples Of What You Can Do

4.1 Numerical Manipulation and Plotting

Stat is a collection of numerical and plotting commands. All of these commands operate on vectors. A vector is a text file that contains numbers separated by delimiters, where a delimiter is anything that is not a number.

For example:

1 2 3 4 5, and
arf tty47 Mar 5 09:52

are both vectors. The latter is the vector:

47 5 9 52.

Here is an easy way to generate a Celsius-Fahrenheit conversion table using **gas** to generate the vector of Celsius values:

```
gas -s0,t100,i10 | af " C,9/5*C+32"        2>/dev/null
```

The output is:

```
0       32
10      50
20      68
30      86
40      104
50      122
60      140
70      158
80      176
90      194
100     212
```

where:

- **gas -s0,t100,i10** generates a sequence that starts at 0, terminates at 100, and the increment between successive elements is 10.

- **af "C,9/5*C+32"** generates the table. Arguments to **af** are expressions. Operands in an expression are either constants or file names. If a file name is given that does not exist in the current directory it is taken as the name for the standard input. In this example **C** references the standard input. The output is a vector with odd elements coming from the standard input and even elements being a function of the preceding odd element.

- **2>/dev/null** suppresses the printing of warning messages. It redirects error message to **/dev/null**.

Here is an example that illustrates the use of vector titles and multiline plots:

```
gas | title -v "first ten integers" >N
root N >RN
root -r3 N >R3N
root -r1.5 N >R1.5N
plot -FN,g N R1.5N RN R3N | td
```

where:

- **title -v** "*name*" **associates a** *name* with a vector. In this case, **first ten integers** is associated with the vector output by **gas**. The vector is stored in file **N**.
- **root -r***n* outputs the *n*th root of each element on the input. If **-r***n* is not given, then the square root is output. Also, if the input is a titled vector, the title will be transformed to reflect the root function.
- **plot -F***X,g Y(s)* generates a multiline plot with *Y(s)* plotted versus *X*. The **g** option causes tick marks to appear instead of grid lines.

The resulting plot is shown in Figure 2-3.

The next example generates a histogram of random numbers:

```
rand -n100 | title -v "100 random numbers" | qsort |
    bucket | hist | td
```

where:

- **rand -n100** outputs random numbers using **rand**(3C). In this case 100 numbers are output in the range 0 to 1.
- **qsort** sorts the elements of a vector in ascending order.

- **bucket** breaks the range of the elements in a vector into intervals and counts how many elements from the vector fall into each interval. The output is a vector with odd elements being the interval boundaries and even elements being the counts.

- **hist** builds a histogram based on interval boundaries and counts.

The output is shown in Figure 2-4.

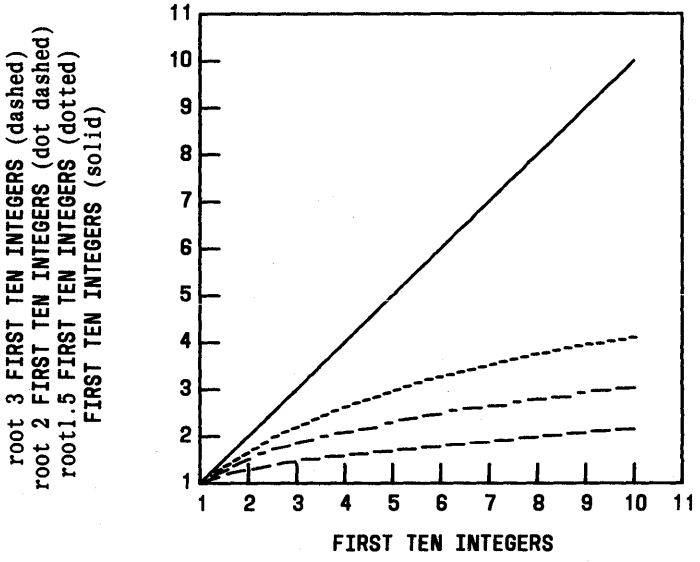

Figure 2-3. Some Roots of the First Ten Integers

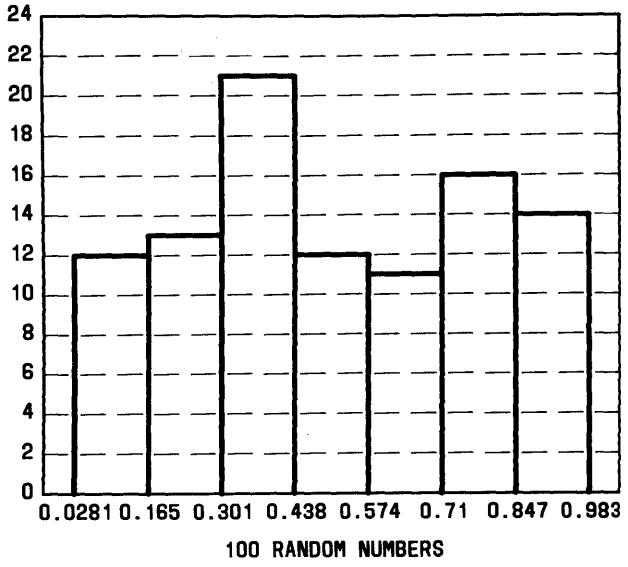

Figure 2-4. Histogram of 100 Random Numbers

4.2 Drawings Built From Boxes

There is a large class of drawings composed from boxes and text. Examples are structure charts, configuration drawings, and flow diagrams. In **graphics** the general procedure to construct such box drawings is the same as that for numerical plotting; namely, gather and transform the data, build and display the layout.

As an example, for hierarchy charts, the command line

```
dtoc | vtoc | td
```

outputs drawings representing directory structures.

- The **dtoc** command outputs a table of contents that describes a directory structure (Figure 2-5). The fields from left to right are level number, directory name, and the number of ordinary readable files contained in the directory.

- The **vtoc** command reads a (textual) table of contents and outputs a visual table of contents, or hierarchy chart (Figure 2-6). Input to **vtoc** consists of a sequence of entries, each describing a box to be drawn. An entry consists of a level number, an optional style field, a text string to be placed in the box, and a mark field to appear above the top right-hand corner of the box.

```
0.      "source"        2
1.      "glib.d"        1
1.1.    "gpl.d"         12
1.2.    "gsl.d"         14
2.      "gutil.d"       6
2.1.    "cvrtopt.d"     7
2.2.    "gtop.d"        8
2.3.    "ptog.d"        5
3.      "stat.d"        54
4.      "tek4000.d"     5
4.1     "ged.d"         37
4.4.    "td.d"          8
5.      "toc.d"         3
5.1.    "ttoc.d"        3
5.2.    "vtoc.d"        22
6.      "whatis.d"      108
```

Figure 2-5. Output of dtoc Command

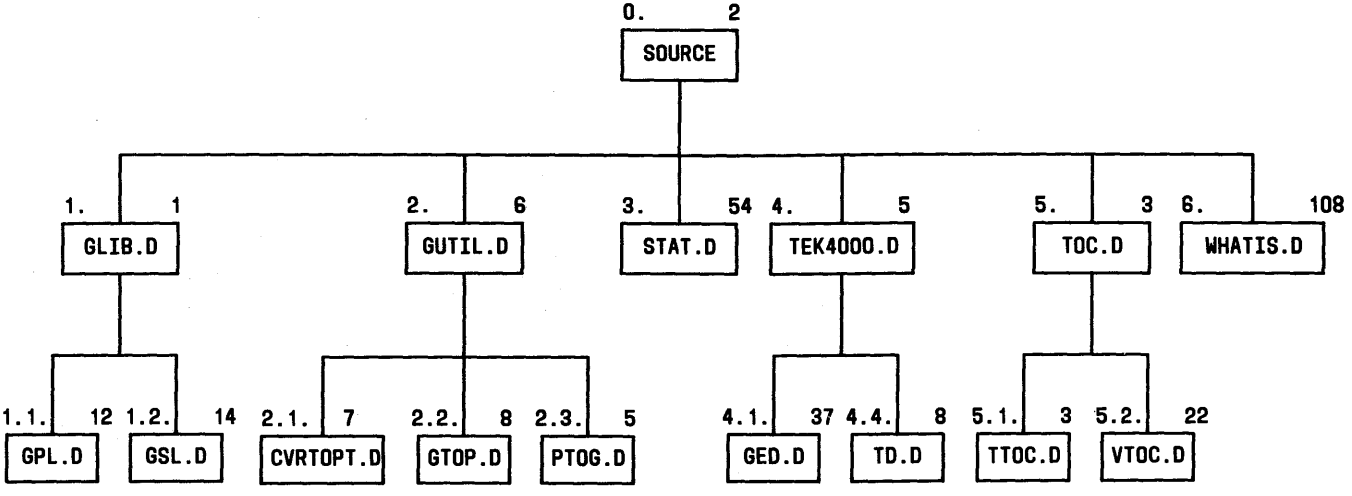

Figure 2-6. Output of vtoc Command

5. Where To Go From Here

The best way to learn about **graphics** is to log onto a UNIX operating system and use it. Other chapters in this guide contain tutorials for **stat**(1G) and **ged**(1G) and administrative information for **graphics**. Additional information can be found in the *UNIX System User Reference Manual* in the following manual entries:

gdev(1G), a collection of commands to manipulate TEKTRONIX 4000 series terminals; and
ged(1G), the graphical editor;
graphics(1G), the entry point for **graphics**;
gutil(1G), a collection of graphical utility commands;
stat(1G), numerical manipulation and plotting commands;
toc(1G), routines to build tables of contents;
gps(5), a description of a graphical primitive string.

Chapter 3

STAT-A TOOL FOR ANALYZING DATA

PAGE

Chapter 3

STAT-A TOOL FOR ANALYZING DATA

1. Chapter Introduction

This chapter introduces **stat** concepts and commands through a collection of examples. Also, a complete definition of each command is provided.

Stat is a collection of numerical programs that can be interconnected using the UNIX system shell to form processing networks. Included within **stat** are programs to generate simple statistics and pictorial output.

Much of the power for manipulating text in the UNIX operating system comes from the DOCUMENTER'S WORKBENCH* software text processing package. The general interface is an unformatted text string. The interconnection mechanism is usually the UNIX system shell. The programs are independent of one another, new functions can easily be added and old ones changed. Because the text editor operates on unformatted text, arbitrary text manipulation can always be performed even when the more specialized routines are insufficient.

Stat uses the same mechanisms to bring similar power to the manipulation of numbers. It consists of a collection of numerical processing routines that read and write unformatted text strings. It includes programs to build graphical files that can be manipulated using a graphical editor. And since **stat** programs process unformatted text, they can readily be connected with other UNIX operating system command-level (i.e., callable from shell) routines.

It is useful to think of the shell as a tool for constructing processing networks in the sense of data flow programming. Command-level

---

* Trademark of AT&T Technologies.

routines are the nodes of the network, and pipes and tees are the links. Data flows from node to node in the network via data links. Throughout this chapter, the operator := means *defined as.*

2. Basic Concepts

All numerical data in **stat** are stored in vectors. A *vector* is a sequence of numbers separated by delimiters. Vectors are processed by command-level routines called *nodes.*

2.1 Transformers

A *transformer* is a node that reads an input element, operates upon it, and outputs the resulting value. For example, suppose file **A** contains the vector

1 2 3 4 5

then the command

root A (typed input is **bold**)

produces

1 1.41421 1.73205 2 2.23607

the square root of each input element. Also,

log A

produces

0 0.693147 1.09861 1.38629 1.60944

the natural logarithm of each element of vector **A.**

Af, for arithmetic function, is a particularly versatile transformer. Its argument is an expression that is evaluated once for each complete set of input values. A simple example is

```
af "2*A^2"
```

which produces

```
2       8       18      32      50
```

twice the square of each element from **A**. Expression arguments to **af** are usually surrounded by quotes since some of the operator symbols have special meaning to the shell.

2.2 Summarizers

A *summarizer* is a node that calculates a statistic for a vector. Typically, summarizers read in all of the input values, then calculate and output the statistic. For example, using the vector **A** from the previous example,

```
mean A
```

produces

```
3
```

and

```
total A
```

produces

```
15
```

2.3 Parameters

Most nodes accept parameters to direct their operation. Parameters are specified as command-line options. **Root**, for example, is more general than just square root, any root may be specified using the **r** option. For example,

```
root -r3 A
```

produces

```
1   1.25992   1.44225   1.5874   1.70998
```

the cube root of each element from **A**.

2.4 Building Networks

Nodes are interconnected using the standard UNIX system shell concepts and syntax. A pipe is a linear connector that attaches the output of one node to the input of another. As an example, to find the mean of the cube roots of vector **A** is simply

```
root -r3 A | mean
```

which produces

```
1.39991
```

Often the required network is not so simple. Tees and sequence can be used to build nonlinear networks. To find the mean and median of the transformed vector **A** is

```
root -r3 A | tee B | mean; point B
```

which produces

```
1.39991
1.44225
```

Beware of the distinction between the sequence operator, (;), and the linear connector, the pipe (|). Because processes in a pipeline run concurrently, each file written to in the pipeline should be unique. Sequence implies run to completion (so long as **&** is not used) hence files may be used more than once.

There is a special case of nonlinear networks where the result of one node is used as command-line input for another. Command substitution makes this easy. For example, to generate residuals from the mean of **A** is simply

```
af "A-`mean A`
```

which results in

```
-2      -1      0       1       2
```

2.5 Vectors, a Closer Look

Thus far vectors have been used but not created. One way to create a vector is by using a generator. A *generator* is a node that accepts no input and outputs a vector based upon definable parameters. **Gas** is a generator that produces additive sequences. One of the parameters to **gas** is the number of elements in the generated vector. As an example, to create the vector **A** that we have been using is

```
gas -n5
```

which produces

```
1       2       3       4       5
```

Vectors are, however, merely text files. Hence, the text editor can be used to create and modify the same vector.

A useful property of vectors is that they consist of a sequence of numbers surrounded by delimiters, where a delimiter is anything that is not a number. Numbers are constructed in the usual way

[sign](digits)(.digits)[e[sign]digits]

where fields are surrounded by brackets and parentheses. All fields are optional, but at least one of the fields surrounded by parentheses must be present. Thus vector **A** could also be created by building the file **B** in the text editor as

1partridge,2tdoves,3frhens,4cbirds,5gldnrings,

which, when read by

list B

produces

1 2 3 4 5

A note should be made as to the size of a vector. A vector is a stream containing numbers terminated by an end of file (control-d from the keyboard). A good illustration of this is to use the keyboard as the source of the input vector, as in

cusum −c1
2<cr>
2
16.3<cr>
18.3
25.4<cr>
43.7
−14<cr>
29.7
<control d>

which implements a running accumulator. Since no vector was given

to **cusum**, the input is taken from the standard input until an end of file.

2.6 A Simple Example: Interacting with a Data Base

When used in conjunction with the UNIX operating system tools for manipulating text, **stat** provides an effective means for exploring a numerical data base. Suppose, for example, there is a subdirectory called **data** containing data files that include the lines:

path length = *nn* (*nn* is any number)
node count = *nn*

To access the value for **node count** from each file, sort the values into ascending order, store the resulting vector in file **A**, and get a copy on the terminal by typing

```
grep "node count" data* | qsort | tee A
17      19      22      32      39
50      68      78      125     139
```

If some of the data files have numbers in their name, we must protect those numbers from being considered data. Using **cat**, this is easy:

```
cat data/* | grep "node count" | qsort | tee A
```

To get a feel for the distribution of node counts, shell iteration can be used to advantage.

```
for i in .25 .5 .75
do point -p$i A
done
24.5
44.5
75.5
```

This generates the lower hinge, the median, and the upper hinge of the sorted vector **A**.

2.7 Translators

Translators are used to view data pictorially. A *translator* is a node that produces a stream of a different structure from that which it consumes. Graphical translators consume vectors and produce pictures in a language called GPS. Among the programs that understand GPS is **ged**, the graphical editor, which means that the graphical output of any translator can be edited at a display terminal. **Hist** is an example of a translator; it produces a GPS that describes a histogram from input consisting of interval limits and counts. The summarizer **bucket** produces limits and counts, thus

bucket A | hist | td

generates a histogram of the data of vector **A** and displays it on a display terminal (Figure 3-1). **Td** translates the GPS into machine code for TEKTRONIX 4010 series display terminals.

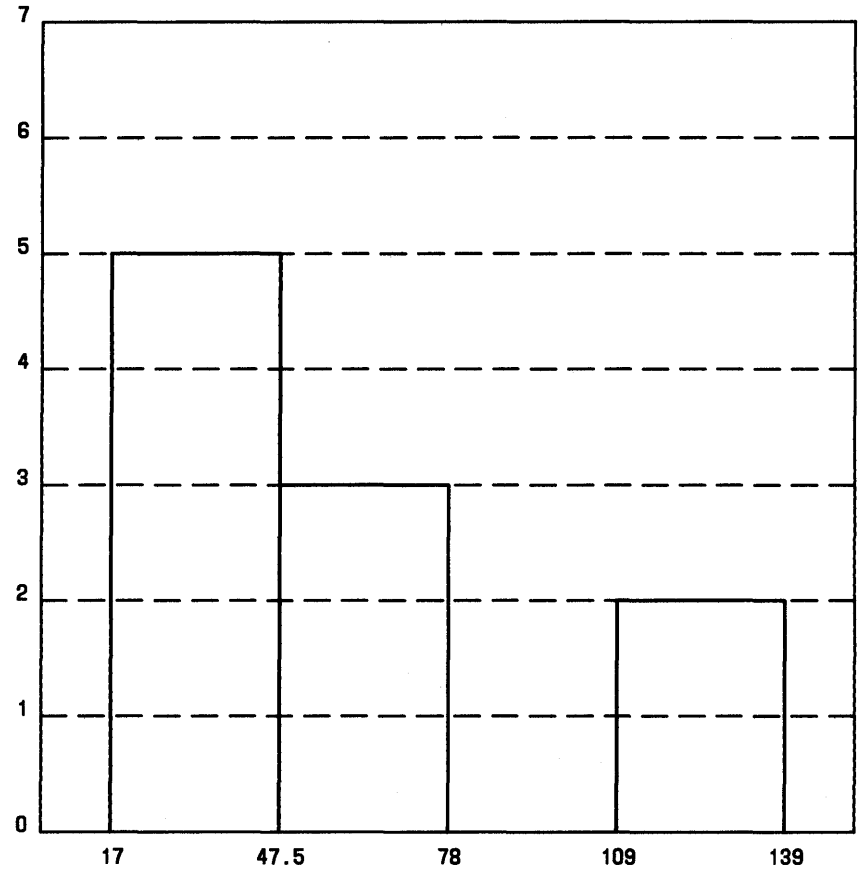

Figure 3-1. bucket A | hist | td

A wide range of X-Y plots can be constructed using the translator **plot**. For example, to build a scatter plot of **path length** with **node count** (Figure 3-2) is

```
grep "path length" data/* | title -v"path length" >A
grep "node count" data/* | title -v"node count" | plot
    -FA,dg | td
```

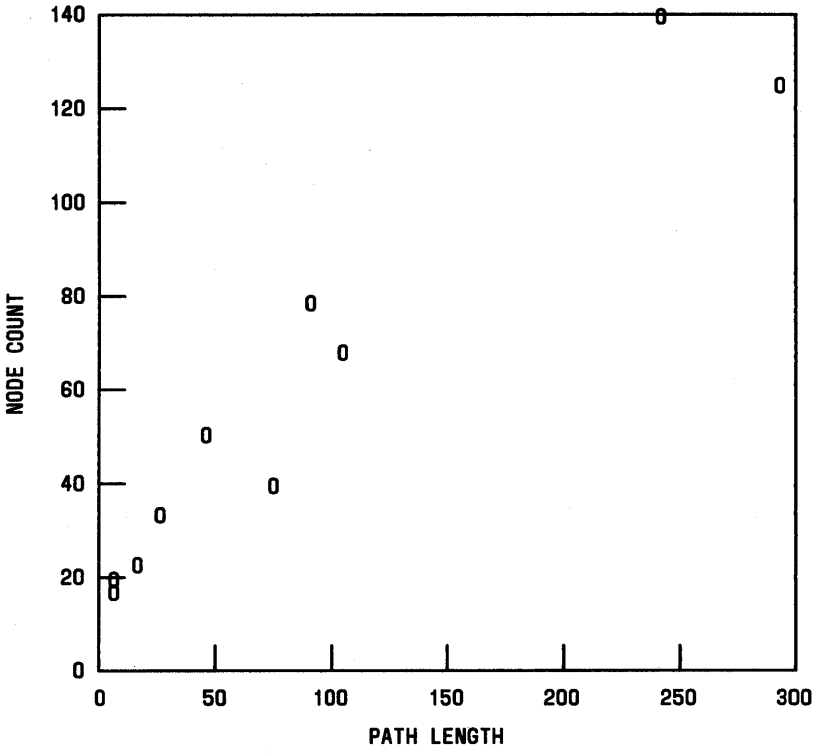

Figure 3-2. Scatter Plot

A vector may be given a title using **title**. When a titled vector is plotted, the appropriate axis is labeled with the vector title. When a titled vector is passed through a transformer, the title is altered to reflect the transformation. Thus in a graph of log **node count** versus the cube root of **path length**, i.e.,

```
grep "node count" | title -v "node count" | log >B
root -r3 A | plot -F-,dg B | td
```

the axis labels automatically agree with the vectors plotted (Figure 3-3).

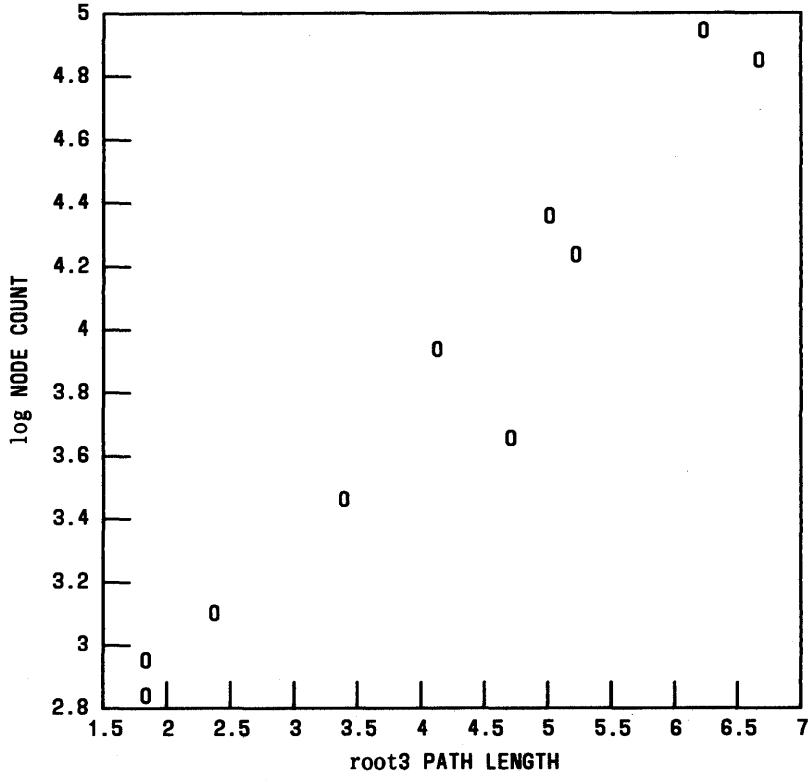

Figure 3-3. Transformed Scatter Plot

3. Node Descriptions

In this section a more formal description of each node is given. The mathematical formula given with each description corresponds to the algorithm implemented by the command. The descriptions are organized by node class. The **stat** nodes are divided into these four classes:

- *Transformers*
- *Summarizers*
- *Translators*

- *Generators*

All of the nodes accept the same command-line format:

- A *command* is a *command-name* followed by zero or more *arguments.*
- A *command-name* is the name of any **stat** node.
- An *argument* is a *file-name* or an *option-string.*
- An *option-string* is a — followed by one or more *options.*
- An *option* is one or more letters followed by an optional value. Options may be separated by commas.
- A *file-name* is any name not beginning with —, or a — by itself (to reference the standard input).

Each file argument to a node is taken as input to one occurrence of the node. That is, the node is executed from its initial state once per file. If no files are given, the standard input is used. All nodes, except generators, accept files as input, hence it is not made explicit in the synopses that follow.

Most nodes accept command-line *options* to direct the execution of the node. Some *options* take values. In the following synopses, to indicate the type of value associated with an *option*, the *option* key-letter is followed by:

i	to indicate integer,
f	to indicate floating point or integer,
string	to indicate a character string, or
file	to indicate a file-name.

Thus, the *option* **c***i* implies that **c** expects an integer value (**c** := integer).

3.1 Transformers

Transformers have the form

V_{in} *transform* V_{out}

where, by convention, V_{in} is a vector Y, with elements y_1 through y_k ($y_{1:k}$) and V_{out} is a vector Z, $z_{1:m}$. All transformers have a **c***i* option, where **c** specifies the number of columns per line in the output. By default, **c** := 5.

abs - absolute value

$z_i := |y_i|$

af [**-t v**] - arithmetic function

The command-line format of **af** is an extension of the command-line description given above, with *expression* replacing *file-name*; an *expression* consists of *operands* and *operators*.

An *operand* is either a *vector, function, constant,* or *expression*:

- A *vector* is a file name with the restriction that file names begin with a letter and are composed only of letters, digits, ".", and "_". The first unknown file name (one not in the current directory) references the standard input. A warning will appear if a file cannot be read.

- A *function* is the name of a command followed by its arguments in parentheses. Arguments are written in command-line format.

- A *constant* is an integer or floating point (but not "E" notation) number.

The *operators* are listed below in order of decreasing precedence. Parentheses may be used to alter precedence.

The x_i (y_i) represents the start element from X (Y) for the expression.

- $'Y$ - reference y_{i+1}. y_{i+1} is consumed; the next value from Y is y_{i+2}. Y is a vector.
- $X\hat{}\ \ Y$ - $-Y$ - x_i raised to the y_i power, negation of y_i. Association is right to left. X and Y are expressions.
- $X*Y$ X/Y $X\%Y$ - x_i multiplied by, divided by, modulo y_i. Association is left to right. X and Y are expressions.
- $X+Y$ $X-Y$ - x_i plus, minus y_i. Association is left to right. X and Y are expressions.
- X,Y - yields x_i, y_i. Association is left to right. X and Y are expressions.

Options:

t	causes the output to be titled from the vector on the standard input.
v	causes function expansions to be echoed.

ceil - ceiling

z_i := smallest integer greater than y_i

cusum - cumulative sum

$$z_i := \sum_{j=1}^{i} y_j$$

exp - exponential function

$$z_i := e^{y_i}$$

floor - floor

z_i := largest integer less than y_i

gamma - gamma function

$z_i := \Gamma(y_i)$

list [-**d***string*] - list vector elements

$z_i := y_i$

If **d** is not specified, then any character that is not part of a number is a delimiter.
If **d** is specified, then the white space characters (space, tab, and new-line) plus the character(s) of *string* are delimiters.
Only numbers surrounded by delimiters are listed.

log [-**b***f*] - logarithmic function

$z_i := \log_{\mathbf{b}} y_i$

By default, **b** := e (e ≈2.71828...)

mod [-**m***f*] - modulus

$z_i := y_i$ modulo **m**

By default, **m** := 2

pair [-**F***file* **x***i*] - pair elements

F is a vector X, $x_{1:j}$, and **x** is the number of elements per group from X.
Let % denote modulo and / denote integer division, then

$$z_i := \begin{cases} y_{(i/(\mathbf{x}+1))} & \text{if } i\,\%(\mathbf{x}+1) = 0 \\ x_{(i-i/(\mathbf{x}+1))} & \text{if } i\,\%(\mathbf{x}+1) \neq 0 \end{cases}$$

$rank(Z) = (\mathbf{x}+1)\text{minimum}(k, j/\mathbf{x})$

If **F** is not specified, then X comes from the standard input. If both X and Y come from the standard input, X precedes Y.

By default, **x** := 1

power [-**p***f*] - raise to a power

$z_i := y_i^{\mathbf{p}}$

By default, **p** := 2

root [-**r***f*] - extract a root

$z_i := \sqrt[\mathbf{r}]{y_i}$

By default, **r** := 2

round [-**p***i* **s***i*] - round off values

if **s** is specified, then
$z_i := y_i$ rounded up to **s** significant digits,

else if **p** is specified, then
$z_i := y_i$ rounded up to **p** digits beyond the decimal point.

By default, **p** := 0

siline [-**i***f* **n***i* **s***f*] - generate a line, given a slope and intercept

$z_i = \mathbf{s}\ y_i + \mathbf{i}$

if **n** is specified, then

$Y \equiv 0, 1, 2, 3, \cdots, \mathbf{n}$.

By default, **i** := 0, **s** := 1

sin - sine function

$z_i := \sin(y_i)$

spline [-options] - interpolate smooth curve

Y and *Z* are sequences of X,Y coordinates
(like that produced by **pair**).

For more information about **spline**, see **spline**(1) in the *UNIX System User Reference Manual.*

subset [-**a***f* **b***f* **F***file* **i***i* **l***f* **nl np p***f* **s***i* **t***i*] - generate a subset

Z consists of elements selected from *Y*. Selection occurs as follows:

Let C(*w*) be true if

($w > \mathbf{a}$ or $w < \mathbf{b}$ or $w = \mathbf{p}$) and $w \neq \mathbf{l}$

is true. If neither **a**, **b**, nor **p** are specified, C(*w*) is true if $w \neq \mathbf{l}$ is true.

CASE 1 - **nl** or **np** not specified.

If **F** is specified, then $key_i = x_i$
else $key_i = y_i$

For $r = \mathbf{s}, \mathbf{s} + \mathbf{i}, \mathbf{s} + 2\mathbf{i}, \ldots$ with $r \leq \mathbf{t}$,
y_r becomes an element of Z if C(key_r) is true.

By default, **i** := 1, **s** := 1, **t** := 32767.

CASE 2 - **np** is specified.

F is a vector X, $x_{1:j}$.
For $r = x_1, x_2, \ldots, x_j$,
y_r becomes an element of Z if C(y_r) is true.

CASE 3 - **nl** is specified.

F is a vector X, $x_{1:j}$.
For $r \neq x_1, x_2, \cdots, x_j$,
y_r becomes an element of Z if C(y_r) is true.

For cases 2 and 3, if **F** is not specified, then the standard input is used for X. Either X or Y may come from the standard input, but not both.

3.2 Summarizers

Summarizers have the form

V_{in} *summarize* V_{out}

where, again, V_{in} is a vector Y, $y_{1:k}$, and V_{out} is a vector Z, $z_{1:m}$. For many summarizers, $rank(Z) = 1$.

bucket [-**a***i* **c***i* **F***file* **h***f* **i***i* **l***f* **n***i*] - break into buckets

Y must be a sorted vector.
Z consists of odd elements (parenthesized) which are bucket limits and even elements which are bucket counts.

The count is the number of elements from Y greater than the

lower limit (greater than or equal to for the lowest limit), and less than or equal to the higher limit. If specified, the limit values are taken from **F**. Otherwise the limits are evenly spaced between **l** and **h** with a total of **n** buckets. If **n** is not specified, the number of buckets is determined as follows:

$$\mathbf{n} := \begin{cases} \dfrac{\mathbf{h} - \mathbf{l}}{\mathbf{i}} & \text{if } \mathbf{i} \text{ is specified} \\ \dfrac{\mathrm{k}}{\mathbf{a} + 1} & \text{if } \mathbf{a} \text{ is specified} \\ 1 + \log_2 \mathrm{k} & \text{if neither } \mathbf{a} \text{ nor } \mathbf{i} \text{ are specified.} \end{cases}$$

c specifies the number of columns in the output.

By default:

c := 5
h := largest element of Y
l := smallest element of Y

cor [-F*file*] - correlation coefficient

If **F** is a vector X, $x_{1:k}$,

let $\bar{x} = \dfrac{\sum_{i=1}^{k} x_i}{k}$ and

$\bar{y} = \dfrac{\sum_{i=1}^{k} y_i}{k}$, then

$$z_1 := \frac{\sum_{i=1}^{k} (x_i - \bar{x})(y_i - \bar{y})}{\left[\sum_{i=1}^{k} (x_i - \bar{x})^2\right]^{1/2} \left[\sum_{i=1}^{k} (y_i - \bar{y})^2\right]^{1/2}}$$

X and Y must have the same rank.
If **F** is not specified, the standard input is used for X.
If both X and Y come from the standard input,
X precedes Y.

hilo [**-h l o ox oy**] - high and low values

z_1 := lowest value across all input vectors

z_2 := highest value across all input vectors

Options to control output:

h	Only output high value.
l	Only output low value.
o	Output high, low values in option form (suitable for **plot**).
ox	Output high, low values with "x" prefixed.
oy	Output high, low values with "y" prefixed.

lreg [**-F***file* **i o s**] - linear regression

If **F** is a vector X, $x_{1:k}$, let $\bar{x} = \dfrac{\sum_{i=1}^{k} x_i}{k}$ and $\bar{y} = \dfrac{\sum_{i=1}^{k} y_i}{k}$, then

$$z_1 := \bar{y} - z_2\bar{x} \qquad \text{(intercept)}$$

and

$$z_2 := \frac{\dfrac{\sum_{i=1}^{k} x_i y_i}{k} - \bar{x}\,\bar{y}}{\dfrac{\sum_{i=1}^{k} x_i^2}{k} - \bar{x}^2} \qquad \text{(slope)}$$

X and Y must have the same rank.
If **F** is not specified, then

$$X \equiv 0, 1, 2, \cdots, k$$

Options to control output:

i	Only output the intercept.
o	Output the slope and intercept in option form (suitable for **siline**).
s	Only output the slope.

mean [-**f***f* **n***i* **p***f*] - (trimmed) mean

$$z_1 := \frac{\sum_{i=1}^{k} y_i}{k}$$

Y may be trimmed by

(1/**f**) k	elements from each end,
p k	elements from each end, or
n	elements from each end.

By default, **n** :=0

point [-**f***f* **n***i* **p***f* **s**] - empirical cumulative density function point

z_1 := linearly interpolated *Y* value corresponding to the

100 (1/**f**)	percent point, the
100 **p**	percent point, or the
nth	element.

Negative option values are taken from the high end of *Y*.
Option **s** implies *Y* is sorted.

By default, **p** := .5 (median)

prod - product

$$z_1 := \prod_{i=1}^{k} y_i$$

qsort [-**c***i*] - quicksort

z_i := ith smallest element of Y.

By default, **c** := 5

rank - rank

z_1 := number of elements in Y.

total - sum

$$z_1 := \sum_{i=1}^{k} y_i$$

var - variance

$$z_1 := \frac{\sum_{i=1}^{k} (y_i - \bar{y})^2}{k - 1}$$

3.3 Translators

Translators have the form

F_{in} *translate* F_{out}

where F_{in} may be a vector or a GPS depending upon the translator. F_{out} is a GPS. A GPS is a format for storing a picture. A picture is defined in a Cartesian plane of 64K points on each axis. The plane, or universe, is divided into 25 square regions numbered 1 to 25 from the lower left to the upper right. Various commands exist that can display and edit a GPS. For more information, see **graphics**(1) in the *UNIX System User Reference Manual* and *UNIX System Graphics Overview.*

bar [**-a b f g r***i* **w***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] - build a bar chart

F_{in} is a vector, each element of which defines the height of a bar. By default, the x-axis will be labeled with positive integers

beginning at 1; for other labels, see **label**.

Options:

a	Suppress axes.
b	Plot bar chart with bold weight lines, otherwise use medium.
f	Do not build a frame around plot area.
g	Suppress background grid.
r*i*	Put the bar chart in GPS region *i*, where *i* is between 1 and 25 inclusive. The default is 13.
w*i*	*i* is the ratio of the bar width to center-to-center spacing expressed as a percentage. Default is 50, giving equal bar width and bar space.
x*f* (**y***f*)	Position the bar chart in the GPS universe with x-origin (y-origin) at *f*.
xa (**ya**)	Do not label x-axis (y-axis).
yl*f*	*f* is the y-axis low tick value.
yh*f*	*f* is the y-axis high tick value.

hist [**-a b f g r***i* **x***f* **xa y***f* **ya yl***f* **yh***f*] - build a histogram

F_{in} is a vector (of the type produced by **bucket**) of odd rank, with odd elements being limits and even elements being bucket counts.

Options:

a	Suppress axes.
b	Plot histogram with bold weight lines, otherwise use medium.
f	Do not build a frame around plot area.
g	Suppress background grid.
r*i*	Put the histogram in GPS region *i*, where *i* is between 1 and 25 inclusive. The default is 13.
x*f* (**y***f*)	Position the histogram in the GPS universe with x-origin (y-origin) at *f*.
xa (**ya**)	Do not label x-axis (y-axis).
yl*f*	*f* is the y-axis low tick value.
yh*f*	*f* is the y-axis high tick value.

label [**-b c F***file* **h p r***i* **x xu y yr**] - label the axis of a GPS file

F_{in} is a GPS of a data plot (like that produced by **hist, bar,** and **plot**). Each line of the label *file* is taken as one label. Blank lines yield null labels. Either the GPS or the label *file*, but not both, may come from the standard input.

Options:

b	Assume the input is a bar chart.
c	Retain lower case letters in labels, otherwise all letters are upper case.
F*file*	*file* is the label *file*.
h	Assume the input is a histogram.
p	Assume the input is an x-y plot. This is the default.
r*i*	Labels are rotated *i* degrees. The pivot point is the first character.
x	Label the x-axis. This is the default.
xu	Label the upper x-axis, i.e., the top of the plot.
y	Label the y-axis.
yr	Label the right y-axis, i.e., the right side of the plot.

pie [**-b o p pn***i* **pp***i* **r***i* **v x***i* **y***i*] - build a pie chart

F_{in} is a vector with a restricted format. Each input line represents a slice of pie and is of the form:

[< **i e f c***color* >] value [label]

with brackets indicating optional fields. The control field options have the following effect:

i	The slice will not be drawn, though a space will be left for it.
e	The slice is " exploded" or moved away from the pie.
f	The slice is filled. The angle of fill lines depends on the color of the slice.
c*color*	The slice is drawn in *color* rather than the default black. Legal values for *color* are **b** for

black, **r** for red, **g** for green, and **u** for blue.

The pie is drawn with the value of each slice printed inside and the label printed outside.

Options:

b	Draw pie chart in bold weight lines, otherwise use medium.
o	Output values around the outside of the pie.
p	Output value as a percentage of the total pie.
pn*i*	Output value as a percentage, but total of percentages equals *i* rather than 100. The option **pn**100 is equivalent to **p**.
pp*i*	Only draw *i* percent of a pie.
r*i*	Put the pie chart in region *i*, where *i* is between 1 and 25 inclusive. The default is 13.
v	Do not output values.
x*i* (**y***i*)	Position the pie chart in the GPS universe with x-origin (y-origin) at *i*.

plot [**-a b c***string* **d f F***file* **g m r***i* **x***f* **xa xh***f* **xi***f* **xl***f* **xn***i* **xt y***f* **ya yh***f* **yi***f* **yl***f* **yn***i* **yt**] - plot a graph

F_{in} is a vector(s) which contains the y values of an x-y graph. Values for the x-axis come from **F**. Axis scales are determined from the first vector plotted.

Options:

a	Suppress axes.
b	Plot graph with bold weight lines, otherwise use medium.
c*string*	The character(s) of *string* are used to mark points. Characters from *string* are used, in order, for each separately plotted graph included in the plot. If the number of characters in *string* is less than the number of plots, the last character will be used for all remaining plots. The **m** option is implied.
d	Do not connect plotted points, implies option **m**.
f	Do not build a frame around plot area.

F*file*	Use *file* for x-values, otherwise the positive integers are used. This *option* may be used more than once, causing a different set of x-values to be paired with each input vector. If there are more input vectors than sets of x-values, the last set applies to the remaining vectors.
g	Suppress the background grid.
m	Mark the plotted points.
r*i*	Put the graph in GPS region *i*, where *i* is between 1 and 25 inclusive. The default is 13.
x*f* (**y***f*)	Position the graph in the GPS universe with x-origin (y-origin) at *f*.
xa (**ya**)	Omit x-axis (y-axis) labels.
xh*f* (**yh***f*)	*f* is the x-axis (y-axis) high tick value.
xi*f* (**yi***f*)	*f* is the x-axis (y-axis) tick increment.
xl*f* (**yl***f*)	*f* is the x-axis (y-axis) low tick value.
xn*i* (**yn***i*)	*i* is the approximate number of ticks on the x-axis (y-axis).
xt (**yt**)	Omit x-axis (y-axis) title.

title [**-b c l***string* **v***string* **u***string*] - title a vector or GPS

F_{in} can be either a GPS or a vector with F_{out} being of the same type as F_{in}. **Title** prefixes a **title** to a vector or appends a **title** to a GPS.

Options apply as indicated:

b	Make the GPS **title** bold.
c	Retain lower case letters in **title**, otherwise all letters are upper case.
l*string*	For a GPS, generate a lower **title** := *string*.
u*string*	For a GPS, generate an upper **title** := *string*.
v*string*	For a vector, **title** := *string*.

3.4 Generators

Generators have the form

generate V_{out}

where V_{out} is a vector Z, $z_{1:k}$. All generators have a **c***i* option where **c** specifies the number of columns per line in the output. By default, **c** := 5.

gas [-**i***f* **n***i* **s***f* **t***f*] - generate additive sequence

Z is constructed as follows:

$z_1 := \mathbf{s}$

$$z_{i+1} := \begin{cases} z_i + \mathbf{i} & \text{if } |z_i| \leqq \mathbf{t} \\ z_1 & \text{otherwise} \end{cases}$$

rank(Z) = **n**.

By default, **i** := 1, **n** := 10, **s** := 1, **t** := ∞

prime [-**h***i* **l***i* **n***i*] - generate prime numbers

The elements of Z are consecutive prime numbers with

$1 \leqq z_i \leqq \mathbf{h}$

rank(Z) $\leqq$ **n**.

By default, **n** := 10, **l** := 2, **h** := ∞.

rand [-**h***f* **l***f* **m***f* **n***i* **s***i*] - generate random sequence

The elements of Z are random numbers generated by a multiplicative congruential generator with **s** acting as a seed, such that

$$1 \leqq z_i \leqq \mathbf{h}$$

If **m** is specified, then

$$\mathbf{h} = \mathbf{m} + \mathbf{l}$$

rank(Z) = **n**.

By default, **h** := 1, **l** := 0, **n** := 10, **s** := 1.

4. Examples

4.1 Example 1:

PROBLEM

Calculate the total value of an investment held for a number of years at an interest rate compounded annually.

SOLUTION

```
Principal=1000
echo Total return on $Principal units compounded annually
echo "rates:\t\t\c" ; gas -s.05,t.15,i.03 | tee rate
for Years in 1 3 5 8
do
   echo "$Years year(s):\t\c" ; af "$Principal*(1+rate)^$Years'
done
```

Total return on 1000 units compounded annually

rates:	0.05	0.08	0.11	0.14
1 year(s):	1050	1080	1110	1140
3 year(s):	1157.62	1259.71	1367.63	1481.54
5 year(s):	1276.28	1469.33	1685.06	1925.41
8 year(s):	1477.46	1850.93	2304.54	2852.59

NOTES

Notice the distinction between vectors and constants as operands in the expression to **af**. The shell variables $*Principal* and $*Years* are constants to **af**, while the file *rate* is a vector. **Af** executes the expression once per element in *rate*.

4.2 Example 2:

PROBLEM

Given are three ordered vectors (**A**, **B**, and **C**) of scores from a number of tests. Each vector is from one test-taker, each element in a vector is the score on one test. There are missing scores in each vector indicated by the value -1. Generate three new vectors containing scores only for those tests where no data is missing.

SOLUTION

```
echo Before:
gas -n`rank A` | tee N | af "label,A,B,C"

for i in N B C A
do subset -FA,l-1 $i >s$i; done
for i in N A C B
do subset -FsB,l-1 s$i | yoo s$i; done
for i in N A B C
do subset -FsC,l-1 s$i | yoo s$i; done

echo "\nAfter:"
af " sN,sA,sB,sC"
```

Before:

1	5	6	-1
2	7	10	10
3	-1	10	9
4	10	-1	8
5	6	5	-1
6	5	7	5
7	-1	7	8
8	-1	-1	8
9	3	-1	8
10	6	10	10
11	7	5	7

After:

2	7	10	10
6	5	7	5
10	6	10	10
11	7	5	7

NOTES

The approach is to eliminate those elements in all vectors that correspond to -1 in the base vector. Each of the three vectors takes a turn at being the base. It is important that the base be subsetted last. The command **yoo** (see **gutil**(1) in the *UNIX System User Reference Manual*) takes the output of a pipeline and copies it into one of the files used in the pipeline. This cannot be done by redirecting the output of the pipeline as this would cause a concurrent read and write on the same file.

The printing of the "Before" matrix illustrates a useful property of **af**. The first name in an expression that does not match any name in the present working directory is a reference to the standard input. In this example, **label** references the input coming through the pipe.

4.3 Example 3:

PROBLEM

Generate a bar chart of the percent of execution time consumed by each routine in a program.

SOLUTION

```
prof | cut -c1-15 | sed -e 1d -e "/ 0.0/d" -e" s/^ *//" >P
echo These are the execution percentages; cat P
title P -v"execution time in percent " | bar -xa -yl0,
        yh100 | label -br-45,FP | td
```

```
These are the execution percentages
_fork  32.9
_creat  14.3
_sbrk  14.3
_read  14.3
_open  14.3
_prime  9.9
```

NOTES

Prof is a UNIX operating system command that generates a listing of execution times for a program (see **prof**(1)). **Cut** and **sed** are used to eliminate extraneous text from the output of **prof**. (It is because verbiage can get in the way that **stat** nodes say very little.) Notice that **P** is a vector to **title** while it is a text file to **cat** and **label**.

Figure 3-4 shows the output of these commands.

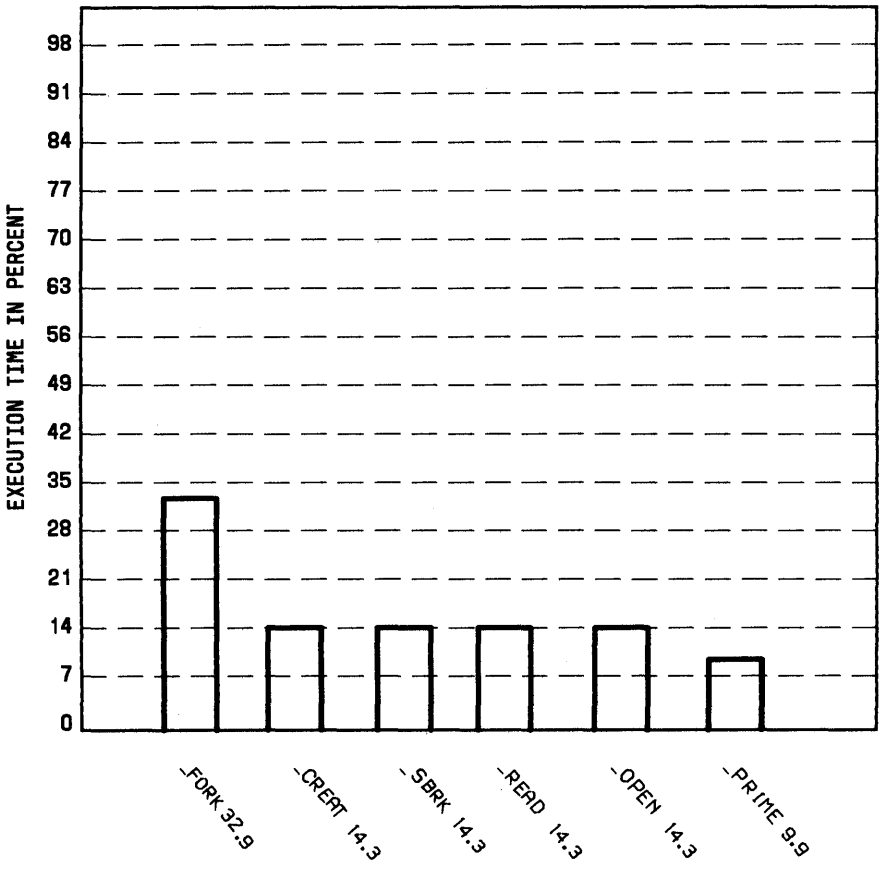

Figure 3-4. Bar Chart Showing Execution Profile

4.4 Example 4:

PROBLEM

Plot the relationship between the execution time of a program and the number of processes in the process table.

SOLUTION

```
# The first program generates the performance data

for i in `gas -n12`
do
        ps -ae | wc -l >>Procs&
        time prime -n1000 >/dev/null 2>>Times
        sleep 300
done

# The second program analyzes and plots the data

for i in real user sys
do
grep $i Times | sed " s/$i//" |
    awk -F: "{ if(NF==2) print \$1*60+\$2; else
        print } " | title -v"$i time in seconds" >$i
    siline -`1reg -o,FProcs $i `Procs >$i.fit
done
title -v "number of processes" Procs | yoo Procs

plot -dg,FProcs real -r12 >R12
plot -ag,FProcs real.fit -r12 >>R12
plot -dg,FProcs sys -r13 >R13
plot -ag,FProcs sys.fit -r13 >>R13
plot -dg,FProcs user -r8 >R8
plot -ag,FProcs user.fit -r8 >>R8
ged R12 R13 R8
```

NOTES

The performance data is the execution time, as reported by the UNIX operating system **time** command, to generate the

first 1000 prime numbers. **Times** outputs three times for each run:

- The time in system routines
- The time in user routines
- Total real time.

The output of the **time** command is saved in the file **Times**. Each of these types of time is treated separately by the analysis program.

In the file **Procs** are the number of processes running on the system during each execution of **prime**. The short **awk** program converts "minutes:seconds" format to "seconds." **Lreg** does a linear regression of the time vectors on the size of the process table. **Siline** generates a line based on the parameters from the regression. One plot is generated for each type of time. Each plot is put into a different region so that they can be displayed and manipulated simultaneously in **ged**.

Figure 3-5 shows the output of these commands.

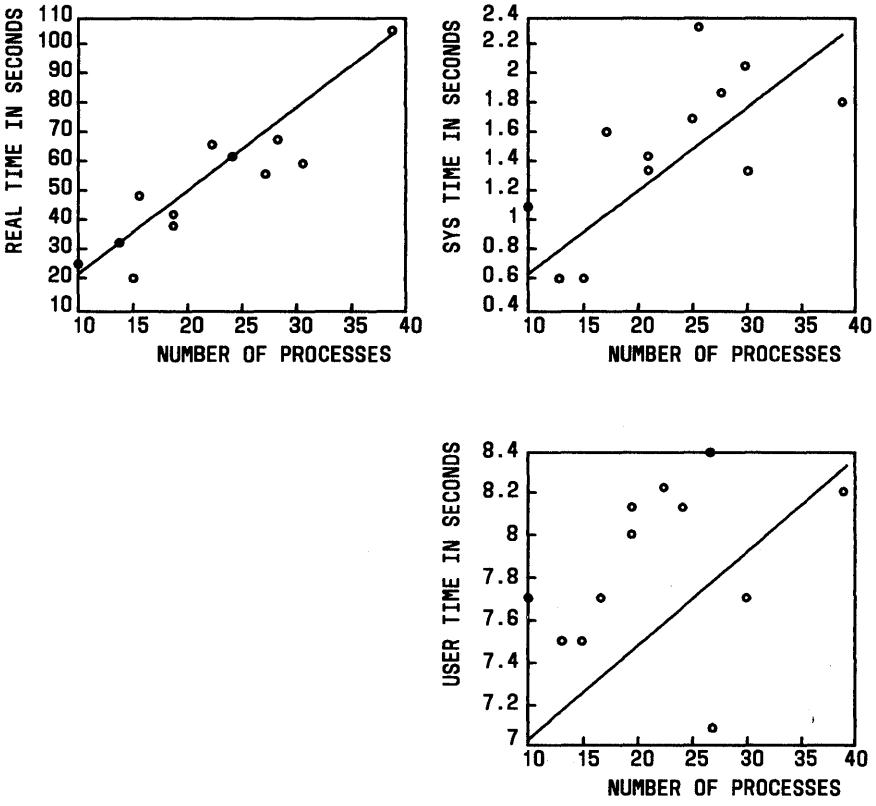

Figure 3-5. Relationship Between Execution Time and Number of Processes

Chapter 4

GRAPHICS EDITOR

PAGE

Chapter 4

GRAPHICS EDITOR

1. Chapter Introduction

The graphics editor, (**ged**), is an interactive graphical editor used to display, edit, and construct drawings on TEKTRONIX 4010 series display terminals. The drawings are represented as a sequence of objects in a token language known as GPS (graphical primitive string). A GPS is produced by the drawing commands in the UNIX System Graphics such as **vtoc** and **plot**, as well as by **ged** itself.

Drawings are built from objects consisting of lines, arcs, and text. Using the editor, the objects can be viewed at various magnifications and from various locations. Objects can be created, deleted, moved, copied, rotated, scaled, and modified.

The examples in this tutorial illustrate how to construct and edit simple drawings. Try them to become familiar with how the editor works, but keep in mind that **ged** is intended primarily to edit the output of other programs rather than to construct drawings from scratch.

As for notation, literal keystrokes are printed in **boldface**. Meta-characters are also in boldface and are surrounded by angled brackets. For example, <**cr**> means return and <**sp**> means space. In the examples, output from the terminal is printed in Roman (normal) type. In-line comments are in Roman and are surrounded by parentheses. Section 2 contains an introduction to the commands understood by the **ged**. A summary of these editor commands and options is given in Section 3 under Command Summary.

2. Commands

To start, it is assumed that while logged in at a display terminal the **graphics** environment (as described in **graphics**(1G) in the *UNIX System User Reference Manual*) has been successfully entered.

> **Note**: In order for **ged** to work properly, the standard strap options need to be set on the terminal. See Section 5.2 under Administrative Information for these standard settings.

To enter **ged** type:

ged<cr>

After a moment the screen should be clear except for the **ged** prompt, *, in the upper left corner. The * indicates that **ged** is ready to accept a command.

Each command passes through a sequence of stages during which you describe what the command is to do. All commands pass through a subset of these stages:

1. Command line
2. Text
3. Points
4. Pivot
5. Destination

As a rule, each stage is terminated by typing **<cr>**. The **<cr>** for the last stage of a command triggers execution.

2.1 Command Line

The simplest commands consist only of a *command line.* The command line is modeled after a conventional command line in the shell. That is

command name [−*option*(s)] [*filename*]<**cr**>

A question mark (**?**) is an example of a simple command. It lists the commands and options understood by **ged**. To generate the list, type

*?<**cr**> (type a question mark followed by a return)

A command is executed by typing the first character of its name. The **ged** will echo the full name and wait for the rest of the command line. For example, **e** references the **erase** command. As **erase** consists only of stage 1, typing <**cr**> causes the erase action to occur. Typing

*<**rubout**>

after a command name and before the final <**cr**> for the command aborts the command. Thus, while

*erase<**cr**>

erases the display screen,

*erase<**rubout**>

brings the editor back to the **ged** prompt, *.

Following the command name, *options* may be entered. Options control such things as the width and style of lines to be drawn or the size and orientation of text. Most options have a default value that applies if a value for the option is not specified on the command line. The **set** command allows examination and modification that the default values. Type

***set<cr>**

to see the current default values.

The option value is one of three types: integer, character, or Boolean. Boolean values are represented by + (for true) and − (for false). A default value is modified by providing it as an option to the **set** command. For example, to change the default text height to 300 units, type

***set −h300<cr>**

Arguments on the command line, but not the command name, may be edited using the erase and kill characters from the shell. This applies whenever text is being entered.

2.2 Constructing Graphical Objects

Drawings are stored as a GPS in a display buffer internal to the editor. Typically, a drawing in **ged** is composed of instances of three graphical primitives: *arcs, lines,* and *text.*

2.3 Generating Text

To put a line of text on the display screen use the **T**ext command.

First enter the *command line* (stage 1):

***Text<cr>**

Next enter the text (stage 2):

a line of text<cr>

And then enter the starting *point* for the text (stage 3).

<position cursor><cr>

Positioning of the graphic cursor is done either with the thumbwheel knobs on the terminal keyboard or with an auxiliary joystick. The **<cr>** establishes the location of the cursor to be the starting point for the text string. The **Text** command ends at stage 3, so this **<cr>** initiates the drawing of the text string.

The **Text** command accepts options to vary the angle, height, and line width of the characters, and to either center or right justify the text object. The text string may span more than one line by escaping the **<cr>** (i.e., \<**cr**>) to indicate continuation. To illustrate some of these capabilities, try the following:

```
*Text -r<cr>              (right justify text)
top\<cr>
right<cr>
<position cursor><cr>
*Text -a90<cr>            (rotate text 90 degrees)
lower\<cr>
left<cr>
<position cursor><cr>     (pick a point below and left of
                           the previous point)
```

Results of these commands are shown in Figure 4-1.

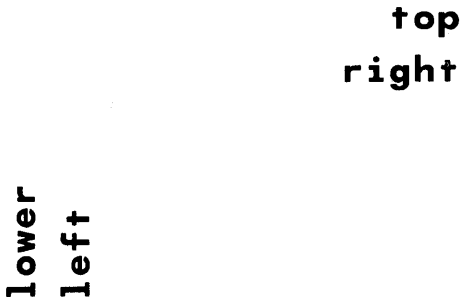

Figure 4-1. Generating Text Objects

2.4 Drawing Lines

The **Lines** command is used to construct objects built from a sequence of straight lines. It consists of stages 1 and 3. Stage 1 is straightforward:

*Lines *options*<**cr**>

The **Lines** command accepts options to specify line style and line width.

Stage 3, the entering of *points*, is more interesting. *Points* are referenced either with the graphic cursor or by name. We have already entered a point with the cursor for the **Text** command. For the **Lines** command it is more of the same. As an example, to build a triangle, type

*Lines<**cr**>
<**position cursor**><**sp**> (locate the first point)
<**position cursor**><**sp**> (the second point)
<**position cursor**><**sp**> (the third point)
<**position cursor**><**sp**> (back to the first point)
<**cr**> (terminate points, draw triangle)

Results of these commands are shown in Figure 4-2.

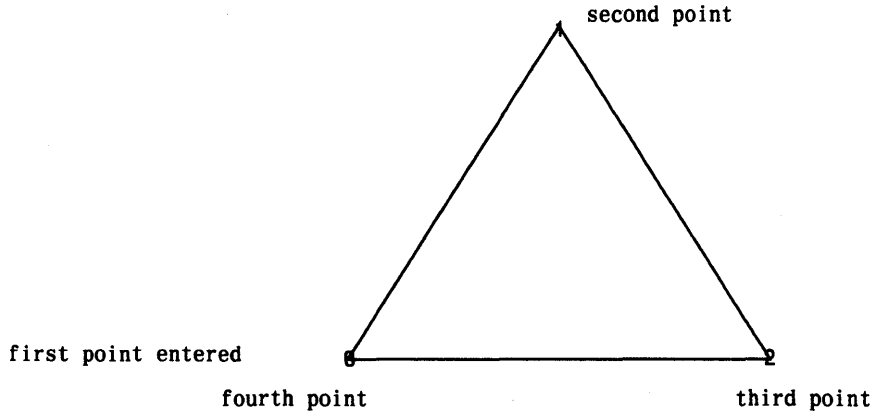

Figure 4-2. Building a Triangle

Typing <**sp**> enters the location of the crosshairs as a point. **Ged** identifies the point with an integer and adds the location to the current *point set.* The last point entered can be erased by typing #. The current point set can be cleared by typing @. On receiving the final <**cr**> the points are connected in numerical order.

2.5 Accessing Points by Name

The points in the current point set may be referenced by name using the $ operator. For instance, $*n* references the point numbered *n*. By using $ the triangle above can be redrawn by entering:

```
*Lines<cr>
<position cursor><sp>
<position cursor><sp>
<position cursor><sp>
$0<cr>                  (reference point 0)
<cr>
```

At the start of each command that includes stage 3, *points*, the current point set is empty. The point set from the previous command is saved and is accessible using the . operator. The . swaps the points in the previous point set with those in the current set. The = operator can be used to identify the current points. To illustrate, use the triangle just entered as the basis for drawing a quadrilateral.

```
*Lines<cr>
.       (access the previous set)
=       (identify the current points)
#       (erase the last point)
<position cursor><sp>       (add a new point)
$0<cr>          (close the figure)
<cr>
```

Results of these commands are shown in Figure 4-3.

Individual points from the previous point set can be referenced by using the . operator with $. The following example builds a triangle that shares an edge with the quadrilateral.

***Lines<cr>**
$.1<cr> (reference point 1 from the previous point set)
$.2<cr> (reference point 2)
<position cursor><sp> (enter a new point)
$0<cr>** (or **$.1, to close the figure)
<cr>

Results of these points are shown in Figure 4-4.

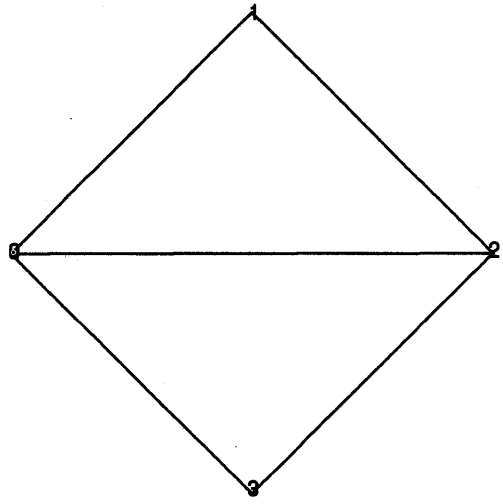

Figure 4-3. Accessing the Previous Point Set

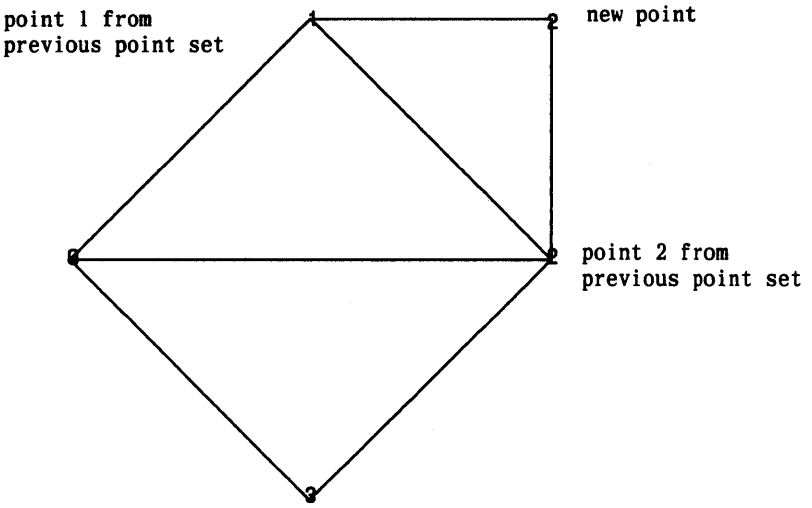

Figure 4-4. Referencing Points from Previous Point Set

A point can also be given a name. The > operator permits an upper case letter to be associated with a point just entered. A simple example is:

```
*Lines<cr>
<position cursor><sp>      (enter a point)
>A<cr>                     (name the point A)
<position cursor><sp>
<cr>
```

In commands that follow, point **A** can be referenced using the $ operator, as in:

```
*Lines<cr>
$A<cr>
<position cursor><sp>
<cr>
```

2.6 Drawing Curves

Curves are interpolated from a sequence of three or more points. The Arc command generates a circular arc given three points on a circle. The arc is drawn starting at the first point, through the second point, and ending at the third point. A circle is an arc with the first and third points coincident. One way to draw a circle is thus:

```
*Arc<cr>
<position cursor><sp>
<position cursor><sp>
$0<cr>
<cr>
```

2.7 Editing Objects

2.7.1 Addressing Objects

An object is addressed by pointing to one of its *handles*. All objects have an *object-handle*. Usually the object-handle is the first point

entered when the object was created. The **o**bjects command marks the location of each object-handle with an **O**. For example, to see the handles of all the objects on the screen, type

*objects −**v**<**cr**>

Some objects, **L**ines for example, also have *point-handles.* Typically each of the points entered when an object is constructed becomes a point-handle. (An object-handle is also a point-handle.) The **p**oints command marks each of the point-handles.

A handle is pointed to by including it within a *defined-area.* A defined-area is generated either with a command line option or interactively using the graphic cursor. As an example, to delete one of the objects that was created on the screen, type

*Delete<**cr**>
<**position cursor**><**sp**> (above and to the left of some object-handle)
<**position cursor**><**sp**> (below and to the right of the object-handle)
<**cr**> (the defined-area should include the object-handle)
<**cr**> (if all is well, delete the object)

The defined-area is outlined with dotted lines. The reason for the seemingly extra <**cr**> at the end of the **D**elete command is to provide an opportunity to stop the command (using <**rubout**>) if the defined-area is not quite right. Every command that accepts a defined-area will wait for a confirming <**cr**>. The **n**ew command can be used to get a fresh copy of the remaining objects.

Defined-areas are entered as **p**oints in the same way that objects are created. Actually, a defined-area may be generated by giving anywhere from 0 to 30 points. Inputting zero points is particularly useful to point to a single handle. It creates a small defined-area about the location of the terminating <**cr**>. Using a zero point

defined-area, the **D**elete command would be

***D**elete<**cr**>
<**position cursor**> (center crosshairs on the object-handle)
<**cr**> (terminate the defined-area)
<**cr**> (delete the object)

A defined-area can also be given as a command line option. For example, to delete everything in the display buffer give the **u**niverse option (**u**) to the **D**elete command. Note the difference between the commands **D**elete –**u**niverse and **e**rase.

2.7.2 Changing the Location of an Object

Objects are moved using the **M**ove command. Create a circle using **A**rc, then move it as follows:

***M**ove<**cr**>
<**position cursor**><**cr**> (centered on the object-handle)
<**cr**> (this establishes a *pivot,* marked with an asterisk)
<**position cursor**><**cr**> (this establishes a *destination*)

The basic move operation relocates every point in each object within the defined area by the distance from the *pivot* to the *destination.* In this case, the pivot was chosen to be the object-handle, so effectively the object-handle was moved to the destination point.

2.7.3 Changing the Shape of an Object

The **B**ox command is a special case of generating lines. Given two points, it creates a rectangle such that the two points are at opposite corners. The sides of the rectangle lie parallel to the edges of the screen. To draw a box, type

***B**ox<**cr**>
<**position cursor**><**sp**>
<**position cursor**><**cr**>

The **Box** command generates point-handles at each vertex of the rectangle. Use the **points** command to mark the point-handles. The shape of an object can be altered by moving point-handles. The next example illustrates one way to double the height of a box (shown in Figure 4-5).

***Move −p+<cr>**	
<position cursor><sp>	(left of the box, between the top and bottom edges)
<position cursor><cr>	(right of the box, below the bottom edge)
<position cursor><cr>	(on the top edge)
<position cursor><cr>	(directly below on the bottom edge)

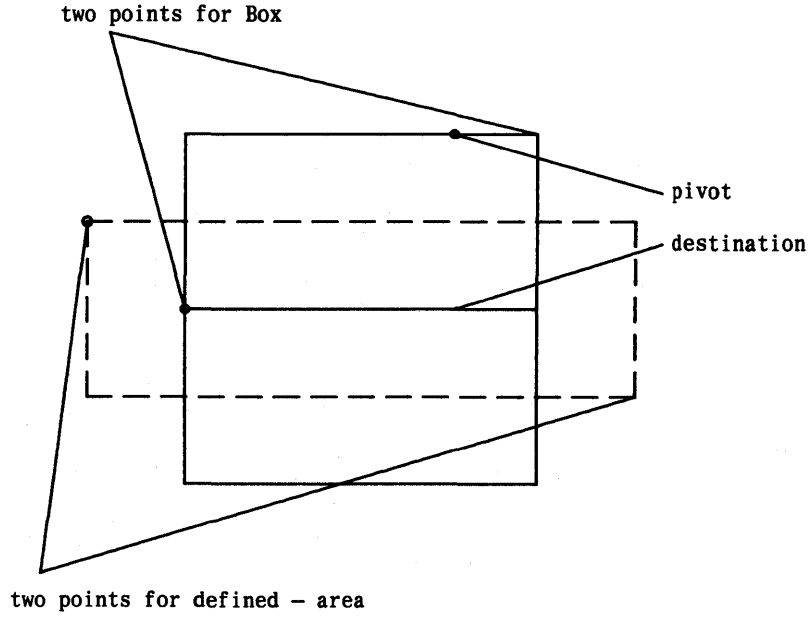

Figure 4-5. Growing a Box

When the **points** flag (**p**) is true, operations are applied to each point-handle addressed. In this example, the **points** flag was set to true using the command-line option **−p+** causing each point-handle within the defined-area to be moved the distance from the pivot to the destination. If **p** was false, only the object-handle would have been addressed.

2.7.4 Changing the Size of an Object

The size of an object can be changed using the **Scale** command. The **Scale** command scales objects by changing the distance from each handle of the object to the pivot by a factor. Put a line of text on the screen and try the following **Scale** commands (Figure 4-6).

```
*Scale −f200<cr>                (factor is in percent)
<position cursor><cr>           (point to object-handle)
<position cursor><cr>           (set pivot to rightmost character)
<cr>

*Scale −f50<cr>
.<cr>  (reference the previous defined-area)
<position cursor><cr>           (set pivot above a character
                                 near the middle)
<cr>
```

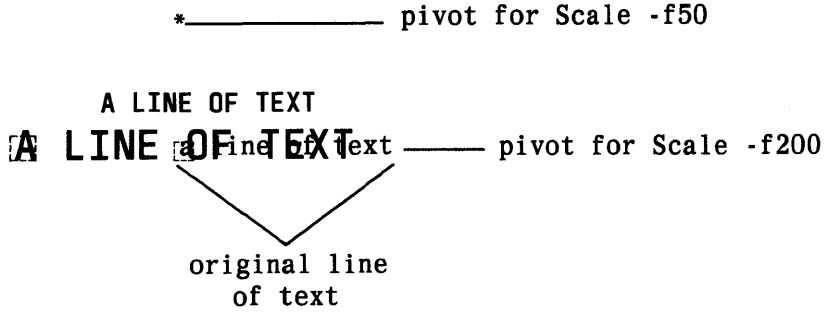

Figure 4-6. Scaling Text

A useful insight into the behavior of scaling is to note that the position of the pivot does not change. Also observe that the defined-area is scaled to preserve its relationship to the graphical objects.

The size of objects can also be changed by moving point-handles. Generate a circle, this time using the **C**ircle command:

```
*Circle<cr>
<position cursor><sp>      (specify the center)
<position cursor><cr>      (specify a point on the circle)
```

The **C**ircle command generates an arc with the first and third point at the point specified on the circle. The second point of the arc is located 180 degrees around the circle. One way to change the size of the circle is to move one of the point-handles (using **M**ove **−p+**).

The size of text characters can be changed via a third mechanism. Character height is a property of a line of text. The **E**dit command allows changing character height as follows:

```
*Edit −hheight<cr>   (height is in universe units,
                      see Section 2.8 View Command)
<position cursor><cr>      (point to the object-handle)
<cr>
```

2.7.5 Changing the Orientation of an Object

The orientation of an object can be altered using the **R**otate command. The **R**otate command rotates each point of an object about a pivot by an angle. Try the following rotations on a line of text (Figure 4-7).

*R*otate −**a90<cr>** (angle is in degrees)
<position cursor><cr> (point to object-handle)
<position cursor><cr> (set pivot to rightmost character)
<cr>

*Rotate −**a−90<cr>**
.**<cr>** (reference previous defined-area)
<position cursor><cr> (set pivot to a character near the middle)
<cr>

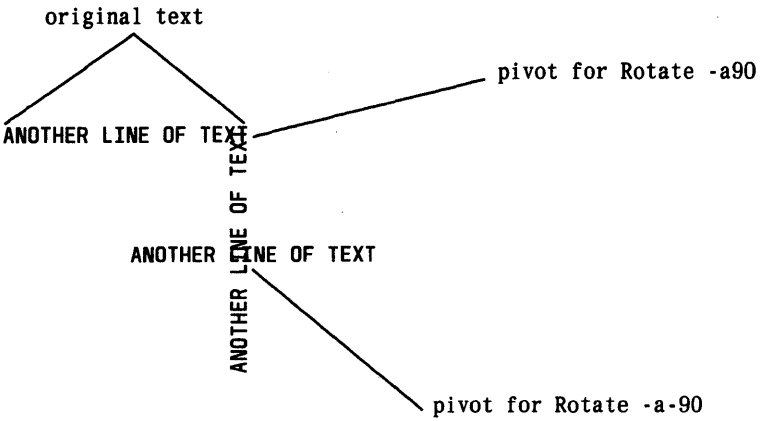

Figure 4-7. Rotating Text

2.7.6 Changing the Style and Width of Lines

In the current editor, objects can be drawn from lines in any of five styles: solid (**so**), dashed (**da**), dot-dashed (**dd**), long-dashed (**ld**) and three widths narrow (**n**), medium (**m**), and bold (**b**). Style is controlled by the **s** option and width by **w**. The next example creates a narrow dotted line:

*Lines −**wn,sdo<cr>**
<position cursor><sp>
<position cursor><sp>
<cr>

Using the **E**dit command, the line can be changed to bold dot-dashed:

```
*Edit −wb,sdd<cr>
$.0<cr>          (reference the object-handle of the previous line)
<cr>             (complete the defined-area)
<cr>
```

2.8 View Commands

All of the objects drawn lie within a Cartesian plane, 65,534 units on each axis, known as the *universe*. Thus far, only a small portion of the universe has been displayed on the screen. The command

```
*view −u<cr>
```

displays the entire universe.

2.8.1 Windowing

A mapping of a portion of the universe onto the display screen is called a *window*. The extent or magnification of a window is altered using the **z**oom command. To build a window that includes all of the objects drawn, type

```
*zoom<cr>
<position cursor><sp>      (above and to the left of all
                            the object)
<position cursor><cr>      (below and to the right, also
                            end points)
<cr>  (verify)
```

Zooming can be either *in* or *out*. Zooming in, as with a camera lens, increases the magnification of the window. The area outlined by *points* is expanded to fill the screen. Zooming out decreases magnification. The current window is shrunk so that it fits within the defined-area. The direction of the zoom is controlled by the sense of the **o**ut flag; **o** true means zoom out.

The location of a window is altered using the **v**iew command. **V**iew moves the window so that a given point in the universe lies at a given

location on the screen.

```
*view<cr>
<position cursor><cr>      (locate a point in the universe)
<position cursor><cr>      (locate a point on the screen)
```

View also provides access to several predefined windows. As seen earlier, view **−u** displays the entire universe. The view **−h** command displays the *home-window.* The home-window is the window that circumscribes all of the objects in the universe. The result is similar to that of the example using **zoom** given earlier.

Lastly, the **view** command permits selection of a window on a particular *region.* The universe is partitioned into 25 equal-sized regions. Regions are numbered from 1 to 25 beginning at the lower left and proceeding toward the upper right. Region 13, the center of the universe, is used as the default region by drawing commands such as **plot**(1) and **vtoc**(1).

2.9 Other Commands

2.9.1 Interacting with Files

The **write** command saves the contents of the display buffer by copying it to a file:

*write *filename*<cr>

The contents of *filename* will be a GPS, thus it can be displayed using any of the device filters (e.g., **td** (1)) or read back into **ged**.

A GPS is read into the editor using the **read** command:

*read *filename*<cr>

The GPS from *filename* is appended to the display buffer and then displayed. Because **read** does not change the current window, only some (or none) of the objects read may be visible. A useful command

sequence to view everything read is

***read** −**e**− *filename*<**cr**>
***view** −**h**<**cr**>

The display function of **read** is inhibited by setting the **echo** flag to false; **view** −**h** windows on and displays the full display buffer.

The **read** command may also be used to input text files. The form is

read [−*option*(s)] *filename*<**cr**>

followed by a single point to locate the first line of text. A text object is created for each line of text from *filename*. Options to the **read** command are the same as those for the **Text** command.

2.9.2 Leaving the Editor

The **quit** command is used to terminate an editing session. As with the text editor **ed**, **quit** responds with **?** if the internal buffer has been modified since the last **write** command. A second **quit** command forces exit.

2.10 Other Useful Information

2.10.1 One-Line UNIX System Escape

As in **ed**, the **!** provides a temporary escape to the shell.

2.10.2 Typing Ahead

Most programs under the UNIX operating system allow input to be typed before the program is ready to receive it. In general, this is not the case with **ged**; characters typed before the appropriate prompt are lost.

2.10.3 Speeding up Things

Displaying the contents of the display buffer can be time consuming, particularly if much text is involved. The wise use of two flags to control what gets displayed can make life more pleasant:

- The **echo** flag controls echoing of new additions to the display buffer.
- The **text** flag controls whether text will be outlined or drawn.

3. Command Summary

In the summary, characters actually typed are printed in boldface. Command stages are printed in italics. Arguments surrounded by brackets (e.g., [...]) are optional. Parentheses surrounding arguments separated by "or" means that exactly one of the arguments must be given.

For example, the **D**elete command accepts the arguments −**u**niverse, −**v**iew, and *points*.

3.1 Construct Commands

Arc	[−**e**cho,**s**tyle,**w**idth] *points*
Box	[−**e**cho,**s**tyle,**w**idth] *points*
Circle	[−**e**cho,**s**tyle,**w**idth] *points*
Hardware	[−**e**cho] *text points*
Lines	[−**e**cho,**s**tyle,**w**idth] *points*
Text	[−**a**ngle,**e**cho,**h**eight,**m**idpoint,**r**ightpoint, **t**ext,**w**idth] *text points*

3.2 Edit Commands

Delete	(− (**u**niverse or **v**iew) or *points*)
Edit	[−**a**ngle,**e**cho,**h**eight,**s**tyle,**w**idth] (− (**u**niverse or **v**iew) or *points*)
Kopy	[−**e**cho,**p**oints,**x**] *points pivot destination*
Move	[−**e**cho,**p**oints,**x**] *points pivot destination*
Rotate	[−**a**ngle,**e**cho,**k**opy,**x**] *points pivot destination*
Scale	[−**e**cho,**f**actor,**k**opy,**x**] *points pivot destination*

3.3 View Commands

coordinates	*points*
erase	
new	
objects	(− (**u**niverse or **v**iew) or *points*)
points	(− (labelled-**p**oints or **u**niverse or **v**iew) or *points*)
view	(− (**h**ome or **u**niverse or **r**egion) or [−**x**] *pivot destination*)
x	[−**v**iew] *points*
zoom	[−**o**ut] *points*

3.4 Other Commands

quit	
read	[−**a**ngle,**e**cho,**h**eight,**m**idpoint,**r**ightpoint,**t**ext, **w**idth] *filename* [*destination*]

set	[−**a**ngle,**e**cho,**f**actor,**h**eight,**k**opy,**m**idpoint, **p**oints,**r**ightpoint,**s**tyle,**t**ext,**w**idth,**x**]
write	*filename*
!*command*	
?	

3.5 Options

Options specify parameters used to construct, edit, and view graphical objects. If a parameter used by a command is not specified as an *option*, the default value for the parameter will be used. The format of command *options* is

−*option* [,*option*]

where *option* is *keyletter*[*value*]. Flags take on the values of true or false indicated by + and −, respectively. If no value is given with a flag, true is assumed. Object options are

angle*n*	Specify an angle of *n* degrees.
echo	When true, changes to the display buffer will be echoed on the screen.
factor*n*	Specify a scale factor of *n* percent.
height*n*	Specify height of text to be *n* universe-units (*n* greater than or equal to 0 and less than 1280).
kopy	The commands **S**cale and **R**otate can be used to either create new objects or to alter old ones. When the **k**opy flag is true, new objects are created.
midpoint	When true, use the midpoint of a text string to locate the string.

out When true, reduce magnification during zoom.

points When true, operate on points; otherwise operate on objects.

rightpoint When true, use the rightmost point of a text string to locate the string.

style*type* Specify line style to be one of the following *types*:
so solid
da dashed
dd dot-dashed
do dotted
ld long-dashed

text Most text is drawn as a sequence of lines. This can sometimes be painfully slow. When the **text** flag (**t**) is false, strings are outlined rather than drawn.

width*type* Specify line width to be one of the following *types*:
n narrow
m medium
b bold

x One way to find the center of a rectangular area is to draw the diagonals of the rectangle. When the **x** flag is true, defined-areas are drawn with their diagonals.

Area options are:

home Reference the home-window.

region*n* Reference region *n*.

universe Reference the universe-window.

view Reference those objects currently in view.

4. Some Examples of What Can Be Done

The following examples are used to illustrate use of the **ged**.

4.1 Example 1--Text Centered Within a Circle

```
*Circle<cr>
<position cursor><sp>      (establish center)
<position cursor><cr>      (establish radius)
*Text -m<cr>               (text is to be centered)
some text<cr>
$.0<cr>                    (first point from previous set,
                           i.e., circle center)
<cr>
```

Figure 4-8 shows the output of these commands.

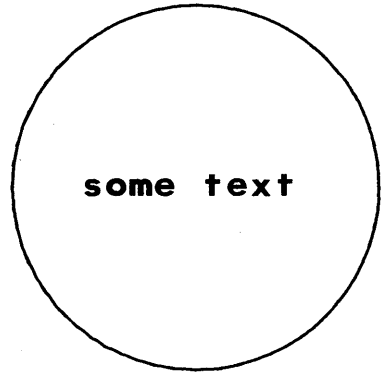

Figure 4-8. Text Centered Within a Circle

4.2 Example 2--Making Notes on a Plot

***! gas | plot −g >A<cr>** (generate a plot, put it in file A)
***read −e− A<cr>** (input the plot, but do not display it)
***view −h<cr>** (window on the plot)
***Lines −sdo<cr>** (draw dotted lines)
<position cursor><sp> (0,6.5 y-axis)
<position cursor><sp> (6.5,5.5)
<position cursor><sp> (5.5,0 x-axis)
<cr> (end of Lines)
***set −h150,wn<cr>** (set text height to 150, line width to narrow)
***Text −r<cr>** (right justify text)
threshold beyond which nothing matters<cr>
<position cursor><cr> (set right point of text)
***Text −a−90<cr>** (rotate text negative 90 degrees)
threshold beyond which nothing matters<cr>
<position cursor><cr> (set top end of text)
***x<cr>** (find center of plot)
<position cursor><sp> (top left corner of plot)
<position cursor><cr> (bottom right corner of plot)
***Text −h300,wm,m<cr>** (build title: height 300, weight medium, centered)
SOME KIND OF PLOT<cr>
<position cursor><cr> (set title centered above plot)
***view −h<cr>** (window on the resultant drawing)

Figure 4-9 shows the output of these commands.

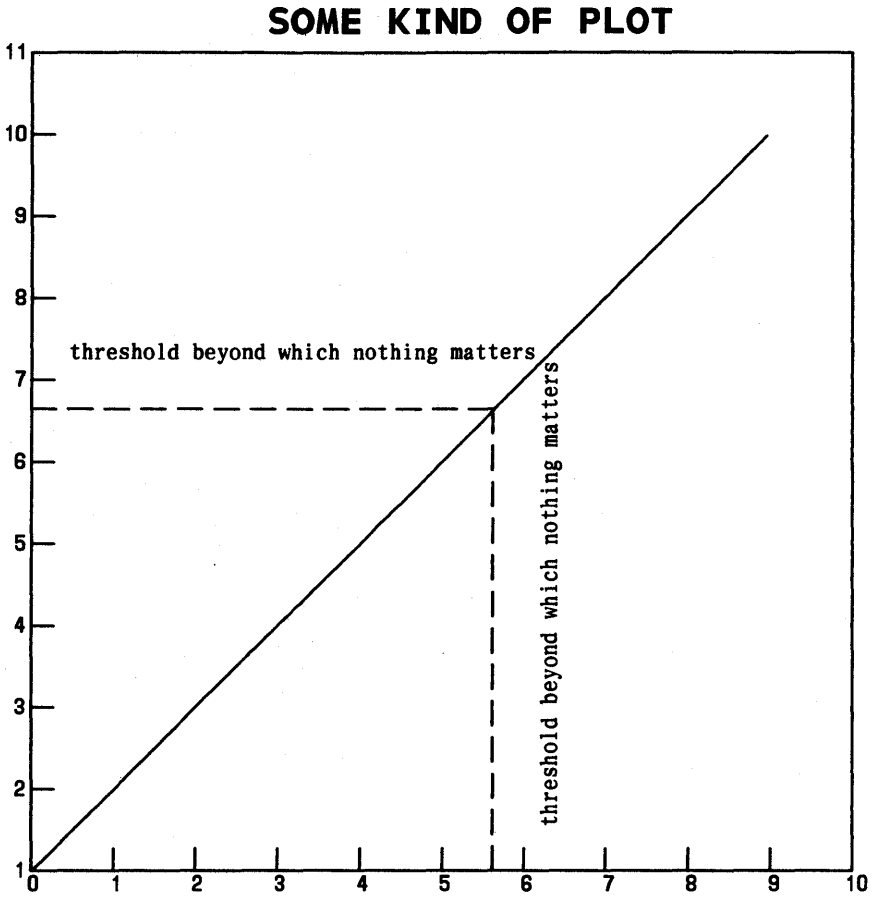

Figure 4-9. Making Notes on a Plot

4.3 Example 3--A Page Layout with Drawings and Text

```
*! rand -s1,n100 | title -v" seed 1" | qsort | bucket |
        hist -r12 >A<cr>    (put a histogram, region
                12, of 100 random numbers in file A)
*! rand -s2,n100 | title -v" seed 2" | qsort | bucket |
        hist -r13 >B<cr>    (put another histogram,
                            region 13, into file B)
*! ed<cr>        (create a file of text using the text editor)
a<cr>
On this page are two histograms<cr>
from a series of 40<cr>
designed to illustrate the weakness<cr>
of multiplicative congruential random number
generators.<cr>
.pl 3<cr>       (mark end of page)
.<cr>
w C<cr>         (put the text into file C)
151
q<cr>
*! nroff C | yoo C<cr>          (format C, leave the output
                                in C)
*view -u<cr> (window on the universe)
*read -e- A<cr
*read -e- B<cr>
*view -h<cr> (view the two histograms)
*read -h300,wn,m C<cr>    (text height 300, line weight
                          narrow, text centered)
<position cursor><cr>     (center text over two plots)
*view -h<cr> (window on the resultant drawing)
```

Figure 4-10 shows the output of these commands.

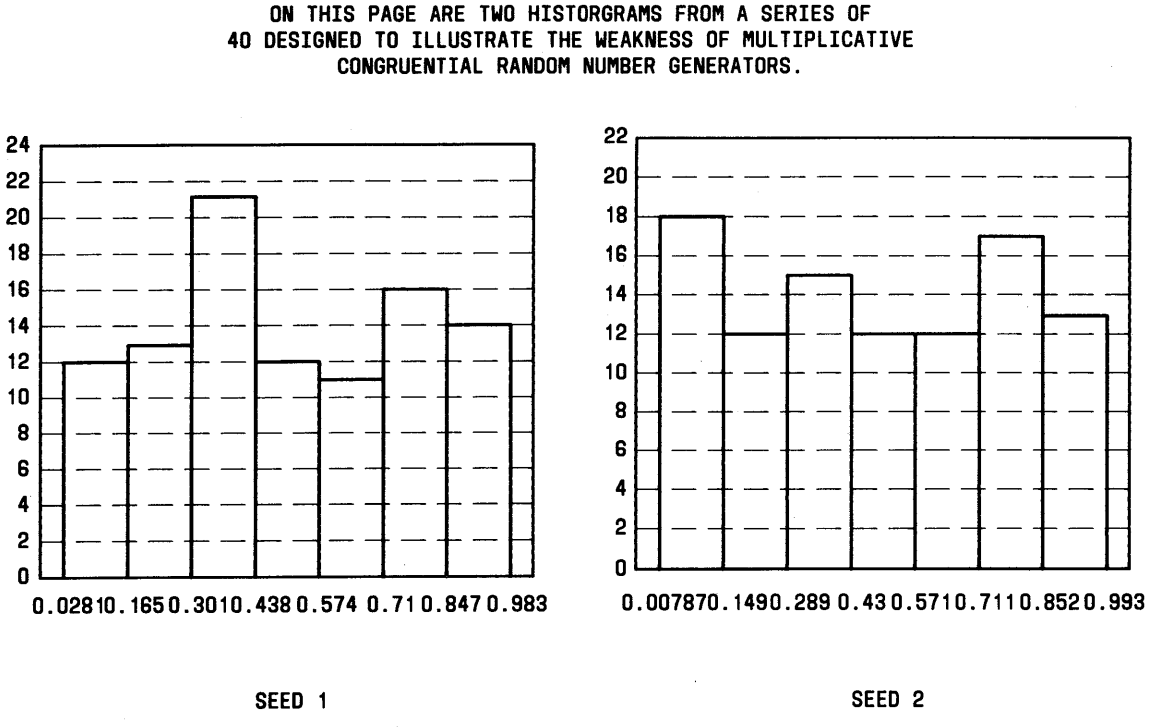

Figure 4-10. Page Layout with Drawings and Text

Chapter 5

ADMINISTRATIVE INFORMATION

PAGE

Chapter 5

ADMINISTRATIVE INFORMATION

1. Chapter Introduction

This chapter is a reference guide for system administrators using or establishing a graphics facility on a UNIX operating system. It contains information about directory structure, installation, makefiles, hardware requirements, and miscellaneous facilities of the graphics package.

2. Graphics Structure

Figure 5-1 contains a graphical representation of the directory structure of graphics. In this part, the shell variable *SRC* will represent the parent node for graphics source and is usually set **/usr/src/cmd**.

The **graphics** command (see **graphics**(1G)) resides in **/usr/bin**. All other graphics executables are located in **/usr/bin/graf**; the **/usr/lib/graf** directory contains text for **whatis** documentation (see **gutil**(1G)) and editor scripts for **ttoc** (see **toc**(1G)).

Graphics source resides below the directory **$SRC/graf**; **$SRC/graf** is broken into the following subdirectories:

- **include** - contains the following header files: **debug.h**, **errpr.h**, **gsl.h**, **gpl.h**, **setopt.h**, and **util.h**.
- **src** - contains source code partitioned into subdirectories by subsystem. Each subdirectory contains its own *Makefile* (or *Install* file for **whatis.d**).
    - **glib.d** - contains source used to build the graphical subroutine library, **$SRC/graf/lib/glib.a**.
    - **stat.d** - contains source for numerical manipulation and plotting routines.

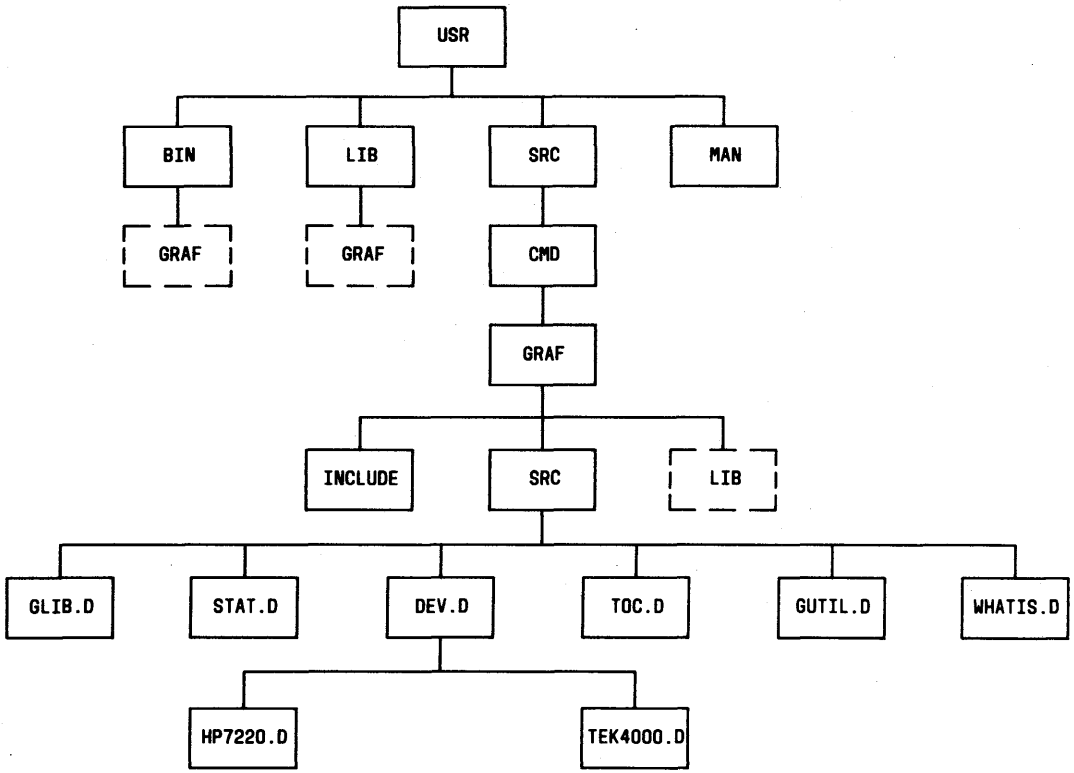

Figure 5-1. Directory Structure of Graphics Program

- **dev.d** - contains source code for device filters partitioned into subdirectories.
    - **lolib** and **uplib** - contains source used to create device independent libraries.
    - **hp7220.d** - contains source for **hpd** (a Hewlett-Packard Plotter display function).
    - **tek4000.d** - contains source for **ged** (the graphical editor), **td** (a TEKTRONIX display function), and other TEKTRONIX dependent routines.
- **gutil.d** - contains source for graphical utility programs.
- **toc.d** - contains source for table of contents drawing routines.
- **whatis.d** - contains **nroff** files and the installation routine for on-line documentation.

- **lib** - contains **glib.a** which contains commonly used graphical subroutines.

The *UNIX System User Reference Manual* entries for **graphics** consist of the following: **gdev**(1G), **ged**(1G), **graphics**(1G), **gutil**(1G), **stat**(1G), **toc**(1G), and **gps**(5).

3. Installing Graphics

Procedures for installing **graphics**:

— To build the entire **graphics** package, execute (as super-user):

/usr/src/:mkcmd graf

— To build a particular **graphics** subsystem, use the shell variable *ARGS*:

ARGS=subsystem **/usr/src/:mkcmd graf**

A *subsystem* is either **glib, stat, dev, toc, gutil,** or **whatis. Glib** must exist before other subsystems can be built. Write permission in **/usr/bin** and **/usr/lib** is needed, and the following libraries are assumed to exist:

- /lib/libc.s - Standard C library, used by all subsystems.
- /lib/libm.a - Math library, used by all subsystems.
- /usr/lib/macros/mm[nt] - Memorandum macros for [**nt**]*roff*, used by the **whatis** subsystem.

The complete build process takes approximately two hours of system time. If the build must be stopped, it is a good idea to restart from the beginning. Upon completion, the following things will be created and owned by **bin**:

- /usr/lib/graf - A directory for data and editor scripts.
- /usr/bin/graf - A directory for executables.
- /usr/bin/graphics - Command entry point for graphics.

Whatis.d contains source files for **whatis** and the executable command **Install.**

Install command name

calls **nroff** to produce **whatis** documentation for *command name* in **/usr/lib/graf**. To install the entire **whatis** subsystem, use **:mkcmd** as described above.

3.1 Makefile Parameters

*Makefile*s use executable shell procedures **cco** and **cca**. **Cco** is used to compile C source and install load modules in **/usr/bin/graf**. The **cca** command compiles C programs and loads object code into archive files.

*Makefile*s use various macro parameters, some of which can be specified on the command line to redirect outputs or inputs. Parameters specified in higher level *Makefile*s are passed to lower levels. Below is a list of specifiable parameters for *Makefile*s followed by their default values in parentheses and an explanation of their usage:

$SRC/graf/graf.mk:

- BIN (/usr/bin) - installation directory for the **graphics** command.
- BIN (/usr/bin/graf) - installation directory for other **graphic** commands.
- SRC (/usr/src/cmd) - parent directory for source code.

$SRC/graf/src/Makefile:

- BIN1 (/usr/bin) - installation directory for the **graphics** command.
- BIN2 (/usr/bin/graf) - installation directory for other **graphic** commands.
- LIB (/usr/lib/graf) - installation directory for **whatis** documentation.

$SRC/graf/src/stat.d/Makefile:

- BIN (../../bin) - installation directory for executable commands.

$SRC/graf/src/toc.d/Makefile:

- BIN (../../bin) - installation directory for executable commands.

$SRC/graf/src/dev.d/Makefile:

- BIN (../../bin) - installation directory for executable commands.

$SRC/graf/src/dev.d/hp7220.d/Makefile:

- BIN (../../../bin) - installation directory for executable commands.

$SRC/graf/src/dev.d/tek4000.d/Makefile:

- BIN (../../../bin) - installation directory for executable commands.

$SRC/graf/src/gutil.d/Makefile:

- BIN (../../bin) - installation directory for executable commands.

The following example will make a new version of the **ged**, installing it in **/a1/pmt/dp/bin** (assuming that necessary libraries were previously built):

```
cd $SRC/graf/src/dev.d/tek4000.d
make BIN=/a1/pmt/dp/bin ged
```

4. Hewlett-Packard Plotter

The **graphics** display function, **hpd**, uses the Hewlett-Packard (HP*) 7221A Graphics Plotter. The HP plotter can be connected to

* Registered trademark of Hewlett-Packard Company.

the computer in series with a terminal via a dedicated or dial-up line. This arrangement allows the plotter to intercept plotting instructions while passing other data to the terminal unaltered and thus providing for normal terminal operation. Plotter switch settings should match those of the terminal. The plotter operating manual contains a more complete discussion.

5. TEKTRONIX Terminal

The graphics display function, **td**, and the **ged**, both use TEKTRONIX Series 4010 storage tubes. Below is a list of device considerations necessary for **graphics** operation.

5.1 Inittab Entry

When a TEKTRONIX 4010 series terminal is connected to a UNIX operating system via a dedicated 4800 or 9600 baud line, **/etc/inittab** should reference speed table entry **6** (the table may vary locally) of **getty**. Speed table entry **6** is designed specifically for the TEKTRONIX 4014 and, among other things, sets a form-feed delay so that the screen may be cleared without losing information and clears the screen before prompting for a login. See **stty**(1), **inittab**(5) and **getty**(8) for more information.

5.2 Strap Options

The standard strap options as listed below should be used (see the Reference Manual for the TEKTRONIX 4014):

- LF effect - LF causes line-feed only.
- CR effect - CR causes carriage return only.
- DEL implies loy - DEL key is interpreted as low-order y value.

6. Miscellaneous Information

6.1 Announcements

The **graphics** command provides a means of printing out announcements to users. To set up an announcement facility, create a readable text file containing the announcements named *announce* in directory **/usr/adm**. If it is desired to use a different directory for announcements, redefined the shell variable *GRAF* in **/usr/bin/graphics**.

6.2 Graflog

The **graphics** command also provides a means of monitoring its use by listing users in a file.

Each time a user executes **graphics**, an entry of the login name, terminal number, and system date are recorded in file *graflog* in directory **/usr/adm**, the same directory used for announcements.

6.3 Restricted Environments

Restricted environments can be used to limit access to the system (see **sh**(1)). A restricted environment for **graphics** can be set up by creating the directories **/rbin** and **/usr/rbin** and populating them with restricted versions of regular UNIX system commands, so that the environment cannot be compromised. In particular, **ed**(1), **mv**(1), **rm**(1), and **sh**(1) require restricted interface programs which do not allow users to move or remove files whose names begin with ".".

To create a restricted environment for **graphics**:

- Create a restricted **ged** command in **/usr/rbin** as follows:

  exec /usr/bin/graf/ged −R

- Create restricted logins for users or create a community login with a working directory (reached through **.profile**) set up for each user. A restricted login specifies **/bin/rsh** as the terminal interface program and is created by adding **/bin/rsh** to the end of the **/etc/passwd** file entry for that login.

- Call **graphics −r** from **.profile**.

The execution of **graphics −r** changes $*PATH* to look for commands in **/rbin** and **/usr/rbin** before **/bin** and **/usr/bin** and executes a restricted shell. The **−R** option is appended to the **ged** command so that the escape from **ged** to the UNIX operating system (!*command*) will also use a restricted shell.

NOTES

READER COMMENT FORM

UNIX SYSTEM V GRAPHICS GUIDE

Altos Computer Systems
2641 Orchard Parkway
San Jose, CA 95134

This document has been prepared for use with your Altos Computer System. Should you find an error or problem in this manual, please write it down (noting page number), and return this form to the ALTOS PUBLICATIONS DEPARTMENT.

System Model Number_____

Serial Number_____

Document Title_____

Revision Number 690-15842-001 Date_____

Name_____

Company Name_____

Address_____

Printed in U.S.A.
P/N 690-15842-001

2641 Orchard Park Way, San Jose, California 95134
(408) 946-6700, Telex 470642 Alto UI

July 1985