A Tutorial Introduction to the Graphics Editor

A. R. Feuer

Bell Laboratories
Murray Hill, New Jersey 07974

1. INTRODUCTION

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 (for graphical primitive string). GPS is produced by the drawing commands in UNIX† Graphics [1] such as *vtoc* and *plot*, as well as by *ged* itself.

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. A summary of editor commands and options is given in Section 3.

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.

2. COMMANDS

To start we will assume that you have successfully entered the graphics environment (as described in *graphics*(1G) of [2]) while logged in at a display terminal. To enter *ged* type:

ged <cr>

After a moment the screen should be clear save for the *ged* prompt, *, in the upper left corner. The * tells you 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 The 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>**

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

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

† UNIX is a trademark of Bell Laboratories.

A command is executed by typing the first character of its name. *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 *****.

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 you to examine and modify the default values. Type:

***set <cr>**

to see the current default values.

The value of an option is either of type 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. (Actually, this applies whenever text is being entered.)

2.2 Constructing Graphical Objects

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

2.2.1 Generating Text. To put a line of text on the display screen use the *Text* 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.

Text 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)

top
right

lower
left

Figure 1. Generating text objects

2.2.2 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 *possible options* <cr>**

Lines 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 *Lines* it is more of the same. As an example, let us build a triangle:

***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)

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.

The points in the current point set may be referenced by name using the **$** operator. **$*n*** references the point numbered ***n***. Using **$** we can redraw the triangle above by entering:

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

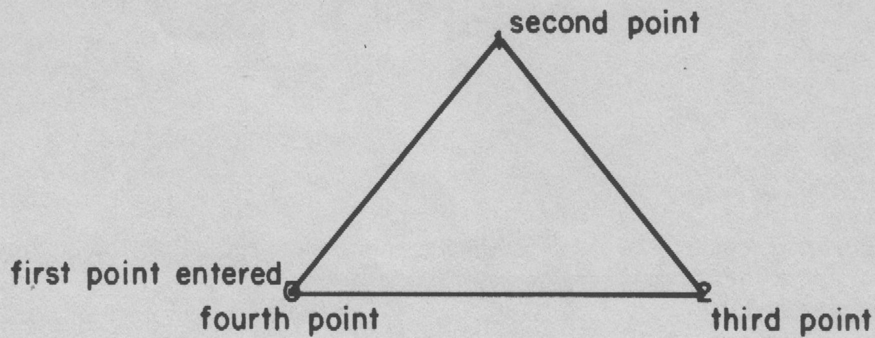


Figure 2. Building a triangle

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; . 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, let us use the triangle just entered as the basis for drawing a quadrilateral:

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

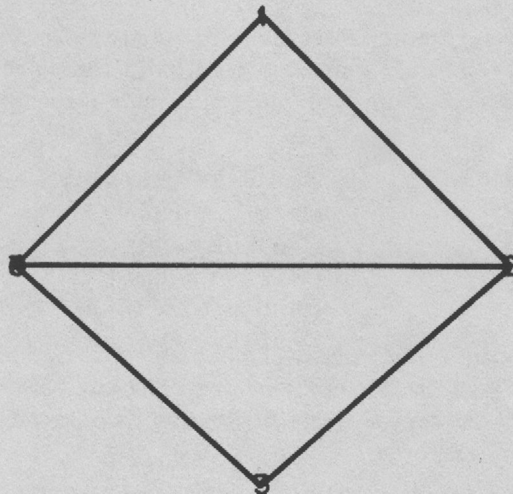


Figure 3. Accessing the previous point set

Individual points from the previous point set can be referenced by using the . operator with **$**. We will build 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)
<sp>	(enter a new point)
$0 <cr>	(or **$.1**, to close the figure)
<cr>	

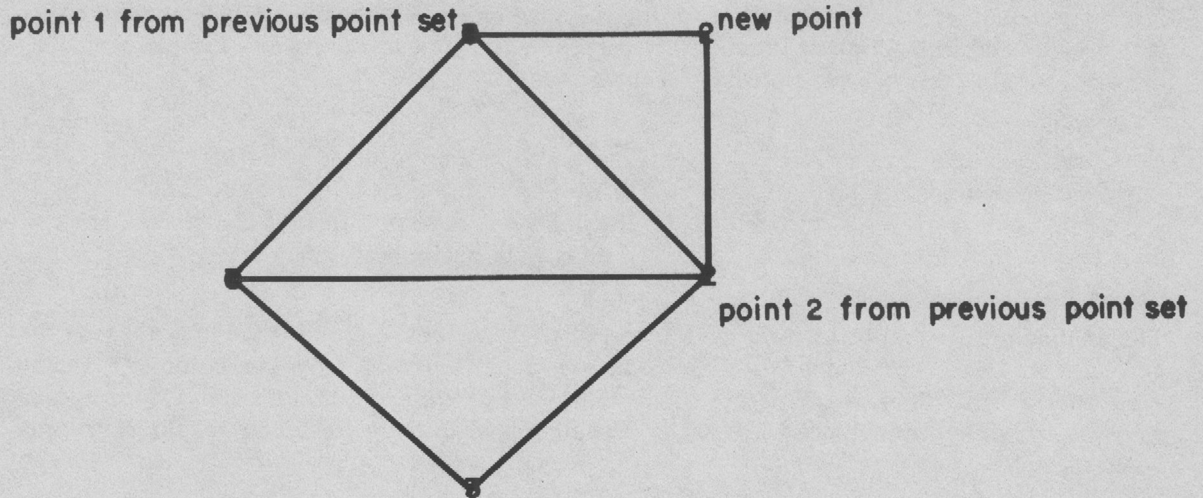


Figure 4. Referencing points from the previous point set

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

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

In commands that follow you can now reference point A using the $ operator, as in:

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

2.2.3 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>

2.3 Editing Objects

2.3.1 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 *objects* command marks the location of each object-handle with an **O.** Type:

*objects −v <cr>

to see the handles of all the objects on the screen.

Some objects, *Lines* for example, also have *point-handles*. Typically each of the points entered when an object is constructed becomes a point-handle. (Yes, an object-handle is also a point-handle.) The *points* 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, try deleting one of the objects you have created on the screen.

*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 *Delete* command is to give you 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>** . Use the *new* command to get a fresh copy of the remaining objects.

Notice that defined-areas are entered as *points* in the same way that objects are created. Actually, a defined-area may be generated by giving anywhere from zero 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 *Delete* command would be:

*Delete <cr>
<position cursor> (center the 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 **universe** option to the *Delete* command. Note the difference between the commands **Delete −u** and **erase**.

2.3.2 Changing the Location of an Object. Objects are moved using the *Move* command. Create a circle using *Arc*, then move it as follows:

*Move <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 addressed by the distance from the *pivot* to the *destination.* In this case we chose the pivot to be the object-handle, so effectively we moved the object-handle to the destination point.

2.3.3 Changing the Shape of an Object. The *Box* 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. Draw a box:

<**cr**> throughout: commands are set in bold mixed with <angle-bracket> keys.

*Box <**cr**>
<**position cursor**> <**sp**>
<**position cursor**> <**cr**>

Box 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.

*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)

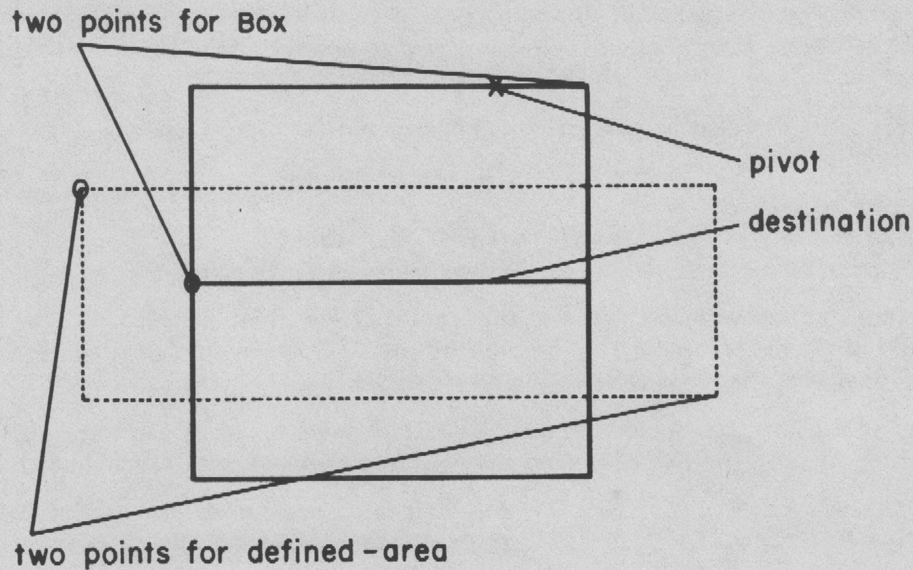


Figure 5. Growing a box

Because the **p**oints flag is true, the operation is applied to each point-handle addressed. In this case each point-handle within the defined-area is moved the distance from the pivot to the destination. If **p** were false only the object-handle would have been addressed.

2.3.4 Changing the Size of an Object. The size of an object can be changed using the *Scale* command. *Scale* scales objects by changing the distance from each handle of the object to a pivot by a factor. Put a line of text on the screen and try the following *Scale* commands:

*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**>	

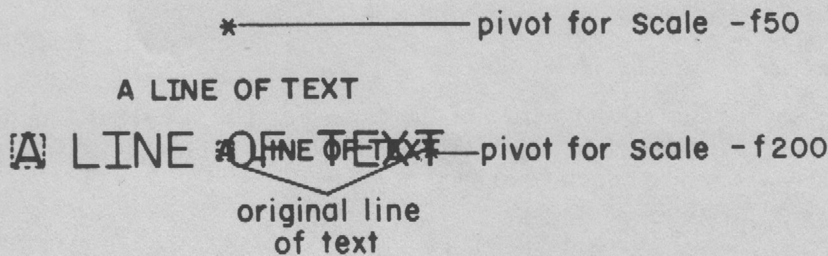


Figure 6. Scaling text

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 *Circle* command:

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

Circle 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° around the circle. One way to change the size of the circle is to move one of the point-handles (using Move **−p**).

The size of text characters can be changed via a third mechanism. Character height is a property of a line of text. The *Edit* command allows you to change character height as follows:

*Edit **−h***height* **<cr>**	(*height* is in universe units, see Section 2.4)
<position cursor> <cr>	(point to the object-handle)
<cr>	

2.3.5 Changing the Orientation of an Object. The orientation of an object can be altered using *Rotate*. *Rotate* rotates each point of an object about a pivot by an angle. Try the following rotations on a line of text:

*Rotate **−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>	

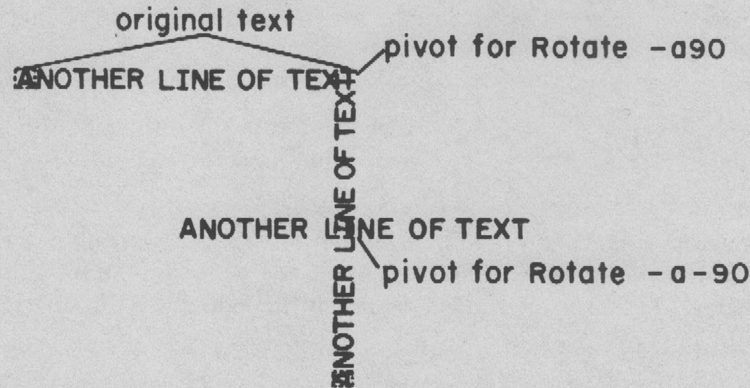


Figure 7. Rotating text

2.3.6 Changing the Style or Width of Lines. In the current editor objects can be drawn from lines in any of five styles (**s**olid, **d**ashed, **d**ot-**d**ashed, **d**otted, **l**ong-**d**ashed) and three widths (**n**arrow, **m**edium, **b**old). Style is controlled by the **s** option, width by **w**:

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

creates a narrow width dotted line.

***Edit −wb,sdd <cr>**	
<position cursor> <cr>	(point to object-handle of the line)
<cr>	

changes the line to bold dot-dashed.

2.4 View Commands

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

***view −u <cr>**

displays the entire universe.

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 *zoom* command. To build a window that includes all of the objects you have drawn, type:

***zoom <cr>**	
<position cursor> <sp>	(above and to the left of any 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 *view*. *View* 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. We have already seen **view −u**. **view −h** 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, using *view* you may select to 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* and *vtoc* (see [1]).

2.5 Other Commands

2.5.1 Interacting with Files. To save the contents of the display buffer copy it to a file using the *write* command:

*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 *read* are the same as those for the *Text* command.

2.5.2 Leaving the Editor. Use the *quit* command 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*. A second *quit* forces exit.

2.6 Other Useful Things to Know

2.6.1 One-Line UNIX Escape. As in *ed*, **!** provides a temporary escape to the shell.

2.6.2 Typing Ahead. Most programs under UNIX allow you to type input 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.

2.6.3 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 are optional. Parentheses surrounding arguments separated by "or" means that exactly one of the arguments must be given. For example, the *Delete* command (Section 3.2) accepts the arguments **−universe**, **−view**, and *points*.

3.1 Construct commands:

Arc	[**−echo,style,width**] *points*
Box	[**−echo,style,width**] *points*
Circle	[**−echo,style,width**] *points*
Hardware	[**−echo**] *text points*
Lines	[**−echo,style,width**] *points*
Text	[**−angle,echo,height,midpoint,rightpoint,text,width**] *text points*

3.2 Edit commands:

Delete	(**−** (**universe** or **view**) or *points*)
Edit	[**−angle,echo,height,style,width**] (**−** (**universe** or **view**) or *points*)
Kopy	[**−echo,points,x**] *points pivot destination*
Move	[**−echo,points,x**] *points pivot destination*
Rotate	[**−angle,echo,kopy,x**] *points pivot destination*
Scale	[**−echo,factor,kopy,x**] *points pivot destination*

3.3 View commands:

coordinates	*points*
erase	
new	
objects	(**−** (**universe** or **view**) or *points*)
points	(**−** (**labelled-points** or **universe** or **view**) or *points*)
view	(**−** (**home** or **universe** or **region**) or [**−x**] *pivot destination*)
x	[**−view**] *points*
zoom	[**−out**] *points*

3.4 Other commands:

quit	
read	[**−angle,echo,height,midpoint,rightpoint,text,width**] *filename* [*destination*]
set	[**−angle,echo,factor,height,kopy,midpoint,points,rightpoint,style,text,width,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:

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 ($0 \leq n < 1280$).
kopy	The commands *Scale* and *Rotate* can be used to either create new objects or to alter old ones. When the **kopy** 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 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 is false, *text* strings are outlined rather than drawn.
width*type*	Specify line width to be one of 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:

home	Reference the home-window.
region*n*	Reference region *n*.
universe	Reference the universe-window.
view	Reference those objects currently in view.

4. ACKNOWLEDGEMENTS

Ged borrows freely from the ideas and code of the *gex* program by D. J. Jackowski. The first version of *ged* was written by D. E. Pinkston.

5. REFERENCES

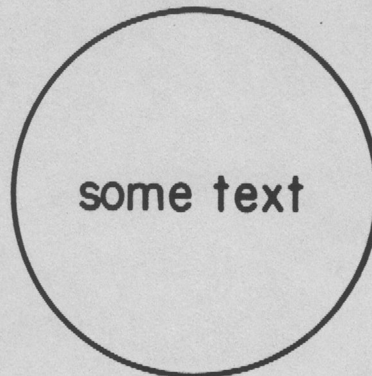
[1] Feuer, A. R. *UNIX Graphics Overview,* Bell Laboratories (1979).

[2] Dolotta, T. A., Olsson, S. B., and Petruccelli, A. G. (eds.). *UNIX User's Manual*—Release 3.0, Bell Laboratories (June 1980).

APPENDIX: Some Examples of What Can be Done

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>	

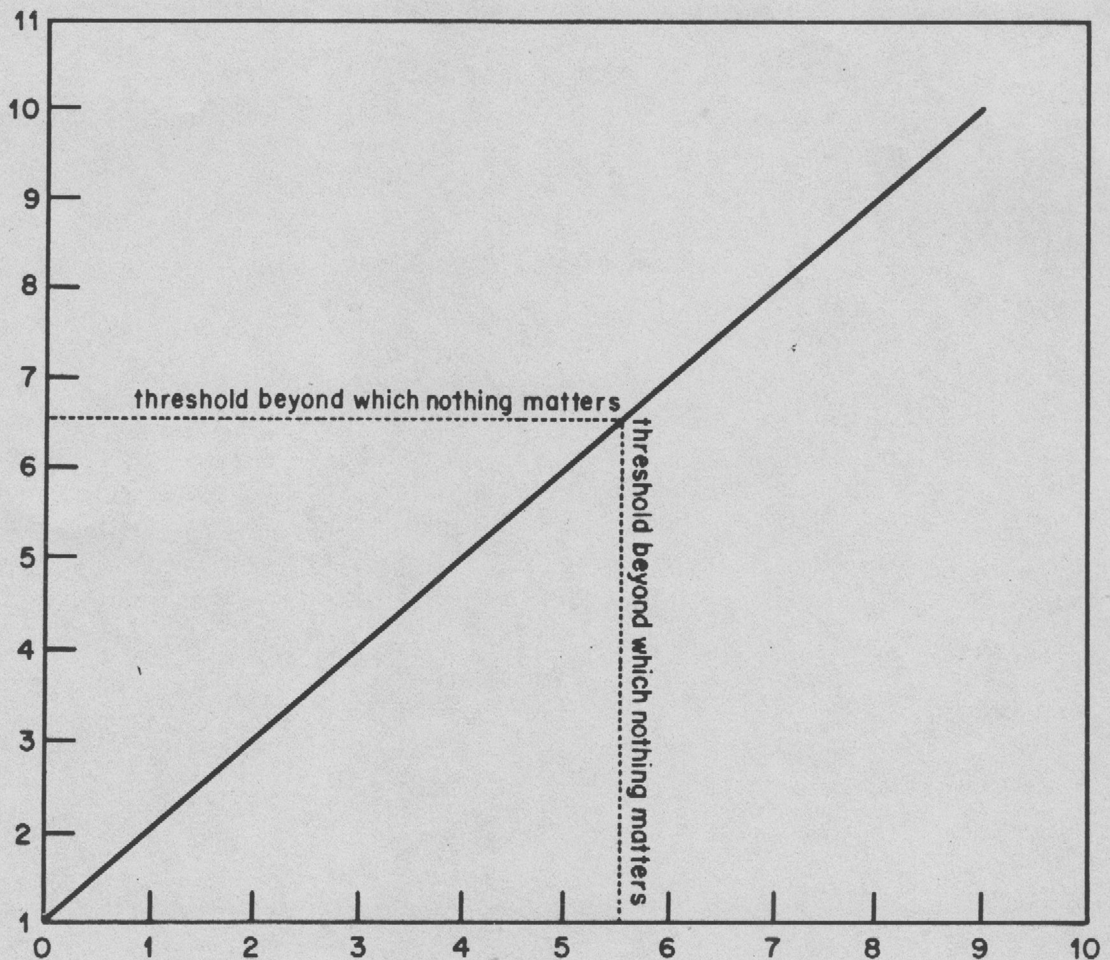


2. Making Notes on a Plot

```
*! 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>
<position cursor> <sp>
<position cursor> <sp>
<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 of plot)
<position cursor> <cr>                     (bottom right)
*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)
```

SOME KIND OF PLOT



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 into 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 \n(nlu <cr>                         (mark end of page)
. <cr>
w C <cr>                                (put the text into file C)
156
q <cr>
*! nroff C | yoo C <cr>                 (format C, leave the output in C)

*view −u <cr>                           (window on the universe)
*read A <cr>
*read B <cr>
*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)
```

On this page are two histograms from a series of 40 designed to illustrate the weakness of multiplicative congruential random number generators.