Vplot: SEP's plot language

Steve Cole and Joe Dellinger

ABSTRACT

Vplot is a plotting system that has been developed at SEP over the last 10 years. It is portable to UNIX systems. It supports at least fourteen screen and hardcopy display devices, including every device at SEP. It supports line, area, text, and raster graphics primitives, and can support all four on any device. One drawback is that it is not interactive.

PREFACE

The Vplot graphics package has been distributed on a worldwide computer network. A copy of the distribution is available to SEP sponsors on request. This document is an introduction to Vplot, and a brief tutorial. In Appendix 1 a historical review of the development of Vplot at SEP is presented. Attached in Appendix 2 are copies of the most useful on-line manual pages distributed with Vplot. These describe the software in greater detail.

INTRODUCTION

Vplot is SEP's home-grown plot language. It accounts for most of the plotting done at SEP. In this report, for example, about 85% of the figures were prepared using Vplot. The remaining 15% are mostly figures produced using commercial Apple Macintosh graphics programs or by an SEP seismogram plotting program specialized to the Imagen laser printer.

Scope of Vplot

Vplot should run on any machine that uses UNIX[1] as an operating system. It is known to have been successfully ported to the IBM PC-RT, Apple Macintosh II,

[1]UNIX is a registered trademark of AT&T.

Sun 3, Sun 4, Vax, Gould PN9050, and Convex computers. Vplot currently consists of roughly 30,000 lines of C source code and 32,000 words of documentation, not including the 5844 words in this paper.

Vplot graphics can be plotted on many different output devices. At SEP we have supported the following devices:

- terminals
  - RasterTek
  - Envision
  - Gigi (DEC Regis)
  - Tektronix 4105
  - Tektronix 4010
- hardcopy devices
  - Imagen laser printer
  - Apple LaserWriter (PostScript[2])
  - Printronix line printer
  - HP pen plotter
- workstation window systems
  - Sunview
  - InterViews
  - NeWS
  - X Windows
  - Masscomp gps
- virtual devices
  - generic raster formats
  - generic vector formats
  - the virtual Vplot device

A great deal of care has been taken in Vplot to keep *device-dependent* and *device-independent* portions of the source code well segregated, so that only a few small C subroutines need be written to support any new graphics device.

Vplot instruction set

Vplot supports a wide variety of graphics primitives, which can be grouped into four broad categories:

[2]PostScript is a registered trademark of Adobe Systems Incorporated.

- Vector. Vectors include such things as lines, fat lines, dashed or dotted lines, and dots.
- Polygon. Polygons are any sort of filled area. Vplot allows a polygon to be filled with a solid color, a simple dither pattern, a user-defined cross-hatch pattern, or a user-defined bitmap.
- Text. Vplot includes a number of device-independent fonts which are quite flexible, with such niceties as non-English alphabets, ligatures, superscripts, and subscripts. Access is also provided to any built-in fonts that a particular device may have.
- Raster. Vplot's raster is device-independent, scaling consistently with the other graphics primitives. Raster on monochrome devices is automatically dithered to simulate grey levels.

Vplot's single most important attribute is that it supports this entire range of graphics primitives on *any* device, and does so in a completely device-independent way.

Structure of Vplot

Vplot consists of three major parts:

(1.) Vplot metafiles. Vplot metafiles contain encoded descriptions of what is to be plotted. The metafile format is device-independent. By saving important figures as Vplot metafiles, one ensures that they can be reproduced at a later date with their full original resolution. See the *Vplot* manual page in Appendix 2 for more detailed information about Vplot metafiles.

The Vplot distribution includes several utility programs for editing and manipulating Vplot metafiles; See the section of this document entitled **Helpful utilities** for more information.

(2.) "Pen filter" programs. Pen filters are the only device-specific part of Vplot; they accept as input a Vplot metafile, interpret the encoded graphics commands in it, and create the desired plot on a particular graphics output device.

Pen filters have many useful command-line options that can be used to modify their behavior. Refer to the section of this document entitled **Using pen filter options.**

(3.) A subroutine library, *libvplot.* This library provides C and FORTRAN programmers with a simple graphics interface. For more information, refer to the **Libvplot** section of this document.

When a user program calls a libvplot subroutine to draw something, the subroutine writes out the corresponding Vplot command. Under UNIX this

metafile output can be saved into a file on disk, or directed into the input of a pen filter to plot immediately without saving the metafile.

LEARNING ABOUT VPLOT

To illustrate how Vplot is used, we will now discuss Vplot from several different perspectives.

We first examine Vplot from the perspective of the naive end user, who uses Vplot when running graphics application programs but probably doesn't realize it. For the more experienced end user, we show how plotting can be tailored by using pen filter command line arguments. For the most advanced end user we discuss several Vplot utilities which can be used to modify and combine Vplot metafiles in complicated ways.

From the perspective of the applications programmer, we show what can be done with Vplot by examining the routines that a program can use to produce graphics.

Finally, for the Vplot hacker, we briefly discuss how to add support for new devices, and point to the documentation in the source code for further exploration.

VPLOT FOR THE END USER

At SEP, there are a large number of programs that output Vplot graphics. Some of the more widely-used ones are *Wiggle*, which displays seismic data as wiggle traces, *Graph*, which draws a simple graph, *Ta2vplot*, which displays seismic data in raster form with shading to indicate amplitude values, *Contour*, which creates a contour plot, and *Thplot*, which draws a 3-D perspective plot.[3]

All of these programs are used in the same way:

```
program < datafile > out.vplot
pen < out.vplot
```

where `program` is a graphics utility, `datafile` in input data, `out.vplot` is the Vplot metafile it creates, and `pen` is a Vplot pen filter. The particular pen filter program used is determined by the desired output device; for example at SEP

```
rpen < out.vplot
```

would plot on the RasterTek,

[3]These programs are not included with the Vplot distribution because they also depend on other software developed at SEP, including the less portable data cube processing library known at SEPlib.

```
pspen < out.vplot
```

would produce hardcopy on the Apple LaserWriter, and

```
spen < out.vplot
```

would plot on a Sun 3 workstation.

To make things simpler at SEP we have a program "tube" which examines the terminal the user is using, and picks the appropriate pen filter for that device automatically. Thus at SEP using any Vplot graphics utility to plot on your screen is as simple as typing

```
program < datafile | tube
```

Here we have used the UNIX "|" (pipe) construction to skip saving the intermediate Vplot metafile to disk.

The advantage in having all these programs written to use Vplot is that the graphics output of all these programs can effortlessly be displayed on any of the graphics devices at SEP, from workstations to dot-matrix line printers. Also, since the various "pen" filters all share a common interface, the end user can use the same commands to manipulate a Vplot image no matter what device is being used. This common interface is described in the next section.

The Pen interface

Pen filters contain device-dependent and device-independent code. Interaction with the user is handled mainly by the device-independent code, assuring a standard common interface. One specifies the same command-line option to scale a plot, whether it is being plotted on a laser printer or a workstation screen.

This user interface or set of options common to all pen filters is described in detail in the *Pen* manual page, which is included in Appendix 2. Here we will list a few of the most frequently used pen filter options for reference:

- scale
  Scale a plot by this factor.
- hshift,vshift
  Shift a plot by a number of inches horizontally or vertically.
- xscale,yscale
  Stretch the plot along the x or y axis by this factor.

- rotate
  Rotate a plot this many degrees clockwise. Rotation is performed about the Vplot origin, wherever that is on the screen.

- erase
  This option lets you control how the pen filter interprets erase commands. For example, "erase=once" means to erase once at the beginning and ignore erase commands after that. This lets you plot several things on top of each other.

- xcenter, ycenter
  Nail this Vplot coordinate to the center of the screen, no matter what other scaling or rotation may also be applied.

- txfont
  Selects the default text font. Programs that output text should generally not define the font, leaving it to the user to select the font at this stage.

Using pen filter options

All the options described above (as well as all the other options listed in the *Pen* manual page in Appendix 2) are specified on the command line when the pen filter is run. For example, earlier we showed the command sequence necessary to plot a Vplot file named `out.vplot` on the RasterTek screen using the pen filter `rpen`. If you wanted to do this and have the image be twice as large, you would type:

```
rpen scale=2 < out.vplot
```

Since Vplot is device-independent, you would use the same `scale=2` to magnify the image on any device, using the appropriate pen filter.

Device-dependent options

Some pen filters have additional options that are unique to the particular device. For example, the filters for our Imagen and Apple laser printers have a `hold` option that instructs the printer to wait for special paper to be inserted into the printer's manual feed slot. Such device-dependent options are described in the manual page for the appropriate pen filter.

Helpful utilities

There are a few important utilities that are of use to Vplot users.

Pldb and plas

Earlier it was mentioned that Vplot has its own special format for storing Vplot instructions in a file. While users seldom need to know about this format or examine a Vplot file, there are two utilities that let you examine a Vplot file if the need does arise. The program **pldb** converts a Vplot file into the readable ASCII format described in the *Vplot* manual page in Appendix 2. The program **plas** converts an ASCII Vplot file back into Vplot's mixed ASCII-binary format.

These two programs provide a simple way to make minor modifications to a Vplot file. You can convert the file to ASCII using pldb, edit it manually, then convert it back using plas. While this is not generally a recommended procedure (especially for large complicated plots), it does have the advantage of being completely general. Usually you should use pen filter options to modify the image as you plot it if you can, or modify and rerun the program that wrote the Vplot file in the first place.

Another way to modify Vplot files is described in the next section.

Vppen: the Vplot to Vplot filter

Most pen filters convert Vplot instructions to the format particular to some display device. One filter, Vppen, reads *and* writes Vplot. This filter may seem useless at first glance, but it useful in several ways:

- Suppose that while plotting a Vplot image on some device you discover a set of parameters that give a particularly nice rendition of the image. It would be nice to save the parameters, so that when you look at the file later you can obtain the same picture without needing any extra command line arguments. Sending the file through Vppen and specifying those same command-line options (such as scaling, shifting, changing fonts) in effect *records* those options into the file itself.

- You can use Vppen to merge two Vplot files into one. Specify both files on input and save the result to a file. (E.g. `vppen a b > a+b`.) Note that when you plot the new file, you will see the first image, then there will be a screen erase (or page feed for hardcopy devices), then you will see the second image. To plot both images on top of one another, specify `erase=once` in the Vppen step that merges the files.

- Vppen will find plot statistics for you, such as the position and size of an image in Vplot space. Simply specify `stat=y` in the Vppen step and you will get a one-line message telling you the plot's size and position.

- Vppen can position and scale your plot automatically to fit in a desired space. The `align` option can be used to position the image, and the `xsize` and `ysize`

options can be used to have Vppen scale the image to fit in a box of the desired size.

- Vppen can also be used to combine two or more plots into a grid. Scientific research often necessitates the comparison of a number of images in this way. For example, in SEP reports we often show side-by-side a seismic section before and after the application of some process. This feature of Vppen is enabled by using the `gridnum` option.

All these features are described in more detail in the *Vppen* manual page, which is included in Appendix 2.

VPLOT FOR THE APPLICATIONS PROGRAMMER

What sort of graphics can Vplot produce? A programmer writing a program that will output graphics has a number of Vplot routines that he or she can call to generate various graphics objects. In this section we will discuss the graphics programmer's interface to Vplot.

Libvplot

The set of routines that a program can call to output Vplot graphics is known as *libvplot*. A full description of this library, including the arguments for each routine, is presented in the *libvplot* manual page in Appendix 2. The discussion here is a brief overview of the library. The intent is to start with the most basic calls, and gradually get more complicated, with an emphasis on how Vplot handles the peculiarities of a graphics device in deciding how to respond to a request.

Let's begin with some basic calls:

- **vp_move(x,y)**
  Moves the pen to coordinates (x,y). Doesn't draw anything.

- **vp_draw(x,y)**
  Draws from the current position to (x,y). vp_draw draws its line using the current color, line fatness, and line style. The following subroutines set these attributes:

- **vp_color(col)**
  Sets the current drawing color to color number 'col', which is a number between 0 and 511. Colors 0 through 7 are pre-defined by Vplot as black (background), blue, red, purple, green, cyan, yellow, and white, respectively. C programmers may refer to these colors by name (in all caps) rather than by number since they are defined in an include file "<vplot.h>". To use a color other than these seven, you must first call **vp_coltab** to define the color:

- **vp_coltab(col,red,green,blue)**
  Sets color number 'col' to have the color (red, green, blue). Each of red, green, and blue is a floating point number between 0 and 1. Thus (1.,1.,1.) is white and (1.,1.,0.) is yellow.

  The **vp_color** and **vp_coltab** calls are easy to define, but using them properly so that your program works well on all different kinds of devices can be tricky. Read the section "Color vs. monochrome devices" to find out more. An important thing to remember is that color 0 is always the background color, even if your "screen" is white, like a piece of paper. Drawing in color 0 erases.

- **vp_fat(fatness)**
  Sets the line width. At SEP this is in units of 200 to the inch. (Such arbitrary numbers in Vplot are historical artifacts. They can always be changed at installation time.) The default is 0, which is not an infinitely thin line, but the thinnest line that the graphics device can plot.

- **vp_dash(dash1,gap1,dash2,gap2)** and
  vp_setdash(dashp,gapp,lp)
  Set the current line style. Normally Vplot draws solid lines, but you can use either of these to specify a line dashing style. **vp_setdash** is the more general of the two, but check out both subroutines in the libvplot documentation to see which is more suited to your needs.

So far we've learned only how to draw lines. Here are some more tools related to polygons and filled areas:

- **vp_area(xp,yp,lp,fat,xmask,ymask)**
  Fills an arbitrarily shaped area with a color or pattern. You supply arrays containing the x and y coordinates of the vertices of a polygon to be filled. How Vplot does the filling depends on whether the output device has color. If it does the area is simply filled with the current color. If it's monochrome, Vplot constructs a simple dither pattern to fill with based on the xmask and ymask parameters.

- **vp_fill(xp,yp,npts)**
  This is a more powerful area fill call that lets you specify your own patterns of arbitrary complexity. The fill is performed with the pattern whose number is given by the current drawing color (from 0 to 511). You can specify patterns with either **vp_hatchload** or **vp_patload**, described below. If you haven't set that pattern, Vplot fill solidly with the current color.

- **vp_hatchload(angle,numhatch,ipat,array)**
  Defines a hatch pattern to fill polygons with. The angle, color, and thickness of the lines making up the hatch pattern are all settable.

- **vp_patload(ppi,nx,ny,ipat,raster)**
  Defines a pattern by specifying its actual bitmap representation.

There is only one raster call, but it has too many arguments to list on one line:

- **vp_raster(array,blast,bit,offset,**
  xpix,ypix,xll,yll,ppi,xur,yur,orient,invert)
  This subroutine lets you plot a raster image, with each byte in the input array specifying the color of one pixel in the raster image. Note that since the raster block can be arbitrarily stretched and scaled, pixels in the raster block do not necessarily correspond to pixels on the output device.

  The interaction between raster and color is complicated enough to deserve a man page of its own; refer to the *Vplotraster* manual page in Appendix 2 and to the section **Color vs. monochrome devices** in this paper to find out more.

Finally there are several subroutines related to text:

- **vp_pmark(npts,mtype,msize,xp,yp)**
  Draw "markers" (any of a number of symbols defined by Vplot, or a character in the current font) at a number of points.

- **vp_text(x,y,size,orient,string)**
  Display a text string. The size and orientation of the text are specified in this call; the position is set by a previous call to **vp_move**. In software text Vplot recognizes several TEX-like escape sequences. A user can exploit these in text given to a graphics program to specify fancy things like mathematics, non-English characters, color changes, etc, which then automatically appear in the output plot with no extra work required of the programmer.

- **vp_tfont(font,prec,ovly)**
  Choose the font to be used for displaying text. Vplot has a number of built-in software fonts. In addition, some devices may have their own fonts, which are also accessible through Vplot. The font precision parameter is meant to control the text quality used; it basically has no effect for software fonts other than to enable ligatures (automatic grouping of adjacent symbols, for example "affine" becomes "affine"). The text overlay mode controls whether the text is boxed or not, and whether the inside of the box is erased before drawing the text.

There are a number of other commands available in libvplot. Consult the *libvplot* manual page (attached in Appendix 2) for information on those commands and for detailed instructions on how to use the commands described here.

User coordinates

In all the commands given above, any positions need to be specified in inches (much to the dismay of all the world except for the USA and Burma). You can define your own user coordinate system to avoid having to work in inches. To do that, there are user-unit versions of most routines (for example, **vp_uraster** instead of **vp_raster**) that let you specify positions in your user coordinate system. You will also need to call **vp_orig**, **vp_uorig**, and **vp_scale** to describe the relationship between your user coordinate system and Vplot's own coordinate system. This is discussed in the *Libvplot* manual page.

Color vs. monochrome devices

Devices vary considerably in their capabilities with respect to color. At one extreme are devices with 256 user-definable colors; at the other are monochrome devices with 2 hard-wired colors, usually black and white. In between there are devices that have 16 or 128 user-definable colors, or 4 or 8 hard-wired colors. Proper use of color is tricky, if you want your program to work well no matter what the color capabilities of the output graphics device.

Vplot's exact algorithm for handling color in all these different cases is described under the "Set color table" command in the *Vplot* manual page. The underlying principle is that when asked to plot in a particular color, if the color itself is not available on a device, Vplot finds the color that *best matches* the requested shade from among those that are available. When writing applications programs it suffices to follow a few guidelines:

- Color 0 is always the background color; even if your screen is white, not black, setting to color 0 and drawing erases things from the screen. Any color the same exact shade as color 0 behaves exactly the same as color 0. Any color not the same exact shade as color 0 does not erase like color 0.

  For example, suppose you set color 0 to be green. On most color devices this will turn the screen green. Plotting in color 4, green, erases on such a screen. What would happen on a laserprinter, that has a white background and only draws in black or white? When color 0 was set to green, nothing would happen: the background physically can't turn green. But color 4, green, will now plot white. Any color that is not green will plot black (the foreground color).

- The higher the color number you try to use, the less chance Vplot will be able to oblige. The most important colors should be given the lowest numbers. For example, suppose in your program you redefine colors 1 through 7 to be several nice shades of brown, and then define colors 8 through 58 to be a rainbow sweep. This is a bad way to do things, because a device with only 8

definable colors will be stuck having to display everything in the rainbow as the closest shade of brown.

- Vplot guarantees that colors 0 through 7 are predefined and standard on all devices. Unless you have good reason, don't change these. That way on any color device you'll at least be left with the standard {black, blue, red, magenta, green, cyan, yellow, white} set for Vplot to choose an approximation of the color you want from. Note that if color 0 is white, color 7 will be black.

FORTRAN vs. C

The libvplot subroutine names given here are those that you would call from a C program. In FORTRAN you should omit the underscore in the names (for example, use **vpraster** instead of **vp_raster**).

VPLOT FOR THE GRAPHICS PROGRAMMER

This section is aimed only at a small subset of Vplot users, those who are interested in adding support for new devices or tinkering with the device-independent innards of Vplot.

Supporting new devices

Detailed instructions for constructing a pen filter to support a new device are given in the *Vplothacker* manual page, which is distributed along with the Vplot source. As this manual page refers repeatedly to various source files, we have not included it in this paper. Here we will simply outline the process of supporting a new device.

A key point in supporting a new device in Vplot is that the device can be supported at *various levels of complexity.* In writing a pen filter, you are telling Vplot how to convert its various primitives (lines, areas, text, raster) into forms that your device can understand. For some primitives, such as a line, this may be very easy. For other primitives, such as raster, this may be very difficult.

As a shortcut you can tell Vplot to convert complicated primitives such as text, polygons, and raster into simple primitives such as points and lines. In this way, you can simplify the process of supporting a new device. This lets you construct a pen filter that will plot all of Vplot's primitives on your device, even though you only implemented a small subset of those primitives. We have seen people construct new Vplot filters in a matter of an hour or two using this approach.

Of course, your pen filter may not be very efficient if it needs to have complicated objects broken down into points and lines in order to plot them. Plotting may be very slow. With Vplot you can go back at any time and add hardware support for new primitives, gradually improving the speed and efficiency of your pen filter.

Inside Vplot

At this point, if you are interested in learning more about Vplot the best way to do that is to obtain a copy of the source and begin browsing it. The main Vplot directory has a number of subdirectories, which we will describe briefly in order to get you started. There is a **Documentation** directory containing all the on-line documentation; a directory named **lvplot** that contains the source for the libvplot subroutines that programs call to output Vplot; and a **utilities** directory that contains the utility programs *plas* and *pldb*. The most important directory is **filters**. It contains the device-independent Vplot source, and a subdirectory containing the device-dependent code for each pen filter.

APPENDIX 1: A HISTORY OF VPLOT[4]

Origins

Vplot originated in 1978 with Rob Clayton. Rob defined the original device-independent metafile format and wrote the first pen filter, which he called "Pen". The "V" in Vplot originally stood for "vector", as the original Vplot supported only vectors and a single basic text font. At that time Vplot ran on a PDP-11-34 with 48K of RAM. The only output device was a 200 dot per inch Gould electrostatic plotter, which fitfully made slimy carcinogenic plots amid a great deal of noise. Later a Gould-dependent raster command was added to "Pen".

Dave Hale wrote the original version of libvplot around 1980.

After a year or so, the first DEC Gigi screen terminals arrived. Jon Claerbout and Stew Levin hacked the source code for the Gould pen filter to make it work on DEC Gigi terminals. The new pen filter was christened "Gpen", and had scattered through its source chunks of dead code leftover from "Pen" related to vector rasterization.

On the Gould plotter, the X axis pointed in the paper feed direction, and so could be infinite. It was decided that the Gould's paper feeding corresponded to scrolling on the Gigi, and so Gpen put the origin in the upper left corner of the screen, with the X axis going down and the Y axis across.

Around this time SEP transferred all its software from the PDP-11 to a fancy new VAX 11/780 called Mazama, which nearly filled the computer room. Also around this time a new Varian electrostatic plotter arrived. Software-wise, it looked exactly the same as the Gould, but its plots were not nearly as slimy.

Degeneration

In 1982 SEP purchased an AED terminal, and Jeff Thorson hacked together a new Vplot filter for this device. The AED pen filter "Apen" put the origin in the lower left, with the X axis running across and the Y axis running up, so Apen plots were rotated 90° relative to Gpen plots. They came out a different size, too. To fix this problem a short program "Vzoom" was written which would scale and rotate vector commands in Vplot metafiles. Later an AED-dependent polygon command was added to "Apen".

Sometime around 1982 a copy of Vplot migrated across the hall to Erebus, a VAX 11/750. Within a year groups on both sides of the hall had separately tinkered with their copies of the Vplot source code. As a result each side used versions of the Vplot metafile language, libvplot, and Pen incompatible with the other.

[4]Some of the facts and most of the dates are probably wrong, but the lessons to be learned about the evolution of software are valid nevertheless!

At SEP the metafile resolution was changed from the Gould's original 200 units to the inch to 600, to prepare for higher resolution devices to come. Unfortunately whoever made this change in the code missed changing one hardwired "magic" number, with the result that text went from being measured in units of 1/100'ths of an inch to 1/33.333333'rds of an inch. The Gigi position of the coordinate origin in the upper left hand corner was declared standard at SEP.

On Erebus the metafile resolution was left the same as it had been, but the names of all the routines in libvplot were changed instead. On Erebus the AED coordinate origin in the lower left hand corner was declared standard. In the process of diverging, Glenn Kroeger on Erebus created a new Vplot filter called "Tekpen" for the Tek 4014, which was the first Vplot filter since "Pen" not to contain large hunks of dead code.

Regeneration

In 1983 several Envision terminals were bought, and Joe Dellinger was assigned the task of hacking together "Epen" out of "Gpen" as his first programming project not in FORTRAN or Basic. A few months later, dissatisfied with the very buggy Epen which resulted, he obtained a copy of Glenn Kroeger's Tekpen from Erebus and converted it back to SEP's Vplot standards. He then moved all the device dependent parts he could identify out of the single original source file and into separate subroutines organized by function. As he created each new subroutine he documented what its responsibilities were. He then turned the new "Tekpen" into a new less-buggy "Epen". This apparently fruitless activity was generally derided.

Later Glenn stole back the improved "Epen" and converted it back to Erebus's Vplot standards. Using the documentation describing what each of the device-dependent subroutines did, he was able to use the new "Epen" as a template for supporting several more devices. In the process he used his superior knowledge of C to replace Joe's hard-wired numbers and names with proper C defines, pointers, and structures.

Joe then stole back Glenn's improved device-independent structure, redocumented it, added capabilities to support all the incompatible Vplot definitions, and rewrote all the SEP Pen programs to use the new, unified system. By 1985 the current structure had jelled. Later that year, amid much complaint, Joe and Glenn used their Vplot dictatorial powers to enforce a new standard libvplot and metafile format on both Erebus and the new SEP computer, Hanauma (which was so small it made the computer room look huge). Unfortunately, to avoid breaking all Jon's archived figures from FGDP, fatness stayed measured in units of 1/200'ths of an inch, and text in units of 1/33'rds of an inch.

Using the new structure, Joe found he could get *other* people to support new devices by getting them to write a few simple subroutines. They didn't have to understand the device-independent kernel of Vplot, and he didn't have to under-

stand their device-dependent subroutines. This allowed everybody, especially Joe, to get more research done. Using this technique new Vplot filters have been fairly painlessly created as necessary at the rate of about 2 per year.

Maturity

In 1985 Joe deleted the now completely useless Gould-dependent raster command and replaced it with a new device-independent raster command. This Vplot raster capability was not used by anyone for about a year. Later Steve Cole added dithering to the Vplot device-independent kernel, and was surprised when Joe demonstrated to him that not just the Imagen, but *every* monochrome device at SEP was now able to display grey-level raster because of his work. Immediately the popular imagen-dependent program *Tiplot* lost all of its dithered-raster users to a new Vplot utility *Ta2vplot*, because with Ta2vplot it was possible to plot on other devices besides the imagen, and even more importantly, it was possible to position and scale the plot arbitrarily.

In 1986 Joe added the capability to generate Hershey software text fonts to the Vplot kernel (he ran across the fonts circulating around the network). This made text under Vplot look much more professional, and allowed Chuck Sword to label his plots in Russian.

Also in 1986 a copy of Vplot migrated to the Center for Wave Phenomena at the Colorado School of Mines with SEP graduate Shuki Ronen. At the time they only had a few graphics devices to support, and so their version of the Vplot source was simplified to make the code easier to understand. Less used graphics primitives such as raster were discarded, and most of the device-independent code was either moved into device-dependent subroutines or deleted. CWP currently distributes this somewhat device-independent Vplot-derived graphics system with their "SU" seismic processing system under the name "cplot".

In 1987 Vplot was officially distributed on the usenet newsgroup "comp.sources.unix". Usenet is an international computer network of UNIX computers, connecting many companies, universities, government installations, bulletin boards, and miscellaneous sites.

Joe has vowed to wash his hands of Vplot as much as possible, but that didn't keep him from helping to write this article.

APPENDIX 2: VPLOT DOCUMENTATION

This appendix contains copies of the on-line documentation that is distributed with Vplot. These manual pages include:

- *Vplot.* A description of the format in which Vplot instructions are stored. Generally users do not need to know about this.
- *Pen.* A description of the device-independent user interface.
- *Vppen.* A special pen filter that is used to
- *Libvplot.* The package of subroutines that a program can call to create Vplot graphics. manipulate Vplot metafiles.
- *Vplottext.* A guide to using text in Vplot.
- *Vplotraster.* A guide to using raster in Vplot.

Several devices are complicated enough to warrant their own Pen filter manual pages. These are not included here. Also the *Vplothacker* manual page which discusses the internal structure of Vplot and other programming issues has not been included.

NAME

vplot - definition of the vplot graphics metalanguage

INTRODUCTION

Vplot is a graphical meta-language that is interpreted by the **pen** filters to produce plotted images in a device-independent way on many different graphics terminals and hard copy devices. New devices can readily be supported. Unfortunately, **vplot** has only a primitive interactive capability. The reason for writing your plots in a device-independent intermediate plot language such as **vplot** is to make your plot programs portable and long lasting.

Vplot originally stood for 'vector plot', but this name is now insufficient because vplot supports not only vectors, but also filled areas (either raster patterns or hatched), text (Hershey fonts), and raster images (including grey-scale dithered ones on monochrome devices such as laser printers).

COPYRIGHT

Vplot is not in the public domain, although the copyright is not very restrictive. Here is the official Vplot copyright notice (the one that all the other manual pages and the source code tell you about):

Copyright 1987 the Board of Trustees of the Leland Stanford Junior University. All Rights Reserved. Permission is hereby given to use, copy, modify, and distribute this software provided that (1) copyright and proprietary notices are retained in each copy, (2) any files which are modified are identified as such, and (3) you do not copy or distribute the software for payment or for commercial use without prior written consent from Stanford. STANFORD MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND concerning this software or its use.

Vplot consists of everything in this distribution, except for the contents of ".../vplot/Hershey" and ".../vplot/filters/loclib". These are in the public domain. The contents of ".../vplot/Hershey_Fonts", which may or may not be included in your copy of the distribution, are under their own restrictions contained in the file README_Hershey.

You are invited to make improvements to Vplot, support new devices, fix bugs, use Vplot as a spare parts junkyard, etc, as long as you abide by the copyright given above. Older public domain versions of Vplot exist, although not as complete and as bug-free as this "official" version. Before embarking on any major project, you might want to ask me (Joe Dellinger) for advice as to how to proceed. Other than that, you're on your own. When all else fails, read the manual! (And when that fails, read the source code.)

You can get a tape of this distribution at cost by writing:

Office of Technology Licensing
Stanford University
350 Cambridge Avenue, Suite 250
Palo Alto, CA 94306

ASSOCIATED PROGRAMS

Self-documenting Utilities: pldb plas

For info on pen programs: man pen
For info on libvplot: man libvplot
For info on text under vplot: man vplottext
For info on raster under vplot: man vplotraster
Library of test routines and examples: .../vplot/filters/Tests/*

COORDINATE SYSTEMS

Vplot fits the largest 3-high by 4-wide rectangle (or whatever ratio is defined by SCREENRATIO in vplot.h) onto the device's screen that it can. The rectangle is left and bottom justified. Clipping is not done to the edge of this rectangle, but to the edge of the device's screen. For convenience, from here on in the documentation I will consider the 3 by 4 rectangle inscribed inside the device's screen to BE the device's screen for the purposes of describing Vplot's coordinate system.

Vplot coordinates can either be measured in integer vplot units (the actual integer numbers that go into the metafile) or as real inches. How many integer units there are per inch for various sorts of Vplot primitives is defined in vplot.h. RPERIN is the number of integer vplot units per inch for most things. Unfortunately, in order to be backwards compatible we have to measure some primitives in different units: TXPERIN gives the number of text-height units per inch, and FATPERIN gives the number of line-fatness-width units per inch. (HATCHPERIN is only used by libvplot, not vplot itself.) At Stanford, RPERIN=600, FATPERIN=200, and TXPERIN=33. These are not particularly convenient numbers, and new sites installing Vplot from scratch might consider selecting more reasonable values for all of these. (In theory, all you should have to do is change the definitions in vplot.h, update the documentation, and recompile everything in sight.) Whenever any of these "magic numbers" are referred to in code, they should be "#define"d by vplot.h.

Two different scaling systems are possible, "relative" and "absolute". The so-called "real inches" in Vplot can either correspond to absolute inches on your device's screen (size=absolute), or can scale with the size of the screen (size=relative). How many inches tall the screen is considered to be unfortunately depends on the "plot style", which is defined next.

Two different coordinate systems are possible, "standard" and "rotated". (This is called the "plot style".) The plot style can be set with the vplot 'S' command, set from the command line, set from the environment, or allowed to default, with that order of precedence. Rotated style is mostly supported for backwards-compatibility reasons; new programs should avoid using it whenever possible.

Standard Style: The origin is in the lower left hand corner, with the X axis horizontal and the Y axis vertical. If size=relative, the plot is scaled so that the screen is considered to be 10.24 (STANDARD_HEIGHT in vplot.h) inches tall. That is, if you move to the Vplot coordinate (in inches) of (x=0.,y=10.24), you will be at the top left corner of the screen no matter how big or small your device's screen actually is.

Rotated Style: Plot is scaled so that the screen is considered to be 7.5 inches tall. The origin is in the upper left hand corner, with the X axis vertical running down the screen and the Y axis horizontal. If size=relative, the plot is scaled so that the screen is considered to be 7.5 (ROTATED_HEIGHT in vplot.h) inches tall. That is, if you move to the Vplot coordinate (in inches) of (x=7.5,y=0.) you will be at the bottom left corner of the screen no matter how big or small your device's screen actually is.

DESCRIPTION

The following list constitutes the plot commands for the various plotting programs. The commands are universal in the sense that they are properly read in and appropriately used or ignored. The commands are specified by a literal one byte command character followed by zero or more half-word (2 byte) integers (sometimes characters or bytes if so stated in the text).

The program "pldb" converts the binary Vplot format described here into a human-readable ASCII one. Pldb-format Vplot follows the documentation here closely. Commas show where newlines occur in 'pldb' format vplot that are not shown as newlines here.

A final message: official vplot dogma says that all programs should only WRITE vplot by calling libvplot. They should NOT actually write out the bytes described here directly. Official vplot dogma also says that only the subroutine "dovplot" should READ vplot. Plas and pldb violate both of these rules!

COMMANDS

e **erase** the screen and start a new plot. All plot frames begin with an erase. (The "erase" pen filter option can be used to add an erase as the very first thing in an input plot file, but will only do so if the plot file doesn't already have an initial erase. Erases that occur near, but not at, the beginning of the plot file are NOT considered initial erases, even if the commands preceding the erase produce no actual output on the device.)

b **break,** same as erase in that it is a possible place to pause, but doesn't erase the screen. The erase command basically has 2 functions: first, it signals that the current frame of the plot is completed, and secondly it erases the current frame of the plot so that the next frame can start fresh. The break command only carries the first meaning. The "next frame" of the plot builds upon the previous ones. This is useful when you want to be able to stop at various times as an image is being built up.

o x y redefine **origin** (0,0) on the device (the lower left hand corner of the screen for standard plot style with no global translations via pen command line options) as vplot (x,y). This shift applies to everything until the next origin command. An origin command is not affected by previous ones. Changing the plot style resets the origin to zero.

m x y **move** the current position to (x,y).

d x y **draw** a line from the current position to (x,y), and make this the new current position.

w xmin ymin xmax ymax
set clipping **window. (Note that this only changes** It does not change the vplot-to-device transformation at all!) The initial clipping window is set to the maximum dimensions of the particular plot device. If the coordinate system is rotated, the smallest clipping window with vertical and horizontal sides that contains the desired rotated clipping window will be used. If xmin > xmax or ymin > ymax, then an inverted window will be used that should clip away everything.

T size orientation, string
Start **text** mode. A character **string** is plotted starting at the old pen position (as set by a previous move, draw, or text command). A nullbyte terminates the string. The character **size** (the height) is in units of 1/33'rds of an inch (Ug!). The character width is font and character dependent. The character **orientation** is in degrees, and the angle is measured counter-clockwise from the X-axis. The carriage controls newline, carriage return, and backspace are correctly interpreted. The origin for newline and carriage return is taken as the pen position as of the last move.

A large number of special escape sequences for accessing multiple fonts, colors, sizes, fatnesses, symbols, and positions of text are available. See man vplottext.

The archaic text command **t** has been left in the code only for backwards compatibility. It should no longer be used, so I'm not going to even tell you the ridiculous format it used.

G xpath ypath xup yup, string
An alternate text command, which uses **GKS** notation. The vector given by **xpath** and **ypath** is the "text path vector", and the vector given by **xup** and **yup** is the "text up vector", as defined in the GKS standard. Since it is important for these vectors to be very accurate, both are measured in units TEXTVECSCALE (defined in params.h, set to 10 at Stanford) times finer than normal vplot units.

f k Set line **fatness.** The integer **k** specifies the number of additional lines to be drawn parallel to, and one bit over from all future lines. The extra lines are added symmetrically. k=0 resets back to single-line mode. The fatness is scaled to approximate the behavior of a 200 pixel per inch device.

c k Select new **color** from menu of 512 colors. **K** is an integer from 0 to 511. Colors 0 through 7 are guaranteed to be by default the Regis colors: 0-background, 1-blue, 2-red, 3-magenta, 4-green, 5-cyan, 6-yellow, and 7-white. (This is the order of colors in a color bar pattern on a TV. It is used for that because on a black and white TV it makes a grey scale.)

Color zero is always the background color. Monochrome devices should consider ANY non-zero color to NOT be the background

color.

See important comments in the 'set color table' command with regards to color mapping.

C color red green blue

set **Color** table: Color number **color** is set to have red value **red,** green value **green,** and blue value **blue.** These are integers from 0 to 255, (or floats from 0. to 1. when using plas and pldb in inch or centimeter units) with 255 being fully on and 0 being fully off. The corresponding 'grey level' for grey-scale devices is given by floor((4 * green + 2 * red + blue + 6) / 7). Note that the only way to get grey level 0 is if all three color levels are 0.

Vplot will try to give you the color you want, but this may not always be possible. If the device has N settable colors, then colors 0 through N-1 will be set exactly as you request. If you attempt to define a color X outside of this range, Vplot will pick which color in the range 0 through N-1 is the closest and map all requests for color number X to that available color value. Whenever any color in the range 0 through N-1 is changed, all the mappings will be re-calculated (although of course what is already drawn or stored cannot be changed.)

This scheme is especially useful when for example mixing grey scales for raster information and bright colors for labels. The few bright colors which you might need are allocated to the first few color numbers. Then the remainder of the lower half of the color scale is filled with the grey level values you will need, but ordered so that the most important grey levels come first. In the upper half of the color table you define a normally ordered grey scale. Now when you want to draw grey scales, you refer only to the upper half of the grey scale and vplot will automatically map your requests down onto the real available colors of the particular device.

Color zero is always the background color. Color 0 is only mapped to if the match is exact.

a npts

afat xmask ymask

x1 y1, x2 y2, ...

Define polygonal **area** for stippling (on monochrome devices) or solid area fill with current drawing color (for devices with color). The polygon is specified by its vertices **(x1,y1),...,(xn,yn). Npts** is the number of vertices; the polygon is automatically closed only if necessary. **Afat** is the fatness of the outline; 0 is the minimum, and -1 draws no outline. The shading for monochrome devices is determined by the integer mask. A point within the polygonal area is darkened if ((x mod **xmask** == 0) and (y mod **ymask** == 0)) is true, where x and y are all points within the polygon. For example, xmask=1 and ymask=1 fills the area completely, and xmask=1 ymask=5 makes horizontal bars spaced 4 vertical bits apart. The origin for x and y coordinates used in shading is

device dependent but will usually be the lower left hand corner of the device screen.

As a special case, if either **xmask** or **ymask** is zero no shading is done. (The area filling routine for the device is never even called.) This is true for both color and monochrome devices.

A npts

x1 y1, x2 y2, ...

Fill polygonal **area** with pattern or color. Fills polygon with patterns loaded with the 'l' command (q.v.). The pattern number used is the same as the current drawing color. Note that a pattern can either be a hatch-line pattern or an arbitrary raster pattern, depending on the format of the 'l' command used.

The purpose of this scheme is to associate each color with a particular fill pattern. Once a current drawing color is set, all vectors will have that color and all filled areas will have the associated fill pattern.

Devices that can't fill with arbitrary patterns should simply fill solidly with the current drawing color. If no pattern has been loaded, the 'A' command will fill solidly with the current drawing color automatically.

l ppi nx ny lpat

XXXX..., ... (nx*ny X's)

Load raster pattern for 'A' command. Each **'X'** is an integer containing a color number. (These color numbers can be mapped, etc, just like color numbers in the 'c' command.) The **X's** represent pixels in a rectangular pattern of dimensions nx by ny. (In pldb format the **X's** are single-digit Hex numbers.) The pattern is repeated to fill the polygon. The pattern is scanned so that the first ny numbers are the first line, the second ny numbers are the second line, etc, for nx lines. The scanning of the pattern is done TV-style, ie starting at the upper left and working left-to-right and then top-to-bottom. (If in doubt, remember that patterns should appear on the device in the same orientation as in pldb-format vplot files.)

The pattern is given the pattern number **lpat.** This pattern will be used for filling by the 'A' command when the current color is number **lpat. Ppi** is used to scale the pattern; it says how many pixels per inch you wanted this pattern to have. E.g. on a 300 pixel per inch device

l 300 2 2 17
0 1
2 0

represents

01
20

whereas

l 150 2 2 17
0 1

2 0
represents
0011
0011
2200
2200
where the digits are color numbers for the corresponding pixels. The result when you load a new pattern with the same pattern number as one already defined is device-dependent.

l angle -1 numhatch ipat

fat color offset interval,

...,

(numhatch * 2 sets of 4)
Load hatch pattern for 'A' command. The **angle** gives the amount to rotate off horizontal. The **'-1'** is a flag to distinguish this case from the previous command. **Numhatch** is the number of sets of hatched lines to place in each direction (X and Y). Each set of hatched lines has four parameters: **fat, color, offset,** and **interval.** The **fat** and **color** give the fatness and color of every line in the set. The **interval** is the spacing (in regular vplot units) between the lines. The **offset** parameter displaces all the lines in the set. It should be less than the repeat interval. The purpose of **offset** is so that you can have two different sets of lines (differently colored, perhaps) interleaved. All the X sets are specified, and then all the Y sets.

Ipat is the pattern number. (See above for explanation.)

v flag Change **overlay** mode.
Overlay mode controls the opacity of the background color of polygon and raster fills (see 'a','A','l','r','R'). If flag=0, the polygon area is cleared before the new pattern is filled in. If flag=1, the new pattern is simply added to whatever is already there with what is underneath showing through where the new pattern is color 0. The default overlay flag can also be set in the command line. flag=0 mode takes significantly longer to fill the polygon on many devices. Some devices that have hardware raster or area fill may ignore this option if they are unable to implement it.

n **no-op** for word aligning. Does nothing.

p **purge** or flush plot buffers.

z, string
Print **string** out on the text screen of the plot device.

L npts

x1 y1, x2 y2, ...
Draw a **polyLine** through the given points.

M npts mtype msize

x1 y1, x2 y2, ...
Draw **Markers** of type **mtype** and size **msize** at the given set of points. For small integer values of **msize,** this is as defined in GKS. For larger values, it is taken to be a font glyph number (for most fonts this will correspond to the ASCII character number. However **mtype** may still be valid even when greater than 255 for some fonts.) **msize** is measured in the same units as text is.

s ndash

dash1 gap1, dash2 gap2, ...
Set line **style.** This command is used to make dashed or dotted lines. It affects all vectors produced by the 'm', 'd', and 'L' commands. **Ndash** is the number of dash-gap pairs in the pattern. If **ndash** is 0, then continuous vectors will be drawn.

J hjust vjust
Set text **Justification.** Values have meanings set in the include file '/usr/include/vplot.h'. The finer points of text justification are discussed in 'man vplottext'.

F font precision overlay_mode
Set text **Font** and **precision. Font** is an integer from 0 on up. Font numbers less than 100 are reserved for device-independent 'gentext' fonts. **Precision** is an integer zero through two, meaning: string, 0; char, 1; stroke, 2. The generic vplot text routine gentext draws everything with stroke precision, but only parses for ligatures at precision 2. The standard 'pen' font should be font number 0. How many others are available is site-dependent, and should be documented in 'man vplottext'.

The overlay mode is an integer from 0 through 3, with the following meanings: 0: nothing unusual. 1: Draw a box around the text. 2: Clear behind the text with a polygon drawn in the background color before drawing the text. 3: both 1 and 2.

All fonts, precisions, and text overlay modes are''#define''d in vplot.h.

S styleflag
Set Vplot **Style.**
Style is a single character, with possible values 'r' rotated, 's' standard, 'a' absolute. Case distinctions don't matter. These affect the scaling and orientation of the plot. ''Standard'' and ''rotated'' styles are defined in the ''Coordinates'' section at the beginning of this document. Absolute style is like standard style, except the plot is scaled to be in true inches. Some other archaic styles are recognized (old, mazama) but should no longer be used. The style is reset to the default at the start of every plot frame. It is probably a good idea to set the style immediately after every **erase** command, especially if the style you want to use is not STANDARD.

R orient offset

xll yll, xur yur

xpix ypix

{linerep1, {numpat1 numbyte1, b1, b2, ...}}

Byte deep **Raster** data, each point having **offset** added to it as it is read in. It is **xpix** pixels wide and **ypix** pixels tall, with lower left hand pixel at **(xll, yll)** and width **xur-xll** and height **yur-yll. Linerep** tells how many times to repeat the following raster line. **Numpat** gives the number of repetitions of the following **numbyte** bytes **b1, b2, ... bN** within a line. This is repeated until all the bytes within the raster line have been accounted for. (The last byte of the line must also be the last byte of the repeat pattern.) The same holds for the loop over lines.

The orientation of the axes of the raster is determined by the value of **orient.** For **orient=0** the raster is oriented TV-style, with the fast axis left-to-right and the slow axis top-to-bottom. For each increase of 1 in the value of **orient,** the raster rotates 90 degrees clockwise.

Note that the lower-leftmost pixel of the raster image will be exactly at the vplot coordinate **(xll,yll).** However, the point at vplot coordinate **(xur,yur)** will be the pixel just outside the raster image, touching the upper-rightmost corner. Things were done this way so that **xur-xll** for example gives the width of the raster data box in vplot units.

The **offset** parameter is provided so that there is some way to access color table values greater than 255. (For example, 4 different 64-value color schemes could be defined in the color table in slots 256-511. You could switch between these by merely changing the value of **'offset'.**)

r orient rastermult

xll yll, xur yur

xpix ypix

{linerep1, {numpat1 numbit1, b1, b2, ...}}

Same format as the 'R' command, except that the data is encoded in bits and **rastermult** is multiplied by the value of each bit. 8 bits are encoded into each byte, in each byte the bits ordered from highest to lowest. As many bytes are used as are needed to hold **numbit** bits. For example, 13 bits would be encoded as **b1: (1 2 3 4 5 6 7 8) b2: (9 10 11 12 13 X X X).**

[, string

Begin group named **string.**

]

End last-opened group (it is an error to use this command if no group is open). Groups may be nested. Groups may not contain an erase command. Groups may not extend across files.

SEE ALSO

pen vplotlib vplottext vplotraster vplothacker

BUGS

A non-technical "tutorial and overview" manual page is desperately needed, but I don't have time to write one.

There may be no bugs in the vplot language, but there are in the programs which read and write it.

I had a heck of a time with troff bugs eating words and characters while trying to write this, so if something seems to be missing you might refer back to the unformatted original.

"Dovplot", the subroutine which reads the vplot language, should be broken down into subroutines. There should be one subroutine to process each vplot command, and one subroutine to read the vplot metafile. These subroutines could then be hooked into programs that want to have the option of reading and writing vplot, without actually having to be a vplot filter.

Vplot at Stanford suffers somewhat by the restriction of having to remain backwards compatible to the stone age (1980). The vplot coordinate system is integers, with 600 pixels to the inch. A much higher resolution should be used! This isn't too hard to change, all you have to do is to make plas and pldb write and read 4 bytes instead of 2. (Plas and pldb already work with integers anyway, and there are no short ints used in the code anywhere.) Several other 'magic numbers' show up in this documentation as well. These are defined in 'vplot.h' and 'params.h'. I recommend that any new site starting to use vplot from scratch change these numbers to more reasonable values. If ALL code uses the #define's in vplot.h, as all Vplot source code (hopefully) does and all user code ought, then these values can be changed without breaking any programs, only old metafiles.

AUTHOR

Robert W. Clayton began the vplot software while in the Geophysics Department at Stanford University (circa 1979) on a PDP-11. He was last seen at Cal Tech. Since then Jeff Thorson, Stew Levin, Jon Claerbout, Dave Hale, Ron Ullmann, Glenn Kroeger, Michel Debiche, Shuki Ronen, Doug Wilson, Wes Monroe, Chuck Karish, Steve Cole, Clement Kostov, Kai Lanz, Jean-Luc Guiziou, Dave Nichols, and especially Joe Dellinger have altered and expanded it almost beyond recognition, for the most part preserving backwards compatibility, even when that required compromising of ideals. This manual page was hacked together by Joe Dellinger out of an ancient and out-of-date original by Michel Debiche, which was itself a hack of something even more ancient of unknown origin.

Joe Dellinger,
ARPA: joe@hanauma.stanford.edu
UUCP: apple!hanauma!joe
Stanford Exploration Project
Dept of Geophysics, Stanford University, Jan 1988

NAME

pen - VPLOT graphics output filters for all devices

SYNOPSIS

pen plot_file1 [plot_file2 ...] [options]

Seplib version: Pen < Plot1.h [Plot2.h ...] [options]

DESCRIPTION

Pen filters accept input in the **vplot(9)** graphical metalanguage and translate it into code for any of about 20 graphics output devices including graphics terminals, laser printers, electrostatic plotters, and dot-matrix line printers. Interactive input (mouse, cursor keys, keyboard) is not yet supported.

The output of a **pen** filter can be sent to a plot device or saved as a file of device-language commands. If the output is not redirected it is understood that you want to plot on the standard output device.

There are now many different **pen** filters for different devices. These programs differ only in the device-dependent output routines. They share a common **vplot** interpreter, and most filters make use of common intermediate-level code. This manual entry describes those options that are implemented in the device independent part and thus are common to all pen filters.

DEVICES SUPPORTED

Well supported devices: Imagen, Tektronix 410x, Envision, DEC Regis, Rastertech, Printronix.

Partially supported devices: PostScript, Tektronix 401x, XWindows, SunView, Sun NeWS, Masscomp (GPS), HPGL.

Virtual devices: raspen (generic raster device), vppen (generic vplot device)

OPTIONS

Pen has many options. Though many of these are included mostly for backward compatibility with older versions of **vplot**, they also add quite a lot of flexibility to the system.

Those options for which the default is device- or host-dependent are marked with an 'X', since the default varies from machine to machine or device to device.

All options have the form **a=b**, where **'a'** is the name of the option being set and **'b'** is the value it is being set to. Alternatively, commonly used sets of options may be listed in a parameter file and **par=parameter_file_name** can be put as an option on the command line. The last occurrence of any given option on the command line is the one that is used (see also **man getpar.**) The form a= has the effect of causing the option **a** to be unset.

in=string (numvplot=0 in1=string1 in2=string2 ...)

String is an input file containing vplot-format plot commands. The **'in='** is optional; any text on the command line which doesn't contain an '=' is assumed to be an input file. The standard input is also read. It is valid to have inputs in all three of

these forms at the same time; they will be read in order. This is not quite the same as concatenating the files and then processing the result, as the clipping window, fatness, current color number, text justification mode, text font, text precision, text overlay mode, and plot style get reset to the default at the start of each new input file (and at the start of each frame within a file). (Non-sep filter versions only: **numvplot=0** controls how many extra input file specifications of the form **inX=stringX**, where X runs from 1 to numvplot, will be searched for.)

wstype=

Some pen filters may support several related, but not identical, devices. This variable allows you to tell the pen filter the exact device type that you are using. The work station type variable can also be set via the environment variable WSTYPE. The command line takes precedence. Normally the presence of any options on the command line inhibits self-documentation. The **wstype** option, if alone on the command line, will not.

hshift=0, vshift=0

This is an amount to translate the plot, in inches, in the horizontal and vertical directions, respectively. This is independent of all other scaling and origin setting. These are useful when including graphics into a paper via TeX. The program **vppen** (a special sort of pen filter) can be used to calculate an appropriate hshift and vshift.

xcenter= ycenter=

If specified, the coordinate given (in inches) is forced to be in the middle of your screen. In combination with the **interact** option, this allows you to zoom in on an interesting part of your plot without having it zoom off the screen. This is independent of everything except translation (hshift and vshift).

pause=0

Number of seconds to wait in between plots. -1 means to prompt and stop and wait for a carriage return before continuing.

fat=0 Base line thickness. Add this fatness to all vectors drawn. Fatnesses are measured in units of 1/200'th of an inch. A fatness of 1 or 2 may very well have no effect on a low resolution device, but it will on a high resolution one. **Fat** scales as does fatmult.

fatmult=1.

Fatness multiplication factor. This is mainly useful when you want to turn off fatness for speed of plotting. This option can also be set via the environment variable FATMULT.

patternmult=1.

Pattern multplication factor. This multiplies the size of patterns inside polygons, both bit patterns and hatch patterns. Like FATMULT, can be set via an environmental variable PATTERNMULT. If you set "patternmult=0' " then polygons will only be filled solidly or not at all.

scale=1. xscale=1. yscale=1. txscale=1. mkscale=1. dashscale=1.
Amounts by which to stretch the plot. **Scale** stretches both axes equally, **xscale** X only, and **yscale** Y only. Again, this is vplot's X and Y, not necessarily the screen's if the plot has been rotated. Fatnesses, patterns, dashed line pattern lengths, and marker sizes do NOT scale geometrically along with the rest of the plot. Patterns and dashed line lengths do not scale at all except by their own special scaling options given below. Markers, text fatnesses, and vector fatnesses scale with screen size and with the **scale** option, but not with **xscale** and **yscale.** Special scaling options that control specific things are **txscale,** which scales only text size; **mkscale,** which scales only markers; **dashscale,** which scales only dashed line patterns; **fatmult,** which scales all fatnesses; and **patternmult,** which scales only patterns.

shade=y overlay=n
These options can be used to control shading of polygons. If shading is turned off, the interior is not filled. ('A' style polygons normally do not have any outline. If **shade=n,** one will be drawn so that you can still see where the polygon is.) For devices that support polygons filled with patterns, the overlay option sets the default overlay mode: n = replace, y = overlay. This can be reset during plotting by the **overlay** command in vplot. The difference between these two options is whether color zero is treated as transparent or not. This option also applies to raster. Not all devices will support all overlay options. On some devices that do support both overlay options (in particular those that simulate raster in software) **overlay=y** may be much faster than the default.

style=standard
This option sets the default coordinate system. (It is overridden if the coordinate system for the frame is set by the VP_SETSTYLE command in the vplot file.) **style=rotated** sets the origin in the upper left corner of the plot with y horizontal and x positive downwards. **style=standard** sets the origin in the lower left corner of the plot with y vertical and x horizontal. **style=absolute** is the same as **standard** but with 'size=absolute' implied as well. Other possibilities are 'old' and 'mazama', both obsolete standards. This option can also be set using the environment variable PLOTSTYLE.

txfont=X txprec=X txovly=0 font?= txsquare=n
Default text font, text precision, and text overlay mode. Alternate font file. If **txsquare=y,** then xscale and yscale will not be allowed to distort text. These are described in more detail in vplottext(9).

window=y frame=n xwmin= xwmax= ywmin= ywmax=
Window=n turns off all clipping windows (except that of the edge of the screen). This is useful if you have accidentally clipped your plot away to nothing, and can't figure out what's going wrong. **Frame=y** shows you where your windows are, by outlining them in white. (The first window should always simply

frame the screen, but often one side or another is clipped by the device because the device's coordinates are off by one.) **xwmin, xwmax, ywmin, ywmax** allow you to specify a global clipping window in addition to any specified in the Vplot. The window's position is given in vplot units of inches. (The 4 corners of the window specified will be translated into device coordinates, and the smallest rectangle with sides perpendicular to the edge of the screen containing these 4 points will be used as a global clipping window.) Together with concatenation of vplot files and hshift and vshift, this gives a primitive sort of cut and paste capability.

rotate=0
Global rotation of entire plot by this many degrees clockwise, about the origin. Vshifts and/or hshifts may be needed to keep the plot from rotating off the screen. Since clipping windows must be standard up-down left-right rectangles, they cannot rotate correctly except to multiples of 90 degrees. The smallest clipping window that is possible that contains the desired clipping window will be used in this case.

erase=y break=n
Erase=y forces an erase at the start of each input file, and follows the vplot literally thereafter. **Erase=literal** does not force an erase at the start. **Erase=n** ignores ALL erases. **Erase=once** forces one erase at the start of plotting and thereafter ignores all erases. These options are especially useful when combining several vplot files. "Forcing an erase at the start" creates an erase as the very first thing in the input plot file, but only if the plot file didn't already start with one. **Break=n** treats the Vplot 'break' command as a chance to pause, but not to erase. **Break=y** makes 'break' and 'erase' synonymous.

size=X **size=relative** scales the plot workspace, a 10.24 inch high by 13.65 inch wide rectangle, to fit the size of the screen. **size=absolute** scales to real inches. Hardcopy devices with large or semi-infinite plotting surfaces should default to **size=absolute, while** screen devices and "page-oriented" hardcopy devices like laserwriters default to **size=relative.**

echo=X Echoing is typically turned off for the duration of plotting for most (but not all) pen filters. **echo=y** forces echoing to be left on, and **echo=n** forces it to be turned off in case the default picked by the pen filter is inappropriate.

endpause=X
Some filters by default pause at the end of plotting (usually with a beep) and wait for a carriage return. This is so you can see the last frame of the plot before the text is turned back on, obliterating it. For some devices this is not necessary, and there is no default pause at the end. Should the default not be appropriate (for example when plotting from within a shell repeatedly), **endpause=y** or **endpause=n** can be used to force this 'end pause' on or off.

mono=n

If **mono=y,** the plot will come out in black and white, possibly speeding up the plotting and allowing the use of such monochrome features as dithering and halftoning on color screen devices.

dither=X

For **mono=y** (monochrome devices), dithering is a means of representing a continuous-tone grey image. See the manual page for vplot raster for a discussion of dithering methods. If **dither=0,** no special treatment will be given to raster colors. (This means that all the points in the raster image will be shaded except those with color 0.) Available dithering methods are:

1 Random Dither
2 Ordered Dither
3 Minimized Average Error Method
4 Digital Halftoning Method

Devices that do their own dithering may not support all of these methods.

greyc=1.

This parameter applies only when using dithering (described above) to plot grey rasters on a monochrome device. It has been observed that grey scale reproduction on hardcopy devices is quite different than that on graphics displays. Briefly, the transition from black to white occurs more rapidly on a display device, leaving both ends of the grey scale clipped at black or white. This nonlinearity in the perceived grey scale is a useful feature that can be simulated on a hardcopy device by use of the **greyc** parameter. Greyc values less than 1. alter the grey scale to simulate this nonlinearity. The smaller the greyc value, the more rapidly the transition from black to white occurs. See the vplot raster manual page for a more complete discussion of this parameter and how to use it.

pixc=1. This parameter also applies only when using dithering to plot grey rasters on monochrome devices. It has been noted that grey rasters come out significantly darker than expected on a hardcopy device than when they are displayed on a graphics screen. This parameter applies a correction based on the assumption that this darkening is due to the overlap of pixels on the hardcopy device. Pixc values less than 1. shift the grey color scale toward white to compensate for the darkening due to pixel overlap. See the vplot raster manual page for a more complete discussion of this parameter and how to use it.

invras=y

Color 0 is the background color on all devices. For monochrome paper plotters, this is usually WHITE, the color of the paper. On these devices color 7 (WHITE) comes out as BLACK since it has to be the opposite of color 0. If **invras=y,** this convention is reversed for the purposes of dithered raster, so that color 0 is

black and color 7 is white despite the fact that this color scheme is the reverse of that used for vectors on the same plot. This option has no effect on raster that is not dithered.

wantras=y

If **wantras=n,** all raster data will be displayed as a solid white block. This shows where the raster data is, but on most devices will plot significantly faster. This allows you to get the general layout of your plot right without having to wait hours for the raster data to plot on your screen.

interact=

If **interact=file_name** is specified, at the end of each plot the cursor will be turned on and you may pick points around the screen. The Vplot coordinates of these points will be stored into the file **file_name.** Picking the far upper-right hand corner of the screen will end the picking. (The may also be device-dependent ways to end the picking, such as hitting Escape or q.) Many of the pen options are specified in Vplot coordinates, in inches. This option allows you to find the coordinates of specific points of interest. It is a primitive sort of interaction, but useful.

selfdoc=n

If all else fails, **selfdoc=y** will force pen to self document no matter what it thinks it should do. (This applies to the non-SEP version only.)

signal= If the **signal** option is set to anything, then no signal catching will be done. Useful for debugging.

colormask=y,y,y,y,y

The **colormask** option is used to "turn off" certain color planes in the device. This option is useful if you wish to submit color figures to printers, who often require the plot to be broken up into 4 different "color planes". It is somewhat limited. It only affects colors set by the user, so if you want it to affect the default colors 0 through 7 you'll have to set these yourself. (You can easily create a file to do this using plas. Prepend this to your plot file.) Each of the first 4 "y/n" toggles controls a color plane. "Masked off" color planes are set to the value of the corresponding plane of the current background color, color 0, which is always left untouched. (So make sure to set color 0 first thing!) The 4 color planes will be red, green, blue, and white if the background color is black; and cyan, magenta, yellow, and black if the background color is white. The 5th "y/n" toggle controls whether the planes are plotted in color. (This option should not be used if more than one plane is on (in which case it is undefined) or if the 4th plane is on (in which case it is unnecessary).)

SEE ALSO

Vppen, Ipen, Vplot, Libvplot, Vplottext, Vplotraster, Vplothacker

AUTHOR

The original filter was **pen,**

written by Rob Clayton, who also invented **vplot.** Originally it was only possible to do vectors and that on only one device. Vplot has since been expanded beyond recognition, and is supported on about 20 devices. Michel Debiche, Glenn Kroeger, Chuck Karish, Steve Cole, and especially Joe Dellinger have all worked on Vplot at one time or another.

This documentation was written by Joe Dellinger.

COPYRIGHT

Please read the official Vplot copyright notice, which is contained in the Vplot manual page.

BUGS

There are a few "magic numbers" referred to in this documentation. These are defined in "vplot.h", and are subject to change.

Vplot was never intended to be used in an interactive way, but does OK for hard-copy purposes. It would be nice if someone out there took Vplot apart and rebuilt it again in an interactive framework, using a standardized meta-language. (We did not use a standard meta-language because none existed at the time Vplot was started, circa late 70's. Since then Vplot has slowly grown, for the most part staying backwards compatible.)

Pen has a huge number of options supporting backwards compatibility, and would need twice as many to make absolutely everyone happy. Most of the completely arcane ones have finally died a merciful death.

Not all devices support all the primitives, but at least the software capability exists. Several generic software routines inside 'pen' should be redone when possible, as I was just learning C when I wrote them. They seem to work reliably, however. We use vplot a lot and haven't seen bugs in the device-independent code in a long time now. Bugs in device-dependent code is another story. (There is a tendency for people to only support a device to the extent that it does what they themselves need, which creates problems when someone else comes along and tries to do something else with it.)

Why are we still using inches? Everything should be changed to centimeters! (Except that, strangely enough, every device we have here has a screen resolution that is defined by the manufacturer in terms of pixels per INCH! So how serious is the US on standardizing with the rest of the world anyway?)

Bona-fide bugs that I know about but haven't bothered to fix are listed in the file 'known_bugs'. There are also such files in device subdirectories where appropriate.

NAME

vppen - VPLOT filter for VPLOT

SYNOPSIS

vppen plot_file1 [plot_file2 ...] [options] > vplot_out

Seplib version: Vppen < Plot1.h [Plot2.h ...] [options] > Vplot_out.h

DESCRIPTION

All **pen** filters accept input in the **vplot** graphical metalanguage. **Vppen** is the filter for the "virtual vplot device". Both its input and output are vplot. Why is this useful? The full set of pen options allows you to rotate, combine, clip, scale, shift, etc, plots. At some point you may make something on your screen using these options which you would really like to have as a single file. How to do this? Simply use the same set of options with vppen. Vppen will draw the same picture, but will save the picture as a vplot file. You can then rotate, combine, clip, etc this new plot.

Vppen also has options to do useful things such as centering, making arrays of plots, and finding plot statistics.

VIRTUAL DEVICE DESCRIPTION

The virtual vplot device has a screen 10.24 inches (STANDARD_HEIGHT in <vplot.h>) high, with a height/width ratio of 3/4 (SCREEN_RATIO in <vplot.h>). It has 600 pixels to the inch (RPERIN), and has 256 colors, all settable. It knows how to do its own clipping and has "hardware text", etc. This is the device that vppen "plots" on.

Ideally vplot coming into vppen with no options set should be passed on through completely unchanged. This is true (except for a few exceptions like polygon fill patterns and such) if both the input and output are absolute style. (If you don't know what terms like "absolute style" mean, read the other vplot documentation.) For this reason, vppen by default uses "size=absolute" and outputs absolute style output. This is convenient if you are repeatedly sending files through vppen, but you may want to override it otherwise.

GENERIC OPTIONS

For a list of all generic pen options, which also apply to vppen, do "man pen". Generic options that you may find particularly useful are:

erase The erase=once option is useful when combining several plots into one. (Otherwise you're likely to have the entire screen cleared before each plot.)

xshift, yshift
These options are useful for positioning a plot.

scale The scale options are useful in getting a plot the right size.

xcenter, ycenter, interact
When you're going crazy trying to get something centered in an oddly shaped screen made by "gridnum", this comes in very handy.

xwmin, xwmax, ywmin, ywmax
Useful for clipping out just the part of the plot you want.

rotate Useful for rotating a plot, usually by 90 degrees.

Remember that ALL generic pen options apply, not just the ones listed above. Only vppen-specific options will be covered in the next section.

VPPEN-SPECIFIC OPTIONS

gridnum=0 gridsize= grid=-1
These commands are provided as a quick way to make a grid of plots. Gridnum=X,Y divides the output screen into X rectangles horizontally and Y rectangles vertically. If Y is not specified, then it is set equal to X. If X is zero, then gridding is not done. Gridsize=X,Y sets the X and Y dimensions of each rectangle in inches. If the gridsize is not set, then the array of rectangles will be sized to fill the screen. (The "screen" has an X of 13.65 inches and a Y of 10.24 inches.) Grid sets the fatness of a white border drawn around each rectangle. If it is negative, no border will be drawn. Each rectangle is its own miniature device screen. Clipping is set to the edges of this screen just as it is for the full-sized screen. All generic commands such as rotate, xshift, etc, will apply to each rectangle and NOT to the plot as a whole. Each rectangle contains one (or zero) plot. When an erase command is received, nothing will be erased but instead plotting will begin in the next rectangle, and a "break" command will be output. The rectangles are filled left to right and then top to bottom. It is not an error to have more plots that you have room for, the rectangles will march right off the bottom of the page, but they will still be there. If you have less plots than you have room for some of the rectangles will be empty. Normally the generic pen option "size" defaults to "absolute" in vppen, as it does on most hardcopy devices. This doesn't work very well if your screen is a few inches on a side, so if the gridnum option is used "size" will instead default to the normal screen-device default "relative". For similar reasons, the gridnum option will also change the default options to "big=n" and "vpstyle=n". You may find it useful to use size=relative when plotting a gridded vplot file.

big=y If big=y, the output screen is expanded to the entire range of possible vplot coordinates, -54.6 to +54.6 inches (VP_MAX in <vplot.h>) in each direction. A coordinate shift is thrown in to move the origin back to (0,0) where it belongs. This has some good effects and some bad effects. The chief good effect is that nothing is clipped at the edge of the screen, since the "screen" contains all possible vplot coordinates. (It is still possible for things to be clipped, but only at the edge of the vplot coordinate system.) The chief bad effect is that rotated style objects will be positioned at the top of the expanded screen, which is to say 54 inches away from the origin and right at the edge of the largest coordinates vplot can handle. Thus big=y with rotated style input is an all around bad idea; the solution is to make one pass through vppen with big=n to turn it into non-rotated style. Another bad thing to do is to try to mix big=y with size=relative. You get a very big plot indeed!

stat=n If stat=y, no vplot will be written to standard out. Instead vppen will find the largest and smallest coordinates used in each input file and print out a summary of the height, width, etc, information. Statistics are also kept for all the input plots together as a whole.

align=uu

This option is used to left, right, top, bottom, or center justify a plot. The format is align=xy, where "x" controls the left-right justification and "y" controls the up-down justification. "X" is one of:

l	for left justified
r	for right justified
c	for centered
u	for unaligned, (no shifting done)

and "Y" is one of:

b	for bottom justified
t	for top justified
c	for centered
u	for unaligned, (no shifting done)

The align point is set to have coordinate (0,0). Note that points shifted into negative coordinates are still there, they just may be off the screen. (Use the big=y option to avoid clipping problems.) The "xcenter=0, ycenter=0" option is very handy to use when plotting "aligned" files.

xsize= ysize=

These options allow you to scale your plot to fit within a rectangular area of the desired size. If you specify both xsize and ysize, the size of the area will be xsize inches wide by ysize inches high. Such scaling can change the aspect ratio of the plot. If you specify EITHER xsize or ysize, vppen picks the other so that the original aspect ratio is maintained.

vpstyle=y

Normally vppen inserts a "set style absolute" vplot command at the beginning of every plot frame. If vpstyle=n, it does not do this.

dumb=n

If dumb=y, then all output will be digested down to the bare essentials - color changes, erases, moves, draws, and nothing else.

blast=y bit=0

If blast=n, then raster output will be compacted as much as possible as it is being written. This is slower, but the resulting file may be substantially smaller. If bit=integer > 0, then bit raster will be used, with the integer giving the color number of the "on" pixels.

COMMENTS

Some options (stat, align) will not work with piped input, as they involve making repeated passes through the input vplot files.

Beware letting vppen dump raw binary vplot to your screen!

Vppen outputs an initial erase only if an initial erase is forced from the command line (erase=yes (the default) or erase=once).

Set the text font, precision, etc, that you want on your first pass through vppen, because it will be hardwired after that.

Some vplot commands as yet have no corresponding command in libvplot, and so vppen currently eats these. Provisions are made in the code for the day when these commands will exist; they just have to be uncommented out.

Vppen has four uses. One is as a sort of cheap vplot editor. The second is as a filter for digesting complicated things into the bare essential moves and draws. The third is that it provides a library of routines that can be linked into other vplot filters (perhaps an interactive vplot editor), so that they can easily "dump a hardcopy of the screen". (The code has been carefully written with this in mind.) Lastly, it provides an example of how to do several tricky things: First, how to support a device that can do EVERYTHING in "hardware", and so needs to have the highest possible level of support. Second, how to do multiple passes through the vplot input (again something paving the way for a vplot editor). Third, how to find the length of a text string without actually drawing it.

SEE ALSO

pen, libvplot, vplot

EXAMPLES

On Hanauma, look in the directory /usr/src/generic/tex for a sample showing how this program together with ipen can be used to automatically center Vplot figures included in TeX files.

COPYRIGHT

The Vplot source is copyrighted. Please read the copyright which can be found in the accompanying Vplot manual page.

AUTHOR

Joe Dellinger

BUGS

There are still rotated style plots in common use at SEP, which have problems with the default big=y. There's not much I can do about this one, either, and still be backwards compatible. See the discussion under the "big" option to find out how to get around this problem.

It is not clear how options such as "stat", "align", and "gridnum" behave when used in combination. Absolute sizing can be a pain.

NAME

libvplot - programmer's interface to the vplot - pen graphics system

SYNOPSIS

#include <vplot.h>

Use

{cc,f77} myprog.o -lvplot -lm

to link in the **libvplot** functions.

One of the functions {vp_file(), vp_filep(), or vp_unit()} must be called to initialize the output file.

INTRODUCTION

Libvplot is a set of subroutines which write graphics calls in the **vplot(9)** graphics language. The output file may be an ordinary file or a pipe to a **pen(9)** display program.

The subroutines are callable either from C or from FORTRAN. C function names have an embedded underbar (e.g., 'vp_break()') while the corresponding FORTRAN name will not use the underbar ('CALL VPBREAK()').

Input to **vplot** plot files should be done ONLY through the **libvplot** calls. Programs which write **vplot** language directly are doomed to early obsolescence. Future versions of **vplot** will provide backwards compatibility at the **libvplot** subroutine call level, not necessarily at the **vplot** language level.

Separate functions are available to accept geometric arguments in inches and in arbitrary user units. The routines which accept user units have 'u' prepended to their root names. Absolute coordinates (x,y) in inches are computed internally from user coordinates (xu,yu) in user units via:

x = x0+(xu-xu0)*xscl
y = y0+(yu-yu0)*yscl

This transformation is set by calls to **vp_orig**, which sets x0 and y0; **vp_uorig**, which sets xu0 and yu0; and **vp_scale**, which sets xscl and yscl.

COORDINATES

Vplot fits the largest rectangle with a 3 (high) by 4 (wide) aspect ratio (SCREENRATIO in vplot.h) onto your device screen that it can. This window defines your output space. (Although clipping is still done at the edges of the actual screen.)

Two different coordinate system orientations are available, "standard" and "rotated". In the standard coordinate orientation, the origin is in the lower left hand corner and the X axis is horizontal and the Y axis is vertical. In the rotated coordinate orientation, the origin is in the upper left hand corner and the X axis is vertical and the Y axis is horizontal. The standard coordinate orientation is the default. (See vp_style.) New programs should avoid using rotated style if at all possible!

Two different scaling methods are available, "relative" and "absolute". Which mode is used can be set by the device, options to pen, or vp_style.

In absolute mode inches in vplot correspond to real inches on the device display screen. In relative mode the plot is scaled to the size of the screen. In the standard coordinate system orientation the screen is considered to be 10.24 (STANDARD_HEIGHT) inches tall, and in the rotated coordinate orientation system the screen is considered to be 7.5 (ROTATED_HEIGHT) inches tall.

The routine vp_stretch is a utility routine which calls vp_scale, vp_orig, and vp_uorig for you if you just want to "fill the screen".

For more on coordinate systems and "magic numbers", refer to the Vplot manual page.

FORTRAN

All of these routines can also be called from FORTRAN. The argument types are given for C, but these can be easily converted to FORTRAN by the following table:
int x -- integer x
float y -- real y
float *x -- real x() (unless otherwise noted)
int *y -- integer y() (unless otherwise noted)
char *s -- character(*) s
unsigned char *a -- No Standard Fortran Equivalent

SUBROUTINES

geth
int geth (iop)
register FILE * iop;

Get a two-byte integer from file or pipe 'iop'. Uses the same byte order on all machines. (This is important!) Usually called by other **libvplot** functions, not by user programs.

puth (w, iop)
register int w;
register FILE * iop;

Put the low two bytes of 'w' on stream 'iop'. Uses the same byte order on all machines. Usually called by other **libvplot** functions, not by user programs.

vp_area(xp, yp, lp, fat, xmask, ymask)
vp_uarea(xp, yp, lp, fat, xmask, ymask)
float *xp, *yp;
int lp, fat, xmask, ymask;

Fill the area within the polygon defined by the points in the 'xp' and 'yp' arrays. Uses the **VP_OLDAREA** vplot command, which knows how to do solid fills or simple halftone-style stipples. 'Lp' is the number of vertices. 'Fat' is the fatness of the border line. If fat<0, no border is drawn. The filling style varies depending on whether or not the device has color. If the device has color, the fill is done solidly using the current drawing color. If the device is monochrome, the masks tell how many blank pixels to use to pad each pixel to be drawn. Filling is done by tiling the area with rectangles of size 'xmask' by 'ymask', with only the

lower right pixel turned on. Thus, both masks 1 gives a solid fill; xmask=1 and ymask=2 produces horizontal lines spaced one pixel apart; xmask=4, ymask=4 gives a 9.1% 'gray' color. If either xmask or ymask is zero, then the interior is not filled (even on color devices).

vp_arrow(x0, y0, x, y, r)
vp_uarrow(x0, y0, x, y, r)
float x0, y0, x, y, r;

Plot an arrow from (x0,y0) to (x,y) with arrowhead-size 'r'. The arrowhead is half a square. 'r' is the length of a side of the square. If 'r' is negative, the arrowhead is not filled. If (x0,y0) = (x,y), a centered box is drawn of size r. All sizes are in inches (vp_arrow) or user units (vp_uarrow). If your user-unit coordinate system is not square, you'll probably want to use vp_where and vp_arrow instead of vp_uarrow.

vp_bgroup(string)
char *string;

Begin a new group. Groups are used to combine several vplot primitives into one entity. Groups may be nested. Groups may not contain an erase.

string: The name of this group.

vp_egroup()

End a group. Each begin group must be paired with an end group, like parenthesis in formulas.

vp_break()

Interrupt the output processing of the vplot file. Prompt the user (or pause, depending on the 'pause' argument to **pen(9)**) allow interaction, and reset global plotting parameters, but don't erase the previous output.

vp_clip(xmin, ymin, xmax, ymax)
vp_uclip(xmin,ymin,xmax,ymax)
float xmin,ymin,xmax,ymax;

Define the current clipping rectangle, in inches or in user units. There is currently no call for simply turning off clipping. Just define a clipping window larger than the space you're working in. The best way to do this is with the call vp_clip(-VP_MAX, -VP_MAX, VP_MAX, VP_MAX). VP_MAX is defined in vplot.h.

vp_color(col)
int col;

Set the current drawing color to a number 'col' between 0 and 511 (MAX_COL in params.h). The first eight colors are predefined:

BLACK	0
BLUE	1

RED	2
PURPLE	3
GREEN	4
CYAN	5
YELLOW	6
WHITE	7

These numbers are defined in <vplot.h>, so the color name (all in caps) can be used instead of the number (within the to-be-compiled program, NOT as input to plas). These 8 colors may be re-assigned, if desired, but usually aren't. Color 0 is always the background color; drawing in color 0 erases. Color/pattern numbers larger than 7 should be set by the program, using the **vp_coltab()** call, before being used (what color you get when you plot with an unset color is device-dependent). The numbers 0-511 are used both for colors and for area-fill patterns. The user is responsible for keeping track of how the color/pattern table is set. See the descriptions of **vp_fill()**, **vp_area()**, **vp_patload()**, and **vp_hatchload()** for more information.

vp_coltab(col, red, green, blue)
int col;
float red, green, blue;

Color number 'col' is set to have the color (red, green, blue). The color levels are all between 0. and 1. Thus, (1.,1.,1.) is white and (1.,1.,0.) is yellow. 'col' is between 0 and 511 (MAX_COL in params.h). (The handling of colors on devices that don't have many colors, or don't have settable colors at all is a tricky business. In general Vplot will try to take care of all of this for you and come as close as is possible to what you want. Read the Vplotraster manual page to find out more about this.) Color table settings last until changed by the user, except that colors 0 through 7 are reset to the default when a pen filter is first fired up. Normally colors 0 through 7 are left at their default settings, so that simple plots using only those colors can be freely interspersed with a complex many-hued plot without problems. Color 0 defines the background color.

vp_dash(dash1, gap1, dash2, gap2)
float dash1, gap1, dash2, gap2;

Set the current line style, by defining a special line type. 'Dash' and 'gap' are in inches. This dashing is done within vplotlib itself, and so these dashes are guaranteed to scale with the rest of the plot. **Vp_setdash** allows a more complicated (and efficient) dashed line pattern to be defined.

vp_draw(x, y)
vp_udraw(x, y)
float x, y;

Draw from the current position to (x,y) using the current color, fatness, and line style.

vp_endplot()

Close the plot file (output stream).

vp_erase()

Clear the screen, feed a page of paper, or flush a raster file to the output stream. Reset most global plotting parameters to their default values. Prompt the user (or pause, depending on the 'pause' argument to **pen(9)**) and allow interaction. Normally every plot file should start with a **vp_erase** call, as the very first thing (after opening the output file). A common mistake is to call routines like vp_style, vp_fat, etc, and only THEN vp_erase to erase the screen before actually drawing something. This is a mistake, as the vp_erase command will start a new frame. This is not noticeable on most screen devices (unless you are pausing between frames), but it will be very noticeable on hardcopy devices that will spit out a blank first page before "doing the plot". Since **vp_erase** resets most of the vplot global parameters (fatness, current color, text attributes, current clipping window, dash line pattern, etc), you should set these to their desired values again after every call. Color table settings are NOT reset at erases.

vp_fat(fatness)
int fatness;

Set the line width, 200 (FATPERIN) per inch. Default: 0, which is the thinnest line possible on the device.

vp_file(filename)
char *filename;

Open the file called 'filename' as the output stream. Works for C and for FORTRAN. One of the functions { **vp_file()**, **vp_filep()**, or **vp_unit()** } MUST be called to initialize the output.

vp_filep(filepntr)
FILE *filepntr;

Connect the already-open stream pointer 'filepntr' to the output stream. C language only. (Example: vp_filep(stdout).) One of the functions { **vp_file()**, **vp_filep()**, or **vp_unit()** } MUST be called to initialize the output.

vp_unit(lunit)
int *lunit;

Connect the Fortran logical unit 'lunit' to the output stream. Works for 4.2/4.3 BSD Unix only. This routine was written using details of the Berkeley Fortran implementation, which have been changed in ports to some machines other than VAXes. One of the functions { **vp_file()**, **vp_filep()**, or **vp_unit()** } MUST be called to initialize the output.

vp_fill(xp, yp, npts)
vp_ufill(xp, yp, npts)
float *xp, *yp;
int npts;

Fill the area within the polygon defined by the points in the 'xp' and 'yp' arrays using the **VP_AREA** vplot command. The polygon fill style is determined by the current fill pattern, which is the loaded pattern with the same index number as the current drawing color. Patterns are loaded with the **vp_patload()** and **vp_hatchload()** calls. If no pattern has been loaded for this number, the polygon will be filled solidly with the current drawing color. 'Npts' is the number of vertices of the bounding polygon.

vp_hatchload(angle, numhatch, ipat, array);
int angle, numhatch, ipat;
int *array;

Load a hatch pattern as pattern number 'ipat'. (When the current drawing color is number 'ipat', this is the pattern that will be used with the **vp_fill()** call.) The hatch pattern consists of 'numhatch' sets of lines drawn at an angle of 'angle' to each of the two coordinate axes. For each set of lines (numhatch * 2 of them), the array contains 4 elements specifying: 'fatness', 'color', 'offset', 'repeat interval'. (Thus the array is of dimension numhatch * 2 * 4.) The 'repeat interval' determines the distance between the lines in a set. The 'offset' parameter is used to shift all the lines in the set. Both of these parameters are in units of 1/100'ths of an inch (HATCHPERIN in vplot.h). Color and fatness are just as you'd expect.

vp_message(string)
char *string;

Have the vplot filter issue the message given in "string".

vp_move(x, y)
vp_umove(x, y)
float x, y;

Resets the current position to (x,y), but doesn't actually draw anything. Used in conjunction with vp_draw(), vp_udraw(), or vp_where(). Units are inches (vp_move) or user units (vp_umove).

vp_orig(x0, y0)
float x0, y0;

Sets where on the device screen (in inches) the origin of the user coordinate system will be plotted. The initial reference point is the lower left corner of the display area for style=STANDARD, and the upper left corner for style=ROTATED.

vp_uorig(x0, y0)
float x0, y0;

Assigns values, in user units, to the origin of the user coordinate system, whose location on the screen was set by the **vp_orig()** command.

vp_patload(ppi, nx, ny, ipat, raster)
int ppi, nx, ny, ipat;
int *raster;

Load a raster pattern as pattern number 'ipat'. The pattern is designed at 'ppi' pixels per inch. The raster pattern itself is in the array 'raster', which is an array of color table numbers. It will be represented on the display scanned TV-style, starting in the upper left hand corner and working left-to-right (up to ny) and then top-to-bottom (up to nx scan lines).

vp_pendn(x, y)
vp_upendn(x, y)
float x, y;

Move to the location (x, y) in inches (vp_pendn) or user units (vp_upendn) and then put the pen down.

vp_penup()

Pick the pen up.

vp_pline(xp, yp, npts)
vp_upline(xp, yp, npts)
float *xp, *yp;
int npts;

Draw a polyline through the points given in the 'xp' and 'yp' arrays.

vp_plot(x,y,down)
vp_uplot(x,y,down)
float x, y;
int down;

Draw a vector from the current position to (x, y) with the pen up ('down' = 0) or down ('down' = 1). Uses the current line style. (Fat, dashed lines may look odd on some devices.) Usually called by other **libvplot** functions, not by user programs.

vp_pmark(npts, mtype, msize, xp, yp)
vp_upmark(npts, mtype, msize, xp, yp)
int npts, mtype, msize;
float *xp, *yp;

Plot 'npts' polymarkers, of type 'mtype' and size 'msize'. 'xp' and 'yp' are arrays of x and y values in inches (vp_pmark) or user units (vp_upmark). 'msize' is in vplot text height units. 'Mtype' gives the following special symbols from font 15 for these special values: 2, plus; 3, asterisk; 4, circle; 5, cross; 20, square; 21, triangle; 22, diamond; 23, star. 'mtype' = 0 or 1 sets a single pixel, the smallest displayable dot. Other values of 'mtype' refer to

glyph numbers in the current font (which should be compatible with ASCII for most fonts).

vp_purge()

Flush the output buffer. Flushes both the vplot stream and the device stream.

vp_raster (array, blast, bit, offset, xpix, ypix, xll, yll, ppi, xur, yur, orient, invert)
vp_uraster (array, blast, bit, offset, xpix, ypix, xll, yll, ppi, xur, yur, orient, invert)
unsigned char *array;
float xll, yll, *xur, *yur, ppi;
int xpix, ypix, bit, offset, blast, orient, invert;

Display the raster data pointed to by 'array'.

blast	If 'blast' = 1, don't try to compact the output. If 'blast' = 0, compaction will be done. Compaction does run-length encoding and compacts repeated lines. Compaction can make the vplot file considerably smaller, but it also takes longer to create the file.
bit	If 'bit' = 1, the raster is saved as one bit per pixel; if 'bit' = 0, as one byte per pixel. Byte raster is faster!
[xy]pix	'xpix', 'ypix': Number of pixels in each direction.
[xy]ll	'xll', 'yll': The display coordinates of the lower left corner of the array, in inches (vp_raster) or user units (vp_uraster).
ppi	'ppi': If 'ppi' is non-zero, scale the image by duplicating or dropping image pixels to a factor of 'ppi' output pixels per inch. The point (xur, yur) is returned as the upper right hand corner. If 'ppi' is zero, use whatever resolution is required to make (xur, yur), which must then be supplied, the upper right hand corner.
[xy]ur	'xur', 'yur': The display coordinates of the upper right corner (Well, actually, not quite. Read the vplotraster manual page.). Either in inches (vp_raster) or user units (vp_uraster). These two variables should NOT be arrays in FORTRAN, despite being declared 'float *' for C. Note that 'xll' <= 'xur', 'yll' <= 'yur'.
array	'array': The 'array' contains values from 0 to 255. For byte raster these values have 'offset' added to them and then they are interpreted as color values. For bit raster, a value of 0 is interpreted as color 0, and anything else is interpreted as color 'offset'.
orient	For 'orient' = 0 and 'invert' = 0, the array is drawn on the screen television style. The filling starts from the upper left and works to the right and then down. 'Orient' 1 rotates this 90 degrees clockwise, 'orient'

2 180 degrees, etc.

invert — 'Invert' reverses the 'array' over the slow axis. The data occupies the same area of the output regardless of the value of 'orient' and 'invert'.
Note that the point (xll, yll) is the lower leftmost pixel of the data, whereas the point (xur, yur) is one to the right and above the upper rightmost pixel of the data. This is so that (xur-xll) and (yur-yll) give the length of the two dimensions of the data box.

offset — The 'offset' parameter is provided so that several different raster color tables can be maintained at once, perhaps from 0-255 for ordinary colors and from 256-511 for a gray scale.

vp_scale(xscl,yscl)
float xscl, yscl;

Set the scale of user units for the plot, in inches per user unit.

vp_setdash(dashp,gapp,lp)
float *dashp, *gapp;
int lp;

Set the dashed line pattern. The pattern is set to dashp[0], gapp[0], dashp[1], gapp[1], ..., dashp[lp-1], gapp[lp-1], where dashp contains the lengths of the solid-line segments (dashes) of the pattern and gapp contains the lengths of the gaps between dashes. All lengths must be specified in INCHES. This dashing will be done either by the device hardware itself or by code in the pen filter. This dash pattern will be of the specified size in inches regardless of any scaling done to the plot as a whole!

Generally, unless you have some reason to require that the dash pattern scale exactly with the size of the plot, vp_setdash is preferable to vp_dash. This is especially true when using devices that support fat dashed lines in hardware.

vp_stretch(xmin,ymin,xmax,ymax)
float xmin, ymin, xmax, ymax;

This routine makes calls to vp_orig, vp_uorig, and vp_scale for you so that (xmin,ymin) is the user coordinate of the lower left hand corner of the output space and (xmax,ymax) is the user coordinate of the upper right hand corner.

As an example, suppose that you want to display everything in the unit box (0. to 1.) on both axes, but that you also want your coordinate system to remain isotropic. You would call vp_stretch(0.,0.,1./SCREENRATIO,1.);

vp_style(st)
int st;

Set the overall display style. Choices are:

STANDARD — (default, for terminals and laserprinters) Origin in lower left, scaled so that the maximum Y value

(top of the screen) is 10.24 (STANDARD_HEIGHT) 'inches'.

ABSOLUTE — (default, for large bed or electrostatic plotters) Origin in lower left, plotted in physical inches on the device.

ROTATED — Origin in upper left, Y-axis horizontal increasing to the right, X-axis vertical and increasing down, scaled so that the maximum X value (bottom of the screen) is 7.5 (ROTATED_HEIGHT) 'inches'. Use of rotated style is discouraged; it is kept around only to support many ancient programs that still use it.

I write 'inches' because unless absolute scaling is being used vplot's 'inches' do not correspond to real, physical inches on the device screen.

Note that the STANDARD style scales the output so a square 10.24 inches on a side will fit on the terminal screen. This allows a full 8.5x11-inch image to be shown on most terminals. The magic numbers 10.24 and 7.5 are defined in **vplot.h.** More about coordinate systems can be found in the Vplot manual page. The plot style reverts to the default at the start of every frame.

vp_text(x, y, size, orient, string)
vp_utext(x, y, size, orient, string)
float x, y;
int size, orient;
char *string;

Display text, using the currently-defined font, precision, and text alignment.

x, y: — The coordinates of the reference point for the string. Either in inches (vp_text) or user units (vp_utext).

size: — Height of a character, 33 (TXPERIN in vplot.h) units per inch (!!!).

orient: — Text drawing direction, in degrees counter-clockwise from horizontal, right-facing.

string: — The text string that is to be displayed. 'man vplottext' explains the many special escape sequences recognized by the generic vplot text routine gentext.

vp_gtext(x, y, xpath, ypath, xup, yup, string)
vp_ugtext(x, y, xpath, ypath, xup, yup, string)
float x, y;
float xpath, ypath;
float xup, yup;
char *string;

Display text, using the currently-defined font, precision, and text alignment. The text coordinate system is defined in the GKS standard.

x, y: The coordinates of the reference point for the string. Either in inches (vp_gtext) or user units (vp_ugtext).

xpath, ypath: A vector pointing in the direction the text string will extend. Either in inches (vp_gtext) or user units (vp_ugtext).

xup, yup: A vector pointing in the "up" direction for individual letters. Either in inches (vp_gtext) or user units (vp_ugtext).

For normal, unstretched text, the up vector is 90 degrees counterclockwise from the path vector, and both vectors are the same length. This length is the text size. Projecting these two vectors will project the entire text string accordingly.

string: The text string to be displayed, just as in the other text command.

vp_tfont(font, prec, ovly)
int font, prec, ovly;

Choose the font to be used for output. (All the following values are defined in vplot.h.)

font: Which font to use. Sixteen software fonts are defined (if you have the Hershey fonts installed, otherwise you've only got font 0. You may also have hardware fonts starting at 100 (NUMGENFONT in params.h) available on some devices). The fonts names are defined in vplot.h. Font 0 (PEN in vplot.h), the traditional vplot font, is fastest and is the default on terminals (unless they have an even faster hardware font). Another better-looking font should be the default on hard copy devices. NO_CHANGE leaves the font unchanged from that in effect before the call.

prec: Font precision. This doesn't matter much if you're using a software font to begin with. 'Prec' is one of

STRING	Use the hardware text capabilities to write the whole string. The default for screen devices.
CHAR	Use hardware characters, but position them individually.
STROKE	Software text. This should be the default for hardcopy devices. Ligatures in the Vplot software fonts are only enabled at STROKE precision. (See the vplottext manual page for more about this.)
NO_CHANGE	Use the previous value.

ovly: Overlay mode:

OVLY_NORMAL:	just draw the text over what's there in the standard way;
OVLY_BOX:	draw a box around the text first;
OVLY_SHADE:	clear a box under the text first;
OVLY_SHADE_BOX:	box the text and clear under it too;
NO_CHANGE:	use the previous value.

vp_tjust(hjust, vjust)
int hjust, vjust;

Set the position of the reference point for locating strings or characters. Choices are (these are defined in vplot.h):

hjust:

TH_NORMAL	Use the default.
TH_LEFT	Left justify.
TH_CENTER	Center the string.
TH_RIGHT	Right justify.
TH_SYMBOL	Position the character for use as a symbol marking the point.

vjust:

TV_NORMAL	Use the default.
TV_BOTTOM	Reference point is at the bottom of the letters.
TV_BASE	Reference point is at the bottom of the descenders.
TV_HALF	Centered.
TV_TOP	Reference point at the top of the writing area, including space above letters.
TV_CAP	Reference point is at the tops of capital letters.
TV_SYMBOL	Position the character for use as a symbol marking the point.

It is possible to change font size, change font, and do many other tricks "on the fly" by including escape codes in the text string. See the vplottext manual page for information about the details of interpretation of special characters within vplot text.

vp_where(x,y)
float *x, *y;

Return the current position, in inches. Fortran users note that x and y are not arrays, despite being declared 'float *' in C.

ACCESSING VPLOTLIB VARIABLES

At present, few inquiry functions are provided by **libvplot.** It would be wonderful if someone could turn the one-way pipes into two-way sockets and provide some. The changes required to the pen filters themselves would be minimal, as the required device dependent structures already exist.

The current values of the global parameters used in libvplot are stored in the external structure **vp_pc.** This structure is defined in the file vp_pc.h, which is "#include"ed by the libvplot routines that need to refer to the variables in vp_pc. User programs should probably have no need to do this.

MAGIC NUMBERS

The magic numbers (200, 100, 33, 7.5, 10.24, etc) used in this document are subject to being changed at new sites. At Stanford we are forced to live with these off-the-wall numbers in order to not break a large mass of ancient code. They are all defined in "vplot.h" (which should be in /usr/include, and which you should probably #include) and "params.h" (which you probably don't need to worry about).

I would have loved to have turned everything into inches or centimeters, but it just wasn't possible here. It is especially annoying that text is currently quantized in such rough units. Sometimes it is impossible to get text the exact size you need!

COPYRIGHT

The Vplot code is copyrighted. Refer to the Vplot manual page for the official copyright notice.

SEE ALSO

pen(9), vplot(9), vplottext(9), vplotraster(9), plas, pldb

AUTHOR

Dave Hale, Joe Dellinger, Chuck Karish, and Steve Cole

LIMITATIONS

Vector fonts and hardware fonts cannot be intermixed in a single string.

Inquiry functions should be provided.

Vpunit() only works under 4.x BSD FORTRAN.

Vplot.h should have defines for the various special markers.

BUGS

None, that I know of. It is likely that some of the lesser-used FORTRAN routines still harbor bugs.

NAME

vplot text - A guide to using the Hershey fonts in vplot

DESCRIPTION

Vplot fonts are mixtures of vectors and filled areas which simulate typeset fonts on graphics devices. Since they are not raster, they can be arbitrarily scaled, stretched, skewed, and rotated, and can be plotted on any device.

There are 17 fonts. Font 0 is just the regular old PEN font we've been using all along. The default font is device and installation dependent, or can be set from the command line (see options below). Here is a complete list of fonts and their number:

0 pen
1 roman simplex
2 roman duplex
3 roman complex
4 roman triplex
5 italic complex
6 italic triplex
7 script simplex
8 script complex
9 greek simplex
10 greek complex
11 Cyrillic complex
12 German style gothic triplex
13 Greek style gothic triplex
14 Italian style gothic triplex
15 mathematics
16 miscellaneous

Font 0 was designed by Rob Clayton at Stanford. Fonts 1 through 16 are debugged versions of the 'Hershey' fonts available from mod.sources. In order to have permission to use these fonts, I must now display this message:

- The Hershey Fonts were originally created by Dr.
A. V. Hershey while working at the U. S.
National Bureau of Standards.

INTERFACE

This section describes the control sequences which are accessible to any user of **vplot** text, via whatever program.

Vplot text recognizes '\' as a special character used to signal the start of an escape sequence. There are two sorts of escape sequences, those that take an argument and those that do not.

Here is a complete list of escape sequences that do not take an argument:
\> Advance one interletter space

\< Back up one interletter space
\^ Raise one half of a capital letter height
_ Lower one half of a capital letter height
\g Continue processing text, but don't actually print it ('ghostify it'). This is useful if you want to leave space to go back and add something by hand.
\G Start printing text again
\n Newline
\h Backspace (control-h also works) back up over the last character
\- Does nothing; used to prevent a group of characters from being formed into a ligature.
\\ Print a backslash

The following take an integer argument immediately after, with a **required space** after the integer to delineate the end. This space is not printed.

\s Size change. Change to this percent of the size set in the text vplot call. \s100 restores the default height.
\f Add this to current fatness. Goes out of effect when text printing is finished.
\F Switch to this font number. (-1 restores the default font).
\k Move by this many space widths to the right.
\r Move up this many character heights (the height of a standard capital letter, such as 'A').
\w Print this ASCII character number in the current font, stripping it of any special meaning. This and ligatures are the only way that glyphs numbered greater than 255 are available.
\c Switch to this color. -1 restores the current drawing color. Vplot's current drawing color is not changed by changing the color inside text.
\m Save current position in this number register.
\M Restore position saved in this number register.

EXAMPLES

Since programs such as 'graph' do not interpret text themselves, but rely on vplot to do it, the above control sequences may simply be used wherever any graphics text is used. For example, instead of specifying as an option to graph label='beta', you can specify label='\F9 b' and actually get the Greek letter beta from the Greek Simplex font.

Here are some more examples:

To get an integral sign:
\F15 \w168
Translation: Switch to font 15 (math symbols), print character #168 (integral sign).

To get 'X squared plus Y squared':
X\^2_ + Y\^2
Translation: Print X, go up half, print 2, go back down half, print

+, print Y, go up half, print 2.

As an advanced example, to get 'A sub b sup beta':
A\m1 _b\M1 \^\F9 b
Translation: Do A, save this spot, go down half, do b, restore the saved spot, go up half, switch to font 9 (Greek Simplex) do beta. (Notice the use of marks to stack characters on top of each other.)

Warning: \ is a special character in C. To get a backslash, you have to use \\

GENERIC PEN OPTIONS

txsquare=no
Xscale and yscale distort text along with the entire plot. To avoid this, set **"txsquare=yes"**, and then for the purposes of text xscale and yscale will be forced equal to their geometric mean.

txfont=font_number
Txfont specifies the default text font. The default is device and installation dependent. Generally, screen devices will default to the 'pen' font since it is by far the fastest to draw, and hardcopy devices will default to either roman simplex or roman complex since these are the closest to looking like standard typeset text. Out-of-range font numbers (starting at 100) may be used in a device-dependent fashion to access hardware fonts on devices that have them. Devices that don't have hardware fonts will use the font number modulo 100. Fonts 0-99 are guaranteed device-independent.

txprec=0
Txprec specifies the default text precision (a GKS term). Possible values are character (0), string (1), and stroke (2). For vector fonts, the text precision parameter is ignored except that ligatures are only enabled at precision 2. The precision parameter may be significant if device-dependent hardware fonts are used, however. The default precision is device and installation dependent.

txovly=0
Txovly sets the default text overlay mode. This controls whether or not to clear out an area behind the text before writing it, and whether or not to put a box around it. Overlay modes 1 and 3 draw a box, and modes 2 and 3 shade the inside of the box with the background color. The default text overlay mode is 0 (do nothing). All text overlay modes may not be supported on all devices. (Some devices cannot shade to background color, for example.)

The default font, default precision, and default text overlay mode may all be reset by the appropriate vplot command.

fontN=file_name
The binary format vplot font file for font number N is located in file_name. If the font number is not one normally defined on the system, file_name defaults to fontN, where again N is the font number. If the device does not have hardware fonts, the font number modulo 100 will be used for all internal purposes (so fonts 201 and 101 must be the same, for example). The N in the above option will still be the original font number, however. Fonts that are included at compile time are hardwired and cannot be changed.

LIGATURES

Fonts may specify that certain combinations of adjacent glyphs be combined into a single new glyph whenever they occur. For example, 'ffi' in font 3 (it's a ligature in troff, too). This takes place automatically at precision 2. The only fonts which currently contain ligatures are the Roman Complex, Italic Complex, and Cyrillic Complex fonts. You can prevent a group of letters from being formed into a ligature by breaking them up using the '\-'' special sequence (just like in TeX). (Ligatures may be used in fonts for foreign languages to good effect to automatically de-transliterate English approximations. For example, 'sh' in the Cyrillic font automatically comes out as a single Shah character. 'KHrushchev', for example, contains only 6 glyphs: KH-r-u-shch-e-v. The K and H here are both capitalized since they are both part of the first character. The 'Tests' directory contains the example TEST_Cyrillic which shows how to generate every letter in the Cyrillic alphabet.)

TEXT JUSTIFICATION

Text justification is set by the appropriate command in vplot, and follows the GKS standard. Since this text routine allows character size and active font to be changed within a single text string, the GKS standard is extended so that 'TOP' and 'BOTTOM' alignments align on the highest top and the lowest bottom of the characters used. (Note that this makes the alignment font and size dependent, but not dependent on the actual characters used.) The 'CAP' and 'HALF' alignments are determined by those for the default font.

All sizes of all fonts are aligned together by their baselines.

Horizontal alignment is based according to the beginning and ending horizontal positions. You can use marks to make the ending position be anywhere in the string you wish.

A 'SYMBOL' justification mode has also been added to those defined by GKS. Each glyph has a 'hot spot' defined, which is where the glyph is to be centered if it is used as a symbol. The last printable character in a string is the character centered on as the symbol. To center on a character in the middle of the string as a symbol, mark the spot with the \m command immediately after that character and then restore that location at the end of the string with the \M command. That character is now the 'last

character' as far as all centering is concerned. Size changes and horizontal shifts are taken into account in the symbol justification mode, but shifts up and down are not.

Vplot polymarkers are drawn by setting the SYMBOL text justification mode, setting the correct font, creating a special single-character string, and then calling the text routine to draw the symbol.

DIAGNOSTICS

Nonexistent glyphs are rendered as a special 'error' glyph. Which glyph is used for this is installation dependent, and is set by defines in gentext.c. As distributed, it is glyph 30 in font 0, which is a '?' inside a diamond. (I stole this idea from the Imagen laser printer.) An error message will be produced the first time that a nonexistent glyph is referenced. (Only the first time so that you don't get overwhelmed with error messages if you're trying to use a font that doesn't exist on that system.) Attempts to use nonexistent fonts produce an error message and font 0 (or whatever other font has been designated as the error font in gentext.c) is used instead.

Various other non-fatal errors can occur and all give self-explanatory error messages.

COPYRIGHT

The Vplot source code is copyrighted. Please read the copyright notice you will find in the Vplot manual page.

FONT FILES

Font files used by the vplot text subroutine 'gentext' are kept in an 'include' form and a 'binary' form. The include form is compiled in as an #include file into gentext. This is more efficient, but makes the executable code considerably longer and slows down compilation. The binary form is loaded on the fly as needed at runtime. Which method is used for which font is system dependent and is determined by the file .../vplot/include/font_de finitions.h.

ASSOCIATED PROGRAMS

The subroutine **gentext.c** is the device-independent generic subroutine that vplot uses to draw vector fonts. Gentext.c is constructed in such a way that it could easily be fit into a GKS package. While Vplot itself thinks in terms of 'text orientation' and 'text size', gentext thinks in terms of 'character up vectors' and 'character path vectors'. Squashed, slanted, etc, text can be produced by having these two vectors not be perpendicular or not the same length. (This can only happen when using vplot when xscale does not equal yscale.)

The program **makefont.c** takes a 'vplot font' and encodes it into the forms required by gentext. The format of a 'vplot font' is designed such that it can be read into the program 'plas' and then piped into a vplot filter. IE, the command

plas < cyrilc.vplot_font | ?pen xcenter=0 ycenter=0 scale=50 pause=1
will display each glyph in the cyrillic font, one per second.

The exact format of a 'vplot font' file is given in comments at the beginning of makefont.c, and the file pen.vplot_font serves as an example. Since vplot_fonts are just open-format vplot files, it is very easy to create your own special purpose glyphs, markers, or fonts and add these to the system.

The program **hertovplot.c** converts 'Hershey' font data as distributed by mod.sources (mod.sources is a UNIX network 'news' group for distributing useful UNIX software.) into vplot fonts. Its use is described in comments at the beginning of the program. The program **hertogrid.c** draws all the glyphs in a Hershey glyph file on a grid, labeling each glyph with its number. (The Hershey glyph files have to be organized into reasonable fonts by hand! The organization of my fonts mostly follows those provided by Mr. Hurt, but I have added in a few things he missed and reordered the non-English fonts to make more sense. I have stolen bits of Mr. Hurt's code to write hertovplot.c and hertogrid.c.)

FONT INSTALLATION

To install all the fonts, you must first obtain the Hershey fonts for yourself as distributed by mod.sources (if you don't know how to get stuff from them, then how did you get this file that you're reading now?). Follow the instructions to create the four '.oc' files, and copy these into the .../vplot/Hershey directory. Do not copy the '.hmp' files. Follow the instructions you will find in the file .../vplot/Hershey/README to make all the required 'vplot_font's. (Note that pen.vplot_font is not part of the Hershey font distribution. This font was created by Rob Clayton around 1980, and slightly modified by Joe Dellinger in 1986.) Once you have done this, go into the .../vplot/filters/include/vplotfonts directory, and do 'make'. This will create the required '.include' and '.bin' fonts.

SEE ALSO

pen(9) vplot(9) libvplot(9)

AUTHOR

Joe Dellinger (joe@hanauma.stanford.edu, convex!convexw!hanauma!joe, decvax!hanauma!joe) You may ask me for advice if you wish to help do any of the things in the wish list below.

BUGS

The letter 'J' in the Roman Triplex font in the Hershey font distribution has mangled width information. The vplot font for this font has to be edited by hand to correct this mistake.

Gentext.c does not yet support all the required GKS features. However, as all the internal calculations are done in a way consistent with the GKS model modifying the code to support these is a very straightforward job

which I leave to someone else.

Somebody should make sure that all the 'hot spots' defined for characters are reasonable. (The 'hot spot' is the point that is aligned on when symbol alignment mode is used.) (The ones installed now are the defaults chosen by makefont.c.) Somebody should create a special 'polymarker' font. Currently several of the more esoteric symbols called by the vplot polymarker command are not very well centered (notably triangles).

Somebody should write an interactive program to create new glyphs with.

Somebody should turn some of the better Hershey vector fonts into filled-area fonts. (Unfortunately, Mr. Hershey didn't see fit to order his vectors in such a way as to make this easy.) The only thing lacking is the font itself, because vplot itself already supports arbitrary mixtures of filled-areas and vectors in fonts. Currently there are just a few glyphs containing filled areas in font 0, as examples showing how to do it.

Dr. Geller (the first tenured American professor at a Japanese university, and a former Stanford Earth Sciences professor) should have his students create a katakana and a hirigana font. Ligatures can be used to pair the appropriate consonant-vowel pairs. There are also quite a few Kanji glyphs available as well, from which a Kanji font could be constructed. A ligature could be constructed to turn each English word into its Kanji equivalent! (The number of glyphs in a font is not limited to 256! That's why I did things that way.)

Somebody should extend the way ligatures are read from vplot font files, so that you can just give the ASCII character itself instead of the glyph number if one exists. Since I only had a few ligatures I didn't want to bother with this.

NAME

vplot raster - A guide to using raster plotting commands in vplot graphics

DESCRIPTION

Despite the name **vplot** (where the 'v' once stood for 'vector') the vplot metalanguage includes raster plotting capability. This document describes more detailed aspects of raster plotting than are included in the **vplot** manual page.

A raster is a rectangular array of bytes. The value stored in each byte determines the number of the color that is to be displayed at the corresponding location in the raster. The mapping of array values to colors is set up by the programmer by calling the **vp_coltab()** routine, which may be found in **libvplot.a**. To define a color using **vp_coltab()**, the red, green, and blue components of the color are each specified as floats from 0. to 1., with 1. being fully on and 0. being fully off. The corresponding 'grey level' for grey-scale devices is given by the formula floor((4 * green + 2 * red + blue + 6)/7).

Each device has a number of settable colors (call this number N). Calling **vp_coltab()** for colors 0 through N-1 will redefine the desired color. If you attempt to define a color X outside of the range 0 through N-1, **vplot** will pick the color number in the range 0 through N-1 that is closest to color X in color and map all requests for color X to that color number. Whenever any color in the range 0 through N-1 is changed, all the mappings will be recalculated (although of course what is already drawn cannot be changed). Note that color 0 is the background color. Other colors will be mapped to color 0 only if the match is exact.

As an example, consider the common case where an eight bit per pixel raster is to be displayed using a grey scale, ranging from black to white, and various colors are to be used to display labels on the plot. The calling program should call **vp_coltab** to allocate the first few color numbers to the label colors, then fill the remainder of the lower half of the color scale (up to color 255) with the needed grey scale colors. Because different devices have different numbers of settable colors, it is important to order the grey scale colors so that the most important ones such as black (0, 0, 0), white (1, 1, 1), and medium grey (.5, .5, .5) come first. Then whatever the number of settable colors, the raster reproduction will be as good as possible. More explicitly, the calling program should repeatedly cover the 0 to 1 range with a decreasing step size, taking care not to repeat colors, i.e., (0, 0, 0), (1, 1, 1), (.5, .5, .5), (.25, .25, .25), (.75, .75, .75), (.125, .125, .125), (.375, .375, .375), etc. Then in the upper half of the color scale (colors 256 through 511), the calling program can define the grey scale colors corresponding to the raster array values, setting, for example, color 256 to (0, 0, 0), color 257 to (1./255., 1./255., 1./255.), color 258 to (2./255., 2./255., 2./255.),..., and color 511 to (1, 1, 1). The **offset** parameter in **vp_raster** is used to reference these colors. For this example, **offset=256** would be the appropriate specification. Then for each value in the raster array, **vplot** adds 256 to it to find the grey scale color defined for that value, then (using the mapping to available colors) uses the available color that best matches the requested color to plot that raster element.

Monochrome devices such as plotters are unable to give a good representation of grey rasters because of the lack of definable colors. For such devices, **vplot** can dither the image to simulate a grey or continuous-tone image. Briefly, dithering methods represent a continuous-tone image on a bilevel display by varying the threshold value that determines whether a given output pixel is set to 'on' or 'off' given its value on input. Consider the above example of an eight bit per pixel raster. This means that, on an appropriate graphics display, each pixel can take on any of 256 different intensity levels. The task of dithering is to represent such an image on a device that has only two different intensity levels. A crude way of transforming from continuous-tone to bilevel form would be to divide the continuous-tone intensity range in half, and set input intensities 0-.5 to 'off' and .5-1 to 'on'. But this would neglect most of the continuous-tone information. Vplot has four better ways of performing this transformation built in.

Random dither randomly selects a threshold value for determining whether a given output pixel is set to 'on' or 'off'. This preserves much of the continuous-tone information, but gives a rather 'noisy' image, since within a constant-intensity region of the raster, some pixels will be set to both 'on' and 'off' due to the random thresholds.

Ordered dither applies a sixteen by sixteen matrix of threshold values to the input image in checkerboard fashion. This gives a much more regular appearance on output.

The **minimized average error method** or **Floyd-Steinberg algorithm** takes the 'error' in converting a single pixel to bilevel form into account when converting neighbors, with the goal of minimizing the difference between the continuous-tone and bilevel images. This method generally produces the best representation of the original image, though it is the slowest of the four algorithms.

Digital halftoning is an ordered-dither scheme that is designed for images that are going to be reproduced photographically. The high-frequency alternation of black and white samples that is used to produce grey levels in ordered dither is not reproduced faithfully by photocopiers. This algorithm uses lower-frequency alternation of samples to produce the same grey level. While the resulting image has a coarser texture, it will be reproduced accurately. This method is only recommended for images that will be reproduced photographically.

The dithering method can be selected by use of the **dither** parameter with any **pen** filter. The available dithering methods are:

0 No dither (0=off, anything else=on)
1 Random Dither
2 Ordered Dither
3 Minimized Average Error Method
4 Digital Halftoning

Dithered output can be displayed on polytone or color devices by specifying **mono=y** in the **pen** filter call.

On a typical graphics display, a linear grey scale has a nonlinear appearance. The transition from black to white is more rapid than expected, leaving the scale clipped at both ends. This perceived nonlinearity is due to the characteristics of the human eye, which sees a linear change in brightness as a logarithmic change, as well as to the display characteristics of the device being used. When a grey scale is displayed on paper using the dithering methods described above, the nonlinearity is no longer present. Such images seem to have a washed out appearance because of the relative scarcity of pure black and white shades. The nonlinearity of display devices seems to be a useful feature that one might wish to duplicate on paper. The pen filter parameter **greyc** (short for 'grey correction') modifies the grey scale used to display a raster to simulate the nonlinearity of displays. The grey scale is multiplied by a cubic polynomial with greyc determining the exact shape of the polynomial. The function has been constructed such that the ends and middle of the grey scale are left undisturbed, but on either side of the middle the grey scale is skewed towards the end of the scale (black or white). Greyc=1. leaves the grey scale undisturbed, and values smaller than 1. skew the scale as described above. We have found greyc=-0.5 to yield plots on our Imagen laser printer that are very similar to images displayed on our Rastertek graphics display. Some experimentation is undoubtedly required for other devices. Once a desirable value has been obtained for a particular device, that value should be made the default for the particular device.

A further complication with plotting raster images is the dot size used by the plotter. In displaying a linear grey scale on our laser printer, we found the result to be much darker than expected, especially in the darker half of the scale. Our hypothesis is that the plotter causes adjacent dots to overlap. When the image is mostly white (as in the white end of the grey scale), this overlap is not very important, since it is rare that a given empty spot is surrounded by black dots. However, in the black half of the grey scale, white places in the plotted image are usually surrounded by black dots, and the overlap causes the white space to be much smaller than expected. Hence the plot is darker than expected.

From this hypothesis, we have constructed a function that alters the grey scale to compensate for dot or pixel overlap. The **pixc** (short for 'pixel correction') parameter controls this alteration of the grey scale. Pixc=1. leaves the grey scale undisturbed. Values smaller than 1. shift all the grey values toward white in such a way as to compensate for the darkening that results from pixel overlap. The shift is larger for the dark half of the grey scale, since (from the above discussion) it is more seriously affected by pixel overlap. We have found pixc=0.6 to be an appropriate value for our Imagen laser printer. This value gives accurate reproduction of a linear grey scale.

As with the greyc parameter, once a suitable value has been obtained for a particular printer, the appropriate value should be made the default for that device.

COPYRIGHT

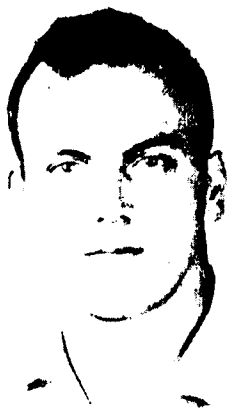
The Vplot source code is copyrighted. Please read the copyright notice which can be found in the Vplot manual page.

SEE ALSO

pen vplotlib vplot vplothacker

AUTHOR

Joe Dellinger and Steve Cole



JON F. CLAERBOUT

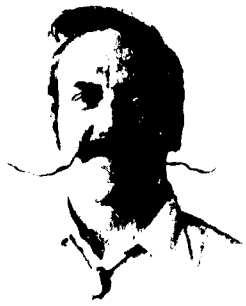
1964, Vol. 29, No.2

CLAERBOUT

1968, Vol. 33, No.2

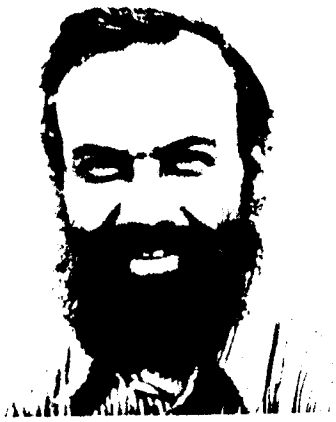
CLAERBOUT

1970, Vol. 35, No.3



CLAERBOUT

1973, Vol. 38, No.5



JON F. CLAERBOUT

1974, Vol. 39, No.1

CLAERBOUT

1978, Vol. 43, No.4