# Interface for system independent plotting

*Jon F. Claerbout*


## ABSTRACT

I define a graphics coordinate interface (*GI*) to bridge the gap between application programs and various plotting systems such as vplot, Sunview, X and Postscript. The principal activity is coordinate management. Users declare windows and coordinate systems at the beginning. This eases portability across systems and it helps merging plots. *Pictures* expand *isotropically* to a bounding rectangle whereas *plots* expand *differentially* to fill a rectangle of any aspect ratio. A drawing for landscape mode is well displayed in portrait mode and vice versa. A tutorial program illustrates Fortran in an interactive environment adjusting travel time plots under Sunview and outputting hardcopy under Vplot.


## PREFACE

This could have been an internal document at SEP, but the software is more portable than typical of times past. Even for those groups with neither Sunview nor vplot, the bulk of the software described herein is coordinate management and is available in about 600 lines of Ratfor. (Ratfor is a public domain preprocessor to Fortran). For those with Sun or Convex computers, large parts should work without change because Sunview is a Sun product and Vplot is a Stanford product. An aspect of limited portability is that the test programs and vplot initialization use our local SEP parameter data basing (seplib) which we have not found to be as portable as we wish. Rick Ottolini began the Xwindows platform and Jean Claude Dulac finished it.

## INTRODUCTION

I am interested in writing plot code with a fair degree of permanence. This is because of my interest in maintaining lecture notes and books, and my interest in

maintaining SEP plot code over several generations of students. I find that plot code is hard to maintain when it has built into it the orientation and size of some prescribed coordinate set, such as that of the hardware and plotting systems in use when the software was written.

### Boring review of my previous plot utility work

An earlier software project called *svplot* [Claerbout, 1987], directed outputs of three big software projects (Balloon, ed1D, Zplane) interactively to Sunview and to a file for vplot hardcopy. Compared to the usual screen dump, svplot retains full resolution, which on a screen is about one megapixel and in hardcopy is about six megapixels. Further, vector information is stored in vector format, so the files are much smaller than raster files, and they can be kept on line indefinitely. And, the Sunview end of svplot handles raster well enough for movies.

Svplot encompasses both vector and raster for both interaction and hard copy. This makes it more complete than either Sunview or vplot. But svplot is so complex that I never used it for plotting simple figures from Fortran which is unfortunate because many figures have adjustable parameters and could recompute and readjust in seconds, and so are especially suitable for education.

If the difficulty of preparing interactive plots cannot be greatly reduced, then they won't be used much in research.

### Claim

To try to overcome the forementioned difficulties I wrote a *graphics coordinate interface* and embedded in it access to two local plotting systems, vplot and Sunview.

### GI, a graphics coordinate interface

GI is intended to be a graphics coordinate interface to simplify plotting, especially

- portability over plotting systems
- suitable for simple Fortran jobs
- cut and paste diverse plotting programs.

The characteristics of GI are:

1. GI requires coordinate declarations at the outset, and then derives all coordinate transformation information from the declarations.
2. GI is a switchboard between application programs and plotting systems.
3. GI pushes system and device coordinates to the background and recognizes that users need page coordinates as well as user-world coordinates.

4. GI defaults to *filling* a rectangular page window with a rectangular user window regardless of the ratio of aspect angles. So the default is to plot a circle as an ellipse. A switch allows preservation of isotropy.

5. GI allows a simple switch between landscape mode (video screen) and portrait mode (paper page).

## DESIGN PRINCIPLES

I will avoid using the terms $x$-axis and $y$-axis which cause confusion because they seem to link the *algebraic* 1-axis and 2-axis with the *geometric* horizontal and vertical axes. Programmers use the 1-axis and the 2-axis, and hardware and system suppliers specify horizontal and vertical directions and lengths. There is a distressing disparity of ways to link algebra with geometry.

Although mathematical coordinate systems generally have the origin at the center or the lower left, there are many important coordinate systems with the origin at the upper left, such as written European languages, mathematical matrices, television, and the many related video display systems.

Three points or six numbers determine a coordinate system in a plane. These three points are the vectors to the coordinate origin $\vec{o}$, to the tip of the horizontal axis $\vec{h}$, and to the tip of the vertical axis $\vec{v}$. Requiring the coordinate system to be orthogonal drops the number of parameters from six to five because $(\vec{h}-\vec{o})\cdot(\vec{v}-\vec{o}) = 0$. Unfortunately, it is an oversimplification to suppose that forcing the coordinate system to be aligned vertically and horizontally drops the number of free parameters to four. Since four would be more intuitive than five, let us see why four parameters are not enough: Any corner of a rectangle is specified by giving the values of its 1-axis and its 2-axis. Choose diagonally opposite corners, say the bottom left corner specified by $(bl1, bl2)$, and the top right corner by $(tr1, tr2)$. In almost all cases the top left corner is $(bl1, tr2)$ (and the bottom right is $(tr1, bl2)$). The exceptions are when the 1-axis is vertical. Three such exceptions are (1) matrices in mathematics, (2) seismograms, and (3) plotting a landscape style figure on a portrait style device, or vice versa.

### Rectangles

We will discuss only two rectangles which I call the frame `f` and the window `w`. The frame is defined to be the outer bound of the drawable area. Users define rectangles on the frame called windows.

Windows are the vehicle for setting margins, cutting and pasting plots into matrices, and combining cooperating interactive programs. (At present GI does not support windows in windows which is a theoretical limitation for compounding software efforts.)

### Coordinate types

There are three coordinate *types* for the three essential aspects of plotting.

d The manufacturer imposes coordinates on the frame f in its own *device* units d. Likewise many software plotting systems have an intrinsic coordinate system, that we also call device coordinates.

p You the user prescribe coordinates for the frame f using your *page* coordinates p. To give the location of any window w in the frame f you also use your page coordinates. For page units, I like percentage points. Postscript encourages typesetters points or inches or cm.

u Each user's application uses the coordinates of its own world. More often than not, the physical units differ between the 1-axis and the 2-axis. Some people call such coordinates *world* coordinates. Since the letter 'w' is the initial letter of both "world" and "window" I call such coordinates *user* coordinates, u.

The general principal is this: For each rectangle, two declarations are required, one giving corner positions in interior coordinates, and the other giving corner positions in exterior coordinates. So the frame needs to be given in both d and p units and the window needs to be given in both p and u units. Looking to the future, one more rectangle, a square in device units, is declared to allow for nonisotropic devices. Rectangle bounds are named as follows:

sd an upright *square* in *device* coordinates.

fd the limits of the viewable *frame* in *device* coordinates.

fp the *frame* in user's *page* coordinates.

wp the *window* rectangle in user's *page* coordinates.

wu the *window* rectangle in the *user*-world coordinates

Ordinarily you draw on the frame using page coordinates and on the window in user coordinates. You should never draw with device coordinates.

## PROGRAM LANGUAGE

The first thing a user program does is call the routine `gi_setframe()`. (The underscore character in the name gets absorbed by the Fortran compiler.) This subroutine declares sd and fd and it also declares my favorite defaults for fp, wp, and wu which are percent of full page (0.0 to 100.0) measured from the lower left corner. To override these defaults, you can specify your coordinates either as a *coordinate triad*, or by the *five parameter method.*

### Definition of coordinate triad

A *coordinate triad* ties together the algebraic concept of coordinates with the geometrical concept. A coordinate triad is defined by three geometrical points, the top left corner $\vec{t}$, the bottom left corner $\vec{o}$, and the bottom right corner $\vec{r}$. Although I have found it convenient to name the bottom left point $\vec{o}$, there is no presumption that $\vec{o}$ is the origin. The origin could be anywhere inside or outside the rectangle. To specify a coordinate triad, each of these corners must be given in algebraic coordinates, i.e. a value must be given for the corner's 1-axis and another value for the corner's 2-axis. The arguments to the triad setting program `gi_triad()` are first the triad name (such as `'wu'`) and then the six values `(t1,t2,o1,o2,r1,r2)`.

### Five parameter coordinate specification

An alternate means of providing coordinate definition is the five parameter method used by the routine `gi_axes()`. Its first argument is the triad name, next is an integer flag called `h1v2` which if nonzero specifies that the 1-axis is horizontal and the 2-axis is vertical, and finally the range on the 1-axis `(bl1, tr1)` and the range on the 2-axis `(bl2, tr2)`. Mnemonically, `bl` means bottom left and `tr` means top right. Here are some examples:

```
gi_axes('fd', 1, 0., 13.65,  0., 10.24)  # vplot
gi_axes('fp', 1, 0.,  100.,  0.,  100.)  #  frame percentage
gi_axes('wp', 1,.05,   .95, .05,   .95)  #  5% page margins.
gi_axes('wu', 1,-1.,    1., -1.,    1.)  #  math quadrants.
gi_axes('fd', 1, 0., n1-1., n2-1.,  0.)  # sunview (note floating point)
gi_axes('wp', 1, 1.,   80.,   24.,  1.)  # video screen characters.
```

An axis range can be negative as illustrated by the 2-axis of a video screen.

Any time you call an axis or triad routine, all the coordinate transformations are recomputed.

### Isotropic plots

The coordinate system building routines need to know if the user is displaying a *plot* or a *picture*. I define a *picture* to be anything which resizes isotropically, i.e. vertical and horizontal expand and compress in a constant proportion. The aspect ratio, i.e. the diagonal across the picture, must be kept at a constant angle. I define a *plot* to be anything that expands differentially to fill space. Since *pictures* do not necessarily fill space, it is the job of the transformation generating programs to arrange for empty space either on the sides, or else above and below. So the picture slides either up and down or side to side in the available space. (Some house windows slide up and down while others slide sideways). The coordinate transformation generation programs need to know whether you want to center the

picture, or slide it towards the coordinate system origin. The default is an area filling *plot*, and if you choose a *picture*, the default is that it is centered.

All the plotting systems I have seen seem to assume the user wants to plot pictures, not plots. But in geophysics I find we have ten times as many plots as pictures. So the default is area-filling plots. To override this, see `gi_isotropic()`. The GI plot utility assumes this small but irritating burden of scaling incommensurate physical units into isotropic device units. The significance is that GI forces you to declare your intentions at the beginning of your code, and then prevents you from device or system dependent activity deep in your code. This enhances maintainability and encourages patching together various plot programs written at different times by different people.

You can center an isotropic page of any aspect ratio on the frame. You can center an isotropic user space of any aspect ratio in any window. Or you can uncenter either of them and they slide towards the origin of the coordinate axes. To override the centering assumption, see `gi_center()`.

Ordinarily clipping of output is done at the boundary of a window. To override this, see `gi_clip()`.

## SAVING WINDOWS

The graphics interface GI keeps the coordinate systems in a private block of labeled common. The reason for privacy is to simplify the programs of users who do not wish the wealth of detail that is available. As soon as you call `gi_setframe()` this all gets initialized and transformations are built among the device, page, and user systems. After that you can override defaults and begin drawing with such routines as `gi_vector( a1, b1, a2, b2)`. The units default to user units. If you reset a frame triad, the units switch to page units. Setting a window triad changes back to user units. You can manually switch units with `gi_select()`. For batch-oriented plotting, you may never feel the need to save and restore transformations. It is easy enough simply to recompute them. Matters are different with interactive computing.

### Windows in interactive computing

In interactive computing, an operator has a pointing device, generally a mouse, and moves a pointer over the plotted page, crossing windows, and finally selecting one on which to click a button. The writer of an interactive program writes a subroutine called an *event procedure* that is called by the system to report pushbutton activity as well as the location of the cursor. The first thing the event procedure generally does is check to see which window the cursor is in. The system automatically restores the coordinate transformations (if any) when the cursor enters the window.

An interesting philosophical question is how overlapping windows should be handled. One philosophy is to forbid them. Another philosophy is to pile them on top of each other like paper on a desk top. Still another philosophy which accommodates windows in windows, ad infinitum is for the system to restore the window of smallest area. I have ignored window ambiguity and will promise you only that if two windows overlap, events will not randomly jump from one to the other, they remain on a window until you move the pointer off.

### What are the components of a window?

A window is made up of the wp and wu triads as well as a collection of flags about clipping, isotropy and centering. You save it by invoking the routine `gi_savewin()` which saves all the parameters relevant to the current transformations. You can get a window number by `n = gi_savewin()` or you can just use the fact that the numbers are assigned consecutively from 1. To restore a window you call `gi_restorewin( n )`. To set the number of saved windows to zero, see `gi_forgetwin()`.

## STRUCTURE OF AN INTERACTIVE PROGRAM

The user generally provides four programs, the `main`, and three subroutines which must be named `mywrapup()`, `myrepaint()`, `myevent()`. They communicate via `common`, usually `labeled common`.

### Main

In pseudo Ratfor, the main program looks like

```
common /mystuff/this,that/
# Fetch parameters and data here.
call gi_setframe()
# Set up and save some windows here.
call gi_run()    # Never return
end
```

### mywrapup()

Here the user outputs data or parameters that are judged to be of later use.

### myrepaint()

Here the user just draws whatever he/she wants. It is almost like a batch program, but not quite. This program may be invoked many times. Thus, if it changes a parameter or a coordinate system halfway through, it should be prepared for the change to be already made before the beginning of the second time through.

### myevent()

Writing an event routine will be a new experience for batch-oriented programmers. The user's event procedure is called by the system when it detects mouse motion, keyboard activity, etc. The simplest event procedure would then change a parameter, erase the screen (with `gi_erase()` ) and then call `myrepaint()`. The user-prepared event procedure must have the six arguments

```
myevent( iwind, ievent, u1,u2, p1,p2)
```

The window number `iwind` refers to the location of the mouse pointer. It will be zero if the pointer is on the frame outside all windows. Besides user coordinates `(u1,u2)` and page coordinates `(p1,p2)` are an event number `ievent`. Your event procedure may compare `ievent` to the constants defined in the include file in /usr/include/event.com

```
#define WARNING!!!!   Source for this file is /sep/jon/taal/gci/event.com

#define GI_MOVE          257
#define GI_DRAG          258
#define GI_FRAMEENTER    259
#define GI_FRAMEEXIT     260
#define GI_LEFT_UP       261
#define GI_LEFT_DOWN     262
#define GI_MIDD_UP       263
#define GI_MIDD_DOWN     264
#define GI_WINEXIT       265
#define GI_WINENTER      266
```

The meanings of these events are perhaps obvious, but I mention that DRAG means to move the mouse while any button is down. UP and DOWN refers to pressing a mouse button down or releasing it up. Although we have a three button mouse, at present we omit the RIGHT button, thereby reserving it for system use. Event numbers not in the table above refer to ASCII codes of keystrokes, for example 'A' is 65, and backspace is 8. Finally, the FRAME events will not be detectable by users because they are internally converted to window events often on window number zero.

Before compilation, I use the C-preprocessor to convert these symbols to numeric parameters. The Fortran preprocessor seems to differ on our various machines whereas the C-preprocessor is a constant across machines. Also, by using the C-preprocessor we maintain consistency across Fortran and C language applications. Alternately, you could put the above numbers in Fortran parameter statements.

### Using `fp` to merge plots

For simplicity, assume each plot program accepted the default page coordinate frame `'fp'` running from zero to one hundred percent (0,100). To put one plot on the left side of a page, before calling it, you would reset the 1-axis of `'fp'` to (0,200). Before calling the right side plot you would set `'fp'` to (-100,100).

A theoretical problem is that your final program manipulates `'fp'`, unlike the components that you merged, so your final program could not be so simply merged again. This is a problem of lack of closure. I am not ready to support windows in windows ad infinitum, but I am ready to discuss it.

### Using fp to merge window programs

Merging plots merges plot programs too, hence allowing new kinds of interaction. Imagine a family of programs using gi that includes user programs, as well as GI versions of familiar utilities like Wiggle, Thplot, Movie, and interactive programs like ed1D and Zplane. The programs could share a block of labeled common that would be the usual seplib n1,n2,n3, etc, as well as an area for a seplib float cube and byte cube, and perhaps an input cube and an output cube. A little consideration shows the event procedures, of merged user programs and plot utilities would both need to be called from a super event proc. This is all a bit theoretical, but seems not impossible.

## SELF DOCUMENTATION

The following was "awk"ed straight from the source code. Routines marked "internal" are not intended for general use.

```
        switch.r: routines that switch to the various graphics systems.

gi_setframe()           # System dependent frame setup
        vplot -filep(output), hclose,axes
        sunview -Open main screen window.
gi_initcolor()
        initialize the color table with ''good'' defaults
        the experts will call gi_setcolortable to set their own colors
gi_setcolortable(rescol,colid)
        set the color table with user preferences
        colid is the color file names or a identifier as I, F(flag)
        rescol is the number of vplot reserved color
gi_run()
        vplot alone -- call myrepaint() and gi_wrapup()
        sunview -- go to window_main_loop() never return
gi_wrapup()             # internal
        first call my_wrapup() and then:
        vplot -- Close plot file.
        sunview -- destroy the base frame.
gi_hardcopy()           # internal
        When both sunview and vplot are defined (libgisv.a)
        this routine makes vplot dumps of sunview screen.
gi_move( x, y)
        Move the hypothetical pen location to (x,y)
gi_draw( x, y)
        Draw a line from current position to (x,y)
gi_nodraw( )
        Raise the pen so that the subsequent gi_draw() becomes a move.
gi_vector( x1, y1, x2, y2)
        Draw a line from (x1,y1) to (x2,y2)
gi_polyline(n,x,y)
        Draw a polyline
gi_text( x, y, size, degrees, text)
        like vp_text(float,float,int,int,'string')
gi_area()
        area fill command not yet installed
gi_setclip( xmin, ymin, xmax, ymax)    # internal
```

```
        invoke vp_clip(), no sunview equivalent.
        arguments are in device units!
gi_erase( irect)
        erase page or current window
        irect = {1,2} = {page,window}
        vplot method: area fill with color=0, then set color=7
gi_see( message)
        Sunview: Put a message in the message window.
        Vplot: Ignore message.
gi_color( n)
        Set color, now=vplot, later = Ottolini color standard.
gi_setcolor (name)
        set the current drawing color to name
gi_rubberlines(n,x,y)
        draw a rubber polyline
        n : number of vertex (max=100)
        x,y : vertex coordinates
gi_rubberdelete()
        delete the last rubber polyline created
gi_float2bytes(n1,n2,array,raster)
        transform float to bytes
        n1,n2 are the dimension of the data array
        raster is the rasterized data
gi_raster( x0,y0, x1,y1, x2,y2, n1,n2, s0,s1,s2, ras)
        plot raster
        ras(i1, 1),i1=1,n1 plots from (x0,y0) to (x1,y1)
        ras( 1,i2),i2=1,n2 plots from (x0,y0) to (x2,y2)
        ras(i1,i2) = ras( 1 + s0 + (i1-1)*s1 +(i2-1)*s2)
gi_button(code,button,feedback)
        generate a new button
        code : character code (q,v,p are system reserved)
        button : icon character string
        feedback : help message
        get the button code corresponding to the button event
gi_wiggle( x0,y0, x1,y1, x2,y2, n1,n2, s0,s1,s2, ras, dir)
        wiggle trace
        ras(i1, 1),i1=1,n1 plots from (x0,y0) to (x1,y1)
        ras( 1,i2),i2=1,n2 plots from (x0,y0) to (x2,y2)
        ras(i1,i2) = ras( 1 + s0 + (i1-1)*s1 +(i2-1)*s2)
        if( dir == 1 ) wiggle traces run on 1-axis.
        if( dir == 2 ) wiggle traces run on 2-axis.
gc_ordinate( x0,y0,x1,y1,x2,y2, n1,n2, s0,s1,s2)       # internal
        Tumble raster pointers until axes run positively.
gi_outerbd( bot, x ,top)       # internal
        Set x at either bot or top depending on which is closer.

        ------------ gc = graphic coordinate subroutines ------------------
        coord.r: system INDEPENDENT routines
        (but call system dependent routines in switch.r)

gi_axes( NAME, h1v2, bl1, tr1, bl2, tr2)
        Import axes by stating a name and rectangle.
        NAME is a string describing one of five rectangles. These are:
        --- 'sd' rectangle that in device coordinates should be square.
        --- 'fd' the viewable Frame in Device coordinates.
        --- 'fp' the viewable Frame in user chosen Page coordinates.
        --- 'wp' A Window in user chosen Page coordinates.
        --- 'wu' A Window in User chosen User's world coordinates.
        integer h1v2 > 0 if horiz axis is axis 1 and vert is 2.
        "bl" is bottom left and "tr" is top right corner.
        See also gi_triad()
gi_triad( NAME, t1,t2, o1,o2, r1,r2)
        Import axes by name and three vectors to rectangle corners.
        NAME is a string describing one of five rectangles.
        (t1,t2)=Top_left (o1,o2)=bot_left (r1,r2)=bot_Right
        See also gi_axes()
gi_select( irect)
        You select a coordinate system for move,draw,etc commands.
        irect = {0,1,2} = {device, page, user}
        Writes in device or page coordinates access the full screen.
        Writes in user coordinates access the current 'wp' window.
gi_clip( iyesno)
        iyesno = {0,1} = {no,yes} clip at window edges.
        Default is to clip.
```

```
gi_isotropic( irect, iyesno)
        irect = {1,2} = {frame, window} rectangle i.e. {page,user} coords.
        iyesno = {0,1} = {fill_area,isotropic}
gi_center( irect, iyesno)
        irect = {1,2} = {frame, window} rectangle i.e. {page,user} coords.
        iyesno = {0,1} = {don't center, do} isotropic {window,page}.
gi_user2page( u1,u2, p1,p2)            # user vector to page vector
gi_savewin()
        Save the present window
gi_restorewin( iwind)
        Restore a saved window
gi_forgetwin( )
        Set number of saved windows to zero

        ------- internal routines below -------------------------------
gc_initwin( )          # internal. Set default window states.
        Turn off isotropy and on clip and centering.
gc_inittri( )          # internal. Set default triads.
        Called by everything, but will run only the first time.
        Default fp, wp, and wu to squares with sides ranging from 0 to 100
        Default fd to vplot and sd to upright unit square.
gc_output( up1,up2, d1,d2)             # Transform output vector
gc_input( d1,d2, p1,p2, u1,u2)         # Transform input vector
gc_decode( h1v2, bl1, tr1, bl2, tr2, tri)      # Decode h1v2
gc_emplace( name, tri)                 # Copy given triad to named one.
gc_maketran()          # Make all transformations.
gc_load( )             # Install transform and clip values.
gc_doclip( triad)
        invoke clip commands at periphery of triad window.
gc_whereclip( triad, xmin,ymin, xmax,ymax)
        Given a triad, find clip boundaries in device units.
gc_pair( p, q, p2q)    # Get transform from 2 triads
gc_image( xx, tt, uu)  # Transform a triad.
gc_tan( tri, tangent)          # Get tangent of triad diagonal.
gc_puff( p, scale, middle, q)  # Differential scale up to enclose.
gc_cade( a, b, c)      # Cascade transforms y=cx=(ba)x
gc_portrait( idir)
gc_sideways( tri, idir)                # internal
        tip a triad 90 degrees, direction idir={+1,-1}
gc_chooser( event, xd, yd)
        Choose which window mouse is in and whether
        to change MOVE or DRAG event to ENTER or EXIT
        FRAMEENTER and FRAMEEXIT changed to WINENTER and WINEXIT
gc_reclip()
        reset clip for all stored windows.
```

### Source code

The code of GI is in Fortran (actually Ratfor, which translates to Fortran). The vplot interpretation is in Fortran. Fortran allows (1) use with the Convex vectorizing Fortran compiler, and (2) transportability for graduating students and to sponsors. The Sunview interpretation is partly in C.

## MATHEMATICAL BASIS

Here we examine some of the theory and design considerations, and internal details.

### The transformation

All the coordinate $(x, y)$ pairs you plan to plot must go through the following transformation:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} A_{xx} & A_{xy} \\ A_{yx} & A_{yy} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} s_x \\ s_y \end{bmatrix} \tag{1}$$

Much new hardware and software has this transformation built in. In any case, a big chore for the applications programmer is relieved by the GI package keeping track of the transformation itself.

### Homogeneous equations

The basic transformation can be rewritten in homogeneous form.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} A_{xx} & A_{xy} & s_x \\ A_{yx} & A_{yy} & s_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

Notice that products of matrices of the above form retain that form. Thus you can see that a cascade of transformations can be done sequentially or combined into a single transformation i.e. $A(Bx) = (AB)x$.

### Fortran encoding the transform in a vector

The transformation equation (1) has six adjustable numbers. I encode the numbers in the matrix of (2) into a Fortran vector as follows:

$$\begin{bmatrix} \texttt{t(1)} & \texttt{t(3)} & \texttt{t(5)} \\ \texttt{t(2)} & \texttt{t(4)} & \texttt{t(6)} \\ \texttt{0} & \texttt{0} & \texttt{1} \end{bmatrix} \tag{3}$$

### Need for floats

On video screens, it is tempting to work with integers. But integers are a trap. They lose precision that is needed for hard copy.

### How routine `gi_pair()` works

We saw that the coordinate transformation took six numbers to specify and that the coordinate triad is also specified by six numbers. Given two triads (twelve numbers) we now see how to find a transform. Take the transpose of equation (1)

$$[u\ v] = [\text{x y 1}] \begin{bmatrix} A_{xx} & A_{yx} \\ A_{xy} & A_{yy} \\ s_x & s_y \end{bmatrix} \tag{4}$$

Stacking the row vectors in (4) on top of each other for three $(x, y)$ and $(u, v)$ pairs gives two sets of $3 \times 3$ equations solvable for the transformation, namely for $(A_{xx}, A_{xy}, s_x)$ and for $(A_{yx}, A_{yy}, s_y)$. These equations are:

$$\begin{bmatrix} u_1 & v_1 \\ u_2 & v_2 \\ u_3 & v_3 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} A_{xx} & A_{yx} \\ A_{xy} & A_{yy} \\ s_x & s_y \end{bmatrix} \tag{5}$$

Now that I have imposed the horizontal and plumb constraint on the transformations, I guess I should solve these systems analytically. It would be more elegant and I could abandon the Gauss-Jordan solver. Since either the diagonals, or the off diagonals should vanish, I chose to set to zero the smaller. This might speed computation in machines that recognize that multiplying by zero is easy.

### How isotropy can be forced

Isotropy is forced as follows: A picture is isotropic if the angle of the diagonal of the window triad expressed in isotropic device coordinates equals the angle of the window triad expressed in user units.

First the triad of the window expressed in page coordinates is transformed into device coordinates as you would transform any three vectors. Now we have the window's rectangle expressed in both device coordinates and user coordinates. Next we will examine and rectify the mismatch of the diagonal angles.

For each triad, we first define vertical and horizontal vectors, the vertical span $\vec{dv} = \vec{t} - \vec{o}$ and the horizontal span $\vec{dh} = \vec{r} - \vec{o}$. The cross product $\vec{dv} \times \vec{dh}$ is the product of the magnitudes. Dividing by the dot product $\vec{dh} \cdot \vec{dh}$ gives the tangent of the diagonal angle, $|\vec{dv}|/|\vec{dh}|$.

When the diagonal angles of the two rectangles are dissimilar, the picture must be shrunk in one direction (leaving white space in the new area). Shrinking the picture means the triad must be expanded. So either $\vec{dh}$ or $\vec{dv}$ is scaled up to bring magnitude ratios into agreement.

### Landscape to portrait via `fp`

To tumble a page to the side and keep the usual benefits of area filling versus isotropy, it is merely necessary to tumble `fp`. By this I mean you find the complementary corner $\vec{c}$ to the rectangle by vector addition, $\vec{c} = \vec{r} + \vec{t} - \vec{o}$ and then you cycle the four points $(\vec{t}, \vec{o}, \vec{r}, \vec{c})$ to either $(\vec{o}, \vec{r}, \vec{c}, \vec{t})$ or for the other sense of rotation to $(\vec{c}, \vec{t}, \vec{o}, \vec{r})$.

## EXAMPLE

This example is a two page Fortran (actually Ratfor) program that enables the operator to adjust three parameters of a simple seismic geometry to see the effect of

NMO on multiple reflection and ghost events. The operator uses a mouse to move three letters on a plot. "F" adjusts the sea Floor depth. "G" adjusts the ghost delay (w.r.t. the first arrival). "V" adjusts the velocity (or vertical exaggeration).

Here is the common block used to communicate between the main, the repaint, the event, and the wrapup procedures.

```
#CCCCCCCCCCCCCCC          seismo.com        CCCCCCCCCCCCCCCCCCCCC
common /seismo/ velocity, ghost, floor
real velocity, ghost, floor

common /seismo/ xhot, thot
real xhot(3), thot(3)

common /seismo/ xmin,xmax, tmin,tmax
real xmin,xmax, tmin,tmax
#CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

The complete program source listing is below. The listing begins with the main program which fetches initial values of the adjustable parameters. Next is the wrapup routine that saves final values of the adjustable parameters. (These may initial values on later runs). Next is the repaint program which is a lot like any other batch plot program. Last is the subroutine `myevent()` which is of greatest interest to Fortran programmers who are writing their first interactive program.

```
 # Gather with adjustable velocity, sea floor and ghost.
integer getch
#include "seismo.com"
if ( getch('velocity','f',velocity) == 0)       velocity = 1.
if ( getch('ghost',   'f',ghost   ) == 0)       ghost = .1
if ( getch('floor',   'f',floor   ) == 0)       floor = .5
call gi_setframe();
xmin  = 0.;     xmax  =  2.;    tmin = .0;      tmax = 2.5;
call gi_triad('wu', tmin, xmin, tmax, xmin, tmax, xmax)
call gi_axes ('wp', 1, 10.,40.,10.,90.)
call gi_savewin()
call gi_axes ('wp', 1, 60.,90.,10.,90.)
call gi_savewin()
call gi_run();
end

subroutine mywrapup()
#include "seismo.com"
call auxputch( 'velocity','f', velocity, 'mypar')
call auxputch( 'ghost',   'f', ghost,    'mypar')
call auxputch( 'floor',   'f', floor,    'mypar')
return; end

subroutine myrepaint()
#include "seismo.com"
integer nmo
do nmo = 0, 1 {
        call mkplan( 0., nmo)
        call mkplan( ghost, nmo)
        }
return; end

subroutine mkplan( ghosty, nmo)
implicit character (a-z)
#include "seismo.com"
integer i, n, nmo
real radian, xx,tt, tau, ghosty, t, dt, arg
n = 20
call gi_restorewin( nmo + 1)
call gi_clip( 0)        # turn off clip while drawing axes and labels
```

```
if( nmo == 0 ) {
        xhot(1) = xmax
        thot(1) = xmax / velocity
        call gi_text( thot(1), xhot(1),  6, 90, 'V')
        }
call gi_vector( 0., xmin, 0., xmax)
call gi_vector( 0.,   0., tmax, 0.)
call gi_text( 0., xmax+.05, 7, 90, 'x')
call gi_text( tmax+.1,  0., 7, 90, 't')
if( nmo == 0 )
        call gi_text( -.1, xmin+.5, 10, 90, 'Raw')
else
        call gi_text( -.1, xmin+.5, 10, 90, 'NMOed')
if( nmo == 0 ) {
        thot(2) = floor
        xhot(2) = -.15
        call gi_text(         thot(2), xhot(2), 6, 90, 'F')
        xhot(3) = -.15
        if( ghosty != 0.) {
                thot(3) = thot(2) + ghosty
                call gi_text( thot(3), xhot(3), 6, 90, 'G')
                }
        }
call gi_clip( 1)
for( tau=0;  tau < tmax;  tau=tau+floor ) {
        call gi_nodraw()
        do i= 0, n-1 {
                radian = 3.14159265 *  (.5*i) / n
                tt = tau / cos( radian)
                xx = velocity * tt * sin( radian)
                t = tt + ghosty
                if( nmo > 0 ) {
                        arg = t*t - xx*xx /(velocity*velocity)
                        if( arg > 0 )
                                dt = t - sqrt(arg)
                        else
                                dt = 0
                        t = t - dt
                        }
                call gi_draw( t, xx)
                }
        }
call gi_restorewin( 1)          # Set up the event proc.
return; end

subroutine myevent( iwin, ievent, t, x, p1, p2)
implicit character (a-h, j-z)
integer iwin, ievent, imin, i
real t, x, p1,p2,   dist, mindist
#include "seismo.com"
#include <event.com>
mindist = 1.e30
switch( ievent) {
        case GI_LEFT_DOWN:
                do i = 1,3  {
                        dist = abs( x - xhot(i)) + abs( t - thot(i))
                        if( dist < mindist ) {
                                imin = i
                                mindist = dist
                                }
                        }
        case GI_LEFT_UP:
                call gi_see('Repainting.')
                switch( imin) {
                        case 1:
                                velocity = x / t
                        case 2:
                                floor = t
                        case 3:
                                ghost = t - floor
                        }
                call gi_erase(1)
                call myrepaint()
        }
```

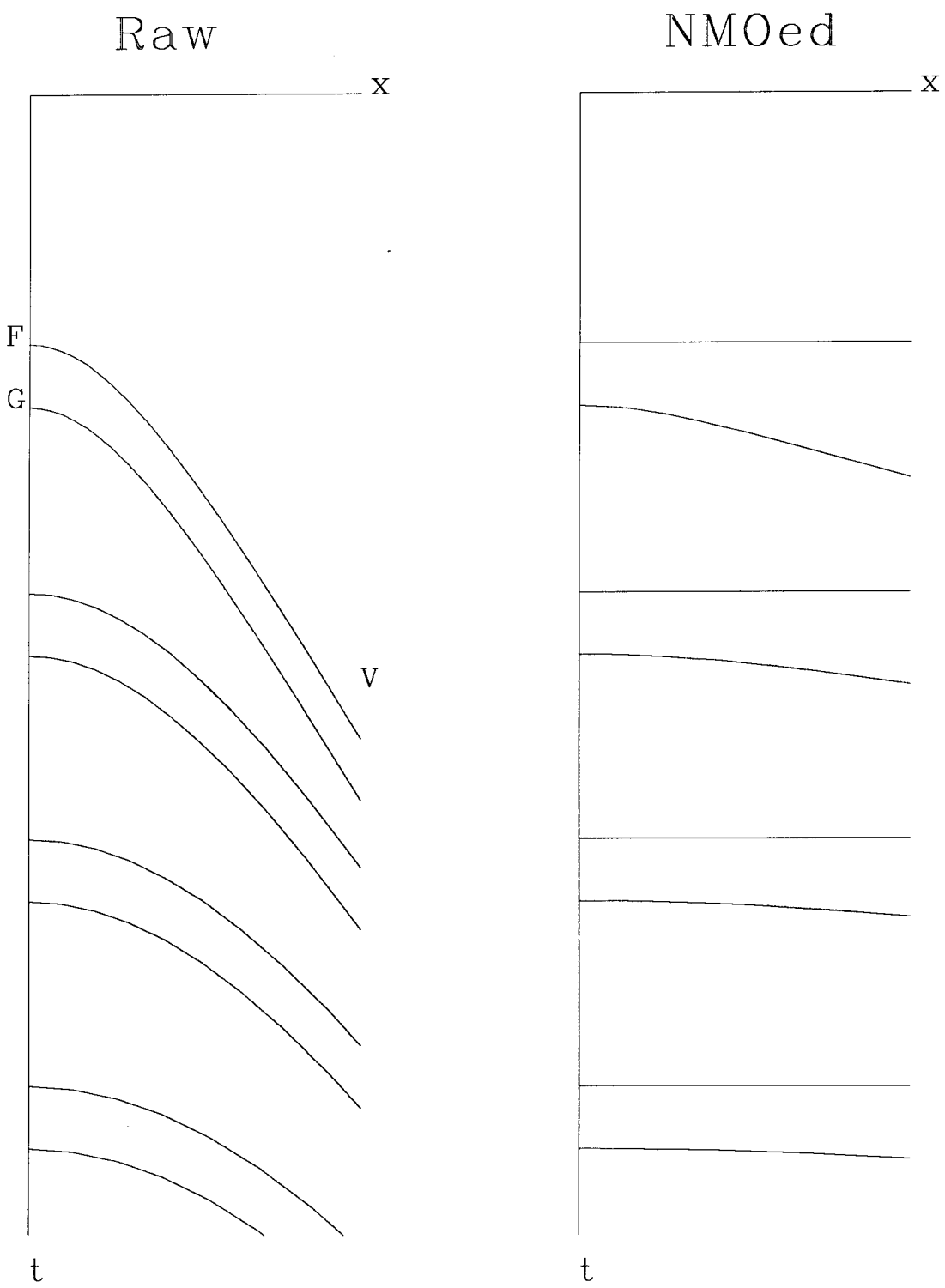


FIG. 1. Use of GI in a deconvolution tutorial. A simple two window plot.

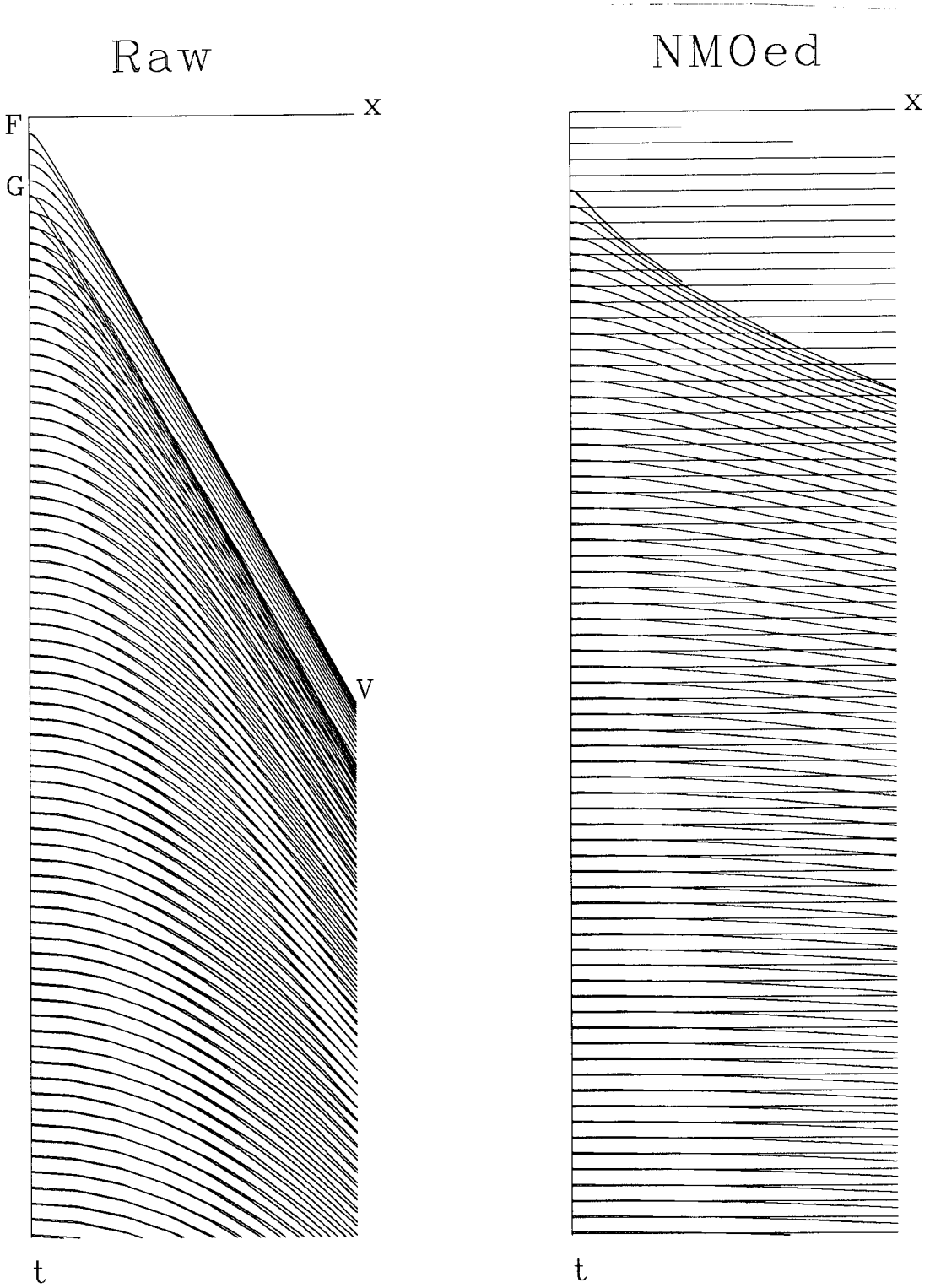


FIG. 2. Deconvolution tutorial example after user has readjusted the water-bottom depth.

```
return; end
```

## TEST CASE

Below is the test program that I use for debugging. It is somewhat long winded, but I include it anyway because it tests most of the features of *gi*. After the program is the result of the test program plotted in hardcopy both before and after the landscape-portrait toggle switch.

```
 # Test area-filling and isotropic windows and frames.
if( getpar('fiso',  'i',fiso  )==0)     fiso  =0        # nonisotropic frame
if( getpar('wiso',  'i',wiso  )==0)     wiso  =0        # nonisotropic window
if( getpar('center','i',center)==0)     center=1        # vrs nocenter
call gi_setframe()
call gi_see ('starting plot ...');
call gi_axes ('fp',1,  0.,200., 0.,100.)# Two side by side 100% pages.
 #call gi_triad('fp',200.,100.,0.,100.,0.,0.)# demo portrait.
call gi_isotropic( 1, fiso)             # set frame isotropy
call gi_axes ('wp',1,  1., 99., 2., 98.)# window left page
call gi_triad('wu',0.,-2.,4.,-2.,4.,2.) # user coords, (sec,km)
call gi_axes ('wu',0,  4.,  0., -2., 2.)# user coords, (sec,km)
call gi_isotropic( 2, 0)                # area fill.
call gi_savewin()
call gi_axes('wu',1, -1., 1., -1.,  1.) # four quadrants.
call gi_axes('wp',1,101.,149., 2., 98.) # left side of right page.
call gi_savewin()
call gi_isotropic( 2, 1)                # Set window coords to be isotropic.
call gi_center( 2, center)              # on window, center={0=no,1=yes}
call gi_axes('wp',1,151.,199., 2., 98.) # right side of right page.
call gi_axes('wu',1, -1., 1.,  0.,  1.) # top two quadrants.
call gi_savewin()
call gi_run();  end

subroutine mywrapup()
return; end

subroutine myrepaint()
character*1 traces(100,5)
integer i, i1, i2
call gi_restorewin(1)
do i2=1,5
        do i1=1,100 {
                i = 128 + 60* cos( .2*i1 - .5*i2 )
                i = 2*(i/2)
                call int2byte( i, traces(i1,i2) )
                }
call gi_raster(2.,1.,4.,1.,2.,2., 100,5, 0,1,100, traces)
call gi_setcolor ('red')
call gi_wiggle(0.,0.,4.,0.,0.,2., 100,5, 0,1,100, traces, 1)
v = 1.5
call gi_setcolor ('green')
for( tau = .2; tau < 4.; tau =tau+.5) {
        call gi_nodraw()                # signifies next draw is really a move.
        for(x=-2.; x<3.; x=x+.2) {      # draw outside window.
                t = sqrt( tau*tau + x*x/(v*v) )
                call gi_draw( t, x)
                }
        }
 #call gi_erase(2)
call gi_restorewin(2)
call gi_nodraw()
for( theta=0.; theta <= 315.; theta=theta+10.) {
        x = cos( 2*3.1416 * theta/360.)
        y = sin( 2*3.1416 * theta/360.)
        call gi_draw( x, y)
        }
```

```
call gi_restorewin(3)
call gi_text( 0., .2 , 7, 0, 'SEP')
call gi_nodraw()
for( theta=0.; theta <= 315.; theta=theta+10.) {
        x = cos( 2*3.1416 * theta/360.)
        y = sin( 2*3.1416 * theta/360.)
        call gi_draw( x, y)
        }
                        call gi_move( 1., 0.);  # Box the window
call gi_draw(-1., 0.);  call gi_draw(-1., 1.);
call gi_draw( 1., 1.);  call gi_draw( 1., .5);
call gi_color( 6);      call gi_select( 1)      # select page coordinates.
                        call gi_move(100.,50.); # Square box at upper right
call gi_draw(50.,50.);  call gi_draw(50.,100.);
call gi_draw(100.,100.);call gi_draw(100.,75.);
return; end
```

Here is the diagnostic test program for interactivity that I used during debugging:

```
subroutine myevent( iwind, ievent, u1,u2, p1,p2)
 #@ event procedure to diagnose window coordinate handling.
#include "event.com"
 integer iwind, ievent, unit
#ifndef VPLOT
 unit = 6
 switch( ievent ) {
        case GI_MOVE:           write( unit, "('GI_MOVE')")
        case GI_DRAG:           write( unit, "('GI_DRAG')")
        case GI_FRAMEENTER:     write( unit, "('GI_FRAMEENTER')")
        case GI_FRAMEEXIT:      write( unit, "('GI_FRAMEXIT')")
        case GI_WINENTER:       write( unit, "('GI_WINENTER')")
        case GI_WINEXIT:        write( unit, "('GI_WINEXIT')")
        case GI_LEFT_UP:        write( unit, "('GI_LEFT_UP')")
        case GI_LEFT_DOWN:      write( unit, "('GI_LEFT_DOWN')")
        case GI_MIDD_UP:        write( unit, "('GI_MIDD_UP')")
        case GI_MIDD_DOWN:      write( unit, "('GI_MIDD_DOWN')")
        default:
               write( unit,20) ievent, ievent
               20 format('Numerical value of letter is =', i12, a2)
               # How to print letters in FORTRAN ?????
               call gi_see('You typed a letter.')
        }
 write( unit,10) iwind, ievent, u1,u2, p1,p2
 10 format(2i6, 4f7.2)
#endif
return; end
```

## COMPILING WITH LIBRARIES AT SEP

On the Convex the library -lgiv supports vplot. On the Suns, the library -lgivs supports vplot and sunview. Also on the Suns is a library -lgis without vplot which frees use of the standard output for other tasks. The X library is invoked by -lgix, and the library outputs both vplot and X is -lgivx.

## UNFINISHED BUSINESS

Just because GI is ready for general use does not mean there is not more to do. Unfinished business is listed below along with the name of the local person with the most expertise.

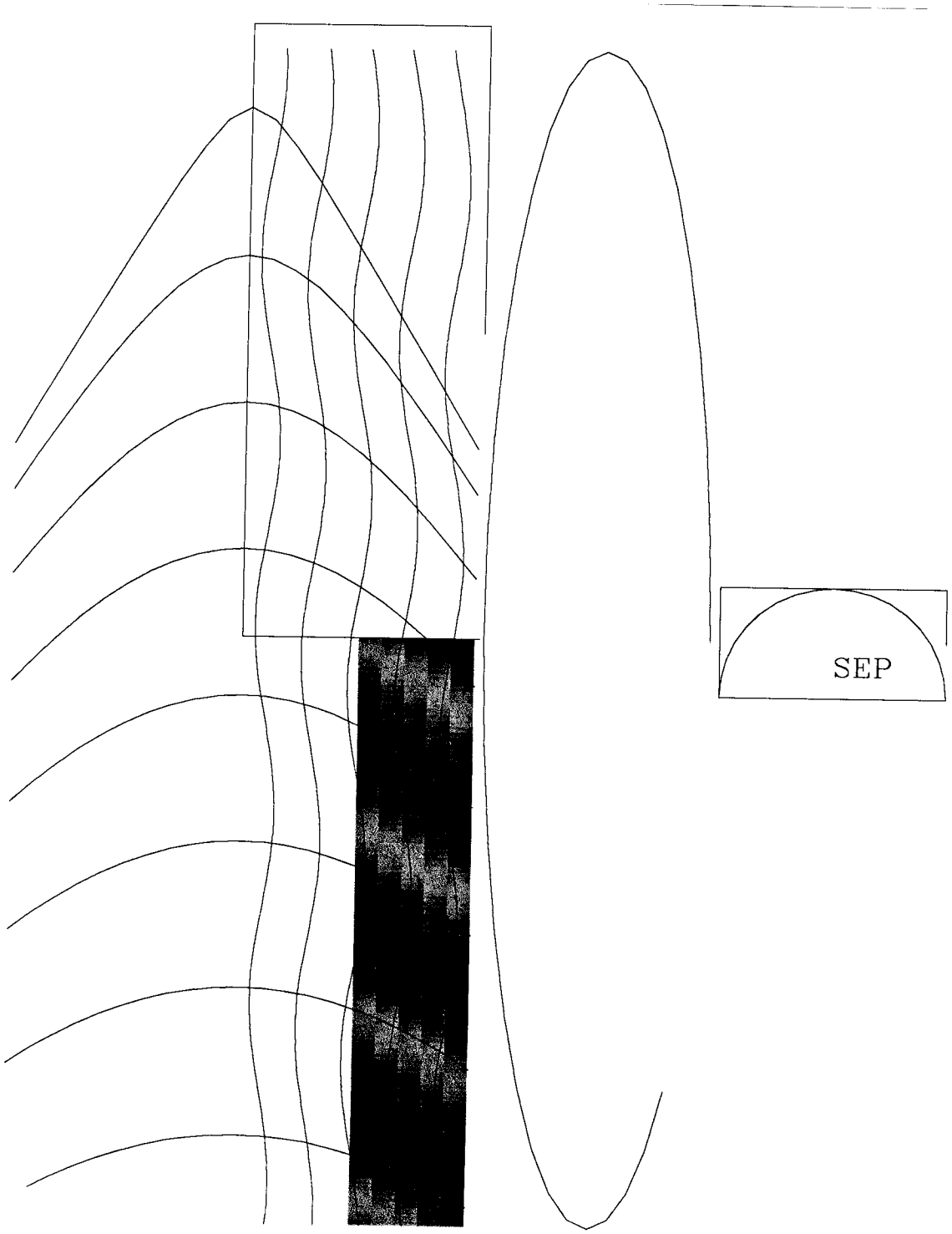


FIG. 3. Test case for isotropic and nonisotropic windows and frames.

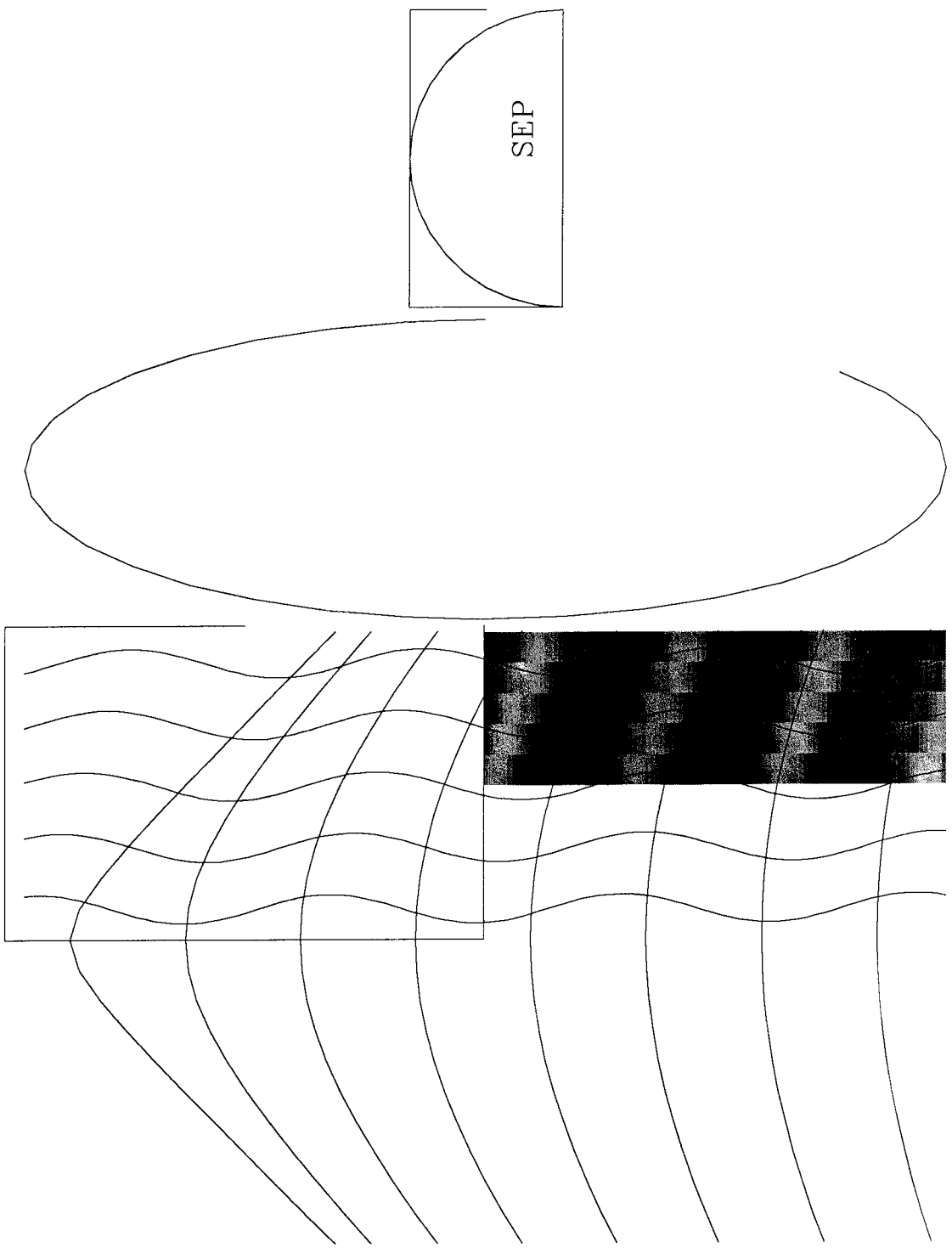


FIG. 4. Test case after portrait-landscape transformation.

- Need `gi_interrupt()` for compute intensive interruption. (JFC)
- Official SEP color manager (Rick or Dave)
- Vector clip subroutine to code from Foley and Van Dam's pseudo Pascal. (anyone at SEP) At present we just throw out vectors that have both ends outside the window.
- extract crude area subroutine from svplot (JFC)
- Good quality area routine (Joe)
- Good quality text. (Jos or joe)
- Overlays—transient movable shapes like in svplot. This cannot be done effectively until Rick finishes the official SEP color table management system. (JFC, Dave, Jclau)

## REFERENCES

Claerbout J., 1988, Overlay plotting with svplot, SEP-57, p. 539-548.

Dellinger, J., 1989, Vplot, this report.

Foley and Van Dam, Fundamentals of Interactive Computer Graphics.

Nye, A., 1989, Xlib Programming Manual for Version 11, O'Reilly & Associates, Inc.

Nye, A., 1989, Xlib Reference Manual, ibid.

Sunview programmers manual, Sun Microsystems Inc.