

DSGI: The OTHER Way To Do SAS Graphics

C. Michael Whitney - USAF Environmental Technical Applications Center

Abstract

This paper will discuss the use of the SAS® Data Step Graphics Interface (DSGI). DSGI provides a flexible means of creating graphics and is able to handle most of your special graphic needs. The basics of DSGI will be discussed and we'll see how complex graphs can be created using simple shapes.

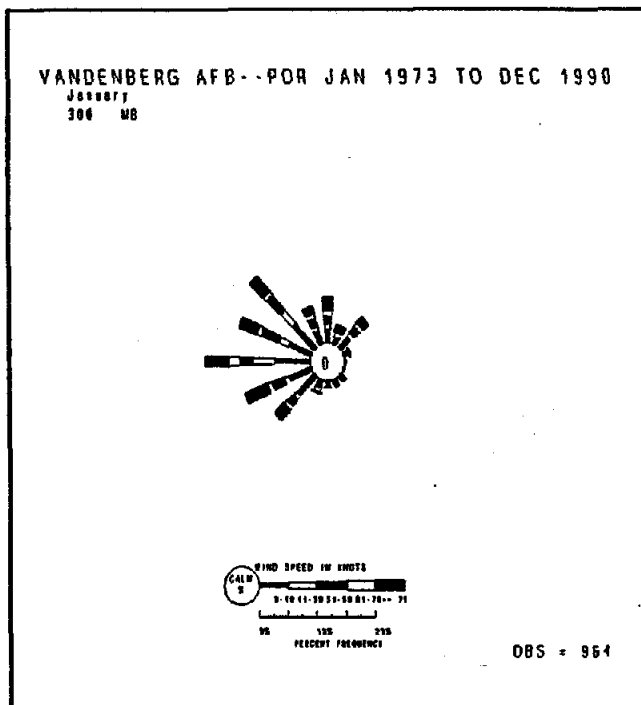
Introduction

In early 1991, I was handed a program called 'Windrose', and told it was now my baby. Windrose created a graphic representation of the average wind speeds and directions, and the frequencies of their occurrence for a given site. At that time, I was unfamiliar with SAS graphics. The graphics generation routine for Windrose was being handled by something called PROC ANNOTATE. I had no idea what it all meant, yet I was being asked to change the program to create multiple graphs for each site, by user-specified heights. What to do? Well, it made more sense to me to handle all the graphics generation from within the Data Step, but how to do it? Ah-hah!...SAS has the Data Step Graphics Interface that does exactly what I needed to do.

Comparison With the Annotate Facility

Both Annotate and DSGI allow you to enhance graphics output produced by other SAS/GRAPH procedures. Most of the SAS graphics tools allow for calling in an Annotate data set for this purpose, making modifications via Annotate a lot easier than doing it with DSGI.

DSGI is best used when creating custom graphics without another graphics procedure. With the Annotate facility, you must first create an Annotate data set and then submit a PROC step to display the graphics output. DSGI allows you to do all of this within the DATA step, eliminating the need for a separate PROC to be run. Since this extra step doesn't need to be performed, DSGI graphic programs are generally faster than Annotate ones. DSGI functions



created on one screen.

There are also five operating states: GKCL (graphics kernel closed), the initial state of DSGI, where no graphics resources have been allocated; GKOP (graphics kernel open), where you can check the attribute settings; SGOP (segment open), where you can generate graphics; WSAC (workstation active), when active, you can issue commands; and WSOP (workstation open), which allows access to the graphics catalog.

There are five types of commands: utility functions that allow you to initialize DSGI; GRAPH commands to perform graphics library management functions; GASK routines, that allow you to check the current attribute settings; GDRAW functions, that allow you to create your graphic elements; and GSET functions that allow you to set the attributes of the graphics elements that your GDRAW commands will use.

The basic DSGI output area is region of your screen, using a cartesian coordinate system of 100 x units by 100 y units. (0,0) is in the bottom left while (100,100) is in the top right. The center point is at the (50,50) coordinate location. These units are arbitrary, and can be set to some other amount if you wish, say -200 through 200 x, -50 through 50 y. I've never had a need to change from the default.

Using DSGI

I generally create the dataset I want to use in generating my graphs before the actual data step in which I use them. For windroses, I create a dataset containing the month, the wind directions, wind speeds in 5 categories, and the frequency of occurrence of the winds speeds per direction. The dataset sometimes has the height and pressure levels as well. Once you have your data, you're ready to start building your graphs!

DSGI commands are issued as function calls, in the form of

```
return_code = function(operator, arguments);
```

where return_code is a numeric variable. A good run returns a 0, and a not so good returns a wide variety

of numbers. The SAS/GRAPH Software Vol 1 book lists each commands possible return codes in the DSGI Interface Dictionary sections, with a complete list of their meaning in the DSGI Return Codes section. Both of these sections are located in chapter 21.

Initializing DSGI

To initialize the system, we use GINIT(). I also set the HSIZE and VSIZE of my output in this portion of code as well.

Here's a description of the commands we will use to do our initialization:

GINIT() - Initializes the DSGI system. You can't do anything with it until this is done.

GSET('HSIZE', n); - Used to set the horizontal size of the graph, same as HSIZE= goption.

GSET('VSIZE', n); - Used to set the vertical size of the graph, same as VSIZE= goption.

GSET('CATALOG', libref, catalog-name); - Allows graphs to be copied into a specified library.

GRAPH('CLEAR' <name><des><byline>); - Opens a graphs for output into current catalog. CLEAR is the only required parameter. Name, description and byline will appear in catalog listings and info.

GSET('LINCOLOR', color-index); - Sets the line color, using color-index. color-index is between 1 and 256.

GSET('TEXALIGN', halign, valign); - Sets the alignment of your text. halign is the horizontal alignment, and can be center, left, or normal, or right. valign is the vertical alignment, and can be top, half, base, or bottom.

GSET('TEXCOLOR', color-index); - Sets the text color, using color-index. color-index is between 1 and 256.

GSET('TEXFONT', font); - Sets the font, where font is any valid SAS font name enclosed in quotes.

GSET('TEXHEIGHT', *height*); - Sets the height of your text, using a numeric value greater than 0.

Now, let's see how all this goes together. We'll create a macro we can call later to initialize our DSGI session, and set up some defaults.

%MACRO INITDSGI;

/* Set graphic size to 16x11 */

RC=GSET('HSIZE', 16);
RC=GSET('VSIZE', 11);

/* Set up the graphics library & catalog to use.
(We'll use Work and Test) */

RC=GSET('CATALOG','WORK',
TEST);

/* Initialize the DSGI system, and clear the graph
screen, calling it 'Demo' */

RC=GINIT();
RC=GRAPH('CLEAR', 'DEMO');

/* Initialize the text alignment, text height, text
font, and colors here */

RC=GSET('TEXALIGN', 'CENTER',
HALF);

/* Set text color to black, which is the first
color on my device */

RC=GSET('TEXCOLOR', 1);

/* Set text height to 2 units, and the font to
one of the CGM hardware fonts */

RC=GSET('TEXHEIGHT', 2);
RC=GSET('TEXTFONT', "HGWCGM01");

/* Set the line color to black*/

RC=GSET('LINCOLOR', 1);

%MEND INITDSGI;

Ending DSGI

To end DSGI, we use the GTERM() command.

The commands we will use are:

GRAPH('UPDATE'<,<show>); - Closes the graph and shows it, if the *show* keyword is included.

GTERM(); - Ends the graph, and closes DSGI.

%MACRO ENDDSGI;

/* Close off graph & output */

RC=GRAPH('UPDATE');

/* End DSGI */

RC=GTERM();

%MEND ENDDSGI;

Building a Windrose

Circles

The first thing we need is an inner circle, and to make one, we use this command:

GDRAW('ARC', *x*, *y*, *radius*, *start*, *end*); - Draws a circular arc. *x,y* are the origin points; *radius* is the length of the radius; *start* is the starting angle, 0 is at the 3 o'clock position; *end* is the ending angle (counter clockwise from our 0 position).

DATA _NULL_;

/* Initialize DSGI */

%INITDSGI;

/* Make a circle centered at (50,50) with a
radius of 3. Start at 0 and end at 360. */

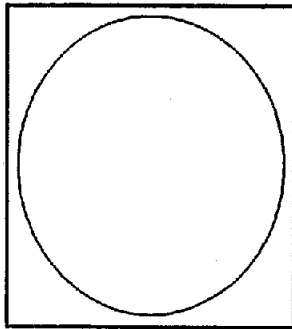
RC=GDRAW('ARC', 50, 50, 3, 0, 360);

/* End DSGI */

```
%ENDDSGI;
```

```
RUN;
```

We end up with a circle like this (but smaller):



Polygons

The next step in creating our graph is to draw the 'leaves' of the rose. The commands used are:

GDRAW('FILL' *n*, *x-vals*, *y-vals*); - Draws a filled polygon. *n* is the number of sides the polygon has; *x-vals* is a list of the x coordinates for each vertex point; *y-vals* is the same for the y coordinates.

GSET('FILTYPE', *interior*); - Sets the fill type for interior fills. *interior* can be set to hatch, hollow, pattern, or solid.

I used some simple trigonometry to calculate the start and end points of each leaf, from a starting X,Y coordinate, at the edge of the circle, to an ending one several units out (determined by the percent frequency of the particular windspeed in the direction indicated by the leaf).

X_4, Y_4	X_3, Y_3
X_0, Y_0	X_2, Y_2
X_1, Y_1	

We start at our origin X_0, Y_0 and draw our lines connecting our calculated corner points, ending at the origin. After we're done, we set X_1, Y_1 to be our new origin for the next leaf. The distance each corner X,Y is from the origin depends on how great the

windspeed grouping is. The faster the winds, the thicker the leaf.

Here's some example code for creating a windrose, using our circle and the filled polygon:

```
%MACRO DRAWROSE;
```

```
/* Our circle from the last example */
```

```
RC=GDRAW('ARC', 50, 50, 3, 0, 360);
```

```
/* Set up arrays holding our trig values to calculate
our leaf angles, and the corners of our leaf
boxes. */
```

```
/* Sines of X and Cosines of Y, in increments of
22.5: sin 0 = 0, sin 22.5 = .383, etc
cos 0 = 1, cos 22.5 = .924, etc */
```

```
ARRAY DX(0:15) (0 .383 .707 .924 1 .924
.707 .383 0 -.383 -.707 -.924 -1 -.924
-.707 -.383);
```

```
ARRAY DY(0:15) (1 .924 .707 .383 0 -.383
-.707 -.924 -1 -.924 -.707 -.383 0 .383
.707 .924);
```

```
/* Subtract 90° from each angle, get the sines of X
and cosines of Y. This is because winds are
reported with north being 0, while math starts
with east being 0. */
```

```
ARRAY PDX(0:15) (-1 -.924 -.707 -.383 0
.383 .707 .924 1 .924 .707 .383 0
-.383 -.707 -.924);
```

```
ARRAY PDY(0:15) (0 .383 .707 .924 1 .924
.707 .383 0 -.383 -.707 -.924 -1 -.924
-.707 -.383);
```

```
/* Array that holds the values of our leaf thickness.
*/
```

```
ARRAY WIDTH(1:5) (.25 .5 .5 .75 .75);
```

```
/* Set our initial X-Y locations to work from */
```

```
INITX = 50;
INITY = 50;
```

```
/* In the real world, we'd have 16 wind directions,
   but we'll just do a loop to simulate them. */
```

```
DO ANGLE = 0 TO 15;
```

```
/* Set the origin for each leaf set. The multiplier
   of three sets the origin at the edge of our
   circle. We get the angle from our DX and DY
   array. */
```

```
ORIGX = INITX + DX(ANGLE) * 3;
ORIGY = INITY + DY(ANGLE) * 3;
```

```
/* Normally, we'd have 4 or 5 wind speed
   categories, but we'll just loop through instead.
   */
```

```
DO LEAF = 1 TO 5;
```

```
/* And, the length of each leaf is controlled by the
   frequency of observations matching the direction
   and speed category group. However, we'll just
   put a random length in here for this
   demonstration. */
```

```
RANDLGTH =
  RANUNI(ANGLE*LEAF) * 4;
```

```
/* Now we'll set the end coordinate of the leaf by
   getting the right trig angle from our array and
   multiplying it by our randomly generated length.
   */
```

```
ENDX = ORIGX + DX(ANGLE) *
  RANDLGTH;
ENDY = ORIGY + DY(ANGLE) *
  RANDLGTH;
```

```
/* We calculate the corner points of our leaf in the
   next segment, using our angle and width arrays.
   This moves our points a tad to either side of
   our origin and end points */
```

```
P1X = ORIGX - PDX(ANGLE) *
  WIDTH(LEAF);
P2X = ENDX - PDX(ANGLE) *
  WIDTH(LEAF);
P3X = ENDX + PDX(ANGLE) *
  WIDTH(LEAF);
P4X = ORIGX + PDX(ANGLE) *
  WIDTH(LEAF);
```

```
PIY = ORIGY - PDY(ANGLE) *
  WIDTH(LEAF);
P2Y = ENDY - PDY(ANGLE) *
  WIDTH(LEAF);
P3Y = ENDY + PDY(ANGLE) *
  WIDTH(LEAF);
P4Y = ORIGY + PDY(ANGLE) *
  WIDTH(LEAF);
```

```
/* We want every other leaf to be solid colored,
   alternating with hollow ones. */
```

```
IF LEAF IN (2,4) THEN
  INTERIOR = 'HOLLOW';
ELSE
  INTERIOR = 'SOLID';
```

```
/* Now we actually draw our leaf. We use the
   FILTYPE to set solid or hollow. With the FILL
   command, we're connecting 5 points, starting
   and ending at our origin. */
```

```
RC = GSET('FILTYPE', INTERIOR);
RC = GDRAW('FILL', 5, P1X, P2X,
  P3X, P4X, P1X, P1Y, P2Y, P3Y,
  P4Y, P1Y);
```

```
/* Now, set the origin coords to our end coords
   so that our next leaf starts where the first left
   off. */
```

```
ORIGX = ENDX;
ORIGY = ENDY;
```

```
/* End our loops. */
```

```
END;
END;
```

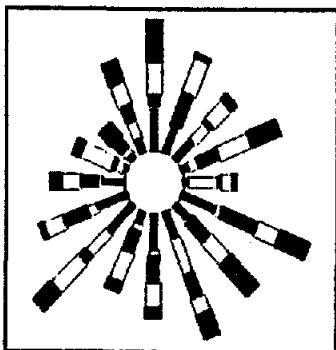
```
%MEND DRAWROSE;
```

```
DATA _NULL_;
%INITDSGI;
```

```
/* Draw our rose */
```

```
%DRAWROSE;
%ENDDSGI;
RUN;
```

Here's our output from this step:



Text

Looks pretty good, but we need some text! Text is handled by the following command:

`GDRAW('TEXT', x, y, string);` - Prints text string at x-y coordinates. The string can be either a quoted text, or a variable that equates to a string of text.

And we use it like this:

```
%MACRO DRAWTEXT;
```

```
/* We normally put the number of calm  
winds in the middle of our rose. We'll just  
put a zero in there. */
```

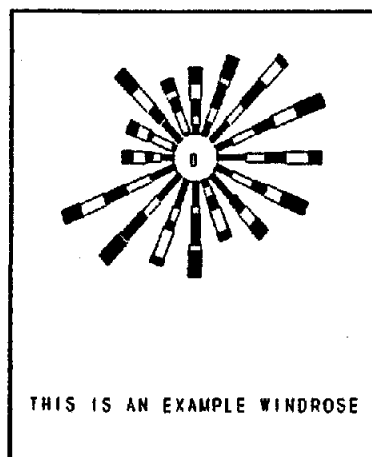
```
RC=GDRAW('TEXT', 50, 50, '0');
```

```
/* And all good graphs have a title of some sort.  
Here we set a variable to hold our string and  
place it below our graph. */
```

```
WRTITLE = 'AN EXAMPLE WINDROSE';  
RC=GDRAW('TEXT', 50, 20, WRTITLE);
```

```
%MEND DRAWTEXT;
```

After we add our text macro to our code, the output looks like this:



Lines

I use the line command to draw the lines in my legend. I'm not going to show an example of it here (because I re-draw a rose to make the legend, then add the extra graphics), but here is the syntax to use it:

`GDRAW('LINE', n, x-vals, y-vals);` - Draws a line, or series of connected lines. *n* is the number of corner points used; *x-vals* is a list of the x coordinates for each vertex point; *y-vals* is the same for the y coordinates.

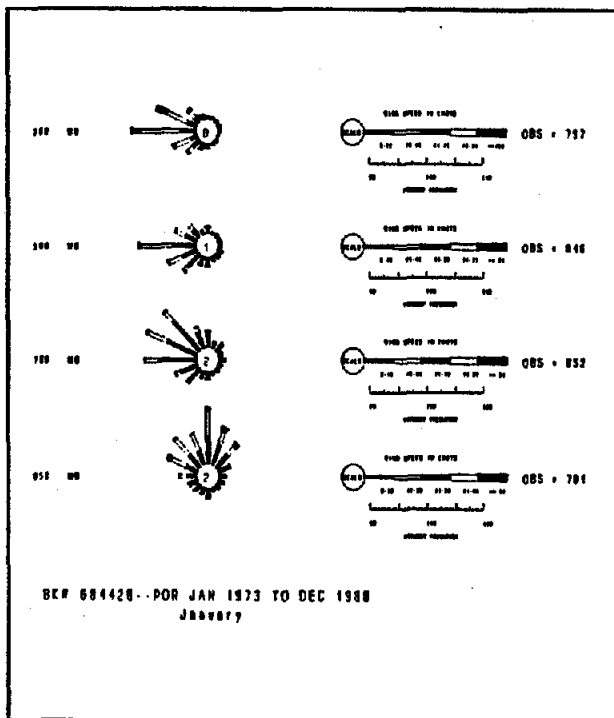
In the windrose program, I calculate the position for each little segment in the legend scale by looping through and moving a bit over each time, drawing a segment each time.

Other Graphs Created by DSGI

Here's a couple other graphs that are created in the Windrose program, using DSGI, and a brief description of how they are created using our graphic basics:

Multiple Roses On One Chart

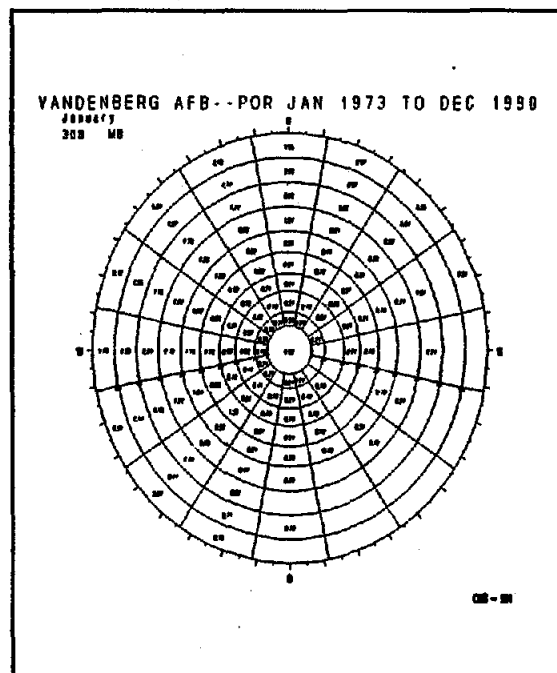
In this chart, we don't close the DSGI, but rather change the origin point of each graph. Also, the origin of the legend has been shifted to the right of each rose, instead of under it.



Engineering Rose

The following rose type was requested by an engineering client, who wanted hard numbers instead of the graphical rose. For this, the bulls-eye was created using the Arc and Line commands, and the numbers were placed using Text command.

Note that this graph is for the same data as the graph on the first page of this article.



Conclusions

Hopefully, this is enough to get you started using the Data Step Graphics Interface. As I've previously shown, these basic building blocks can be combined in many ways to build complex graphs. DSGI is a very useful tool when creating custom graphs.

References

SAS Institute, Inc (1990), SAS/GRAPH Software: Reference, Version 6, First Edition, Volume 1, Cary, NC: SAS Institute, Inc.

SAS and SAS/GRAPH are registered trademarks of SAS Institute, Inc in the USA and other countries.

Contacting the Author

C. Michael Whitney, SSgt, USAF
USAFETAC/SYS
859 Buchanan St, Rm 401
Scott AFB, IL 62225-5116
(618) 256-5323 DSN 576-5323
Internet: syscmw@etacrs1.safb.af.mil