Paper 92-27

Your Shingle in SAS/Graph®

David Johnson FRSA, DKV-J Consultancies, Holmeswood, England

ABSTRACT

SAS/Graph software is a very powerful tool you can use in deploying images and graphical representations of data. This paper explores the use of the SAS/Graph product in Versions 6.12 and 8.2 to develop icons for a web page and graphical symbols for other publishing.

The most complex of these is the company 'Pecten'. The Pecten is the corporate symbol you see on the company's letterhead, their packaging or at the entry to the company's premises (whether the virtual premises of their website, or their actual building).

We will look at ways of drawing this image from within the SAS system. By using the SAS System, we give ourselves substantial power to calculate positions and colour gradients that would otherwise rely on a steady hand and the forgiving nature of our image editor.

It also makes it possible to generate a group of images, and make a global change to them by some simple changes to the source SAS programs.

This paper will also look at the use of the output drivers available to the SAS System, especially the GIF and BMP drivers that allow image export to other applications, and our web pages.

THE PROCESS

We will look at building your web icons and final shingle by breaking the task down into a series of smaller pieces.

- What constitutes a '3D' or three dimensional appearance, and how to achieve it in SAS/Graph
- Setting up your SAS/Graph environment for a graphic file output, and setting the SAS/Graph options.
- Setting some SAS/Graph system options and defining colours using Hexadecimal notation and RGB colour definitions.
- Adding SAS/graph special fonts to create a SUGI shingle
- Creating data defined images for web pages in batch processes.

THE OPTICAL ILLUSION THAT IS '3D'

A three dimensional object is perceived by our brain when the eye sees a change in the object that seems to suggest a change in illumination, or a change in focus. Look at the appearance of a push button on your SAS toolbar, and the upper and left edges appear to be lighter than the right and lower.

Our eyes interpret this lightening effect as increased illumination from a light source to the upper left of the object. Consequently, we see the object as raised. If we now swap the lighter area with the darker area, the image appears to be recessed.

To produce this effect we can instruct SAS to start at the bottom left corner of a square, and draw light coloured lines to the top left and top right corners. Then if we draw darker coloured lines to the bottom right corner and back to the bottom left, the square will appear to be raised.

To produce the square, we can identify each corner to the SAS System by using the co-ordinates we are familiar with on a graph. The corners can then be defined as in the table at figure 1.

Corner	X	Y
1	0	0
2	0	100
3	100	100
4	100	0

Figure 1

To use these co-ordinates, we will associate the positions with the instructions needed to draw the square. We will instruct SAS to 'MOVE' to the first corner, then 'DRAW' in turn around the square. At figure 2 is the table after we have added these commands. We have also added the colouring required for each vector.

Obs	X	Y	FUNCTION	COLOUR
1	10	10	MOVE	
2	10	90	DRAW	STEEL
3	90	90	DRAW	STEEL
4	90	10	DRAW	CHARCOAL
5	10	10	DRAW	CHARCOAL

Figure 2

The column names are variable names that the SAS Graph engine will understand. If we create a SAS data set with these five observations, we can pass the commands to the SAS/Graph engine. We do this through a function called ANNOTATE.

If we invoke the SAS/Graph procedure, and pass this data set as an annotate data set, then the engine will use the commands to draw our square, and colour it appropriately. To easily enable us to see the lines we draw in this demonstration, I have moved our positions a little inside the original boundaries.

We will use the GANNO Procedure because this is simplest for this purpose, although most SAS/Graph procedures will also honour the information contained in an Annotate data set. Here is some of the code used, and the resultant image is at figure 3.

```
Data ANNO;
  Retain SIZE 2;
  FUNCTION = 'MOVE';
    X =  10;  Y =  10; Output;
  FUNCTION = 'DRAW';  COLOR = 'STEEL   ';
    X =  10;  Y =  90; Output;
  FUNCTION = 'DRAW';  COLOR = 'STEEL   ';
    X =  90;  Y =  90; Output;
  FUNCTION = 'DRAW';  COLOR = 'CHARCOAL';
    X =  90;  Y =  10; Output;
  FUNCTION = 'DRAW';  COLOR = 'CHARCOAL';
    X =  10;  Y =  10; Output;
Run;

Proc Ganno Anno = ANNO;
Run;
```

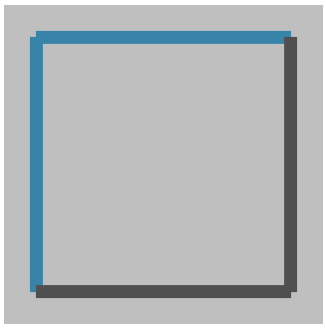

Figure 3

You will notice that the corners of the image are very square and 'chunky looking'. To overcome this we need to give our drawing the appearance of a raised area. We do this by making each side meet the next one on a diagonal that bisects the square through the centre. This gives the appearance of an area gradually raised above the surface. In figure 4 I have added red lines over a watermark of figure 3 diagram to demonstrate the bisecting diagonal. The meeting point will be along these red lines.

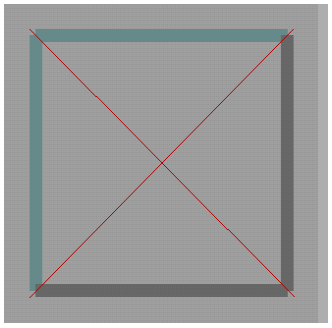

Figure 4

At figure 5 is a detail from a SAS/AF frame, a push button in a container box. The container box appears to be raised, as does the button, because of our use of shading, and the diagonal intersection at each corner.

Figure 5

To get this effect, let's examine the details of the lines we have drawn in figure 3. The width of the line is 2%, as defined in our size parameter in the retain statement. This means that a line starting at 10% across the picture will in fact occupy the space from 9 to 11%. The upper corner then occupies from 89 to 91% and so on around the square. This produced the 'block overlay' effect in our own example. If however we draw very thin lines, and gradually move our position inward from 9% to 11%, we should create the effect of the button we see from the SAS/AF frame depicted in figure 5.

It may seem tedious to calculate the positions, but in fact we can use a loop in our ANNO data step to calculate each intersect. Here is one way we might do this.

```
Data ANNO;
  SIZE = .2;
  ** Move the pointer to the first corner;
  FUNCTION = 'MOVE';  X =  9;  Y =  9; Output;
  POSMIN = 9;  POSMAX = 91;  LOOPMAX = 40;
  FUNCTION = 'DRAW';
  ** We will draw forty very small lines;
  Do LOOP = 1 To LOOPMAX  By 1;
    ** We will alter our drawing definitions once for each side;
    Do SIDE = 1  To 4  By 1;
      ** Top and left are in the light shade and the end Y
values are for the upper edge;
      If SIDE <= 2  Then  Do;
        Y = POSMAX - (LOOP / 10);
        COLOR = 'STEEL    ';
      End;
      ** Right and bottom are darker and the end Y value is the
lower edge;
      Else Do;
        COLOR = 'CHARCOAL';
        Y = POSMIN + (LOOP / 10);
      End;
      ** The second and third sides all draw to the higher X
value;
      If 2 <= SIDE <= 3  Then X = POSMAX - (LOOP
/ 10);
      Else X = POSMIN + (LOOP / 10);
      Output;
    End;
  End;
Run;
```

If you are uncertain about the logic of these calculations, step around the square, and define the X and Y values and their associated colours from the table above.

At figure 6 is the button we have drawn. Note that for clarity, we have drawn quite thick lines for this example. In this case, the sides comprise forty lines each 0.2% of the total picture width.

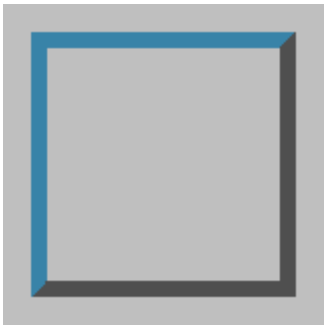

Figure 6

To style up our button a little, we will add a fine black line all the way around the outside, as an indication of the cavity into which the button will be depressed. If we also add a very light or white line on the inner edge of the icon, it highlights the 'upper' area of the image. The following additional code in our data step prior to the output statement performs these functions. The resultant button image appears at figure 7.

```
If LOOP <= 10 Then COLOR = 'BLACK';
Else If LOOP >= LOOPMAX-4 Then COLOR =
'WHITE';
```

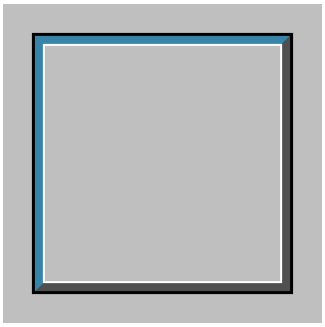

Figure 7

SETTING THE GRAPHIC ENVIRONMENT

If you have been attempting to reproduce the images you have seen, the results may not be as demonstrated. So, having now shown you what can be achieved, let's get a couple of 'set-up' issues out of the way.

To get the grey background, you will need to define the colour of the background appropriately. This is done with a graphic options, or 'Goptions' statement. The following statement is the one I have used, and each of the parameters is explained in turn.

```
GOptions CBack = CXA8A8A8 /* Set the background colour, */
         GsfName = GIF   /* The name of the Graphics stream file. Here it is a file reference, not a file name */
         GsfMode = Replace /* Replace existing entries in the file */
         Dev = SUGIPECT /* The name of the device driver, this is a custom device */
         HText = 2   /* The default height of text, here it is 2 'Units' high */
         GCopies = 0 /* Number of copies to print, 0 specifies 1 graphic output */
         HSize = 4in /* Horizontal size of the graphic image; 4 inches */
         VSize = 4in /* Vertical size of the graphic image; 4 inches */
         NoErase; /* Should graphic be displayed, then erased? */
```

Of the above options, two may be unclear. The first is the colour used in the background of the image. Rather than specifying 'GRAY', we are specifying a shade of grey using 'RGB' colour coding. This method involves specifying the intensity in turn of the Red, Green and Blue components.

We can specify up to 256 levels for each colour, by using a hexadecimal notation. Since A8 is hexadecimal for 168, in this case, the three colours are at 168/256 or 65% saturation. This is quite a light grey, as you can see.

The other option that will be unclear is the specified Graphics device. The name 'SUGIPECT' refers to an entry in a graphics catalog. Using the following code creates the actual entry.

```
/* Assign a library reference to contain the user-defined graphics entries */
  Libname  GDevice0  "D:\Saswork\V6";

  Proc Gdevice   NoFs /* Open the Procedure in batch mode*/
                 Cat = GDEVICE0.DEVICES; /* Use the user defined catalog assigned above */
    Copy GIF    From = SASHELP.DEVICES /* Copy GIF entry from the SAS supplied catalog */
                 Newname = SUGIPECT; /* Give it a user specific name */
    Modify  SUGIPECT  XPixels = 320
                      YPixels = 320 /* Modify the size of the addressable display area to be 320 pixels square */
                      XMax = 4in
                      YMax = 4in; /* Similarly specify the size in inches. This is a low resolution setting, 80 pixels to the inch approximates to a screen display size, rather than a printed format */
  Run;

  /* Label the new entry in the graphics catalog so its purpose is clear to other users */
  Proc Catalog Cat = GDEVICE0.DEVICES ET = Dev;
    Modify SUGIPECT(
      Desc = 'GIF, 4" square SUGI demo');
  Quit;
```

Now that you have the Gdevice procedure code, you can create a suitable device specification for your own needs.

It is just as easy for you to create a custom BMP driver, if you want to make bitmap images available. Be aware though that the code presented here generates vector diagrams, which lend themselves very well to a compressed file format like GIF. Using BMP will create substantially larger images, which will take longer to download, and not provide any obvious improvement in quality.

WHAT ARE IMAGE FORMATS?

The BMP image format is a way of representing each pixel of an image in terms of its colour and brightness. Since we are providing individual data on each pixel, or 'bit' of the image, the file format is called 'Bit Mapped Picture'. Clearly also, a large image will take up a lot of storage space, because each pixel is represented individually in the file definition. BMP image files can be rendered with SAS/Graph by using the IMGBMP or IMGPAINT device drivers, or variations you create on these drivers.

The GIF format was released by Unisys and uses a system of compression called LZW, named after its creators. The compression system works by identifying adjacent pixels in a picture that have the same colour value. Then the adjacent pixels are stored with an abbreviated reference to the adjacent pixel. This reduces the size of the file.

The compression technique is called a 'lossless' compression, because no detail is lost during the compression process. However, it is best suited to images with lines and blocks of colour, where there are large areas with equal colour values. This is why GIF compression works well with vector graphics, which is what we are drawing here. GIF image files can be rendered in SAS/graph by using the IMGGIF, GIF, GIF160, GIF260, GIF373, GIF570 and GIF733 device drivers, or variations you create.

One other image type is worth mentioning; the JPEG image type. This is a compression system developed for photographs by the Joint Photographic Experts Group who used their acronym to name the format. It works by defining the amount of colour shift from between pixels in a block. It is a 'lossy' compression, because the threshold for colour shift can be changed which will cause areas with subtle differences to be rendered in a single colour. It can also cause areas of the same colour to be rendered with variations, because of the pixel grouping process. You can render a SAS/Graph image in JPEG, by using the IMGJPEG device driver, but the corruption of detail it can introduce to the image makes it a poor choice.

Figure 8

SYSTEM SETTINGS AND COLOURS

To get the code to work properly, there are a couple of minor additions to make. You need to instruct the graph procedure output engine on the co-ordinate system that you will use for the output. The SAS/Graph co-ordinate system is specified as laid out in figure 9 below.

Area to use	Unit	Absolute co-ordinates	Relative co-ordinates
Data	%	1	7
	Values	2	8
Graphics output area	%	3	9
	Cells	4	A
Procedure output area	%	5	B
	Cells	6	C

Figure 9

- I usually specify that I will write to the graphics output area. This means that I have control of the whole output page.
- I also specify that my sizes will be defined as percentages, so that if I change graphics output device from 4 inches square to 10 inches square, I don't have to recalculate my positioning.
- Finally, I specify in absolute, rather than relative dimensions. This way I don't have to keep track of my position in terms of the object last drawn.

Instructing the SAS/Graph engine on the co-ordinates in use requires three extra values on the Annotate data set. The first two relate to the X and Y co-ordinate systems, and are assigned with variables called 'XSYS' and 'YSYS'. (If we were using the G3D procedure, which is a procedure to produce three-dimensional representations, then we would also specify 'ZSYS' for the third dimension.)

The third co-ordinate system we must specify relates to the 'SIZE' variable. This is specified with 'HSYS'. In our Goptions statement above, when we specified the Hsize to be "2 units high", we were specifying that it would use units as specified in the 'HSYS' variable. In this case, we want to use 2% of the graphics output.

Referring to our table at figure 9 above, we will use the value 3. (Note that the table is available in your SAS/Graph documentation in the section called 'The annotate data set - about annotate graphics'.) The following line can be added to our data step to use the highlighted value 3.

```
Retain  HSYS  XSYS  YSYS  '3';
```

I'll offer a last word on colours, before we move on to demonstrate the full code. When you develop a graphic image, give a thought to the place where it will be used. The sizing we specified above, as the comment indicates, takes account of the size of the final image on a computer screen. This particular device entry was designed for web display. Similarly, when we colour the graphic image, we need to allow for the limitations of the web browser.

The standard web browser controls a palette of 256 colours. To ensure we get a suitable colour range for our images, it helps if we map out a palette that will give us a comprehensive range of colours. To do this, I assign my colours evenly spread across the colour spectrum.

If I specify the combination of colours produced from 6 levels each of red, green and blue then my colour palette will compass 6*6*6, or 216 colours. To get my evenly spaced colours, I specify them in RGB format, and I use the hex values 33, 66, 99, AA, CC and FF to get my spread of colour. This also leaves me with a handful of colours left over for special purposes, including the grey I have used for the background.

In the code we have already written then; I redefine 'WHITE' to be 'CXFFFFFF' and 'BLACK' to be 'CX000000'. If I now want to produce an icon with grey sides, my light side can be defined with 'CXCCCCCC', and the darker side with 'CXAAAAAA'. At figure 10 is the icon again, using the whole graphic output area, with grey sides, narrower edges, and a text annotation specific to this paper.

Figure 10

PRODUCING YOUR SHINGLE

The Hardest part of our shingle has been getting the three dimensional appearance. Now we have solved the problems of highlighting and shading certain parts of the image, producing a shingle is only a matter of designing the required layout.

Let's explore some of those options with our SUGI icon. We'll give the icon the appearance of depth, by indenting the top of the button, and creating a shadow of the button label.

To create the appearance of depth, we repeat the process that created the edge. We will create a sloping surface that is half the size of the outer edge. If we reverse the colouring we applied for the outside, the upper surface of the button will appear to recess.

```
/* Initialise the outer edges to the edge of the outer slopes */
   POSMIN = X;
   POSMAX = 100 - POSMIN;
   LOOPMAX = 30;
   Do LOOP = 1  To  LOOPMAX  By 1;
     Do SIDE = 1  To  4  By 1;
       If 2 <= SIDE <= 3  Then
          X = POSMAX - ( LOOP / LOOPMAX);
       Else X = POSMIN + ( LOOP / LOOPMAX);
       If SIDE = 1  Then Do;
```

```
      Function = 'MOVE';  Output;
      Function = 'DRAW';
    End;
    If SIDE <= 2  Then Do;
      Y = POSMAX - ( LOOP / LOOPMAX);
/* reverse the colour pairs */
      COLOR = 'CX999999';
    End;
    Else Do;
      Y = POSMIN + ( LOOP / LOOPMAX);
      COLOR = 'CXCCCCCC';
    End;
    Output;
  End;
End;
```

To reinforce the appearance of depth in the upper surface of the button, we can create a shadow on the text. To achieve this, we write the text first in a darker gray, and offset it from the final position. Then we write the final text, in the blue colour, in its normal position. Here is some code we can use to create the two text parts.

```
Function = 'LABEL';
STYLE="ZAPFB";   Text = 'SUGI';    SIZE = 30;
COLOR = 'CX666666';    X = 51;  Y = 79;
Output;
COLOR = 'CX0000CC';    X = 50;  Y = 80;
Output;

STYLE="CENTXI";  Text = '27';    SIZE = 40;
COLOR = 'CX666666';    X = 51;   Y = 39;
Output;
COLOR = 'CX0000CC';    X = 50;   Y = 40;
Output;
```

At figure 11 is our icon image. When we compare it to the previous images, the few simple changes we made to give it the appearance of depth have made it a much more interesting image.

Figure 11

The simple tricks we have explored thus far have given us a quite reasonable icon we can use. If it seems familiar, perhaps you may have seen it in use on our website. For a simple plain effect, it is probably more than adequate. But there are a couple of touches you can add, if you want it to appear more like a shingle that is inviting a visitor into your website.

USING SAS/GRAPH® SPECIAL FONTS

The SAS/Graph documentation provides you with a full list of the fonts available in the product, as well as guidance on loading your own font set. The special font sets are often overlooked. My musical interests have directed me to the 'MUSIC' font on more than one occasion. They also led me to the 'CARTOG' or cartographic font when I wanted to generate an animated GIF in SAS/Graph, although that's the topic for another paper.

You may find something of interest in the 'SPECIAL' font. The lower case 'd' in this font is a circle with a cross in it. I first noticed it when I wondered if I could generate the appearance of a brass plate, or shingle screwed to the virtual wall of my website.

Here is the code I would use to add two such symbols to our SUGI icon. The modified shingle follows at figure 12, with an addition from the 'CARTOG' font to portray the tropical environs of Florida.

```
/* Create a dark yellow fill for the screw head */
   STYLE="MARKER";  Text = 'W';    SIZE = 4;
   COLOR = 'CX999900';    X = 8;    Y = 49;
   Output;
   COLOR = 'CX999900';    X = 92;   Y = 49;
   Output;
/* Overlay appearance of the screw face */
   STYLE="SPECIAL";  TEXT = 'd';    SIZE = 5;
   COLOR = 'CX333333';    X = 8;    Y = 50;
   Output;
   COLOR = 'CX333333';    X = 92;   Y = 50;
   Output;
```

Figure 12

The image at figure 13 is an early design we recently phased out of our web site in favour of a more elegant image. The image has the following key attributes:

- A graphics device driver mapped to 320 pixels * 80 pixels
- The limited use of colours means it is saved as a 16-colour image, with a file size of 1799 bytes. This gives it a fast load time.
- Two standard SAS fonts with a small trick to convert a colon into an elevated decimal point
- A series of apparent 'grooves' in the face to enhance the three dimensional appearance.

Figure 13

Higher resolution versions appeared in our printed and publicity documents, and there is a version of the image with much paler colours, which is used as a watermark on some web pages. Generating these has involved a common piece of code, and a series of device entries on our user catalogue.

In assembling this image we used a technique that we found saves a lot of time and coding. We specify the attributes of the image in a series of data sets, and then interleave them in a

simple data step. This allows us to create a model outer edge for our buttons in a data set called "OUTER", an inner edge in a data set called "RIDGE", and the text and graphics on the face of the icon are defined in a data set called "FACE". When we interleave these data sets, we have a single annotate data set for our SAS/Graph procedure.

We have demonstrated this coding technique, and made all the sample code available on our web site at http://www.dkvj.co.uk/presentations/sugi27/index.html.

WEB IMAGES

To better demonstrate the power of this method, visit http://www.dkvj.biz/sasjobs/index.html. On this part of our site, we generate a daily report of jobs for IT consultants who specialise in SAS® work in the U.K.

The data arrives five days a week, and we can generate around 200 graphic images similar to the following on a daily basis. Perhaps most interesting of all, is that each image has three versions. These versions are:

- A 'standard' version, like a raised push button, that provides a hyperlink to the data displayed on the button (figure 14).
- A 'highlighted' version, with lighter greys, which is swapped in when the mouse cursor moves over the image (figure 15).
- A 'selected' version where the button appears to have been pressed. In this version, the highlight and shaded areas are reversed (figure 16).

Figures 14, 15 & 16

The 'grid lines' on the icons are drawn with simple 'MOVE' and 'DRAW' function statements. The rectangles are defined by using the 'POLY' and 'POLYCONT' functions in the annotate data set. The size of the rectangles is defined by the source data, and provides a 'proportional' comparison of data across four years. Here is some sample code.

```
BAR4 = ( SymGet( 'MYr4Mn') * (83 - 16)) + 16;
COLOR = 'CXCC0000';  SIZE = .5;
Style = 'MSolid';
X = 8;    Y = 16;     Function = 'POLY';
Output;
X = 8;    Y = BAR4;   Function = 'POLYCONT';
Output;
X = 24;   Y = BAR4;   Function = 'POLYCONT';
Output;
X = 24;   Y = 16;     Function = 'POLYCONT';
Output;
```

The first set-up of the macros took a deal of work. However, the process is run almost daily, and now takes less than twenty minutes including data analysis and web site upload.

SUMMARY

We have looked at a number of techniques you can use with Annotate data sets in the SAS/Graph product to create custom icons, and a company shingle. The code we have examined included:

- Definition of a custom GIF output device, to give you the size and resolution you require. The BMP device driver was discussed, as well as its' size limitations.
- Creation of a series of vectors to describe an apparent elevation or depression, giving a three dimensional appearance to a two dimensional image.
- Addition of text to the image to provide a label or message.
- Modifying that text to provide a shadowing effect to increase the three dimensional illusion.
- Addition of text from the SAS/Graph special fonts to add image objects to the final product.
- Using the ability to create multiple images from one basic model, to create a library of web images that are updated by using SAS data steps to analyse any type of data.

TRADEMARK CITATION

REFERENCES

SAS Institute Inc (1999), *SAS/Graph® Software: Reference, Version 8*, Cary, NC, SAS Institute Inc.

ACKNOWLEDGEMENTS

David Ford of the SAS Melbourne office, who, while teaching my class in SAS/EIS, enthused about the possibilities of creating annotated icons with a little care and patience.

Laura LeMay, whose discussion of the benefits of generating web graphics with vector drawing programs, reminded me of those original EIS icons.

My family for their support, patience and late night coffee pots.

CONTACT INFORMATION

Your comments, suggestions and questions are valued and encouraged. Please contact the author:

David Johnson FRSA
DKV-J Consultancies
C/- 'Bonds Cottage'
Holmeswood Rd
Holmeswood nr Rufford
Lancashire England L40 1UA
Work Phone: +44 (0) 7092 25 9556
Fax: +44 (0) 7092 25 9556
Email: sugi92@dkvj.biz
Web: http://www.dkvj.biz