Cool Screen Backgrounds using SAS/GRAPH

Bob Newman, Amadeus Software Limited

ABSTRACT

In this paper, we show how the SAS Data Step Graphics Interface (DSGI) can be used to generate graphics files which can be used as unusual and interesting screen backgrounds for web pages. We begin with patterns of tessellating polyominoes. (A tessellation is a tiling of the plane. A polyomino is like a domino only bigger i.e. it is made up of more than two squares.) Later we move on to "squaring the square" patterns, and show how to incorporate images into them.

DSGI

There are many SAS procedures that will generate graphics for you – GCHART, GPLOT and G3D, to name but a few – but you can also generate graphics from a datastep, using DSGI.

We will be using DSGI to generate graphs from scratch, but it can also be used to enhance existing graphs. This makes it an alternative to using the "annotate" facility of SAS/Graph – although for ease of use, there isn't much to choose between them.

There are a large number of DSGI routines, but which ones are actually available at any given time depends on which DSGI state you are in. At the start of the datastep, the state is GKCL (clear). The two other states we will encounter are WSAC (workstation active) and GSOP (graphics segment open). The basic structure of a DSGI datastep is:

```
data myplotds;
        /* We are in GKCL state.
           This is a good time to set some GOPTIONS etc. */
  rc=ginit();
        /* This routine initialises DSGI, and puts us into
        WSAC     state, where we can perform virtually any
        DSGI function apart from actually generating
        graphics. Routines we can call include GSET to set
        DSGI options, define bundles, windows, viewports
        etc; and GASK to get information. */
  rc=graph('clear');
        /* This routine opens a graphics segment, putting us
        into GSOP state, in which we can generate our
        graphics using GDRAW etc. *
  rc=gdraw('line',1,20,30,40,50);
        /* For instance; */
        /* When we have finished drawing the graph, we
        close the segment: */
  rc=graph('update');
        /* This puts us back into WSAC state. We may now
        want to call some routines to specify where and how
        the graph is to be stored or displayed. */
  rc=gterm;
        /* Terminates DSGI, putting us back into GKCL state
        */
run;
        /* On execution of "run" statement, graphics are
        displayed. */
```

PLOTTING MACRO LIBRARY

For this application, we will generate all our GDRAW calls using macros. The macros are all stored in a catalogue called POLYMACS, each of them in a SOURCE entry whose name is the same as that of the macro. We can then make these macros available to our datastep by defining a FILENAME to point to the macro catalogue, and adding this filename to our macro search path, as follows:

```
libname poly 'c:\Noggs\SAS\poly';
filename polymacs catalog
        'poly.polymacs';
options sasautos=(sasautos,polymacs);
```

The macros themselves are not particularly exciting. At the heart of them is one called SHAPE, which looks like this:

```
%macro shape(c,parms);
  rc=gset('filcolor',&c);
* Set fill colour;
  rc=gset('filtype','solid');
* Set fill type;
  rc=gdraw('fill',&parms);
* Draw filled polygon;
%mend shape;
```

Above SHAPE, there are macros with names like TEE, HPIECE, LOCK25 and even SQUARE, each of which plots a single polygon of a specific shape and colours it in. Here, for example, is the HPIECE macro:

```
%macro hpiece(x,y,c,s=1,f=0,r=0);
  %let xlist=
       0,1,1,4,4,5,5,4,4,1,1,0,0;
  %let ylist=
       0,0,1,1,0,0,3,3,2,2,3,3,0;
  %parmgen (%bquote(&xlist),
            %bquote(&ylist),
            &x,&y,&s,f=&f,r=&r);
  %shape(&c,%bquote(&parms));
%mend hpiece;
```

The parameters to this macro are the position (X,Y) at which the shape is to be plotted, the colour C with which it is to be filled, and three optional parameters. S is the scale, and F and R are flags which can be used to indicate that the shape is to be "flipped" or reflected respectively. The PARMGEN macro combines this information with the basic shape template specified in xlist and ylist to create a parameter string that the GDRAW routines will understand.

Above all these is a macro called REPEAT, which plots a whole line of similar polygons, identically aligned, at a specified spacing, and using colours in a specified sequence.

COLOURS

DSGI handles colours in a slightly non-standard way. Routines such as GDRAW do not use SAS colour names; instead they use colour numbers. You can use GSET to specify what colours you want to associate with what numbers e.g.

```
rc=gset('colrep',3,'cyan');
```

A brief digression on SAS colour naming: There are several different ways of naming colours in SAS.

1. By their English name. RED, GREEN, BLUE, CYAN, GRAY, GREY etc are obvious, but SAS also knows

about a fair number of other colours, including SALMON, TAN, OLIVE, VIOLET, LIME, MAROON, LILAC, STEEL etc.

2. By a SASsy abbreviation of an English phrase e.g. LIG (light green), VIPK (vivid pink), DAGRYBR (dark greyish yellowish brown!)
3. By an RGB specification e.g. CXFFFF00 for yellow.
4. By an HLS (hue/lightness/saturation) specification e.g. H0B480FF for yellow.

If the precise colour specified is not available on the output device being used, SAS will do the best it can.

To see what colours are defined, and what they look like on your screen, visit http://www.devenezia.com/docs/SAS/sas-colors.html.

For use with this application, I have defined a macro called COLDEF, which defines a set of colours suitable for use in a tessellating background e.g. "%coldef(violet);" defines colours 1 to 4 as four different tasteful shades of violet.

A SAMPLE PROGRAM

Assuming the macros are already set up ready for use, here is the remainder of a program to generate part of a tessellation using the shape I have called HPIECE, and to display it on the screen. Note that the units we are using here are "PCT" i.e. percent of the graphics area.

```
Goptions reset=global gunit=pct
targetdevice=pscolor;
%global parms count;

data _null_;
rc=ginit();
rc=graph('clear');
%coldef(green);
%let clist1=1;
%let clist2=2;
%repeat(hpiece,0,0,24,0,10,
      %bquote(&clist1),s=4);
%repeat(hpiece,4,8,24,0,10,
      %bquote(&clist2),s=4,f=1);
%repeat(hpiece,12,12,24,0,10,
      %bquote(&clist1),s=4);
%repeat(hpiece,16,20,24,0,10,
      %bquote(&clist2),s=4,f=1);
%repeat(hpiece,24,24,24,0,10,
      %bquote(&clist1),s=4);
%repeat(hpiece,28,32,24,0,10,
      %bquote(&clist2),s=4,f=1);
%repeat(hpiece,36,36,24,0,10,
      %bquote(&clist1),s=4);
%repeat(hpiece,40,44,24,0,10,
      %bquote(&clist2),s=4,f=1);
%repeat(hpiece,48,48,24,0,10,
      %bquote(&clist1),s=4);
%repeat(hpiece,52,56,24,0,10,
      %bquote(&clist2),s=4,f=1);
%repeat(hpiece,60,60,24,0,10,
      %bquote(&clist1),s=4);
%repeat(hpiece,64,68,24,0,10,
      %bquote(&clist2),s=4,f=1);
rc=graph('update');
rc=gterm();
run;
```

The output looks like this. It's a bit untidy, but good enough for us to convince ourselves that the shape really does tessellate (i.e. tile the plane), and also for us to identify how much of the pattern we need to save in our graphics file for use as a screen background. In this case, it looks as though the pattern repeats every 6 squares horizontally, and also every 6 squares vertically. We'll use squares 16 pixels on a side, so our graphics file will be 96x96 pixels.

CREATING THE GRAPHICS FILE

We are now ready to create the graphics file. We need to change our program in only two places:

First, we must specify a FILENAME for our graphics file, and alter the GOPTIONS to specify output to it, in some suitable format – here I have used PNG. SAS provides drivers for several others – you can use PROC GDEVICE to find what is available. We use the XPIXELS and YPIXELS options to specify the size of our image.

The other change is that we have to arrange for the graphics file to contain only the 6x6 squares (96x96 pixels) of design that we want. To achieve this, we define a *window* using GSET('window'). The co-ordinates and dimensions here are specified in percent of the graphics area. This window is given a transformation number. We clip around the edge of the window using GSET('clip'). We then activate the transformation using GSET('transno').

The revised program looks like this:

```
filename png
"c:\Noggs\SAS\poly\polyh.png";
goptions reset=global cback=WHITE
      device=PNG gsfname=png
      gsfmode=replace
      gaccess=sasgafix gsflen=80
      display gunit=pct xpixels=96
      ypixels=96;
%global parms count;
data _null_;
rc=ginit();
rc=graph('clear');
rc=gset('clip','on');
rc=gset('window',1,44,24,68,48);
rc=gset('transno',1);
```

```
%coldef(green);
%let clist1=1;
%let clist2=2;
%repeat(hpiece,0,0,24,0,10,
      %bquote(&clist1),s=4);
```

and then as before.

The resulting image in the PNG file looks like this:

To see what the finished background looks like, go to http://www.noggs.dsl.pipex.com/ts/ . This page also incorporates a number of other backgrounds, all created in much the same way. You will see them as you move the cursor over the relevant links. (This is achieved using Javascript rollovers, which are not covered in the present paper. I do not claim to be a Javascript expert!)

SQUARING THE SQUARE

The other page in the "ts" part of my website is devoted to a "squaring the square" pattern. Surprisingly, it is possible to dissect a square into a number of smaller squares, all of different sizes. The story of how this was discovered is quite entertaining; there is also an intriguing isomorphism between constructing rectangles from different-sized squares, and solving electrical circuits using Kirchoff's laws. Martin Gardner devoted one of his columns in Scientific American to the subject. For an only slightly heavier account of the underlying mathematics, see http://www.math.uwaterloo.ca/navigation/ideas/articles/honsberger2/index.shtml.

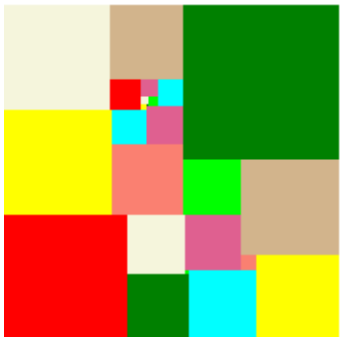
For our present purposes, this is just another geometric pattern that we can generate easily using our existing repertoire of macros. The code gets slightly messier because the total (logical) size of the pattern is 175x175, and yet we are obliged to specify all the coordinates in terms of percent of the graphics area. We will take the side of our smallest square to be 1 pixel, giving an image of 175x175 pixels. (We would have preferred 350x350 or 525x525, but the most the SAS PNG driver can handle is 615x345.)

It also becomes clear, when we are looking at squares, that our SAS code and the graphics drivers are not yet quite on the same wavelength. In order to get in the graphics file something that looks to us to be a 12x12 square, for example, we have to try to plot an 11x11 square. Similarly, to plot a 1x1 square - a single pixel – we have to try to plot a 0x0 square.

So above our SQUARE macro (which plots a square in the obvious way) we define a new macro RSQUARE which subtracts one from the side of the square, and also halves all the coordinate values to make them acceptable in the PCT units we are obliged to use at this stage.

```
%macro rsquare(x, y, c, s=1, q=2);
  %square(%sysevalf(&x/&q),
          %sysevalf(&y/&q),&c,
          s=%sysevalf((&s-1)/&q));
%mend rsquare;
```

With the aid of this macro, and our existing macro library, it is not difficult to write a program to generate an interesting new image that looks like this:



The new concept we are going to introduce is *viewports*, which we will use to superimpose images on our pattern of squares. The image concerned must already exist as an entry in a SAS graphics catalogue. It is possible to paste a wide range of graphics formats into SAS and save them as such entries, although this technique results in a bitmap with no particular structure discernible to SAS. I preferred to try out PROC GIMPORT to import some images "properly". Unfortunately GIMPORT (still!) only supports CGM graphics, which I suspect most other packages have forgotten altogether. I could find only one library of freely-available CGM images on the web, so I used it (with thanks to a gentleman who calls himself Kiyotei). The code to read in a CGM image looks like this:

```
filename cgmfile
'C:\Noggs\SAS\poly\cgm\
  cgmclips\sunprst.cgm';
proc gimport   fileref=cgmfile
filetype=cgm  format=binary;
run;
```

The first such gimport will yield an entry called "cgm" in the graphics catalogue. Subsequent entries will be called "cgm1", "cgm2" etc.

Once the images have been read into SAS graphics entries, the new programming technique we require amounts simply to defining a viewport corresponding to one of the squares of our basic pattern, associating the desired image with it, and then activating it. The image is automatically scaled to fit the chosen square. The relevant code looks like this:

```
rc=gset('viewport',2,94./175,94./175,
        1.,1.);
rc=gset('transno',2);
rc=graph('insert','cgm');
```

and it can go in just before the

```
rc=graph('update');
```

which generates the output. Here "2" is the viewport number, and the other 4 parameters to the "viewport" call are the co-ordinates of the bottom-left and top-right corners of the area into which the image is to be inserted. These are all expressed as fractions of the graphics area. "cgm" in the "insert" call is the name of the entry in the SAS graphics catalogue which contains the image.

The resulting image looks like this:

Repeated application of the same principle to incorporate more of Kiyotei's interesting graphics yields this:

Note that in this final version with many images, each one is scaled properly according to the size of the square that contains it.

To see these patterns on the web, follow the link from the tessellations page, or go directly to http://www.noggs.dsl.pipex.com/ts/sqasqa.htm . As before, move the cursor over the links to make interesting things happen.

GET TESSELLATING!

I found most of the polyomino tessellations given here years ago when confronted with a puzzle from a magazine: "What is the smallest polyomino that tessellates and locks?" The answer given was a polyomino of 25 squares, which didn't look as though it could possibly be minimal. I believe (but have no idea how to prove) that my 14-square and 12-square solutions *are* minimal, in their different ways. If anyone can beat them, please let me know. (I'll be sick as a parrot.)

All the code is appended to this paper. You are welcome to use it to generate your own backgrounds, and adorn your web pages with them.

REFERENCES

SAS System Help, particularly the "DSGI Graphics Summary"
For other references, see the text.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Author Name	Bob Newman
Company:	Amadeus Software Limited
Address:	Orchard Farm, Witney Lane, Leafield, Oxon OX29 9PG
Work Phone:	01993 878287
Fax:	01993 878042
Email:	bob.newman@amadeus.co.uk
Web:	www.amadeus.co.uk