SUGI 15, April, 1990

SAS/GRAPH® SOFTWARE MEETS THE LOGO TURTLE

Michael Friendly, York University

Abstract

This talk describes a set of SAS macros which provide the ability to draw with the SAS/GRAPH® ANNOTATE facility using the "turtle-relative" drawing commands of the Logo language, rather than absolute X-Y coordinates. This makes drawing much easier for many types of figures.

Another key feature of Logo is recursion, which also simplifies many graphics procedures. Since the SAS data step has neither local variables, nor recursion, it is hard to some things that are very easy in Logo. However, SAS macro variables are local variables, and recursive macros which model Logo procedures can be defined. A variety of Logo-like designs carried out with SAS/GRAPH are illustrated.

Introduction

The annotate facility of SAS/GRAPH is a powerful tool for customizing graphs or for designing entirely new graphic displays. A good deal of this power stems from the various data reference systems, specified by the XSYS= and YSYS= annotate variables. The variables X and Y refer to coordinate locations in the display area, but the corresponding values of XSYS and YSYS determine whether the X, Y coordinates are interpreted as, for example, data values, screen cells, or percentage values.

On the other hand, some kinds of figures or shapes are particularly difficult to draw with the annotate facility. For example, to draw a pentagon or other polygon you must calculate the (x, y) coordinate of each point on the polygon using the SIN and COS functions, whose arguments must be specified in radians:

```
data pentagon;
    xsys='2'; ysys='2';
    do i = 1 to 5;
       angle = i/5 * (2*3.14159);
       x = 50 * cos (angle);
       y = 50 * sin (angle);
       if i=1 then function = 'MOVE';
              else function = 'DRAW';
       output;
       end;
proc ganno data=pentagon datasys;
```

This paper describes an approach to make it easier to draw these kinds of geometric figures in SAS/GRAPH by implementing the ideas of "turtle graphics" from the Logo language as SAS macros. From another perspective, this can also be viewed as an example of how extensions to SAS/GRAPH may be explored by incorporating ideas from other programming environments.

What is Logo?

Logo is a computer language designed to make computing easily accessible to young children (Papert, 1980) in an educational setting. Part of its appeal is an extremely simple yet flexible graphics component, called "turtle graphics". However, Logo is actually a descendent of Lisp, and was designed to provide powerful programming features for experts as well as novices; see Friendly (1988).

Logo Turtle Graphics

Contrast the SAS/GRAPH annotate DATA step above with the following command in the Logo language:

```
REPEAT 5 [ FORWARD 50 RIGHT 72 ]
```

This command, entered interactively to the Logo system, causes a pentagon to be drawn on the screen. The Logo primitive `FORWARD 50` (or `BACK 50`) moves a graphic cursor, called "the turtle" forward (or back) 50 "turtle steps" in whatever direction it is currently facing. (The Logo graphics area is approximately 280 by 240 turtle steps. The coordinates range from -120 to +120 in the horizontal direction and to ± 140 in the vertical direction.) The primitive command `RIGHT 72` (or `LEFT 72`) rotates the turtle 72 degrees in a clockwise (or counter-clockwise) direction from its present orientation, but does not change the turtle's position.

Note that the Logo turtle doesn't have to know anything about trigonometry or converting from angles to radians: The Logo commands `FORWARD`, `BACK`, `LEFT` and `RIGHT` are all *relative* to the turtle's current position and orientation, which are maintained internally by the Logo system. (There is also a `SETPOS` command to move the turtle to an absolute (x, y) screen position, and a `SETHEADING` command to turn the turtle to a specified angular orientation.)

The Logo turtle is considered to have a pen which can be lowered or raised. The primitive commands `PENUP` and `PENDOWN` control the turtle's pen position; the command `SETPC` controls the color of ink in the pen. If the pen is down, the turtle draws in the current pen color when it moves; otherwise `FORWARD` and `BACK` cause the turtle to move without drawing a line.

The `REPEAT` command is the Logo primitive for iteration. In the command

```
REPEAT 5 [ FORWARD 50 RIGHT 72 ]
```

the list of commands in square brackets is repeated 5 times.

***Logo procedures*.** The commands to draw any figure can be defined as a new procedure with the `TO` command. For example, these statements define a procedure named `PENTAGON`.

```
TO PENTAGON
   REPEAT 5 [ FORWARD 50 RIGHT 72 ]
   END
```

Once a procedure has been defined, typing the name of the procedure causes the commands in the definition to be carried out.

Logo procedures can also be defined to take variable inputs, just like the Logo primitives FORWARD and RIGHT. This PENTAGON procedure draws a pentagon of any size:

```
TO PENTAGON :SIZE
   REPEAT 5 [ FORWARD :SIZE RIGHT 72 ]
   END
```

In Logo, a variable argument to a procedure is represented by a variable name preceded by a colon, such as ":SIZE", so if the procedure PENTAGON 100 is invoked, the value 100 is substituted wherever :SIZE occurs.

Procedures can take any number of variable inputs. The procedure POLY draws a regular polygon of any :SIZE with any number of :SIDES:

```
TO POLY :SIDES :SIZE
   REPEAT :SIDES [FORWARD :SIZE RIGHT 360/:SIDES]
   END
```

One of the powerful features of Logo, which makes it particularly easy for children to do interesting graphics, is this: Once a procedure has been defined, it can be used in other procedures exactly like the built-in primitive commands. For example, the procedure SPIN.POLY draws a series of :NUMBER polygons, rotating each one from the last.

```
TO SPIN.POLY :SIDES :SIZE :NUMBER
   REPEAT :NUMBER [ POLY :SIDES :SIZE
           FORWARD :SIZE/4 RIGHT 360/:NUMBER ]
   END
```

Recursion & local variables

Another powerful feature of Logo is recursion: procedures can do part of a task, then call themselves to complete the job. To support recursion, the variable inputs of a procedure are local variables: the values assigned to those variables in one invocation of a procedure have no effect on the values of those variables in any other procedure invocation.

The procedure POLYSPI draws a "polyspiral", a figure like a polygon, but with the length of each side decreasing until it becomes zero. The recursive way to do this in Logo is to draw one side, then call the *same procedure* to continue the same process:

```
TO POLYSPI :SIDE :ANGLE
   IF NOT :SIDE > 0 [STOP]
   FORWARD :SIDE RIGHT :ANGLE
   POLYSPI (:SIDE-1) :ANGLE
   END
```

The operation of recursion in Logo can be shown by tracing the execution of the POLYSPI procedure, which causes the procedure to print the values of its input variables on each invocation. The "?" character is the Logo prompt.

```
?TRACE "POLYSPI
?POLYSPI 4 45
-> Entering POLYSPI 4 45
  -> Entering POLYSPI 3 45
    -> Entering POLYSPI 2 45
      -> Entering POLYSPI 1 45
        -> Entering POLYSPI 0 45
        -> Leaving  POLYSPI
      -> Leaving  POLYSPI
    -> Leaving  POLYSPI
  -> Leaving  POLYSPI
-> Leaving  POLYSPI
```

Each recursive call invokes a new copy of the POLYSPI procedure with its own, private values of the inputs :SIZE and :ANGLE. The previous copy is suspended, waiting for the new one to complete. When the value of :SIZE becomes zero, the STOP statement is executed, causing the inner-most version of POLYSPI to terminate. That allows the previous copy to continue, where the values of :SIZE and :ANGLE in that call are reinstated. But since POLYSPI is the last command in the procedure, that copy terminates too. The stacked recursive calls continue to unwind, until the initial call completes.

As another recursive example, the procedure SHRINK.POLY draws one polygon by calling the POLY procedure. It then calls itself to draw another polygon, with the size of one side reduced by the amount specified in the third input, :CHANGE. The process continues until the :SIZE of a side becomes less than zero.

```
TO SHRINK.POLY :SIDES :SIZE :CHANGE
   IF :SIZE < 0 [STOP]
   POLY :SIDES :SIZE
   SHRINK.POLY :SIDES (:SIZE-:CHANGE) :CHANGE
   END
```

Implementing turtle graphics in SAS/GRAPH

In the Logo system, the turtle-relative drawing commands are implemented in such a way that the system maintains the turtle's current position and heading internally. When the command FORWARD 50 is executed, the system simply calculates the new absolute (x, y) screen location using the standard trigonometric relations, and moves or draws to that position.

The Logo graphics primitives can be modelled in SAS/GRAPH with a set of SAS macros. The macros make use of two new data step variables, in addition to those used by the ANNOTATE facility:

PEN	{'UP','DOWN'} determines whether the turtle moves or draws.
HEADING	turtle's current heading, in degrees, where 0=NORTH and angles increase clockwise.

The basic Logo macros are:

%penup	Lift PEN up, so turtle will move
%pendown	Put PEN down, so turtle will draw
%left(degrees)	Turn left (counterclockwise)
%right(degrees)	Turn right (clockwise)
%forward(dist)	Move/draw forward in current HEADING
%back(dist)	Move/draw back in current HEADING
%inilogo	Initialize variables, move turtle home
%home	Move/draw to (0,0)

These macros are used in a DATA step to produce an ANNOTATE= data set.

The macros `%penup`, `%pendown`, `%left`, and `%right` simply set the values of the PEN and HEADING variables. Note that `%left` and `%right` keep the HEADING variable in the range of 0 to 360 degrees.

```
%macro penup;
%*-------------------------------------------;
%* Put PEN up, so turtle will move           ;
%*-------------------------------------------;
    PEN='UP  ';
%mend  penup;

%macro pendown;
%*-------------------------------------------;
%* Put PEN down, so turtle will draw         ;
%*-------------------------------------------;
    PEN='DOWN';
%mend  pendown;

%macro left(degrees);
%*-------------------------------------------;
%* Turn LEFT by DEGREES (counterclockwise)   ;
%*-------------------------------------------;
    HEADING=mod(HEADING-(&degrees),360);
%mend  left;

%macro right(degrees);
%*-------------------------------------------;
%* Turn RIGHT by DEGREES (clockwise)         ;
%*-------------------------------------------;
    HEADING=mod(HEADING+(&degrees),360);
%mend  right;
```

All the work is done in the `%forward` macro. For efficiency the macro recognizes the special cases of HEADING = 0, 90, 180, or 270 degrees. The `%back` macro is implemented in terms of `%forward`.

```
%macro forward(dist);
%*-------------------------------------------;
%* Move/Draw FORWARD in the current HEADING  ;
%* - X & Y are in whatever XSYS, YSYS system ;
%*   is being used, but MUST be absolute.    ;
%* - In turtle coordinates, HEADINGs go clock-;
%*   wise from North. In SAS geometry, angles ;
%*   go counterclockwise from East, so       ;
%*   HEADING = 90-angle                      ;
%* - PEN determines whether to MOVE or DRAW  ;
%*-------------------------------------------;
  select ( HEADING ) ;
    when (   0 )  Y = Y + &dist;
    when (  90 )  X = X + &dist;
    when ( 180 )  Y = Y - &dist;
    when ( 270 )  X = X - &dist;
    otherwise do;        /* general case */
      X=X+(&dist)*cos((90-HEADING)*atan(1)/45);
      Y=Y+(&dist)*sin((90-HEADING)*atan(1)/45);
    end;
  end;
  if PEN = 'UP' then FUNCTION = 'MOVE';
               else FUNCTION = 'DRAW';
  output;
%mend forward;

%macro back(dist);
%*-------------------------------------------;
%* Move/Draw BACK in the current HEADING     ;
%*-------------------------------------------;
    %forward (-1*(&dist));
%mend  back;
```

Two utility macros complete the basic set of Logo commands for SAS/GRAPH. `%home` moves the turtle to the “home” position, (0, 0). `%inilogo` initializes the PEN and HEADING variables and starts the turtle at (0, 0).

```
%macro home;
%*-------------------------------------------;
%* Move/Draw to (0,0)                        ;
%*-------------------------------------------;
    X=0; Y=0;
    if PEN = 'UP' then FUNCTION = 'MOVE';
                  else FUNCTION = 'DRAW';
    output;
%mend  home;

%macro inilogo;
%*--------------------------------------------;
%* Initialize logo variables & move turtle home;
%*  - use at start of data step, like dclanno  ;
%*--------------------------------------------;
    LENGTH PEN $ 4;
    %penup; %home; %pendown;
    HEADING=0;
%mend  inilogo;
```

Using the Logo macros

The Logo macros are invoked in a DATA step to produce an ANNOTATE= data set. The figure can then be drawn with PROC GANNO or PROC GSLIDE.

The example below draws a set of 12 stars equally spaced around a circle, giving the result shown in Figure 1. This example uses the data percentage coordinate system (XSYS='1'; YSYS='1';), so the plot is centered at (50, 50). The Logo macros can use any of the absolute coordinate systems.

```
%include LOGO ;         /*
ge/*LOnoMVS,ruse*%include ddname(file);*/
data example1;
   xsys='1'; ysys='1';    /* data % system */
   %inilogo;
   x = 50; y=50;
   function='MOVE    '; output;
   %pendown;
   do i=1 to 12;
      %forward(15);
      do j=1 to 5;
         %forward(20);
         %right(144);
         end;
      %back(15); %right(30);
   end;
proc ganno anno=example1;
```

The data set EXAMPLE1 contains the usual annotate variables, XSYS, YSYS, X, Y, FUNCTION, and so forth, plus the Logo variables, PEN and HEADING. The observations to draw the first two stars are shown in Figure 2.

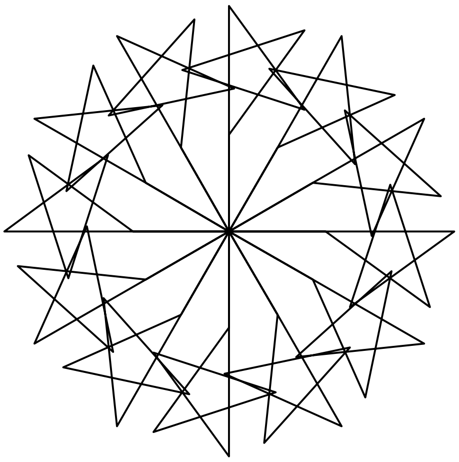

Figure 1: Logo stars

I	J	PEN	HEADING	FUNCTION	X	Y
.	.	UP	.	MOVE	0.0	0.0
.	.	DOWN	0	MOVE	50.0	50.0
1	.	DOWN	0	DRAW	50.0	65.0
1	1	DOWN	0	DRAW	50.0	85.0
1	2	DOWN	144	DRAW	61.8	68.8
1	3	DOWN	288	DRAW	42.7	75.0
1	4	DOWN	72	DRAW	61.8	81.2
1	5	DOWN	216	DRAW	50.0	65.0
1	6	DOWN	0	DRAW	50.0	50.0
2	6	DOWN	30	DRAW	57.5	63.0
2	1	DOWN	30	DRAW	67.5	80.3
2	2	DOWN	174	DRAW	69.6	60.4
2	3	DOWN	318	DRAW	56.2	75.3
2	4	DOWN	102	DRAW	75.8	71.1
2	5	DOWN	246	DRAW	57.5	63.0
2	6	DOWN	30	DRAW	50.0	50.0

Figure 2: Data set EXAMPLE1

Some sample SAS/GRAPH - Logo designs

As in Logo, the basic Logo macros can be used to define new procedures as macros. The possibilities are limited only by your imagination. For simplicity, I'll illustrate some variations and combinations of polygon designs.

Here are SAS/GRAPH versions of the Logo `POLY` and `POLYSPI` procedures:

```
%macro polys(sides,length);
%*--------------------------------------------;
%* Move/Draw a polygon, the Logo way         ;
%*--------------------------------------------;
    do _i_ = 1 to &sides;
       %forward(&length);
       %right(360/(&sides));
    end;
%mend  polys;

%macro polyspi(length,angle,decrease);
%*--------------------------------------------;
%* Draw a polyspiral, decreasing side while >0;
%*--------------------------------------------;
   _len_ = &length;
   do while (_len_ > 0);
       %forward(_len_);
       %right(&angle);
       _len_ = _len_ - &decrease;
   end;
%mend  polyspi;
```

Figure 3 shows a series of six polygons drawn with `%polys`. The DATA step which draws this figure is shown below: The program uses the SAS/GRAPH annotate macros, `%system`, `%frame`, `%label`, etc. in addition to the Logo macros.

```
data poly1;
   %dclanno;
   %system(2,2,4);            /* data system */
   %inilogo;
   do i=4 to 9;
      %polys(i, 15); %right(60);
   end;
   %label(8,30,'Logo polys',RED,0,0,2.5,DUPLEX,5);
proc ganno datasys anno=poly1;
```

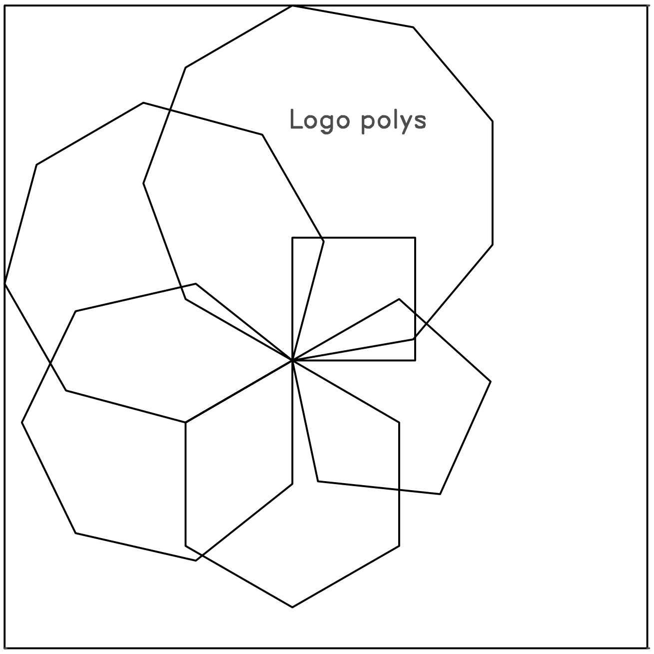

Figure 3: Logo polys

Figure 4 is drawn with `%polyspi` with this DATA step:

```
data poly2;
   %dclanno; %system(2,2,4);
   %frame(BLACK,1,1,EMPTY);
   %inilogo;
   do i=1 to 5;
      %penup; %home; %pendown;
      %forward(25);
      %polyspi(72,75,.5);
      %right( 72);
   end;
   %label(8,12,'POLYSPI',BLACK,0,0,1.5,DUPLEX,5);
proc ganno datasys anno=poly2;
```

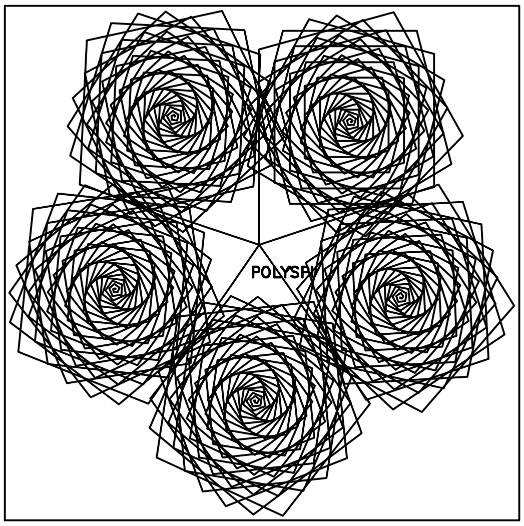

Figure 4: `%polyspi` design

The polygons drawn by `%polys` start and end at the turtle's current position. For use as a building block in larger designs, it is more useful to be able to draw a figure *centered* at the current position, and to be able to control the *radius* of a circumscribing circle, rather than the *size* of one side. The macro `%cpoly` provides these additional controls. The macro `%spingon` uses `%cpoly` to draw a series of rotated, centered polygons of decreasing radius.

```
%macro cpoly(sides,radius);
%*-------------------------------------------;
%* Draw a centered polygon inscribed in a     ;
%* circle of given radius.                    ;
%*-------------------------------------------;
     %* move out to circumscribed circle;
   %penup;
   %forward(&radius);
     %* turn to face along one edge of polygon;
   %right(180 - (90*(&sides-2)/&sides));
   %pendown;
     %* polygon, side as function of radius;
   %polys(&sides,(2*(&radius)*sin(&pi/&sides)));
     %* turn & move back to starting point;
   %left(180 - (90*(&sides-2)/&sides));
   %penup;
   %back(&radius);
   %pendown;
%mend cpoly;

%macro spingon(sides,rad,decrease,times,turn);
%*-------------------------------------------;
%* Centered polygons, of decreasing radius    ;
%*-------------------------------------------;
   _len_=&rad;
   do _j_=1 to &times;
      %cpoly(&sides,_len_);
      %right(&turn);
      _len_=_len_ - &decrease;
      end;
   %left(&turn*&times);
%mend spingon;
```

A design created with `%spingon` is shown in Figure 5. This figure is drawn with the following statements:

```
data spin2;
   %dclanno; %system(2,2,4);
   %frame(BLACK,1,1,EMPTY);
   %inilogo;
   do H = 0, 100;
   do V = 0, 100;
      %move(H,V);
      %spingon(4, 50,2,25,4);
      end; end;
   %move(50,50);
   %spingon(4, 50,2,25,-4);
proc ganno datasys anno=spin2;
```

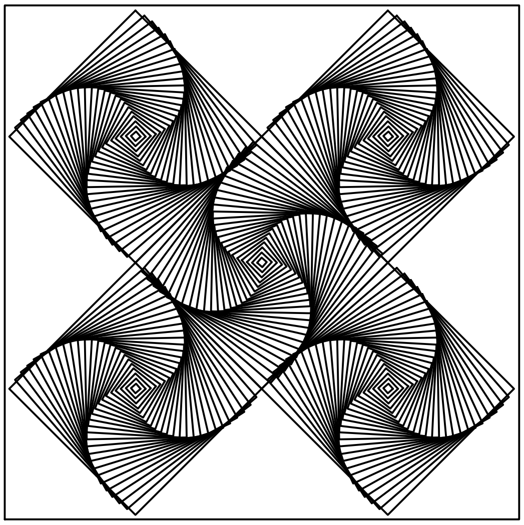

Figure 5: `%spingon` design

Turtle Text

The Logo macros can also be used to construct interesting graphic designs with text. The example below repeats a character string in a square spiral pattern, shown in Figure 6.

```
%annocms;
data text1;
   %dclanno; %system(2,2,4);
   %inilogo;
   str='TURTLES DO IT RECURSIVELY.';
   %penup; %right(90);
   color = 'BLACK';
   style = 'DUPLEX'; size=2.5;
   do s=1 to 20;
      do i = 1 to 20-s;
         text =   substr(str,1,1);
         str = substr(str,2)||substr(str,1,1);
         angle= 90 - HEADING ; rot=angle ;
         %forward(1);
         function='LABEL'; output;
         end;
      %right(90);
      end;
proc ganno datasys anno=text1;
```

The outer loop constructs 20 sides of the spiral, and the inner loop draws the letters along one side. After drawing a letter, the turtle goes `%forward(1)`; after drawing a side, it turns `%right(90)`. The string `STR` of letters is rotated cyclically, so the letter to be drawn is always the first character of the string.

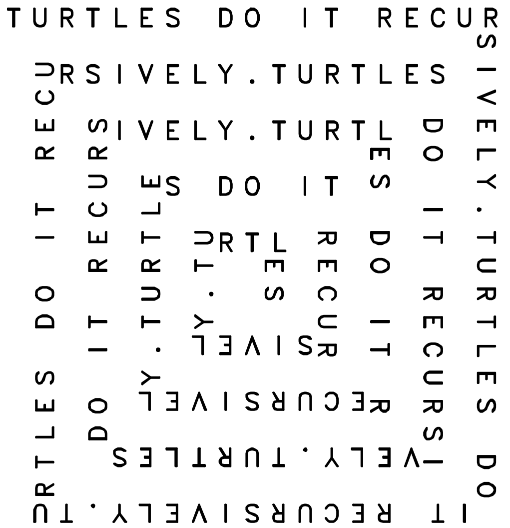

Figure 6: Turtle Text design

Recursive SAS/GRAPH macros

Recursion is possible in almost any programming language in which parameters to procedures or subroutines are local variables. In the SAS DATA step, all variables are global, and subroutines, called by the `LINK` statement do not allow parameters. However, in the SAS macro facility, macro parameters are local variables and other macro variables can be declared local to a macro with the `%LOCAL` statement. This makes it possible to define recursive SAS macros which mirror Logo procedures.

The macro `%repoly` draws a set of recursive polygons. The “outer-level” of the procedure draws a centered polygon, like `%cpoly`. However, at each vertex of this polygon, it calls itself to draw a new polygon centered at that vertex. The number of levels of recursion is specified by the `depth` parameter, and the macro keeps recursing until `depth` becomes zero. At each level, the size of the polygon is multiplied by the scale factor `factor`. Figure 7 shows a sample design drawn with `%repoly`. Figure 8 shows another.

```
%macro repoly(sides,size,factor,rot,depth);
%*------------------------------------------;
%* Recursive polygons                        ;
%*------------------------------------------;
%local s;                          %* MUST be local;
   %if %eval((&depth))=0 %then ;  %* stop rule;
   %else %do;
      %penup; %forward(&size); %pendown;
      %right(180 - (90*(&sides-2)/&sides));
      %do s=1 %to &sides;
         %forward(2* &size * sin(&pi/&sides));
         %left(&rot);
         %repoly(&sides,(&factor*&size),&factor,
&rot,(&depth-1));
         %right(&rot);
         %right(360/(&sides));
         %end;
      %left(180 - (90*(&sides-2)/&sides));
      %penup; %back(&size); %pendown;
   %end;
%mend repoly;
```

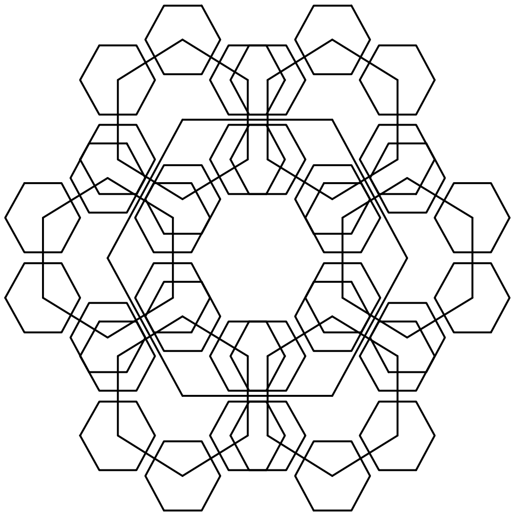

Figure 7: `%repoly( 6, 50, .5, 30, 3);`

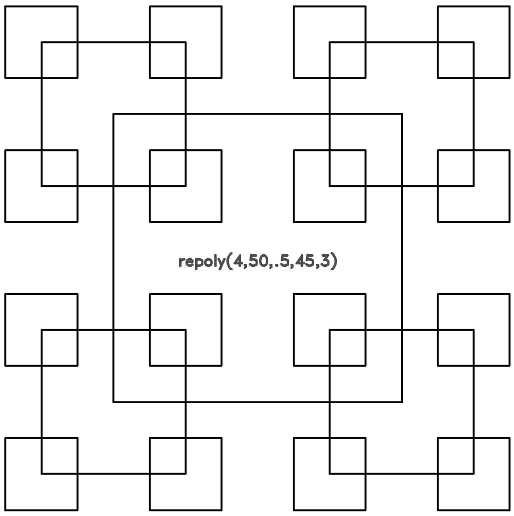

Figure 8: `%repoly( 4, 50, .5, 45, 3)`

Limitations. While recursive macros can be written as illustrated, the SAS macro facility was never designed for this purpose. In languages that explicitly support recursion, the execution path of a recursive procedure is maintained in a “stack frame”, an internal data structure that records the state of each currently-running invocation and the values of local variables. SAS macros, on the other hand simply generate program text.

As a result, a macro like `%repoly` can easily cause the SAS supervisor to run out of memory or symbol table space. The macro call,

```
%repoly( 6, 50, .5, 30, 3);
```

for example, specifies a set of recursive hexagons to a depth of 3. At each of the six sides of the outer hexagon the macro calls itself with `depth=2`: `%repoly( 6, 25, .5, 30, 2);` and each of those calls results in a further recursive call with `depth=1`. That is about all the SAS supervisor can manage.

Logo Thinking

The foregoing examples were designed, in part, to demonstrate some of the power of "Logo thinking", and how it is possible to apply Logo ideas to SAS/GRAPH. Logo was designed to support *cognitive efficiency*—conceptual clarity, understandability, and generalizability—perhaps at the expense of machine efficiency. Some Logo principles reflected implicitly in the macro programs are:

modular procedures: Logo programming is based on the idea of writing small, modular procedures which can then be used as if they were primitives in higher-level procedures. This allows the user to adopt a tool-kit approach, building up complex programs from standardized components, or a top-down approach, breaking up a large problem into manageable sub-problems.

state independence: In order to use one procedure as a building block in a larger design, it is necessary for that procedure to leave the turtle's state—the `PEN`, `HEADING`, and (x, y) position—the same at the end as it was at the beginning. The `%cpoly` macro, for example, uses

```
%forward(&radius);
%right( ... );
```

to move the turtle out into position to draw a polygon, and then reverses those steps with

```
%left( ... );
%back(&radius);
```

to restore the turtle to it's original position.

recursion: Solving a problem by recursion often leads to a much simpler and more elegant solution than solving the same problem by iteration. The key idea of recursion is to think of breaking down a problem into an "easy step", which can be solved directly, and a "hard step", which is just a smaller version of the original problem. This was illustrated in the Logo version of the `POLYSPI` procedure. However, when the recursive call is the last statement in the procedure (called *tail recursion*) as it is in `POLYSPI`, an equivalent iterative version of the procedure can be more efficient, especially in the SAS macro facility. The SAS/GRAPH `%polyspi` macro uses iteration for efficiency, but I think of it as essentially recursive.

I'll illustrate these ideas in the following problem: to design a SAS/GRAPH program to produce a *tessellation*, or tiling design with an arbitrary motif as the unit. Figure 9 shows an example of such a design.

The basic unit in Figure 9 is a "tpgon" (Figure 10), a hexagonal figure, each of whose sides is a "teepee" shape. The teepee shape is drawn by the macro `%teepee` below. `%teepee` is state transparent since the initial and final (x, y) positions are the same

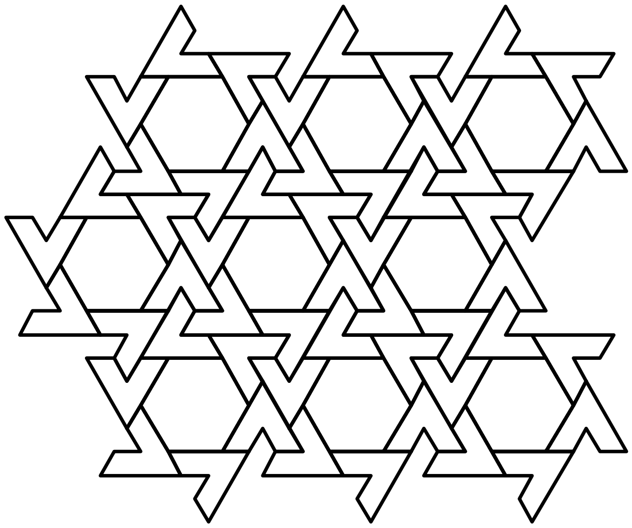

Figure 9: `%tpgon` tiling design

and the sum of the angles turned is 360°. The `%tpgon` macro uses `%teepee` as a building block, and the same idea as `%cpoly` to draw the teepee figures around a central point.

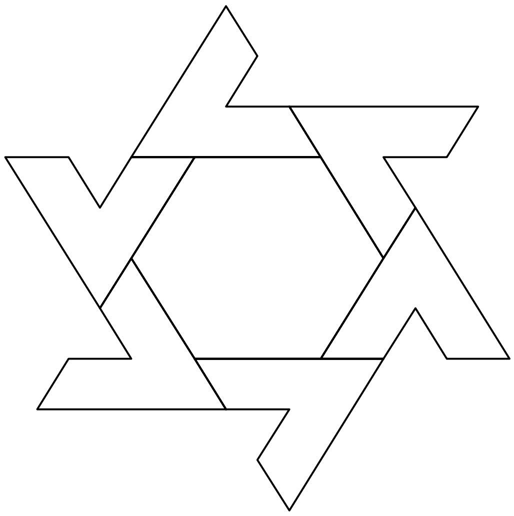

Figure 10: `%tpgon` shape

```
%macro teepee(d);
   %forward(2*(&d)); %left(120);
   %forward(&d);     %left(60);
   %forward(&d);     %right(120);
   %forward(&d);     %left(60);
   %forward(&d);     %left(120);
   %forward(3*(&d)); %left(120);
   %forward(&d);
%mend teepee;

%macro tpgon(size);
%local s;
    %left(30);
```

```
    %penup; %forward(&size); %pendown;
    %right(120);
    %do s=1 %to 6;
        %teepee(&size/2);
        %forward(&size);
        %right(60);
    %end;
    %left(120);
    %penup; %back(&size); %pendown;
    %right(30);
%mend tpgon;
```

Now that we can draw one `%tpgon` figure, how can we produce a space-filling tiling design? The Logo solution is to solve a simpler problem first: given an arbitrary motif, draw a row made up of *n* repetitions of the motif:

```
TO ROW :SPACE :N
   IF :N < 1 [STOP]
   RUN :MOTIF
   PENUP RIGHT 90
   FORWARD :SPACE
   LEFT 90 PENDOWN
   ROW :SPACE (:N-1)
   END
```

The Logo `RUN` command executes whatever commands are stored in the (global) variable `:MOTIF`. For example, the following lines assigns a `TPGON` command to `MOTIF` and draws a row of 5 of them spaced 32 turtle steps from center to center.

```
?MAKE "MOTIF [TPGON 16]
?ROW 32 5
```

Since `ROW` is tail-recursive, it can be translated into this SAS macro:

```
%global motif;
%macro row(space,n);
%*---------------------------------;
%* Paint a row of n images of motif ;
%*---------------------------------;
%local c;
   %do c = 1 %to &n;
      %unquote(&motif);
      %penup; %right(90);
      %forward(&space);
      %left(90); %pendown;
   %end;
%mend row;
```

To draw the same row of 5 tpgon's, we use the following lines in a data step:

```
%let motif=%nrstr(%tpgon(16));
data rowdemo;
   %inilogo;
   %row(32,5);
```

Note that `%nrstr` quotes the command to be assigned to `motif`, and `%unquote(&motif);` in the `%row` macro is the SAS equivalent of the Logo `RUN` command.

We have now reduced the problem to one of drawing several rows, each consisting of several repetitions of the motif. This problem is similar to the one we just solved in `%row`. An additional complication is performing a “carriage return”, to move the turtle back to the beginning of the next row, yet allow the rows to be offset as they are in Figure 9. The macro `%tiles` uses `%row` as a building block:

```
%macro tiles(rows,cols,rspace,cspace,offset);
%*-------------------------------------;
%* Tile a display of rows x cols images ;
%*-------------------------------------;
%local r;
   %do r = 1 %to &rows;
     xsave = x; ysave = y;
     %row( &cspace, &cols);
     %penup;
     %let offset=%eval(-1 * &offset);
     x = xsave + &offset; y=ysave;
     %right(180); %forward(&rspace);
     %left(180);
     %pendown;
   %end;
%mend tiles;
```

Finally, a little trigonometry *is* necessary here to determine the proper row and column spacing in terms of the size of the basic `%tpgon`, so that successive shapes join appropriately to produce a tessellation. Here is the data step that draws Figure 9:

```
data tptile;
   %dclanno;
   xsys='2'; ysys='2';
   %inilogo;
   %let motif=%nrstr(%tpgon(16));
   /* a=16; cs=3*a; rs=cs*cos(30);  */
   %tiles(  3,3, 41.56922, 48, 24);
proc ganno datasys anno=tptile;
```

Author's Address. For further information, contact:

Michael Friendly
Psychology Department, Rm 210 BSB
York University
Downsview, ONT, Canada M3J 1P3
Internet: `<Friendly@VM1.YorkU.CA>`

References

Clayson, J. (1988). *Visual Modeling with Logo* Cambridge, MA: MIT Press.

Friendly, M. (1988). *Advanced Logo: A Language for Learning.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Papert, S. (1980). *Mindstorms.* New York: Basic Books.