The Wild Side of the Annotate Facility
Chip Kelly, SAS Institute Inc.
Cary, NC

Introduction

The annotate facility unleashes the power of the SAS® DATA step programming language for the creation of graphics displays limited only by your imagination. This tutorial will show several facets of that capability. The written portion of the tutorial will consist solely of the code used to generate the examples that were shown at the presentation. The nature of the examples, their complexity, and the use of color gradations prevents the actual pictures from being included in the proceedings.

Your Basic Line

Probably the simplest form of graphics design is drawing lines. The annotate facility provides two commands, MOVE and DRAW, to respectively establish the beginning and ending points for a line. The following code illustrates the beginning of our journey into generating patterns with lines:

```sas
DATA LINE;
LENGTH FUNCTION COLOR $ 8;
RETAIN HSYS '3' XSYS YSYS '2' SIZE 1;
COLOR='BLACK';
FUNCTION='MOVE'; X=1; Y=1; OUTPUT;
FUNCTION='DRAW'; X=100; OUTPUT;
PROC GANNO ANNO=LINE DATASYS;
RUN;
```

This simple program draws a straight line across the page. We use the DATA frame of reference so that we don’t have to worry about the X and Y coordinates falling off the page.

To add interesting texture to the drawing of lines, we can begin to manipulate the beginning and ending points of each line segment. The functions available in the DATA step provide a rich environment for this type of manipulation. The trigonometric functions will enable us to move and draw in a circular fashion. The next example shows the basic shape of the SIN curve:

```sas
DATA SIN;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2' SIZE 1;
COLOR='BLACK';
FUNCTION='MOVE'; X=-2*3.14159; Y=0;
OUTPUT;
DO X=-2*3.14159 TO 2*3.14159 BY .05;
Y=SIN(X); FUNCTION='DRAW'; OUTPUT;
END;
PROC GANNO ANNO=SIN DATASYS;
RUN;
```

A trivial modification of the program changes the amplitude of the waves:

```sas
DATA SIN;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS YSYS '2' HSYS '3' COLOR 'BLACK';
DO START=0 TO 3.14159 BY 3.14159/10;
FUNCTION='MOVE'; X=-2*3.14159; Y=0;
OUTPUT;
DO X=-2*3.14159 TO 2*3.14159 BY .1;
Y=SIN(X)*START; FUNCTION='DRAW'; OUTPUT;
END;
PROC GANNO ANNO=SIN DATASYS;
RUN;
```

Adding Color and Complexity

Before proceeding any further, we should begin to add color to the display. Colors may be specified in many different ways in SAS/GRAPH® software. This tutorial focuses on how to generate displays in the DATA step, so we will assign our colors as a function of line generation itself. This technique is useful for associating gradations of colors with attributes of the display itself. In the previous display, we added an outside loop to control the amplitude of the SIN waves. We use the loop index variable in the next example to assign a different color to each line that is drawn. The HLS color naming convention is used to provide a cyclical progression of colors:

```sas
DATA SIN;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN XSYS YSYS '2';
DO START=0 TO 3.14159 BY 3.14159/50;
COLOR='H84FF';
FUNCTION='MOVE'; X=-2*3.14159; Y=0;
OUTPUT;
DO X=-2*3.14159 TO 2*3.14159 BY .1;
Y=SIN(X)*START; FUNCTION='DRAW'; OUTPUT;
END;
PROC GANNO ANNO=SIN DATASYS;
RUN;
```

By combining the SIN and COS functions, we can begin to use the basic rules of trigonometry to generate interesting designs. In the next example, we advance the iteration variable from 4 times pi to 0, using the iteration value as the starting point for each line; then we calculate the ending point as a function of SIN and COSINE values:
DATA SINCOS;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2' SIZE .5;
DO I=4*3.14159 TO 0 BY -2*3.14159/100;
COLOR='H' || PUT(I*28,HEX3.) || '84FF';
FUNCTION='MOVE'; X=I; Y=I; OUTPUT;
FUNCTION='DRAW'; X=I*COS(I); Y=I*SIN(I); OUTPUT;
END;
PROC GANNO ANNO=SINCOS DATASYS;
RUN;

By iterating over both negative and positive values, we can increase in cyclic fashion the length of each line while linearly advancing the color:

DATA TRY;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2';
DO I=-4*3.14159 TO 4*3.14159 BY 2*3.14159/100;
COLOR='H' || PUT(ABS(I)*20,HEX3.) || '84FF';
FUNCTION='MOVE'; X=I; Y=I; OUTPUT;
FUNCTION='DRAW'; X=I*COS(I); Y=I*SIN(I); OUTPUT;
END;
PROC GANNO ANNO=TRY DATASYS;
RUN;

The SIN and COS functions can also be used to describe the points that make up a circle. The next example shows the use of a fixed origin for each line while the endpoint of each line is moved around the perimeter of a circle:

DATA LINES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2' COLOR 'BLACK' SIZE .5;
DO J=0 TO 2*3.14159 BY .1;
FUNCTION='MOVE'; X=0; Y=0; OUTPUT;
FUNCTION='DRAW'; X=COS(J); Y=SIN(J); OUTPUT;
END;
PROC GANNO ANNO=LINES DATASYS;
RUN;

Vertically displacing the origin of each set of points generates a diamond-like pattern:

DATA LINES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2' COLOR 'BLACK' SIZE .5;
DO J=1 TO 40;
K=J*8..1; COLOR='H' || PUT(I*57,HEX3.) || '84FF';
IF J=40 THEN DO;
FUNCTION='DRAW'; X=50+20*COS(I+K); Y=30+20*SIN(I+K)+J; HX=X; HY=Y; OUTPUT;
IF I=0 THEN DO; FX=X; FY=Y; END;
X=LAG(HX); Y=LAG(HY); OUTPUT;
ELSE DO;
FUNCTION='DRAW'; X=50+20*COS(I+K); Y=30+20*SIN(I+K); OUTPUT;
FUNCTION='PUSH'; OUTPUT;
END;
ELSE DO;
FUNCTION='DRAW'; X=50+20*COS(I+K); Y=30+20*SIN(I+K); OUTPUT;
FUNCTION='PUSH'; OUTPUT;
END;
DO J=0 TO 3.14159 BY .125;
DO I=0 TO 2*3.14159 BY .25;
FUNCTION='MOVE'; X=cos(J); Y=sin(J); OUTPUT;
COLOR='H' || PUT(I*57,HEX3.) || '84FF';
FUNCTION='DRAW'; X=cos(I); Y=sin(I); OUTPUT;
END;
END;
PROC GANNO ANNO=LINES DATASYS;
RUN;

Moving both the origin and the endpoint around the circle allows you to create intricate geometric patterns:

DATA LINES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2' COLOR 'BLACK' SIZE .5;
DO J=0 TO 3.14159 BY .125;
DO I=0 TO 2*3.14159 BY .25;
FUNCTION='MOVE'; X=cos(J); Y=sin(J); OUTPUT;
COLOR='H' || PUT(I*57,HEX3.) || '84FF';
FUNCTION='DRAW'; X=cos(I); Y=sin(I); OUTPUT;
END;
END;
PROC GANNO ANNO=LINES DATASYS;
RUN;

The following example shows how the previous circular line patterns can be offset to form a series of spiral lines. The basic idea is to begin a line segment on the perimeter of a circle, drawing to a point on another circle that is slightly rotated and displaced vertically.

To complete the three dimensional effect, join the first and last points of each line into bottom and top circles, using the PUSH and POP functions:

DATA LINES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2' SIZE .5;
DO J=0 TO 3.14159 BY .125;
DO I=0 TO 2*3.14159 BY .25;
FUNCTION='MOVE'; X=cos(J); Y=sin(J); OUTPUT;
COLOR='H' || PUT(I*57,HEX3.) || '84FF';
FUNCTION='DRAW'; X=cos(I); Y=sin(I); OUTPUT;
END;
END;
PROC GANNO ANNO=LINES DATASYS;
RUN;

Vertically displacing the origin of each set of points generates a diamond-like pattern:

DATA LINES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2';
DO J=0 TO 3.14159 BY .1;
FUNCTION='MOVE'; X=0; Y=0; OUTPUT;
FUNCTION='DRAW'; X=COS(J); Y=SIN(J); OUTPUT;
END;
PROC GANNO ANNO=LINES DATASYS;
RUN;

Vertically displacing the origin of each set of points generates a diamond-like pattern:

DATA LINES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN HSYS '3' XSYS YSYS '2';
DO J=0 TO 3.14159 BY .1;
FUNCTION='MOVE'; X=0; Y=0; OUTPUT;
FUNCTION='DRAW'; X=COS(J); Y=SIN(J); OUTPUT;
END;
PROC GANNO ANNO=LINES DATASYS;
RUN;
Basic Geometric Designs

Thus far we have generated lines that together form simple or complex figures. We can also apply the rules of geometry to generate shapes and manipulate those shapes. Now, instead of using the line as our basic building block, we will use the generated figure.

To conceptually simplify our DATA steps, we can use a macro to generate the multi-sided polygons. The constraints on the macro will be to generate only regular polygons inscribed inside a circle of fixed radius. The macro is shown below:

```
%MACRO POLYGON(B_ANGLE=,SIDES=,XPOS=,YPOS=,RAD=,COLOR=,SIZE=);
XMIN=100; YMIN=100; XMAX=0;
SIZE=.75; COLOR='BLACK';
DO ANGLE=B_ANGLE*3.14159 TO (B.Angle+2)*3.14159
BY 2*3.14159/SIDES;
IF ANGLE=B_ANGLE*3.14159 THEN
FUNCTION='MOVE';
ELSE FUNCTION='DRAW';
X=RAD*COS(ANGLE)+XPOS;
Y=RAD*SIN(ANGLE);
IF X>XMAX THEN XMAX=X;
IF X<XMIN THEN XMIN=X;
IF Y>YMAX THEN YMAX=Y;
OUTPUT;
END;
%MEND POLYGON;
```

With this macro in place, we can now use the same manipulative techniques on its resulting polygons as we used with the origins and ending points of the lines that we used in the previous section. As a first example, let’s try shifting the beginning angle and radius of a series of hexagons:

```data b;
length function color style $ 8;
retain hsys '3' xsys ysys '2';
retain hsys '3' xsys ysys '2';
do i=2*3.14159 TO 0 BY -3.14159/6;
radius=(i+1)*3)-7.99;
%polygon(b-angle=i,sides=6,xpos=50,
ypos=50,radius=radius,
color='black',size=.75);
end;
```

Proc ganno anno=b datasets;
run;

A logical extension of this technique is to begin moving the center point of the circle in which each shape is inscribed by:

```data b;
length function color style $ 8;
retain hsys '3' xsys ysys '2';
do i=2*3.14159 TO 0 BY -3.14159/6;
radius=(i+1)*3)-7.99;
%polygon(b-angle=i,sides=6,xpos=50,
ypos=50,radius=radius,
color='black',size=.75);
end;
```

Proc ganno anno=b datasets;
run;

We can also fill the page with this type of design:

```data b;
length function color style $ 8;
retain hsys '3' xsys ysys '2';
do k=25 TO 75 BY 25;
do j=25 TO 75 BY 25;
do i=2*3.14159 TO 0 BY -3.14159/10;
radius=((i+1)*3)-2.99;
%polygon(b-angle=i,sides=6,
xpos=j-1,ypos=k-1,
radius=radius,
color='black',size=.75);
end;
```

Proc ganno anno=b datasets;
run;

Filling In the Areas

The MOVE and DRAW functions allow us to define the perimeters of the shapes, but we must use the POLY and POLYCONT functions if we want to fill the areas with color. The method of generating the shapes remains the same; we simply substitute POLY for MOVE and POLYCONT for DRAW. The following macro implements the change:

```````
The SIN function can be used to generate solid areas with curved edges. A rainbow could be constructed in this fashion, changing colors across the bands:

```sas
DATA RAINBOW;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN RSYS '1' XSYS YSYS '2' STYLE 'SOLID';
DO B=36 TO 12 BY -1;
DO XX=.5 TO 3.14159 BY 3.14159/25;
  IF XX=.5 THEN DO;
    MOVE=1; X=XX;
    COLOR='H'IIPUT(B*10,HEX3.)II'84FF';
  END;
  ELSE DO;
    FUNCTION='POLYCONT'; MOVE=0;
    X+(XX/LOG(B»);
    END;
    Y=SIN(XX);
    IF MOVE THEN DO;
      FUNCTION='POLY';OUTPUT;
      END;
      ELSE OUTPUT;
  END;
  DO XX=3.14159 TO .5 BY 3.14159/25;
  Y=SIN(XX);
  X=X-(XX/LOG(B»;
  OUTPUT;
  END;
END;
PROC GANNO ANNO=RAINBOW DATASYS;
RUN;
```

We used trigonometric functions to move around the perimeter of a circle. Now we can generate a polygon whose shape is based on a circular motion.

```sas
DATA CIRCLE;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN XSYS YSYS '2' STYLE 'SOLID';
DO ANGLE=O TO 2*3.1416 BY .01;
  IF ANGLE=O THEN DO;
    FUNCTION='POLY'; X=O; Y=O;
    OUTPUT;
    END;
  ELSE FUNCTION='POLYCONT';
    RADIUS=RANUNI(O); X=RADIUS*COS(ANGLE);
    Y=RADIUS*SIN(ANGLE);
    OUTPUT;
  END;
END;
PROC GANNO ANNO=CIRCLE DATASYS;
RUN;
```

Slightly more regular polygons used in this manner lend themselves to a visually placid display resembling a flower. The following example is derived from the SAS/GRAPH User's Guide, Version 5 Edition:

```sas
DATA FLOWER;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN XSYS YSYS '2' STYLE 'SOLID' RLAST 1;
PI2=3.14159*2; DELTA=PI2/500;
NAME='FLOWER';
DO ANGLE=0 TO PI2 BY DELTA;
  A=15*SIN(6*ANGLE); R=A+5*SIGN(A);
  IF ANGLE=O THEN DO;
    FUNCTION='POLY'; X=O; Y=O; PETAL+1;
    COLOR='CYAN';
    END;
  ELSE FUNCTION='POLYCONT';
    RADIUS=RANUNI(O)*CIRCLE;
    X=RADIUS*COS(ANGLE);
    Y=RADIUS*SIN(ANGLE);
    OUTPUT;
  END;
END;
PROC GANNO ANNO=CIRCLE DATASYS;
RUN;
```
Because we are again moving around the perimeter of a circle, the current petal number can be used to determine the color of the petal. Only one statement needs to be changed to accommodate the color changes:

```
COLOR='H' || PUT(PETAL*30,HEX3.) || 'S4FF';
```

replaces

```
COLOR='CYAN';
```

The next logical extension is to layer the petals around diminishing radii, similar to what we did in the previous series. To do this, we must place the petal iterations within a loop that determines the radius of the petals. The entire iteration is shown below:

```
DO INC=1 TO .1 BY -.1;
  IF MOD(INC*10,2)<1 THEN ADDABIT=.75;
  ELSE ADDABIT=0;
  DO ANGLE=0 TO PI2 BY DELTA;
    A=15*SIN(6*ANGLE);
    R=A+5*SIGN(A);
    IF ANGLE=0 | (SIGN(R) + SIGN(RLAST))=0 THEN DO;
      FUNCTION='POLY'; X=0; Y=0;
      PEARL+1;
      COLOR='H'||PUT(INT(PETAL*3),HEX3.) || 'S4FF';
    END;
    ELSE DO;
      FUNCTION='POLYCONT';
      X=R*INC*SIN(ANGLE+ADDABIT);
      Y=R*INC*COS(ANGLE+ADDABIT);
      END;
      RLAST=R;
      OUTPUT;
  END;
END;
```

One thing that should be mentioned about generating your own points around a circle is that the actual circularity of the resulting display is dependent on the characteristics of the display device. Most terminals and hardcopy devices have aspect ratios that prevent X and Y coordinate positions from approximating a circle, even if the points mathematically describe a circle. In the next section we discuss the PIE function and its various uses, including its ability to overcome the problems associated with describing truly round objects.

There are several ways to circumvent an aspect ratio problem. One way is to use the HSIZE= and VSIZE= options to provide a "square" environment in which to draw circular objects, for example, GOPTIONS HSIZE=7 VSIZE=7;

This method yields a round circle, but it is no longer centered on the device. Another method, the one that we will use to draw a round flower, is to use the template facility of the GREPLAY procedure to provide a "square" environment that is centered on the display device:

```
GOPTIONS NODISPLAY;
PROC GANNO ANNO=FLOWER DATASYS GOUT=FLOWER NAME=NAME;
GOPTIONS DISPLAY;
RUN;
PROC GREPLAY IGOUT=FLOWER GOUT=GSEG NOFS;
TO=FLOWER;
TDEF SQUARE 1/LLX=0 LLY=0
ULX=0 ULY=100
LX=100 LY=0
SCALEX=.75
2/LLX=0 LLY=0
ULX=0 ULY=100
URX=100 URY=100
LX=100 LY=0;
TEMPLATE=SQUARE;
TPLAY 1:FLOWER 2:PIE;
QUIT;
RUN;
```

As long as we are using PROC GREPLAY to display this picture, we can use the template facility to our advantage to make a nice little bouquet of virtual flowers:

```
GOPTIONS NODISPLAY;
PROC GANNO ANNO=FLOWER DATASYS GOUT=FLOWER NAME=NAME;
PROC GREPLAY IGOUT=FLOWER GOUT=GSEG NOFS;
RUN;
TO=TEMP;
TDEF SQUARE 1/LLX=0 LLY=0
ULX=0 ULY=100
URX=100 URY=100
LX=100 LY=0
SCALEX=.75
2/LLX=0 LLY=0
ULX=0 ULY=100
URX=100 URY=100
LX=100 LY=0;
TEMPLATE=SQUARE;
TPLAY 1:FLOWER 2:PIE;
IGOUT=GSEG;
MODIFY 1/NAME=ONE;
TPLAY 2:1;
MODIFY 2/NAME=THO;
TPLAY 2:1;
MODIFY 3/NAME=THREE;
TPLAY 2:1;
MODIFY 4/NAME=FOUR;
TPLAY 2:1;
MODIFY 5/NAME=FIVE;
TPLAY 2:1;
MODIFY 6/NAME=SIX;
```
Drawing Truly Round Circles

The problems associated with drawing truly round circles by generating your own points can be easily overcome by using the family of annotate functions designed for generating pie charts:

PIE
PIEXY
PIECNTR.

Drawing a round circle with the PIE function is simply a matter of indicating where you would like the center of the circle to be located (X, Y), how large the circle should be (SIZE), what color you want the circle (COLOR), and what pattern is to be used to fill the circle (STYLE). The ANGLE variable provides the starting angle for drawing the circle; the ROTATE variable provides the ending angle. The following example illustrates the use of these variables to draw a round circle on any graphics device:

DATA PIES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN XSYS YSYS HSYS '2' X Y SIZE 50
STYLE 'SOLID'
FUNCTION 'PIE' ROTATE 2;
DO A=0 TO 358 BY 2;
  COLOR='H' || PUT(A,HEX3.) || '84FF';
  ANGLE=A;
  OUTPUT;
END;
PROC GANNO ANNO=PIES DATASYS; RUN;

A slightly different effect can be wrought from the PIE function by changing the color with decreasing radii instead of rotating around the pie:

DATA PIES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN XSYS YSYS HSYS '2' X Y SIZE 50
STYLE 'SOLID'
FUNCTION 'PIE' ROTATE 360 ANGLE 0;
DO A = 360 to 2 BY -2;
  COLOR='H' || PUT(A,HEX3.) || '84FF';
  SIZE=A/7.2;
  OUTPUT;
END;
PROC GANNO ANNO=PIES DATASYS; RUN;

Although in most cases we would use the PIE function to draw pie charts, in this tutorial we are going for a stroll on the wild side. The values for X and Y provide the center location for a circle. In previous examples we used iterative loops to move an imaginary pointer around the page by providing X and Y values for different functions. The PIE function is no different, but since we are dealing with circles, why not generate our circles with origins that are circularly oriented?

To generate X and Y coordinates that lie in a true circle, we can use the PIECNTR and PIEXY functions. The PIECNTR function allows us to generate a circle without drawing or filling in any lines. The PIEXY function allows us to identify points around any circle previously defined by either the PIE or PIECNTR functions.

As a simple example, let's generate twelve circles located at the hours of a clock face:

DATA PIES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN XSYS YSYS HSYS '2' X Y SIZE 50
STYLE 'SOLID'
FUNCTION 'PIE' ROTATE 360 ANGLE 0;
DO A = 360 to 2 BY -2;
  COLOR='H' || PUT(A,HEX3.) || '84FF';
  SIZE=A/7.2;
  OUTPUT;
END;
PROC GANNO ANNO=PIES DATASYS; RUN;
Since we are advancing around a circle, we can allow the current position of the circle to determine the color of the circle:

```
DATA PIES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN XSYS YSYS HSYS '3' STYLE 'SOLID';
DO B=0 TO 360 BY .360/12;
FUNCTION='PIECNTR'; X=50; Y=50;
SIZE=32; OUTPUT;
FUNCTION='PIEXY'; SIZE=1;
ANGLE=B; OUTPUT;
FUNCTION='PIE';
X=.; Y=.; SIZE=8; ANGLE=0;
ROTATE=360;
COLOR='H' || PUT(B,HEX3.) || '84FF';
OUTPUT;
END;
PROC GANO ANNO=PIES DATASYS;
RUN;
```

We now have a sweep of color that is the beginning of a very interesting display. Suppose we replicate this sweep at ever-diminishing radii, using ever-diminishing sizes of circles for each sweep, and shifting the colors with each change of radius? Even if you have no idea what I am asking, consider the following changes in code:

```
DATA PIES;
LENGTH FUNCTION COLOR STYLE $ 8;
RETAIN XSYS YSYS HSYS '2';
DO I=36 TO 1 BY -.5;
DO A=0 TO 360 BY 36;
FUNCTION='PIECNTR'; X=50; Y=50;
SIZE=1; OUTPUT;
Q=4*A^2/20;
FUNCTION='PIEXY'; ANGLE=A;
SIZE=1; OUTPUT;
FUNCTION='PIE';
COLOR='H' || PUT(A,HEX3.) || '84FF';
X=.; Y=.; SIZE=1/4; STYLE='SOLID';
ANGLE=0; ROTATE=360;
OUTPUT;
END;
END;
PROC GANO ANNO=PIES DATASYS;
RUN;
```

We have probably gone as far as we need to go in this direction, so let's back up a bit and talk again about using the PIEXY function to generate X and Y coordinates. You may have noticed that this function does not produce actual values for the variables X and Y, but rather it updates the values of the internal annotate variables XLAST and YLAST. To make those internal values available to the PIE function we simply allow the X and Y variables to take on missing values.

Suppose we wanted to use the points generated by the PIEXY function to draw a portion or all of a polygon, using the POLY and POLYCONT functions. A regular missing value to the POLYCONT function means that the polygon has discontinuous segments. For that reason, a special missing value, -999, is designated for use with the POLYCONT function when you want to use the values of XLAST and YLAST to continue the polygon. This special value is only available in percentage or screen cell frames of reference, not in data frames of reference.

The following example illustrates how this works:

```
DATA PIESLICE;
LENGTH FUNCTION STYLE COLOR $ 8;
RETAIN XSYS YSYS HSYS '3';
FUNCTION='PIECNTR'; SIZE=50; X=50; Y=50;
OUTPUT;
DO A=0 TO 150 BY 30;
COLOR='H' || PUT(A,HEX3.) || '84FF';
FUNCTION='POLY'; X=50; Y=50;
STYLE='SOLID'; OUTPUT;
FUNCTION='PIEXY'; SIZE=1;
ANGLE=0; OUTPUT;
FUNCTION='POLYCONT'; X=-999; Y=-999;
OUTPUT;
FUNCTION='PIEXY'; ANGLE=A+30; OUTPUT;
FUNCTION='POLYCONT'; OUTPUT;
END;
PROC GANO ANNO=PIESLICE;
RUN;
```

Once again, and for the last time, let's take this concept to the rim of reality. Let's draw a circle that is shaded horizontally from top to bottom, surrounded to the edges of the display with the same sequence of shadings, but arranged from bottom to top. Essentially, we will draw very small horizontal slices of the circle, and their corresponding outside polygons, changing colors in opposite directions as we proceed.

```
DATA PIESLICE;
LENGTH FUNCTION STYLE COLOR $ 8;
RETAIN XSYS YSYS HSYS '3';
FUNCTION='PIECNTR'; SIZE=50; X=50; Y=50;
OUTPUT;
B=181; INC=1;
DO A=0 TO 180 BY INC;
FUNCTION='PIEXY'; SIZE=1;
IF A <=90 THEN ANGLE=90-A;
ELSE ANGLE=450-A; OUTPUT;
FUNCTION='PUSH'; OUTPUT;
FUNCTION='POLY'; X=.; Y=.; STYLE='SOLID';
COLOR='H' || LEFT(PUT(A,HEX3.)) || '84FF';
OUTPUT;
FUNCTION='PIEXY'; SIZE=1;
IF A <=90 THEN ANGLE=90+A+INC;
ELSE ANGLE=450-A-INC; OUTPUT;
FUNCTION='PUSH'; OUTPUT;
FUNCTION='POLYCONT'; X=-999; Y=-999;
STYLE='SOLID'; OUTPUT;
FUNCTION='PIEXY'; SIZE=1;
IF A <=90 THEN ANGLE=90+A+INC;
```
Conclusion

We have only scratched the surface of the capabilities present in the annotate facility. What should be noted is that if you can imagine something as it would appear on a piece of paper or film, chances are very good that it can be accomplished within the realm of SAS/GRAPH software. SAS and SAS/GRAPH are registered trademarks of SAS Institute Inc., Cary, NC, USA.