# Importing Data from External Files to NeoVisuals® Software

James Richard Hagen, SAS Institute Inc.

## ABSTRACT

The application of three-dimensional graphics has evolved into two distinct categories: images based on ideas, concepts, and imagination; and images based on data from tangible, measurable objects. This paper addresses the processes involved in the second area; importing data into NeoVisuals software whether that information is used to recreate an object, change how it looks, or move it. Initial Graphics Exchange Specification (IGES) and Computer Graphics Metafile (CGM) format files are discussed, as well as methods for importing unformatted raw data for visualization.

## INTRODUCTION

Three hundred and sixty years ago, the French philosopher René Descartes divided everything known into two categories: (1) "Mind Stuff" and (2) "Other Stuff". "Mind Stuff" consisted of abstract ideas and concepts, created out of a unique human ability, while "Other Stuff" represented all that is tangible and measurable. Three-dimensional graphics and animation also can be described as having two distinct functions, each supporting one of Descartes' categories.

The first function is communicating ideas.

Communicating ideas by means of computer graphics is very much a creative pursuit. Imagery is conceived, arranged, and articulated using color and composition with specific intents. Communicating ideas is the primary use of computer graphics today. Examples can be found in advertisements, presentation slides, and corporate logos. Even text itself is an abstract symbolic construction used to convey meaning.

Communicating information by means of computer graphics is "show me what we know," "show me what we can measure." The image is produced from data gathered from a tangible or measurable object. These images are not based on creative, artistic, abstract ideas ("Mind Stuff"), but rather they come from the concrete world of substance, measurement, and form ("Other Stuff").

This paper explores the second use of three-dimensional computer graphics—communicating information based upon tangible, measurable sources. External data are represented in one of three ways: creating objects, changing the way objects look, and moving objects. These areas provide the framework of this paper. For clarity, I will refer to these areas as (1) construction, (2) attribute modification, and (3) animation.

## CONSTRUCTION

The most common use of external data is to reconstruct an object electronically. NeoVisuals software achieves this in a variety of ways. Let's first look at the three standardized format file types that can be read directly into NeoVisuals software: IGES, CGM, and FEA. Then, we will examine the manipulation of NeoVisuals software data files to accommodate nonstructured data.

### IGES

IGES stands for Initial Graphics Exchange Specification, a widely used standard for representing three-dimensional objects in the computer-aided design and computer-aided manufacturing (CAD/CAM) industry. An IGES file breaks down the geometry of the object into entity types (lines, arcs, splines, and so on). Files in IGES format can be read directly. However, not all entity types will translate into NeoVisuals software. The types that will translate are

| Entity Number | Entity Type |
|---|---|
| 106 | Copious data (forms 12 or 63) |
| 110 | Line |
| 112 | Parametric spline curve |
| 114 | Parametric spline surface |
| 126 | Rational B-spline curve |
| 128 | Rational B-spline surface |

Here is an example of an IGES file. This file was created using NeoVisuals software and represents the code necessary to represent the letter "A".

```
IGES file generated from a NeoVisuals system by the
translator from SAS Institute Inc., translator IGES
,,8Higes.txt,8Higes.txt,12HNeoVisuals       ,8HIGES 3.0,
32,8,24,8,56,1H0,1.0,3,4HUNIT,1,1.0,13H900315.082251,
-0.45795,1.15, James Richard Hagen,SAS Institute Inc.;
     106      1      0      1      0      0      0
     106      0      0      4     12      0      0 POLYGON
106,2,11,0.0,0.15,-0.45795,0.0,0.15,-0.25165,0.0,0.9562,
-0.00475,0.0,0.517,0.12935,0.0,0.517,-0.13925,0.0,0.3664,
-0.18535,0.0,0.3664,0.17435,0.0,0.15,0.24475,0.0,0.15,
0.45795, 0.0,1.15,0.10205,0.0,1.15,-0.11115;
S      2G      3D      2P
4
```

For a detailed description of the IGES format standard, look in the *Initial Graphics Exchange Specification Version 4.0* (see references).

### CGM

CGM stands for Computer Graphics Metafile. A CGM file contains two-dimensional vector information.

Vectors define the outlines of various shapes from the two-dimensional source. With this information, NeoVisuals software can manipulate each shape independently in three-dimensional space, either keeping a shape planar or extruding it into a voluminous solid. There are some limitations, however, when using CGM files. The *NeoVisuals Software Command Reference Manual* (pg 86) states:

> **NeoVisuals software reads only clear text-encoded CGM files. It does not read binary- or character-encoded CGM formats. Also, NeoVisuals software ignores those commands that are not part of the subset that SAS software exports. Also, the following commands that SAS software generates are ignored:**
>
> **LINEWIDTH SPECIFICATION MODE (1)**
> **MARKERSIZE SPECIFICATION MODE (2)**
> **EDGEWIDTH SPECIFICATION MODE (3)**
> **LINE TYPE (4)**
> **INTERIOR STYLE (5)**
> **EDGE VISIBILITY (6)**
>
> **1. Parameters are ignored, and lines are a single thickness.**
> **2. Sizes are calculated using ABSOLUTE vdc coordinates.**
> **3. Parameters are ignored, and lines are a single thickness.**
> **4. All lines are solid.**
> **5. All interiors are filled solid.**
> **6. Edges are not displayed.**

SAS® Software (Version 6.06) provides users with access to a wealth of maps. These maps can be output into a CGM format file that NeoVisuals software can read directly. Here is a sample program that retrieves the U.S. map and outputs it into a CGM file.

```
x alloc f(gsasfile) da(cgmclear.us)
     NEW LRECL(256) RECFM(V B);
goptions reset=(axis, legend, pattern, symbol, title,
     footnote) htext= ftext= ctext= target=  gaccess=;
goptions device=cgmclear  ctext=yellow   interpol=join
gsfmode=replace  noprompt    display gaccess=gsasfile;

 /*-------------------------------------------------*
  | Summary:                                        |
  |    Creating a map using the data set            |
  |    -NONE- and mapping on                        |
  |    -NONE-.                                      |
  | Generated: 06MAR90 16:34:08                     |
  *-------------------------------------------------*/

 /*-------------------------------------------------*
  | The GOPTIONS statement allows you to have more  |
  | control over the final appearance of your output|
  | such as fonts, colors, text height and so on.   |
  | The output device and destination is also       |
  | specified in the goptions statement.            |
  *-------------------------------------------------*/

goptions                  ctext=yellow   interpol=join;

 /*-------------------------------------------------*
  | PATTERN statements allow you to define colors and|
  | patterns in the chart, map or plot that you are |
  | creating.  SAS/GRAPH uses any pattern statements|
  | that you specify.  If more are needed, default  |
  | PATTERN statements are used.                    |
  *-------------------------------------------------*/

pattern1 color=RED value=MEMPTY;
pattern2 color=BLUE value=MEMPTY;
pattern3 color=YELLOW value=MEMPTY;
pattern4 color=CYAN value=MEMPTY;
pattern5 color=GRAY value=MEMPTY;
pattern6 color=PINK value=MEMPTY;
pattern7 color=WHITE value=MEMPTY;
pattern8 color=GREEN value=MEMPTY;
pattern9 color=MAGENTA value=MEMPTY;
pattern10 color=ORANGE value=MEMPTY;

 /*-------------------------------------------------*
  | This produces a 'dummy' response data set to    |
  | be used with PROC GMAP to produce a default map.|
  *-------------------------------------------------*/

data ASSISTMP;
   length state 3.;
   dummy = 1;
   do state = 1 to 56;
      output;
      end;
run;

 /*-------------------------------------------------*
  | This produces the actual map and any options    |
  | that directly relate to the map.                |
  *-------------------------------------------------*/

proc gmap data=ASSISTMP
          map=MAPS.US
          all;
   id state;
   choro DUMMY/
   nolegend
   discrete
   ;
run;
quit;

x free f(gsasfile);
```

### FEA

FEA stands for Finite Element Analysis. This format is widely used by software packages that analyze stress, strain, temperature, energy, pressure, or fluid flow of objects. FEA data file format has four separate manifestations detailing specific information about the object. The four file types are

- node files
- element files
- attribute files
- displacement files.

Information from these files creates objects that change colors based on temperature or stress, that deform into warped or melted shapes, or that depict fluid resistance. The specific format of each file type is described in the *NeoVisuals Software Command Reference Manual* (pp 151-159).

### Data Files

NeoVisuals data files contain command language statements. This command language file can be manipulated into constructing objects that represent your external data. The structure of these commands is very straightforward. To construct a three dimensional bar for a bar chart, you need only this command: BOX HEIGHT=4. In this example, I specified a height of 4 units. With other options available in the BOX command, I can specify color, location, width, and length. You could easily write a program or shell script to generate a bar chart,

substituting new height variables each time the program is executed.

A more complex example is generating a mathematical surface. The C program below creates a data file that produces the surface represented by this formula:

$$Y = \sin\sqrt{X^2 + Z^2}$$

Some of you may recognize this as the "Cowboy Hat"; the PROC G3D example in the *SAS/GRAPH Guide for Personal Computers*.

```
#include <math.h>              double
#include <stdio.h>
#include <string.h>


float    function_height(float x, float z);


main()
{
 int       vertex;
 float     resolution,xlow,zlow,xhigh,zhigh,neo_inc,
           plot_inc_x,plot_inc_z,nx,nz,px,pz;
 double    ny,py;

 FILE      *datafile;

 datafile = fopen("COWBOY.DAT","w");
 printf("Enter grid resolution:  ");
 scanf("%f",&resolution);
 printf("Enter x range:  ");
 scanf("%f %f",&xlow,&xhigh);
 printf("Enter z range:  ");
 scanf("%f %f",&zlow,&zhigh);
 fprintf(datafile,
   "! Formula:  SIN(X*X + Z*Z)\n"
   "! XRange %f to %f, ZRange %f to %f\n"
   "! Resolution %.0f by %.0f\n\n",
   xlow,xhigh,zlow,zhigh,resolution,resolution);

 fprintf(datafile,"! Make initial flat grid\n\n");

 neo_inc = 10/resolution;
 fprintf(datafile,"FIGURE Type=Vector $P\n",neo_inc);
 for (nx = -5, ny = 0, nz = -5; nx <= 5 + (1.5*neo_inc);
 nx += neo_inc)
   fprintf(datafile,
     "\t%1.3f\t%1.3f\t%1.3f\t/\n",nx,ny,nz);
 fprintf(datafile,"\n\n");
 fprintf(datafile,
   "move FIGURE.F;1 by 0 0 %f $m %1.0f OBJECT.O;1\n\n",
   10*neo_inc, resolution+1);

 fprintf(datafile,"cancel FIGURE.F;1 FIGURE.F;2\n\n"
   "reassign Type=Surface \n  OBJECT.O;1\n\n");

 fprintf(datafile,
   "! Move individual vertices to proper height\n\n");
 plot_inc_x = (xhigh - xlow)/resolution;
 plot_inc_z = (zhigh - zlow)/resolution;

 for (px = xlow,
   pz = zlow,
   vertex = 1;  pz <= zhigh; pz += plot_inc_z)
 {
   for ( ; px <= xhigh; px += plot_inc_x, vertex++)
   {
     fprintf(datafile,
     "\tpmove FIGURE.FM;%d \t 1 by 0 %f 0\n",
     vertex,function_height(px,pz));
   }
   px = zlow;
 }

 fprintf(datafile,"\n\ncancel /\n");

 for ( vertex = resolution+1;
   vertex <= (resolution+1) * (resolution+1);
   vertex += (resolution+1))
   fprintf(datafile,"\tFIGURE.F;%d /\n",vertex);

 for ( vertex = 1 + (resolution * (resolution+1));
   vertex < (resolution+1) * (resolution+1) ;
   vertex++)
   fprintf(datafile,"\tFIGURE.F;%d /\n",vertex);

 fprintf(datafile,"\n\nend\n\n");
}


float    function_height( float x, float z)
{
  return(sin((x*x)+(z*z)));
}
```

This program does more than generate the mathematical surface. It prompts you for range values in both the X and Z direction (with NeoVisuals software the Y direction is up), and for the precision (the smoothness) of the resulting surface. The program then positions the created surface about the origin at a scale of 10 by 10, so that the default position of the camera is pointing directly at the surface, regardless of the selected ranges.

NeoVisuals commands can easily convert data into many different forms. If your data are a series of slices (points outlining a shape in a fixed plane), you can produce figures in the software by surrounding the X Y Z data with the following command:

```
FIGURE $P
x1 y1 z1    /
x2 y2 z1    /
x3 y3 z3    /
    .
    .
    .
```

The $P option connects these points with a straight line from point-to-point. Other modes connect them with curves, splines, or rectangles. The slash is a continuation character. You can have more than one point definition on a line. There are over 36 options with the FIGURE command outlined in the *NeoVisuals Software Command Reference Manual* ; they specify name, materials, various surface attributes, and environmental considerations.

An additional word about slices: Slices can be connected using the CONNECT command, which converts them into solids. My best results occurred when each slice had the same number of definition points, or at least close.

## ATTRIBUTE MODIFICATION

Attribute modification is manipulating the surface characteristics of an object. Rather than using the data to describe the shape of an object, we can use the data (or external image) as a map to be applied to the surface of an existing object.

Two ways to change an object's appearance are (1) map an image or pattern onto its surface, and (2) directly influence the object surface itself.

### Mapping

Mapping takes a two-dimensional image and applies it to a two- or three-dimensional object, much like wrapping a gift with wrapping paper. There are several mapping options in the MAP command that determine the method of wrapping an image over an irregular surface. The image needs to be in a GENFILE format. The GENFILE format is the only image format that NeoVisuals software can directly manipulate.

Let's look at some image sources.

NeoVisuals Software contains a utility to convert a scanned image (from an Imapro OCS-450i® scanner) into a GENFILE. Patterns, textures, photographs, and charts can be scanned and converted into GENFILES. This utility, QCSGEN, is located in the /usr/local/bin directory (on the Silicon Graphics® workstation platforms), along with the current release of NeoVisuals software.

Another utility, TGAGEN, also in this directory, allows you to convert TARGA® (TGA) files into GENFILES. The TGA format is a popular file format for many software paint packages.

NeoVisuals software also produces GENFILE format files. This lets you map previously-created NeoVisuals images onto the surfaces of new objects. I recently did this with the Austin Division's 1989 Christmas Card. I took an image created for SIGGRAPH'89, a steaming teapot serving tea, and used it as a book jacket in the Christmas scene.

Surface mapping can take several forms. It can influence not only the object's surface color, but also other attributes as well. There are three other mapping modes: (1) Transparency Mapping—the lighter the region the more transparent, the darker the region the more opaque, (2) Bump Mapping—the lighter the region the more embossed, the darker the region the more recessed, and (3) Reflection Mapping—the lighter the region the more reflective, the darker the region the less reflective. These alternate mapping modes produce some wonderfully subtle results.

### Direct Modification

The second area of attribute modification is direct modification. You can assign surface color (along with 20 other surface attributes) directly, either to the entire object or to the individual facets within each object. These color changes can reinforce the displacement across a surface, or any other variable representation. When you assign color to a figure with the REASSIGN command, specify the color in one of several ways:

- by name (yellow, brown, white, and so on)
- by RGB values (of red, green, and blue)
- by HSL values (of hue, saturation, and lightness)
- by color table (a value ranging from 1 to 256).

To produce smooth color transitions across the surface, assign color attributes to the individual vertices (the corners of each figure). These vertices are modified with the VREASSIGN command (using the HSL mode). When using this method, you also need to reassign the whole object with the VERTEX ON option.

## ANIMATION

The final area that I would like to cover is animation. We will look at two examples of animations driven by external data: (1) a bar chart graph that changes over time, and (2) the reconstruction of the flight path in a Denver air crash. But first, let's look at the mechanism within NeoVisuals software that accomplishes the animation—the script file.

### Script Files

The script file contains commands that repeat for each image or frame of an animation. This file may contain up to 1000 different variables defined in the data file with the FUNCTION command. These script files may also be defined inside the data file and are called by the following data file command:

```
SCRIPT EXECUTE "file name" CYCLE "start" "end"
```

Besides creating animations, script files also provide a short-hand method of creation. Below is a sample data file that creates the script file, sets three function variables, and then displays the created image. The data file produces a grid of sixty squares, each with a slightly different color ranging from red, to green, to blue, and back to red.

```
FIGURE $P
0.0  5.0  0.0  /
0.0  5.9  0.0  /
0.9  5.9  0.0  /
0.9  5.0  0.0

SCRIPT DEFINE hsl
COPY FIGURE.F;1 1
MOVE FIGURE.F by %2 %1 0
REASSIGN hsl %3 100 50
FIGURE.F
SCRIPT END

FUNC 1 DEF $S
1 0  11 -1  21 -2  31 -3  41 -4  51 -5
FUNC 2 DEF LOOP CYCLE $L
1 0  10 9
FUNC 3 DEF TYPE REAL $L
1 1  60 360

SCRIPT EXECUTE hsl.SCR CYCLE 1 60
CANCEL FIGURE.F;1
PLOT 1 EYE 5 3 -12 FOCAL 5 3 0
SET RENDER DEPTH
NEWFRAME
END
```

When producing motion from external data, you need to manipulate the data into a form that NeoVisuals software can read. One way to do this is to create a

data file containing the function definitions. The form of the initial data file is

```
"create geometry"
"create script file"
DATA "functions.DAT"
"call script file"
END
```

**Bar Chart Example**

A simple bar graph of three blocks can be created in this manner. Suppose we have a file containing the height of each bar (columns) for six weeks (rows).

```
3.0   5.0   2.0
2.5   2.0   1.5
2.0   1.3   3.0
3.0   1.0   4.0
2.5   4.0   4.5
2.0   2.5   4.0
```

Let's produce a function definition using the data in the first column to vary the height of the first bar. Every two seconds, the bar will reach the height indicated.

```
FUNC 1 DEF TYPE REAL $C
 60   3.0   /
120   2.5   /
180   2.0   /
240   3.0   /
300   2.5   /
360   2.0   /
```

The above function example indeed manipulates the bar height, but it does not let the bar ever come to rest. Rather, the bar bobs up and down continuously. By changing the $C to a $S (step mode), the graph would instantly jump from one value to the next every two seconds. We want the bars to move smoothly to a value, pause, move to the next value, pause, and so on. The following function definition accomplishes this fluid movement.

```
FUNC 1 DEF TYPE REAL $L
  1   0.0   $C
 30   3.0   $L
 60   3.0   $C
 90   2.5   $L
120   2.5   $C
150   2.0   $L
180   2.0   $C
210   3.0   $L
240   3.0   $C
270   2.5   $L
300   2.5   $C
330   2.0   $L
360   2.0   /
```

In the example above, the bar moves to its intended height in the first second (3.0), pauses for the next second, then moves smoothly to its next value height (2.5), and so on. You can produce this form by whatever means available to you. SAS data sets can be manipulated by outputting a formatted report. UNIX® files can be manipulated using AWK (a UNIX utility) or by filtering the data through a program written in another programming language.

The function definitions can be kept separate from the data describing the construction of your particular graph. Below is a sample data file that incorporates the functions described above. I am assuming that the functions created are stored in the file "funcs.DAT".

```
BOX center 0.5 0.5 0
AXIS 0 0 0 0 0 -1
      BOX.O;1
MOVE BOX.O;1 by 3 0 0 Sc 2

SCRIPT DEFINE bar
RESET MODEL
RETRIEVE bar.BIN
TRANSFORM BOX.O;1 scale 1 %1 1 BOX.O;1
TRANSFORM BOX.O;2 scale 1 %2 1 BOX.O;2
TRANSFORM BOX.O;3 scale 1 %3 1 BOX.O;3
PLOT 1 eye 2 2.5 -10 focal 2 2.5 0
NEWFRAME
SCRIPT END

DATA funcs

STORE bar.BIN
SCRIPT EXECUTE bar.SCR cycle 1 450

END
```

**Airplane Crash Reconstruction Example**

Reconstructing the Denver air crash from the "black box" flight recorder was an interesting project. This particular plane was outfitted with a flight recorder that is now obsolete; it recorded only the plane's altitude, airspeed, and heading, at tenth-of-a-second intervals. Microphone clicks were also recorded, indicating radio transmissions, but they were not included in our project. The data received were incomplete, however; there were several gaps in the various sequences. A sample of the data is given below.

| TIME | ALT | AIRSPEED | HDG |
|---|---|---|---|
| 0:00:00.2 | . | . | 352 |
| 0:00:00.3 | . | 1 | . |
| 0:00:00.4 | 5340 | 1 | . |
| 0:00:00.9 | . | 2 | . |
| 0:00:01.0 | 5340 | . | . |
| 0:00:01.1 | . | . | . |
| 0:00:01.2 | . | . | 353 |
| 0:00:01.4 | . | 3 | . |
| 0:00:01.7 | . | . | 353 |
| 0:00:02.0 | . | . | . |
| 0:00:02.1 | . | 3 | . |
| 0:00:02.2 | . | . | 353 |
| 0:00:02.5 | . | 4 | . |
| 0:00:02.6 | 5340 | 4 | . |
| 0:00:03.0 | . | . | 354 |
| 0:00:03.3 | . | 6 | 354 |
| 0:00:03.5 | . | . | . |
| 0:00:03.7 | . | . | 354 |
| 0:00:03.9 | . | 9 | . |
| 0:00:04.0 | . | . | . |

This table shows the first four seconds of the take-off. Remember that this plane is taking off from Denver, the "Mile High City," so 5340 feet is the height of the runway above sea-level. The plane crashed shortly after takeoff. The entire flight lasted 55 seconds.

We entered the data into a SAS data set and manipulated them using a SAS program. I might add here that PROC EXPAND was very helpful in interpolating the missing values for us.

The following SAS program produced the NeoVisuals function definitions that we could obtain from the data.

```
%dmscmd(Clear,wid=0);
options tps=60;

proc fsprint data=demodat.den2demo edit;

data demodat.crash1
  (drop=hdg1 alt1 alt2 airspd2);
  format relalt relhdg 10.3;
  retain alt alt1 alt2 relalt
         hdg hdg1 hdg2 relhdg
         airspeed airspd2 0;
  set demodat.den2demo
      (drop=vaccgs mic) end=EOF;
  if  N =1 then do;              /* Set Init Values  */
     hdg1=352;                   /* Retained Head    */
     alt=5340;
     alt1=alt;                   /* Retained Alt     */
     airspeed=0;                 /* Airspeed         */
     end;
  if hdg<180 then hdg=360+hdg;   /* Correct Head     */
  if alt ne . then alt2=alt;     /* Retain Last Alt  */
  if hdg ne . then hdg2=hdg;     /* Retain Last Head */
  if airspeed ne . then          /* Retain Speed     */
     airspd2=airspeed;
  if EOF then do;                /* Retain Last Vals */
     if hdg=. then hdg=hdg2;     /* For Last Obs     */
     if alt=. then alt=alt2;
     if airspeed=. then
        airspeed=airspd2;
     end;
  relalt=alt-alt1;               /* Relative Alt     */
  relhdg=hdg1-hdg;               /* Rel Head (+- Z)  */

proc sort data=demodat.crash1   /* Remove Dupl Time */
        out=demodat.crash2 nodupkey;
  by time;

                                 /* Interpolate Vals */
proc expand data=demodat.crash2 out=demodat.crash3;
  convert alt relalt hdg relhdg airspeed / method=join;
  id time;

data demodat.crash4
  (drop=time1 time2 hdg1 knot ftps );
  format dist x z y 10.3;
  retain time1 hdg1;
  set demodat.crash3;
  y=relalt;                   /* Y-Axis=Rel Altitude  */
  if  N =1 then time1=time; /* Init Retained Time   */

  do;                         /* X-Axis = Calc'd Dist */
     knot=6080.2;                  /* 1 knot (feet) */
     ftps=(airspeed*knot)/(60*60); /* Feet/second   */
     time2=time-time1;          /* Time Int Bet OBS */
     dist=time2*ftps;           /* Calc'd Dist/Time */
     time1=time;                /* Update Retained  */
     end;

  do;                          /* Z-Axis=Calc'd Head  */
     rdn=(.01745329)*relhdg; /* Conv Degs to Rads   */
     x1=(dist*cos(rdn));     /* X Distance          */
     x+x1;                   /* Rel X to Last Pos   */
     z=(x*sin(rdn));         /* Z Distance          */
     end;

data  null ;                    /* Creates          */
  file demos(linpath) notitles;  /* Motion Path to   */
  set demodat.crash4             /* file LINPATH     */
      (keep=time x y z) end=EOF;
  if  N =1 then do;
     Put @1 "FIGURE NAME=LINPATH    $P" ;
     end;
  Put @4 x 10.2 @18 y 10.2 @30 z 10.2 @44 '/' ;
  if EOF then Put / @1 "END" //;
run;


/************************/
/*  To Read X Y Z       */
/*  SDS then output     */
/*  NeoVisuals Command  */
/*  Language Functions  */
/************************/

%dmscmd(clear,wid=0);
%macro rpt;
   %local f funclab;
%do f=1 %to 4;                      /* Function Label */
    %if &f=1 %then %let funclab=XCRASH; /* & Flname */
    %else %if &f=2 %then %let funclab=YCRASH;
    %else %if &f=3 %then %let funclab=ZCRASH;
    %else %if &f=4 %then %let funclab=ROTCRASH;
  data  Null ;
  file demos(&funclab) notitles;
  set demodat.crash4 end=eof;
  if  N =1 then do;                 /* Function Desc  */
     Put @1 "function" @13 "&f" @17 "define"
       @26 "label =" @34 "&funclab" @44 "/" /
       @1 "type = real" @18 "style = direct"
       @37 "loop = const" @52 "time = seconds" @70 "$c" ;
     end;
  Put @4 time  10.3 @;              /* Function Values */
      %if &f=1 %then Put @18 x 10.3 ;
      %else %if &f=2 %then Put @18 y 10.3 ;
      %else %if &f=3 %then Put @18 z 10.3 ;
      %else %if &f=4 %then Put @18 relhdg 10.3 ;
      @40 "/";
  if eof then put / @1 "END" /;  /* Neo End Of File */
%end;
%mend rpt;
%rpt;
run;
```

## SUMMARY

There are only three purposes for incorporating external data into any three-dimensional graphics and animation software package: (1) creating or modifying the shape of an object, (2) creating or modifying the surface of an object, and (3) creating or modifying the motion of an object. We have looked at construction from three readable formats (IGES, CGM, and FEA), and from unformatted data using data files.

We modified surface attributes in two ways: by mapping a two-dimensional GENFILE onto an object, and by modifying the object's vertices directly.

Finally, we looked at the process of animation. Using script files and function definitions, we directed the movement of various objects. I hope the process of importing NeoVisuals data from external files has been made more clear by these examples.

I would like to thank Steve Blackson, SAS Institute Inc. (Austin), for providing me with the Denver air crash SAS program.

You can get additional information about the product by contacting SAS Institute, specifically Karen Lee (919) 677-8000 ext. 7235

## REFERENCES

Penrose, Roger (1989), *The Emperor's New Mind--Concerning Computers, Minds, and the Laws of Physics,* Oxford University Press, 466 pp.

SAS Institute Inc. (1989), *NeoVisuals Software Command Reference Manual, Release 4.5.* Cary, NC: SAS Institute Inc., 618 pp.

SAS Institute Inc. (1987), *SAS/GRAPH Guide for Personal Computer, Version 6 Edition.* Cary, NC: SAS Institute Inc., 534 pp.

Smith, Bradford et al. (1988), *Initial Graphics Exchange Specification (IGES) Version 4.0,* National Bureau of Standards, publication number NBSIR 88-3813, U.S. Commerce Department, 545 pp. Copies may be obtained by contacting the National Technical Information Service, Springfield, Virginia, 22161 ($42.95).

**Trademarks**