PS 390 RENDERING OPTION RELEASE NOTES

EVANS & SUTHERLAND

May 1987
Version A2.V02
E&S #904015-606

CONTENTS

1. Introduction

Many of the capabilities described in the following sections were initially made available in various firmware versions for the PS 340. The PS 340 supports both a calligraphic and a raster display but lacks the advanced hardware and software for displaying dynamic calligraphic-quality lines and static shaded renderings on a single raster scope. Users already familiar with PS 300 graphics systems and new PS 390 users should note that, in addition to documenting changes and additions to functionality, these release notes also reflect changes in terminology (e.g., dynamic and static viewports versus calligraphic and raster displays). This is particularly important when referring to tutorials in the *PS 300 Document Set* that were originally written towards the PS 340 (raster and calligraphic) display configuration. These release notes are intended as a supplement to the *PS 390 Release Notes* and the original *PS 300 Document Set.*

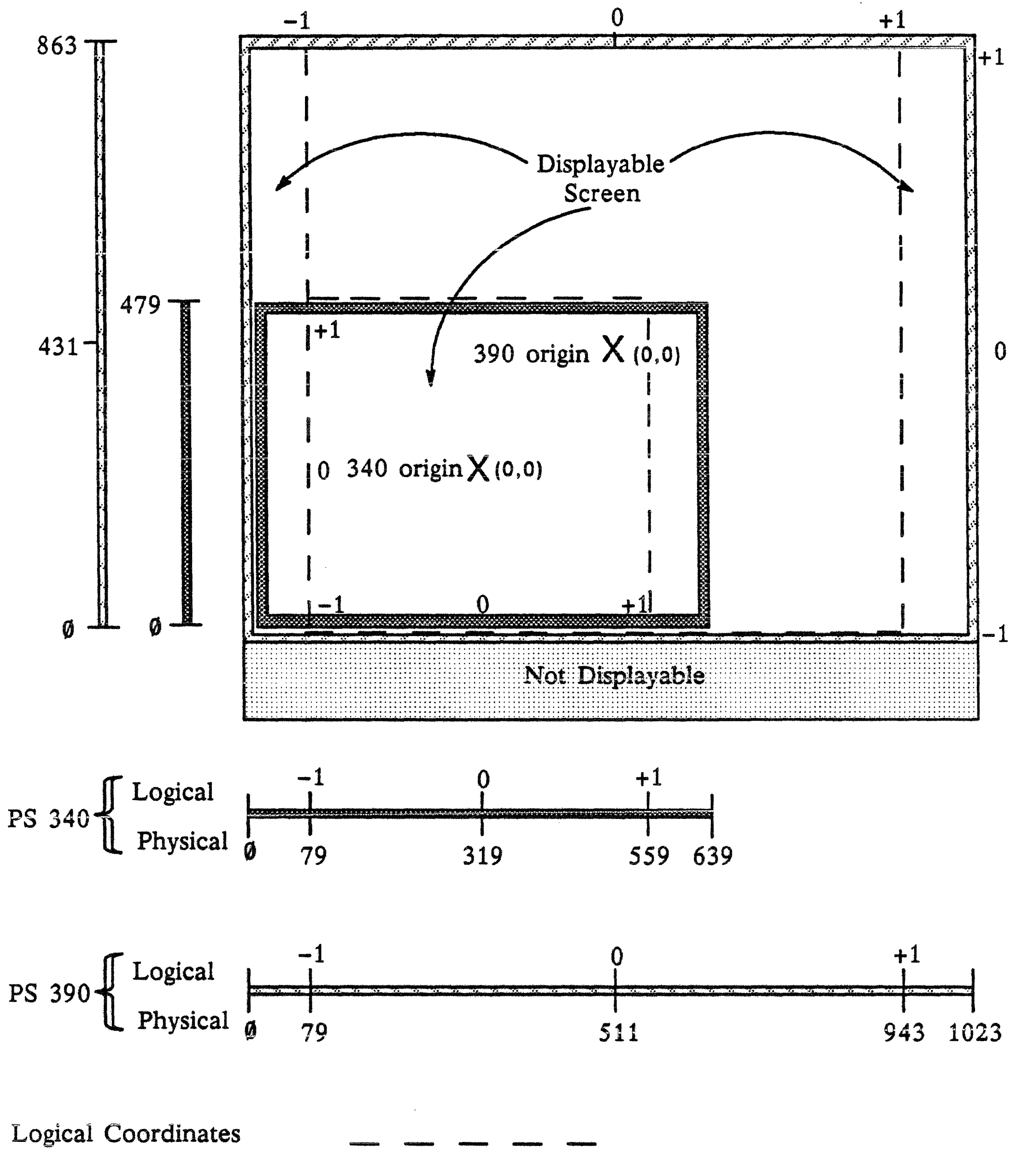
Since the PS 390 rendering firmware allows the display of both dynamic wireframe models and shaded images on the same raster scope and since the commands and operations governing the display of each type of model are inherently different, it is necessary to distinguish between how and where each type of model is displayed. Wireframe models are displayed in what is referred to as a "dynamic viewport." Hidden-line images or shaded renderings are displayed in what is referred to as a "static viewport." An unlimited number and combination of dynamic and static viewports are available within the limits imposed by the useable screen space shown in the display diagram on the following page.

1.1 Dynamic Viewport Operations

The following operations are performed in the dynamic viewport:

- Real-time manipulation of vector or wireframe representations of polygonal models
- Cross-sectioning defined by the sectioning plane (solid wireframe polygonal model)
- Sectioned rendering (wireframe polygonal model)
- Backface removal (solid wireframe polygonal model)

PS 340 Raster / PS 390 Display Comparison



Current PS 340 users will note that if existing PS 340 programs are run on the PS 390 without modification, objects will be displayed in the lower left quadrant of the screen. This is because the PS 390 has more pixels in x and y, and the physical pixel address 0,0 is in the lower left corner for both systems.

1.2 Static Viewport Operations

The following types of rendering styles are displayed in the static viewport:

- Wash shading
- Flat shading
- Phong shading
- Gouraud shading
- Raster hidden-line removal

A type of Gouraud shading has been added as a new shaded rendering style. This style of shading is obtained by sending a fix (8) to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node. Sending a fix (7) to input <1> of this node produces a type of Phong shading as before. Both styles of shading will produce a smooth shading of the object if normals are supplied with each vertex of the polygon. The Gouraud shading is faster but does not produce the quality of picture that the Phong shading will produce.

In Phong shading, the surface normal used is the normal derived by interpolating the normals supplied at each vertex. In flat shading, the normal used is the vector perpendicular to the polygon. In Gouraud shading the degree of light which is reflected is derived by first calculating the degree of light reflected at the vertices using the normal you supply and then interpolating between the vertices. In wash shading, no surface normal is used and no lights are used.

Note that after requesting a shaded image or hidden-line rendering in the static viewport (sending a fix (4), a fix (5), fix (6), fix (7), or fix (8) to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node), the dynamic viewport will go blank while the image is being painted in the static viewport. After the shaded image has been completely painted in the static viewport, the original object can be displayed in the dynamic viewport by sending a fix (0) to <1> of TurnOnDisplay (refer to Section 3).

2. Displaying Pictures

Because of the flexibility in defining the useable screen space to display both wireframe and shaded models, it is necessary that the code be organized so that images or viewports do not overlap each other. The overlapping of dynamic images on static viewports will corrupt the static image.

2.1 Defining Dynamic and Static Viewport Boundaries

Dynamic viewports are defined by viewport operation nodes in the hierarchical display structure (display tree). Refer to Volume 2A of the *PS 300 Document Set* ("Viewing Operation, Specifying a Viewport") for a complete description of how to specify viewports to display dynamic models.

Specifying **static** viewport boundaries is done by sending a 3D vector to input <3> of SHADINGENVIRONMENT. A 3D vector sent to input <3> of SHADINGENVIRONMENT specifies viewport pixel values. For example,

```
Send V3D(80,0,863) to <3>SHADINGENVIRONMENT;
```

would be a valid command to specify a square static viewport of 864 by 864 pixels on the PS 390. Note that specifying viewports by sending to input <3> of SHADINGENVIRONMENT is completely independent from dynamic viewport specification, which is done by creating a viewport node in the display data structure.

NOTE

Specifying static viewports with a 4D vector input (for logical coordinates) is not currently supported.

2.2 Clearing Viewports To Static Or Dynamic

Input <7> of SHADINGENVIRONMENT will clear the entire screen or the current viewport and specify if the viewport is to be treated as a dynamic or static viewport. If a rendering is requested, the rendered object appears in the current static viewport. To clear the current static picture to use that screen space for a dynamic viewport, the static viewport must be "cleared" to dynamic. Images are cleared in both the static and dynamic viewports by sending an integer or a Boolean to input <7> of SHADINGENVIRONMENT.

2.2.1 Clearing to Static

Sending a True to input <7> clears the entire screen to static and causes a screen wash with the current static background color. Sending False to input <7> clears only the currently specified static viewport and causes the viewport to be filled with the current static background color. Sending fix (0) to input <7> clears the entire screen to static and causes a screen wash with the current static background color (This has the same effect as sending a True to this input.) Sending fix (1) to input <7> clears only the currently specified static viewport and causes the viewport to be filled with the current static background color. (This has the same effect as sending a False to this input.) When requesting another rendering to be displayed in a current static viewport, it is not necessary to clear the viewport first, since this is accomplished by requesting a new rendering.

2.2.2 Clearing to Dynamic

Sending fix (2) to input <7> clears the entire screen to dynamic and causes a screen wash with the current dynamic background color. This must be done to clear either a shaded image or a dynamic image (from the entire screen or viewport), before displaying a new dynamic image. Sending fix (3) to input <7>, clears the currently specified dynamic viewport with the current dynamic background color.

To prevent images from being corrupted, do not display an image or use the terminal emulator in an area of the screen that already has an image without first clearing the screen (or viewport) with a dynamic wash. Refer to the SHADINGENVIRONMENT Function Description included in these release notes for more details.

3. Turning Structure Traversal On And Off

To protect an image from being corrupted while it is being rendered, the runtime firmware disables data structure traversal of dynamic images. After the rendered image has been completely painted on the screen, data structure traversal must be re-enabled to continue drawing lines, manipulating objects, rendering, or viewing the terminal emulator. The following command accomplishes this and could be connected to a function network controlled by a function button or key.

```
Send fix (0) to <1>TurnOnDisplay;
```

When the display is turned back on, if an object in a dynamic viewport overlaps a static viewport, the static image will be corrupted.

The following example is taken from the PS 390 function network provided on the standard PS 390 release distribution. The network is composed of a Sync function which clears the screen after a shaded image has been created and you no longer want to display it. This is accomplished by sending a fix (2) to TurnOn which sends a fix (2) to <7> SHADINGENVIRONMENT to clear the screen to Dynamic. At the same time, this network sends a fix (0) to <1> TurnOnDisplay so that the ACP will be re-enabled.

```
{ TurnOnDisplay when clearing screen }

TurnOn :=f:Sync(2);
Setup cness true <2>TurnOn;
Connect TurnOn <1>:<7>SHADINGENVIRONMENT;
Connect TurnOn <2>:<1>TurnOnDisplay;
Send fix(0) to <2>TurnOn;
```

The following network is in the Config.dat file which is used at system boot-up to initialize PS 390 display structures. In this Config.dat file, at the top of the display structure is the name SC0$. Just beneath it is a a Set Level_of_detail node named TurnOnDisplay.

Sending a fix (1) to <1> TurnOnDisplay disables the display of dynamic data structures (which is done automatically by the runtime rendering code), sending a fix (0) turns it back on.

```
{ From the PS 390 Config.dat file }

SC0$:=Set Display All On Then Turn On Display1;
TurnOnDisplay1 := Set Level 0 Then SetLod1;
SetLod1 := If Level = 0 Then VPF1$;
```

3.1 Toggling Between a Dynamic Viewport Rendering and the Original Object

Backface removal, sectioning, and cross-sectioning rendering operations take place in the dynamic viewport and are requested by sending a fix (1), fix (2), or fix (3), to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node). To toggle between the current rendering and the original object, send fix (0) to input <1> of the SOLID_RENDERING or SURFACE_RENDERING node. Hidden-line renderings are no longer performed in the dynamic viewport and cannot be toggled in this way.

4. New Hidden-line Algorithm

A new algorithm has been added to the choices of available rendering operations. When a fix(4) is sent to <1> Surface or Solid rendering node, a raster hidden-line algorithm is used to create a hidden-line rendering in the static viewport. This algorithm is different from the calligraphic hidden-line algorithm released with previous software versions of the PS 340.

This algorithm can properly handle interpenetrations of polygons by resolving obscurity on a sub-pixel by sub-pixel basis.

Currently, (with the initial release of the firmware) readback of pixel data is not supported. Therefore, the hidden-line pixel data in the static viewport may not be read back to the host.

The hidden-line image is displayed on a default black background. Colored backgrounds may be used, but the interior of the object will still be black. Colors for the hidden-line images are defined by the "with outline" clause of the POLYGON command which is used to index into the attribute table. Colors can be changed by updating the Spheres and Lines Attribute table with the name of a tabulated vector list sent to input <14> of SHADINGENVIRONMENT. Refer to section 8.3.

5. Polygon Edge Enhancement

Polygons of shaded images may now be drawn with colored edges in the static viewport. When a TRUE is sent to <15> SHADINGENVIRONMENT, outlines are overlaid on top of all polygon borders. These outlines have z-values, so obscurities are properly resolved. Also, full antialiasing is turned on temporarily.

This feature is useful in applications such as finite element analysis when it is sometimes difficult to distinguish boundaries when polygon color interpolation is used. For the static viewport, polygon edge colors are specified by the "with outline" clause of the POLYGON command. The specifier (h) in the "with outline" clause is an index into the Spheres and Lines Attribute Table loaded at boot time. It is recommended that users encode their own attribute table values in their application programs, as the current default attribute table is subject to change with future releases. The current defaults for the attribute table (for polygon edges) are as follows:

With Outline (h)	Attribute Index	Default Color
0 to 5	1	White
6 to 11	3	Green
12 to 17	5	Yellow
18 to 23	7	Cyan
24 to 29	9	Black
30 to 360	11 to 127	Color Wheel

This attribute table may be altered by encoding new table entries into a PS 300 tabulated vector list and sending the name of the vector list to input <14> of SHADINGENVIRONMENT. This attribute table is also used with raster sphere and line primitives, and a complete description of updating attributes can be found in Section 8.3.

In static viewports, if a Boolean is sent to input <15> SHADINGENVIRONMENT, then all the edges of all the polygons will be toggled on/off. True causes lines to be drawn around the polygons to enhance their edges. False causes polygons to be rendered without enhanced edges (default).

Note that for polygon outlines displayed in the **dynamic** viewport, the specifier (h) in the "with outline" clause refers to a hue value in the color wheel and not to the default attribute table. Refer to the POLYGON Command Summary change pages in these release notes.

5.1 Polygon Edge Smoothing

It is possible to choose between producing an object with jagged polygon edges at a quick rate, or producing an object with smoother edges at a slower rate. This control is accomplished through input <5> of SHADINGENVIRONMENT.

Sending fix (0) to input <5> of SHADINGENVIRONMENT produces no edge-smoothing and is the fastest shading option. Sending fix (1) to input <5> of SHADINGENVIRONMENT produces smooth edges but may not smooth along edges of interpenetrating polygons or correctly resolve obscurity between surfaces that are extremely close. This has a speed intermediate between a fix (0) and a fix (2). Sending fix (2) to input <5> of SHADINGENVIRONMENT produces edge smoothing along all edges, high quality non-jagged intersections, and is slower.

6. Transparent Polygon Attribute - Static Viewport

Polygons can be given a transparent attribute. To accomplish this, the form of the ATTRIBUTE command has been changed to:

[COLOR h[, s[,i]]] [DIFFUSE d] [SPECULAR s] [OPAQUE t]

where t refers to a real number between 0 and 1.

The opaque specifier [t] in the ATTRIBUTE command allows the specification of a transparency level and is input as a real number between 0 and 1, with 1 being fully opaque and 0 being fully transparent. As t decreases from 1 to 0, more of the color of the obscured object(s) will show through. At t=0, the transparent polygon becomes completely invisible. If no opaque attribute is specified, the default is 1 (fully opaque).

Polygons that are rendered as transparent have no color of their own, but merely filter the color of objects appearing behind them. This is according to to the rule that each of the red, green, and blue components of the object behind is multiplied by the red, green, and blue components of the transparent polygon.

This means that a transparent object rendered over a black background will be invisible. This also means that a purely blue transparent object rendered over a purely red object, will make the red object look more black (depending on the value of the Opaque specifier). There are no specular highlights available on transparent objects.

To show polygon orientation relative to the eye point, the color which is transmitted through the transparent object is darkened according to the z-component of a surface normal. This means that with Phong, Gouraud, and flat shading, as the object bends away from the user, the transmitted color becomes darker.

Note that since the PS 390 does not currently allow readback of pixel data, transparency will not function in the overlay mode. All objects, either opaque or transparent, must be rendered together using the scan-line Z-buffer algorithm.

To render any objects as transparent, a TRUE must be sent at to input <11> of SHADINGENVIRONMENT at some time prior to rendering. This is a new input that allows transparency to be turned on (or off). Sending a TRUE to input <11> will cause only the transparent objects to be rendered transparent. Sending a FALSE to input <11> of SHADINGENVIRONMENT will cause all objects to be rendered as fully opaque, regardless of their attributes.

The node created with the ATTRIBUTES command has two new inputs <4> and <14>. Both inputs accept a real number to update the opaque value of the polygon's attributes, for the two sides of the polygon. Refer to the ATTRIBUTES Command Summary change pages in these release notes.

7. Polygon Color Interpolation

You can specify color at the vertices of a polygon to provide color interpolation across the polygon. To invoke this option, you must first specify the color in the polygon command. To accomplish this, the vertex definition in the polygon command has been changed to:

[S] x,y,z [N x,y,z] [C h[,s[,i]]]

where

C – indicates a color to be assigned to the vertex. This color will be interpolated across the polygon to the other vertices.

h, s, i – are coordinates of the Hue–Saturation–Intensity color system.

Polygons may be solidly colored by specifying a color through the attributes command or the colors may be assigned to the vertices by giving a color with each vertex specified. The color is specified by giving, first, the vertex and then, the color (h, s, i). If just the hue and saturation are given, the intensity will default to 1. If just the hue value is given, the saturation and intensity will default to 1. If no vertex colors are given, the vertex colors will default to those specified in the attribute clause.

Vertex colors must be specified for all vertices of a polygon or for none of them. However, as with normals, some polygons may have color at their vertices while other polygons may not have color at their vertices. This means that it is possible to have some objects in the picture color interpolated, while others are not. The default is false.

Although color of polygon vertices is specified h, s, i, the colors are linearly interpolated across the vertices in the Red–Green–Blue color system. If colors are not interpolating as expected, add more vertices to the polygon, or break up large solid volumes into smaller sub–volumes and assign the desired colors to the new vertices in the object.

Color for a polygon can be specified with both the ATTRIBUTES command and the color by vertex specification. A new input to the SHADINGENVIRONMENT function allows switching between attribute-defined color and vertex-defined color. Input <10> of SHADINGENVIRONMENT accepts a Boolean to determine how color will be specified. To use vertex colors rather than surface attributes, send TRUE to input <10> of SHADINGENVIRONMENT. To return to using the attributes specified in the ATTRIBUTE command, send FALSE to input <10> of SHADINGENVIRONMENT.

8. Spheres and Lines Capabilities

Spherical rendering (primarily used in molecular modeling) and raster–lines (primarily used for labeling) have been incorporated with the standard PS 390 raster rendering capabilities. Spheres and raster lines are represented as vector lists instead of an explicit PS 300 data type. Spheres are shaded consistent with the Phong shading style, allowing multiple colored light sources, specular reflections, and depth cueing. If "wash" shading is selected for polygons, spheres in the same rendering will be rendered with just one light source at the default position (straight on). Since spheres are represented as vector lists, no color interpolation or transparency is supported. Lines are rendered with their defined color with no shading, but may have depth–cueing applied. Hidden–element removal with spheres, lines, and polygons has been accomplished with a common z–buffer algorithm. (Additional discussion of the Spheres and Lines capability can be found in the *Enhanced CPK User's Guide E&S #901194–089.*)

8.1 Defining Raster Spheres and Lines

Since there are no explicit PS 300 data types for representing spheres or raster lines, **do not place sphere or raster-line data under a rendering operation node.**

To display line data in a static viewport, a tabulated vector list (3D tabulated vector) must be created. The "P" and "L" indicators specify the "moves" and "draws" for raster-line renderings just as they do for calligraphic display. For spherical data, a dots or itemized vector list (3D tabulated vector) must be created where each x, y, z is interpreted as a spherical center.

These vector lists must be tied to F:XFORMDATA functions which are connected directly to inputs in the SOLID_RENDERING node. Inputs <3>, <4>, and <5> have been added to the rendering node to accommodate sphere and line data. Input <3> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets the vectors as "moves" and "draws" for raster-line rendering. Similarly, input <4> of the rendering node accepts a transformed vector list and interprets each vector as an x, y, z spherical center for raster rendering. Input <5> of the rendering node accepts the original vector list to enable accurate scaling of the rendering.

8.2 Spheres and Lines Attribute Table

The attributes for lines (color) and spheres (radius, color, diffuse, specular) are stored in a default table created at system boot up. It is recommended that users load their own values into the attribute table, as the current attribute default table is subject to change with future releases. This table can be modified via input <14> of the SHADINGENVIRONMENT function. The table has the following components:

Hue	Saturation	Intensity	Radius	Diffuse	Specular

Hue is a real number in the range 0 to 360. Saturation and intensity are real numbers in the range 0 to 1. Radius is a real number greater than 0. Diffuse is a real number in the range 0 to 1. Specular is an integer in the range 0 to 255. The table is initialized as follows:

INDEX	Hue	Sat	Intensity	Radius	Diffuse	Specular
0	0	0	0.5	1.8	0.7	4 (Grey)
1	0	0	1	1.2	0.7	4 (White)
2	120	1	1	1.35	0.7	4 (Red)
3	240	1	1	1.8	0.7	4 (Green)
4	0	1	1	1.8	0.7	4 (Blue)
5	180	1	1	1.7	0.7	4 (Yellow)
6	0	0	0.7	1.8	0.7	4 (Grey)
7	300	1	1	2.15	0.7	4 (Cyan)
8	60	1	1	1.8	0.7	4 (Magenta)
9	0	0	0	1.8	0.7	4 (Black)
10–127						Color Wheel

Spheres use all six of these components. Lines use only the hue, saturation, and intensity components. Refer to Section 6 for polygon edge access to the attribute table.

The t field of each 3D tabulated vector is used as an index into this table. The table contains 128 entries (0–127).

For example, the following vector list represents three spheres with the color indicated.

```
SPHERE := VECtor list  TABulated N = 3
  P  1,2,3      t = 5      {yellow sphere}
  L  4,5,6      t = 6      {grey sphere}
  L  7,8,9      t = 7      {cyan sphere}
  ;
```

The following example represents a square with sides of the indicated colors.

```
RASTERLINE := VEC TAB N = 5
  P  0,1,0
  L  0,0,0      t = 5      {yellow}
  L  1,0,0      t = 2      {red}
  L  1,1,0      t = 3      {green}
  L  0,1,0      t = 4      {blue}
  ;
```

NOTE

Lines use the tabulated index of the point drawn "to" and not the point drawn "from." Thus, the tabulated index of position vectors is ignored and may be omitted.

8.3 Updating the Attribute Table

The attribute table may be updated by encoding the table entries into a PS 300 tabulated vector list and then sending the name of the vector list to <14> SHADINGENVIRONMENT. The six table components are encoded into two consecutive 3D vectors of the vector list. Hue, saturation, and intensity are encoded into the first x, y, z, respectively. Radius, diffuse, and specular are encoded into the second x, y, z, respectively. The table index is encoded into the t field of the second vector.

For example the following vector list would be used to update attribute table entry 5:

```
ATTRIBUTE TABLE := VEC TAB N = 2
   150,0.5,1     5.0,0.3,2  t = 5;
```

Updating would be accomplished by the command:

```
@@ SEND `ATTRIBUTE_TABLE´ TO <14>SHADINGENVIRONMENT;
```

Note that more than one table entry may be encoded into a vector list. The following vector list would be used to update attribute table entries 5, 6, and 7:

```
ATTRIBUTE TABLE := VEC TAB N = 6
   0,1,1        2.0,0.5,4       t = 5
   120,1,1      4.0,0.8,9       t = 6
   240,1,1      3.0,0.3,2       t = 7
   ;
```

If the attribute table is changed (using the above command) to the image of the three atoms defined in the example on the previous page, the yellow, grey, and cyan spheres would change to a small blue sphere, a large dull red sphere, and a shiny medium-sized green sphere.

8.4 Spherical Radii Scaling

To apply the correct radii scaling factor, the rendering node must examine the original or untransformed data. The name of the vector list representing the spherical data must be supplied on input <5> of the rendering node. The name is sent as a string in single quotes.

8.5 Sphere and Line Constraints

Window Restriction – For spheres to be rendered correctly, a cubical orthographic projection must be used (i.e., WINDOW command). Spherical renderings with perspective projections or any other non-cubical orthographic projections will not be displayed correctly.

Viewport Restriction – If spheres are to be rendered in conjunction with either lines or polygons, then only the following raster viewport should be used:

```
      Xleft = 64        Xright = 575
    Ybottom = -32         Ytop = 479
```

The command "SEND V3D (64,-32,511) to <3> SHADINGENVIRONMENT;" will set this raster viewport.

Lines and/or polygons can be rendered correctly in any raster viewport. Spheres alone can be rendered correctly in any raster viewport.

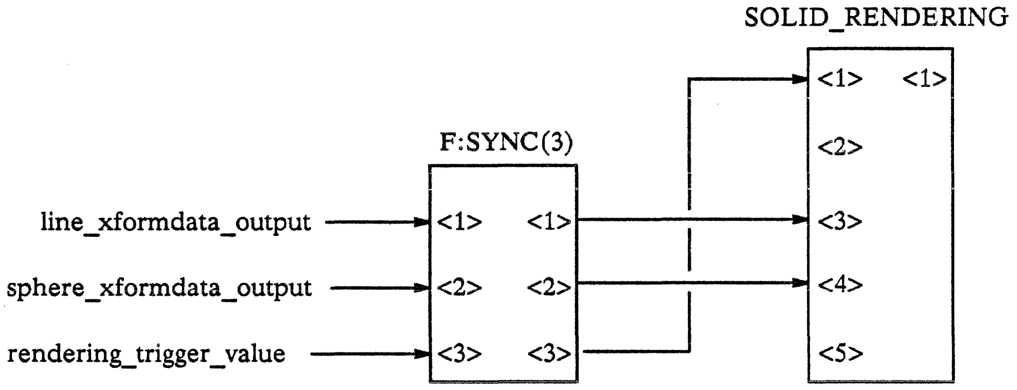
8.6 Edge-smoothing Mode for Spheres and Lines

Input <5> of SHADINGENVIRONMENT accepts an integer defining the level of edge-smoothing. For lines and spheres to be rendered correctly, a 1 or a 2 should be sent to this input. A value of 0 for edge-smoothing may result in incorrect renderings.

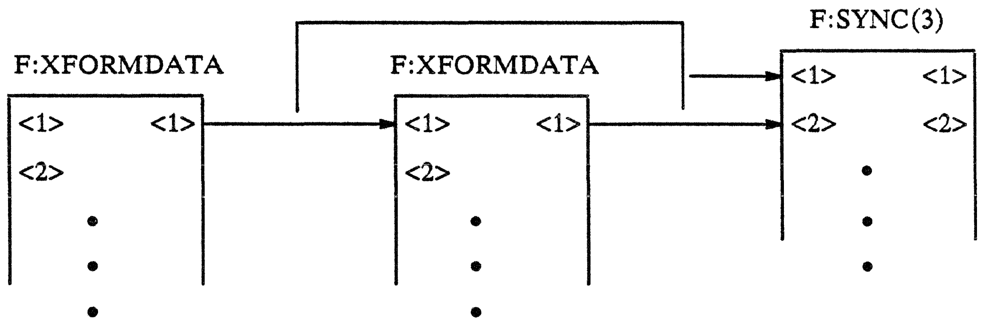
8.7 Function Network Considerations

Two potential timing problems exist with triggering the rendering node. Input <1> of the rendering node is the only active input. Inputs <3> and <4> which accept transformed data for rendering lines and spheres are constant inputs. (These inputs are constant inputs and initialized to NIL so that application programs written prior to A2.V01 will work without any modifications.) Since inputs <3> and <4> are constant inputs, you must guarantee that they have updated before the trigger is sent to input <1> of the rendering node. This is accomplished using the F:SYNC(n) function.

F:SYNC(n) sends its outputs in the order 1 to n. By connecting outputs 1 to n-1 of F:SYNC(n) to the constant inputs of the rendering node and by connecting output n of F:SYNC(n) to input <1> of the rendering node, you can guarantee that the constant inputs will be updated before the rendering node is triggered. This is shown in the following example.



The second potential timing problem deals with the triggering of F:XFORMDATA. An instance of F:XFORMDATA must not be triggered while it or any other instance of F:XFORMDATA is still active. Thus when using multiple instances of F:XFORMDATA, one instance should be used as the trigger for the next. (See the following example.)

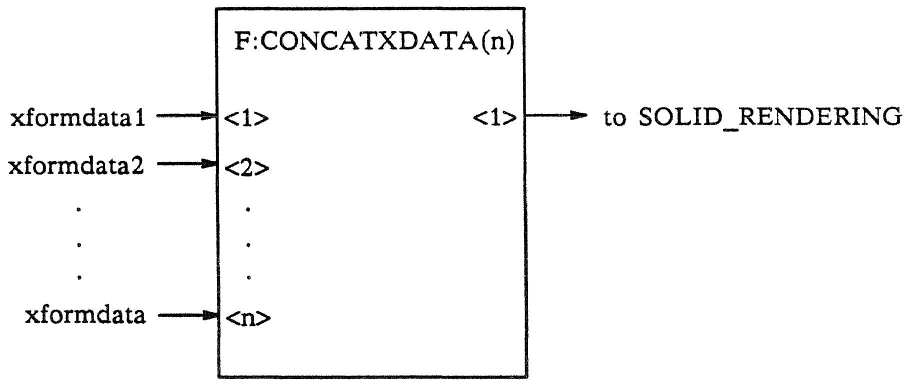


You must be aware of one other restriction when using F:XFORMDATA. Input <3> of F:XFORMDATA typically allows a name specification for the transformed data. However when using F:XFORMDATA in conjunction with a rendering node, this input must be left blank.

CAUTION

Naming the transformed data and then sending it to a rendering node, will result in a system failure.

F:CONCATXDATA(n) accepts up to 127 transformed vector lists (output from XFORMDATA functions) and concatenates them into a single transformed vector list. It is used to avoid the maximum vector restriction imposed on the output of F:XFORMDATA. The XFORMDATA function will return a maximum of 2048 vectors. This restriction passes on to the rendering node since the output of the XFORMDATA function is normally connected directly to the rendering node. To obtain a rendering of greater than 2048 vectors (or spheres), the output of multiple instances of XFORMDATA must be concatenated into a single transformed vector list, which can then be sent to the rendering node.

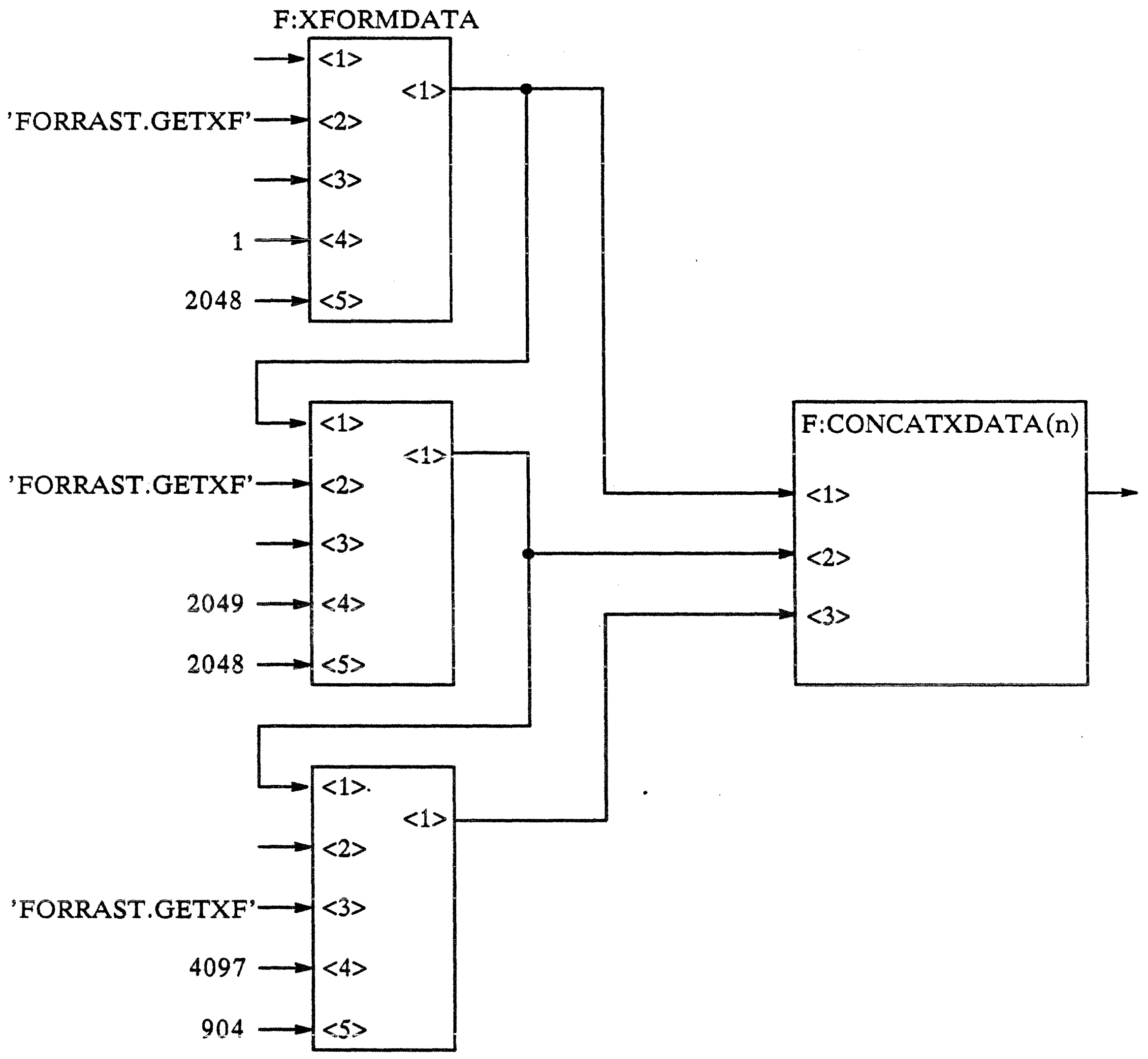


As previously discussed, multiple instances of F:XFORMDATA must be linked together for triggering purposes to ensure that one instance completes before the next one commences.

For example, assume that in the following network, the vector list SPHERES contains 5,000 vectors.

```
FORRAST := BEGIN_STRUCTURE
   GETXF := XFORM VEC;
   INSTANCE OF SPHERES;
END_STRUCTURE;
```

One instance of XFORMDATA could retrieve the first 2048 transformed vectors of SPHERES (vectors 1-2048). A second execution of XFORMDATA could retrieve the second 2048 transformed vectors (vectors 2049 -4096). And a third execution of XFORMDATA could retrieve the last 904 vectors (vectors 4097 - 5000). An illustration of this network follows.



9. SHADINGENVIRONMENT Function Additions

Many of the inputs to the SHADINGENVIRONMENT function have been mentioned throughout these release notes. For a complete description of this function, refer to the change pages to the PS 300 Document Set provided at the end of these release notes. Former PS 340 users should note that changes and additions to the following inputs have been made since the A2.V01 *PS 340 Release Notes*.

Reserved Input: Input <8> is currently reserved for future functionality.

Clear/Overlay Control: Input <9> accepts a Boolean which determines whether the screen is to be cleared with the current background color before the rendering is done, or whether the requested rendering will overlay the object already displayed on the raster screen. Because the PS 390 does not support the readback of pixel data, the object which overlays the original image will not be antialiased with the background color and will have jagged edges.

Spheres and Lines Attribute Table Update: Input <14> accepts the name of a vector list to update attributes for spheres and lines. The number of default colors has been expanded to 127 (refer to Section 8).

Raster Lines Z-value Control: Input <15> now accepts a Boolean or real number in the range of 0–1 which is added to the z-values of lines in raster renderings. Sending a 0 to this input will leave lines in their original z position. Sending a 1 to this input will force lines to be in front of everything else in the image. This feature may be desirable when rendering lines exactly along polygon edges. Leaving lines at their original z-values will cause obscurity problems with the edges of the the polygons. By adding an offset to the lines' z values, this obscurity problem is more easily resolved. The default offset is now 0.05.

If a Boolean is sent to this input, polygon edges are toggled on and off. TRUE causes lines to be drawn along polygon borders and temporarily turns on full antialiasing.

PS 390 PROGRAMMING EXAMPLE

This programming example sets up a display structure and provides a network to perform most of the rendering features of the PS 390. The display structure is set up for a sectioning plane, two light sources, and a display structure for the object. This display structure allows you to put the object in a dynamic wireframe viewport (lower right-hand corner) and three static rendering viewports in the other corners.

NOTE: for this display structure to work, you should instance your object as:

```
OBJECT := INST OF YOUR_OBJECT;
```

The rendering features provided by the network include: shading (Phong, Gouraud, hidden-line removal, etc.), light sources, background colors, color interpolation, viewporting, antialiasing, polygon edges on the rendered image, and real-time dynamic motion. Some of the excluded features of this network are: transparency and raster depth-cueing, but these can be controlled in local mode (@@).

Rendering Control Network

The rendering control network defines the operation of the function keys as follows:

FUNCTION KEYS:
1 - Render Key
2 - Raster Viewports
3 - Style Of Rendering
4 - Overlay Or Refresh The Rendering
5 - Antialiasing Selection
6 - Clears The Screen And Turns On The Display Of The Wireframe Model
7 - Solid Or Surface Rendering
8 - Color Interpolation
9 - Edges Of Polygon On Rendered Picture On/Off
10 - Reset Object
11 - On/Off Selection For Fkey #12
12 - Dial Muxer

NOTE

With the PS 390, after a picture is rendered, the ACP is disabled to allow the rendered image to remain on the screen and not allow the wireframe image to over-write the rendering and corrupt it. To enable the ACP, use the "CLEAR" function key. In "SPLIT-1," "SPLIT-2," and "SPLIT-3" mode, the ACP is enabled by the network as the rendered images are protected, and the wireframe model is displayed in the lower right-hand viewport.

{define a sectioning plane which can be rotated independently}

```
spattributes := attributes;

sect := begin_s
        sectioning_plane;
        trans := translate by 0,0,0;
        rot   := scale by 1;
        with attributes spattributes
        polygon -0.9,-0.9,0.0 -0.9,0.9,0.0 0.9,0.9,0.0 0.9,-0.9,0.0
        polygon 0.1,0.0,0.0 0.1,0.0,-0.3 0.15,0.0,-0.3 0.0,0.0,-0.45
                -0.15,0.0,-0.3 -0.1,0.0,-0.3 -0.1,0.0,0.0
        polygon 0.0,0.1,0.0 0.0,0.1,-0.3 0.0,0.15,-0.3 0.0,0.0,-0.45
                0.0,-0.15,-0.3 0.0,-0.1,-0.3 0.0,-0.1,0.0;
        end_s;
```

{define a light which can be rotated independently}

```
sunset := begin_structure
           persp := fov 90 front=2.2 back=3.6;
           look at 0,0,0 from 0,0,-3;
           set depth_clipping off;
           rot := scale by 1;
           vector 0,0,-.9 0,0,0;
           instance sun;
           translate 0,0,-.9;
           rational polynomial .2,0,8 -.2,-.2,-8 0,.1,4 chords=15;
           rational polynomial .2,0,-8 -.2,-.2,8 0,.1,-4 chords=15;
           vector separate n=15 -.1,0 -.05,0 .05,0 .1,0
                0,-.1 0,-.05 0,.05 0,.1
                -.0707,-.0707 -.0354,-.0354 .0354,.0354 .0707,.0707
                -.0707,.0707 -.0354,.0354 .0354,-.0354 .0707,-.0707;
           end_structure;
sun := illumination 0,0,-1;
```

{define a light which can be rotated with the object}

```
moonset :=begin_structure
           set depth_clipping off;
           rot := scale by 1;
           vector 0,0,-.9 0,0,0;
           instance moon;
           translate 0,0,-.9;
           rational polynomial .2,0,4 -.2,-.2,-4 0,.1,2 chords=15;
           rational polynomial .12,0,4 -.12,-.2,-4 0,.1,2 chords=15;
           end_structure;
moon := illumination 0,0,-1;
```

{set up a place to redisplay a saved hidden-line picture}

```
disphlview := matrix_4x4 1,0,0,0 0,1,0,0 0,0,0,0 0,0,1,1 then hlview;
```

{set up initial display structure}

```
universe := begin_s
   split := SET LEVEL TO 0;
   if level = 0 then full;
   if level = 1 then split;
end_s;
```

{ full screen dynamic window }

```
full := begin_s
      persp := fov 45 front=2.2 back=3.6;
      look := look at 0,0,0 from 0,0,-3;
      vport := viewport horizontal=-1:1 vertical=-1:1 intensity=0:.25;
      inst of world;
end_s;
```

{ 1/4 static viewport and 1/4 dynamic viewport }

```
split := begin_s
      persp := fov 45 front=2.2 back=3.6;
      look := look at 0,0,0 from 0,0,-3;
      vport := viewport horizontal=0:1 vertical=-1:0 intensity=0:.25;
      inst of world;
end_s;

world := begin_s
      bits := set bit 1 on;
      if bit 1 off then disphlview;
      if bit 1 on;
      set depth_clipping on;
      trans := translate by 0,0,0;
      rot   := scale by 1;
      if bit 2 on then sect;
      rendering := surface;
      if bit 3 on then sunset;
      if bit 4 on then moonset;
      inst of object;
end_s;

display universe;
```

{network to translate object}

```
a := f:addc;
connect a<1>:<1>world.trans;
connect a<1>:<2>a;
send v3d(0,0,0) to <2>a;
```

{network to rotate/scale object}

```
m := f:cmul;
connect m<1>:<1>world.rot;
connect m<1>:<1>m;
send m3d(1,0,0 0,1,0 0,0,1) to <1>m;
```

{network to translate sectioning plane}

```
a2 := f:addc;
connect a2<1>:<1>sect.trans;
connect a2<1>:<2>a2;
send v3d(0,0,0) to <2>a2;
```

{network to rotate/scale sectioning plane}

```
m2 := f:cmul;
connect m2<1>:<1>sect.rot;
connect m2<1>:<1>m2;
send m3d(1,0,0 0,1,0 0,0,1) to <1>m2;
```

{network to rotate sun}

```
msun := f:cmul;
connect msun<1>:<1>sunset.rot;
connect msun<1>:<1>msun;
send m3d(1,0,0 0,1,0 0,0,1) to <1>msun;
```

{network to rotate moon}

```
mmoon := f:cmul;
connect mmoon<1>:<1>moonset.rot;
connect mmoon<1>:<1>mmoon;
send m3d(1,0,0 0,1,0 0,0,1) to <1>mmoon;
```

```
{color network}
tripcolor := f:sync(5);
setup cness true <2>tripcolor;
setup cness true <3>tripcolor;
setup cness true <4>tripcolor;
setup cness true <5>tripcolor;
connect tripcolor<2>:<3>shadingenvironment;
connect tripcolor<3>:<7>shadingenvironment;
connect tripcolor<4>:<3>shadingenvironment;
connect tripcolor<5>:<2>shadingenvironment;
send v3d(0,440,39) to <2>tripcolor;
send false to <3>tripcolor;
send v3d(0,0,0) to <5>tripcolor;

suncolor := f:accumulate;
connect suncolor<1>:<2>sun;
connect suncolor<1>:<2>shadingenvironment;
connect suncolor<1>:<1>tripcolor;
send v3d(0,0,1) to <2>suncolor;
send 0 to <3>suncolor;
send v3d(20,.25,.25) to <4>suncolor;
send v3d(360,1,1) to <5>suncolor;
send v3d(0,0,0) to <6>suncolor;

mooncolor := f:accumulate;
connect mooncolor<1>:<2>moon;
connect mooncolor<1>:<2>shadingenvironment;
connect mooncolor<1>:<1>tripcolor;
send v3d(0,0,1) to <2>mooncolor;
send 0 to <3>mooncolor;
send v3d(20,.25,.25) to <4>mooncolor;
send v3d(360,1,1) to <5>mooncolor;
send v3d(0,0,0) to <6>mooncolor;

backgroundcolor := f:accumulate;
connect backgroundcolor<1>:<2>shadingenvironment;
connect backgroundcolor<1>:<1>tripcolor;
connect backgroundcolor<1>:<5>tripcolor;
send v3d(0,0,0) to <2>backgroundcolor;
send 0 to <3>backgroundcolor;
send v3d(20,.25,.25) to <4>backgroundcolor;
send v3d(360,1,1) to <5>backgroundcolor;
send v3d(0,0,0) to <6>backgroundcolor;

{mux the dials}
dialmux := f:croute(5);
connect dialmux<1>:<1>a;
connect dialmux<2>:<1>a2;
connect dialmux<3>:<1>suncolor;
connect dialmux<4>:<1>mooncolor;
connect dialmux<5>:<1>backgroundcolor;

dialmux2 := f:croute(5);
connect dialmux2<1>:<2>m;
connect dialmux2<2>:<2>m2;
connect dialmux2<3>:<2>msun;
connect dialmux2<4>:<2>mmoon;

{network to translate in x}
tx := f:xvec;
connect tx<1>:<2>dialmux;
connect dials<1>:<1>tx;

{network to translate in y}
ty := f:yvec;
connect ty<1>:<2>dialmux;
connect dials<2>:<1>ty;

{network to translate in z}
tz := f:zvec;
connect tz<1>:<2>dialmux;
connect dials<3>:<1>tz;

{network to scale}
s := f:scale;
connect s<1>:<2>dialmux2;
sa := f:addc;
connect sa<1>:<1>s;
send 1 to <2>sa;
connect dials<4>:<1>sa;

{network to rotate in x}
rx := f:xrotate;
connect rx<1>:<2>dialmux2;
sx := f:mulc;
connect sx<1>:<1>rx;
send 100 to <2>sx;
connect dials<5>:<1>sx;

{network to rotate in y}
ry := f:yrotate;
connect ry<1>:<2>dialmux2;
sy := f:mulc;
connect sy<1>:<1>ry;
send 100 to <2>sy;
connect dials<6>:<1>sy;
```

```
{network to rotate in z}
rz := f:zrotate;
connect rz<1>:<2>dialmux2;
sz := f:mulc;
connect sz<1>:<1>rz;
send -100 to <2>sz;
connect dials<7>:<1>sz;

{network to adjust back clipping plane}
backclip := f:fov;
connect backclip<1>:<1>split.persp;
connect backclip<1>:<1>full.persp;
connect backclip<1>:<1>sunset.persp;
setup cness false <4>backclip;
setup cness true <1>backclip;
send true to <1>backclip;
send 45 to <2>backclip;
send 2.2 to <3>backclip;

backclipaccum := f:accum;
connect backclipaccum<1>:<4>backclip;
connect dials<8>:<1>backclipaccum;
send 3.6 to <2>backclipaccum;
send 0 to <3>backclipaccum;
send 1 to <4>backclipaccum;
send 30 to <5>backclipaccum;
send 2.2 to <6>backclipaccum;

{network to reset transformations}
rs := f:sync(2);
connect rs<1>:<1>world.trans;
connect rs<1>:<2>a;
connect rs<2>:<1>world.rot;
connect rs<2>:<1>m;
connect rs<2>:<2>rs;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rs;

rs2 := f:sync(2);
connect rs2<1>:<1>sect.trans;
connect rs2<1>:<2>a2;
connect rs2<2>:<1>sect.rot;
connect rs2<2>:<1>m2;
connect rs2<2>:<2>rs2;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rs2;

rssun := f:constant;
connect rssun<1>:<1>msun;
connect rssun<1>:<1>sunset.rot;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rssun;

rsmoon := f:constant;
connect rsmoon<1>:<1>mmoon;
connect rsmoon<1>:<1>moonset.rot;
send m3d(1,0,0 0,1,0 0,0,1) to <2>rsmoon;

r := f:croute(5);
connect r<1>:<1>rs;
connect r<2>:<1>rs2;
connect r<3>:<1>rssun;
connect r<4>:<1>rsmoon;

{network to turn bits on and off}
bits := f:constant;
connect bits<1>:<5>world.bits;

{network to send to object or sectioning plane}
waylabel := f:inputs_choose(6);
connect waylabel<1>:<1>flabel12;
send 'OBJECT' to <1>waylabel;
send 'PLANE' to <2>waylabel;
send 'SUN' to <3>waylabel;
send 'MOON' to <4>waylabel;
send 'BACK' to <5>waylabel;

diallabel := f:sync(9);
connect diallabel<1>:<1>dlabel1;
connect diallabel<1>:<1>diallabel;
connect diallabel<2>:<1>dlabel2;
connect diallabel<2>:<2>diallabel;
connect diallabel<3>:<1>dlabel3;
connect diallabel<3>:<3>diallabel;
connect diallabel<4>:<1>dlabel4;
connect diallabel<4>:<4>diallabel;
connect diallabel<5>:<1>dlabel5;
connect diallabel<5>:<5>diallabel;
connect diallabel<6>:<1>dlabel6;
connect diallabel<6>:<6>diallabel;
connect diallabel<7>:<1>dlabel7;
connect diallabel<7>:<7>diallabel;
connect diallabel<8>:<1>dlabel8;
connect diallabel<8>:<8>diallabel;
send 'X-TRANS' to <1>diallabel;
send 'X-TRANS' to <1>diallabel;
send 'HUE' to <1>diallabel;
send 'HUE' to <1>diallabel;
send 'HUE' to <1>diallabel;
send 'Y-TRANS' to <2>diallabel;
send 'Y-TRANS' to <2>diallabel;
send 'SAT' to <2>diallabel;
send 'SAT' to <2>diallabel;
send 'SAT' to <2>diallabel;
send 'Z-TRANS' to <3>diallabel;
send 'Z-TRANS' to <3>diallabel;
send 'INT' to <3>diallabel;
send 'INT' to <3>diallabel;
send 'INT' to <3>diallabel;
send 'SCALE' to <4>diallabel;
send 'X-ROT' to <5>diallabel;
send 'Y-ROT' to <6>diallabel;
send 'Z-ROT' to <7>diallabel;
send 'BACKCLIP' to <8>diallabel;

way := f:sync(2);
connect way<2>:<1>dialmux;
connect way<2>:<1>dialmux2;
connect way<2>:<2>way;
connect way<2>:<1>r;
connect way<2>:<6>waylabel;
connect way<2>:<2>bits;
connect way<2>:<9>diallabel;
send fix(1) to <2>way;
send fix(2) to <2>way;
send fix(3) to <2>way;
send fix(4) to <2>way;
send fix(5) to <2>way;
send TRUE to <1>way;  {activate it}
```

{network to change from solid to surface}

```
sslabel := f:boolean_choose;
connect sslabel<1>:<1>flabel7;
send 'SOLID' to <2>sslabel;
send 'SURFACE'  to <3>sslabel;

issolid := f:nop;
connect issolid<1>:<2>world.rendering;
connect issolid<1>:<1>sslabel;
ss := f:sync(2);
connect ss<2>:<2>ss;
connect ss<2>:<1>issolid;
send TRUE to <2>ss;
send FALSE to <2>ss;
send FALSE to <1>ss; { initially a surface }
```

```
{network to toggle anti-alias possibilities}
aalabel := f:sync(2);
aavalue := f:sync(2);
connect aavalue<1>:<5>shadingenvironment;
send 'NO-AA ' to <1>aalabel;
send 'EDGEAA' to <1>aalabel;
send 'FULLAA' to <1>aalabel;
send fix(0) to <1>aavalue;
send fix(1) to <1>aavalue;
send fix(2) to <1>aavalue;
connect aavalue<1>:<1>aavalue;
connect aalabel<1>:<1>aalabel;
connect aalabel<1>:<1>flabel5;
send fix(0) to <2>aalabel;
send fix(0) to <2>aavalue;

{network to control rendering style}
stylab := f:sync(2);
styval := f:sync(2);
style := f:const;
send 'HIDELINE' to <1>stylab;
send 'WASH' to <1>stylab;
send 'FLAT' to <1>stylab;
send 'PHONG ' to <1>stylab;
send 'GOURAUD' to <1>stylab;
send 'XSECTION' to <1>stylab;
send 'SECTION' to <1>stylab;
send 'BACKFACE' to <1>stylab;
send 'SAVE-SEC' to <1>stylab;
send 'SAVE-HL' to <1>stylab;
send fix(4) to <1>styval;   { ###### New algorithm: Raster Hiddenline ###### }
send fix(5) to <1>styval;             { Send fix(4) to this input. }
send fix(6) to <1>styval;
send fix(7) to <1>styval;
send fix(8) to <1>styval;
send fix(1) to <1>styval;
send fix(2) to <1>styval;
send fix(3) to <1>styval;
send 'object' to <1>styval;
send 'hlview' to <1>styval;
conn stylab<1>:<1>stylab;
conn stylab<1>:<1>flabel3;
conn styval<1>:<1>styval;
conn styval<1>:<2>style;
conn style <1>:<1> world.rendering;
send fix(0) to <2>styval;
send fix(0) to <2>stylab;

{ some useful viewports }
piclab := f:sync(2);
picval := f:sync(2);
send 'SQUARE' to <1>piclab;
send 'SPLIT-1' to <1>piclab;
send 'SPLIT-2' to <1>piclab;
send 'SPLIT-3' to <1>piclab;
send 'BIG-PIC' to <1>piclab;
send '1-OF-2' to <1>piclab;
send '2-OF-2' to <1>piclab;
send '1-OF-6' to <1>piclab;
send '2-OF-6' to <1>piclab;
send '3-OF-6' to <1>piclab;
send '4-OF-6' to <1>piclab;
send '5-OF-6' to <1>piclab;
send '6-OF-6' to <1>piclab;
send v3d(82,0,863) to <1>picval; {#PS390 specific# - larger viewports}
send v3d(514,432,431) to <1>picval; {for upper right corner}
send v3d(0,432,431) to <1> picval; {for upper left corner}
send v3d(0,0,431) to <1> picval; {for lower left corner}
send v3d(0,-80,1023) to <1>picval; { big pic }
send v3d(0,170,511) to <1>picval;
send v3d(512,170,511) to <1>picval;
send v3d(0,432,341) to <1>picval;
send v3d(341,432,341) to <1>picval;
send v3d(681,432,341) to <1>picval;
send v3d(0,91,341) to <1>picval;
send v3d(341,91,341) to <1>picval;
send v3d(681,91,341) to <1>picval;

conn piclab<1>:<1>piclab;
conn piclab<1>:<1>flabel2;
conn picval<1>:<1>picval;
conn picval<1>:<3>shadingenvironment;
conn picval<1>:<4>tripcolor;

send 1 to <2>piclab;
send 1 to <2>picval;

{network to invert overlay-picture mechanism}
overinvert := f:xorc;
send true to <2>overinvert;
overlabel := f:sync(2);
connect overinvert<1>:<9>shadingenvironment;
connect overinvert<1>:<2>overlabel;
send 'REFRESH' to <1>overlabel;
send 'OVERLAY' to <1>overlabel;
connect overlabel<1>:<1>overlabel;
connect overlabel<1>:<1>flabel4;

{network to invert color interpolation}
colorinvert := f:xorc;
send true to <2>colorinvert;
colorlabel := f:sync(2);
connect colorinvert<1>:<10>shadingenvironment;
connect colorinvert<1>:<2>colorlabel;
send 'COL-OFF' to <1>colorlabel;
send 'COL-ON ' to <1>colorlabel;
connect colorlabel<1>:<1>colorlabel;
connect colorlabel<1>:<1>flabel8;

{ ###### New feature: Polygon edge enhancement ###### }
{network to invert polyedges (on/off) }
edgeinvert := f:xorc;
send true to <2>edgeinvert;
edgelabel := f:sync(2);
connect edgeinvert<1>:<15>shadingenvironment;
connect edgeinvert<1>:<2>edgelabel;
send 'EDGE-OFF' to <1>edgelabel;
send 'EDGE-ON ' to <1>edgelabel;
connect edgelabel<1>:<1>edgelabel;
connect edgelabel<1>:<1>flabel9;

{ ###### PS390 specific ###### -- TurnOnDisplay when clearing screen }
turnon := f:sync(2);
setup cness true <2>turnon;
connect turnon<1>:<7>shadingenvironment;
connect turnon<2>:<1>TurnOnDisplay;
send fix(0) to <2>turnon;

{ function keys }
fkmo := f:switch;
connect fkeys<1>:<1>fkmo;
connect fkmo<1>:<1>style;
connect fkmo<2>:<2>piclab; connect fkmo<2>:<2>picval;
connect fkmo<3>:<2>stylab; connect fkmo<3>:<2>styval;
connect fkmo<4>:<1>overinvert;
connect fkmo<5>:<2>aavalue;
connect fkmo<5>:<2>aalabel;
connect fkmo<6>:<1>turnon;
connect fkmo<7>:<1>ss;
connect fkmo<8>:<1>colorinvert;
connect fkmo<9>:<1>edgeinvert;
connect fkmo<10>:<2>r;
connect fkmo<10>:<1>original;
connect fkmo<11>:<1>bits;
connect fkmo<11>:<1>original;
connect fkmo<12>:<1>way;

fkm := f:inputs_choose(16);
connect fkm<1>:<2>fkmo;
connect fkeys<1>:<16>fkm;
send fix(1) to <1>fkm;
send fix(2) to <2>fkm;
send fix(3) to <3>fkm;
connect overinvert<1>:<4>fkm;
send fix(0) to <5>fkm;
send fix(2) to <6>fkm;              { ###### PS390 specific ###### }
send fix(7) to <7>fkm;
connect colorinvert<1>:<8>fkm;
connect edgeinvert<1>:<9>fkm;
send v3d(0,0,0) to <10>fkm;
send fix(11) to <11>fkm;
send fix(12) to <12>fkm;
send 'sflabel toggle' to <13> fkm;
send 'sflabel toggle' to <14> fkm;
send 'menu toggle' to <15> fkm;

send 'RENDER' to <1>flabel1;
send 'CLEAR' to <1>flabel6;
send 'RESET' to <1>flabel10;
send 'ON/OFF' to <1>flabel11;
send true to <1>overinvert;
send true to <1>colorinvert;
send true to <1>edgeinvert;

{ some useful colors }
blue    := attributes color 0;
magenta := attributes color 60;
red     := attributes color 120;
yellow  := attributes color 180;
green   := attributes color 240;
cyan    := attributes color 300;
white   := attributes color 0,0,1;

{some other names for shadingenvironment}
se := f:pass(15);
connect se<1>:<1>shadingenvironment;
connect se<2>:<2>shadingenvironment;
connect se<3>:<3>shadingenvironment;
connect se<4>:<4>shadingenvironment;
connect se<5>:<5>shadingenvironment;
connect se<6>:<6>shadingenvironment;
connect se<7>:<7>shadingenvironment;
connect se<8>:<8>shadingenvironment;
connect se<9>:<9>shadingenvironment;
connect se<10>:<10>shadingenvironment;
connect se<11>:<11>shadingenvironment;
connect se<12>:<12>shadingenvironment;
connect se<13>:<13>shadingenvironment;
connect se<14>:<14>shadingenvironment;
connect se<15>:<15>shadingenvironment;

ambient := f:pass(1);
connect ambient<1>:<1>shadingenvironment;
background := f:pass(1);
connect background<1>:<2>shadingenvironment;
rasterviewport := f:pass(1);
connect rasterviewport<1>:<3>shadingenvironment;
exposure := f:pass(1);
connect exposure<1>:<4>shadingenvironment;
quality := f:pass(1);
connect quality<1>:<5>shadingenvironment;
depth := f:pass(1);
connect depth<1>:<6>shadingenvironment;
screenwash := f:pass(1);
connect screenwash<1>:<7>shadingenvironment;
zoffset := f:pass(1);
connect zoffset<1>:<15>shadingenvironment;

{ Setup network for control of turn on display and split screen }
{ This network allows the user to specify 1 dynamic viewport (lower }
{ right-hand corner) and three static rendering viewports (1 - upper }
{ right-hand corner 2 - upper left-hand corner 3 - lower left-hand corner). }
splvec := f:parts;
splvalchk := f:eqc;
splsync := f:sync(2);
splroute := f:broutec;
splflop := f:constant;
splrouted := f:broutec;

conn picval<1>:<1>splvec;
conn splvec<3>:<1>splvalchk;
{ Turnondisplay }
conn world.rendering <1>:<1> splsync;
conn splvalchk <1>:<2> splsync;
conn splsync   <2>:<1> splrouted;
conn splrouted <1>:<1> turnondisplay;
```

```
send fix(0) to <2> splrouted;
setup cness true <2> splsync;

{ Split Selection }
conn splvalchk <1>:<1> splroute;
conn splroute <1>:<1> splflop;
conn splroute <2>:<1> universe.split;
conn splflop <1>:<1> universe.split;

send 431 to <2> splvalchk;
send fix(0) to <2> splroute;
send fix(1) to <2> splflop;
```

CHANGE PAGES TO THE COMMAND SUMMARY, THE FUNCTION SUMMARY, AND GRAPHICS SUPPORT ROUTINES

Version A2.V02

FORMAT

name := ATTRIBUTES **attributes** [AND **attributes**];

DESCRIPTION

Specifies the various characteristics of polygons used in the creation of shaded renderings. This command is only used with the PS 340 and PS 390. For a detailed explanation of defining and interacting with shaded images, consult the "Using the PS 340 - Rendering Operations For Surfaces and Solids" tutorial in Volume 2B.

PARAMETERS

attributes - The attributes of a polygon are defined as follows.

[COLOR h[,s[,i]]] [DIFFUSE d] [SPECULAR s] [OPAQUE t]

where

h - is a real number specifying the hue in degrees around the color wheel. Pure blue is 0 and 360, pure red is 120, and pure green is 240.

s - is a real number specifying saturation. No saturation (gray) is 0 and full saturation (full toned colors) is 1.

i - is a real number specifying intensity. No intensity (black) is 0, full intensity (white) is 1.

d - is a real number from 0 to 1 specifying the proportion of color contributed by diffuse reflection versus that contributed by specular reflection. Increasing d makes the surface more matte. Decreasing d makes it more shiny.

s - is an integer from 0 to 255 which adjusts the concentration of specular highlights. The more metallic an object is, the more concentrated the specular highlights.

t - is a real number from 0 to 1 specifying the transparency of the polygon, with 1 being fully opaque and 0 being fully transparent (invisible).

DEFAULTS

If no color is specified, the default is white (s = 0, i = 1). If saturation and intensity are not specified, they default to 1. If only hue and saturation are specified, intensity defaults to 1. If no diffuse attribute is given, d defaults to .75. If no specular attribute is given, s defaults to 4. If no opaque attribute is given, the default is 1 (fully opaque).

NOTES

1. Polygon attribute nodes are created in mass memory but are not part of a display tree. The attributes specified in an ATTRIBUTES command are assigned to polygons which include a WITH ATTRIBUTES clause. The attributes specified in a WITH ATTRIBUTES clause of a POLYGON command apply to all subsequent polygons until superseded by another WITH ATTRIBUTES clause. If no WITH ATTRIBUTES option is given for a POLYGON node, default attributes are assumed. The default attributes are 0,0,1 for COLOR, 0.75 for DIFFUSE, and 4 for SPECULAR.

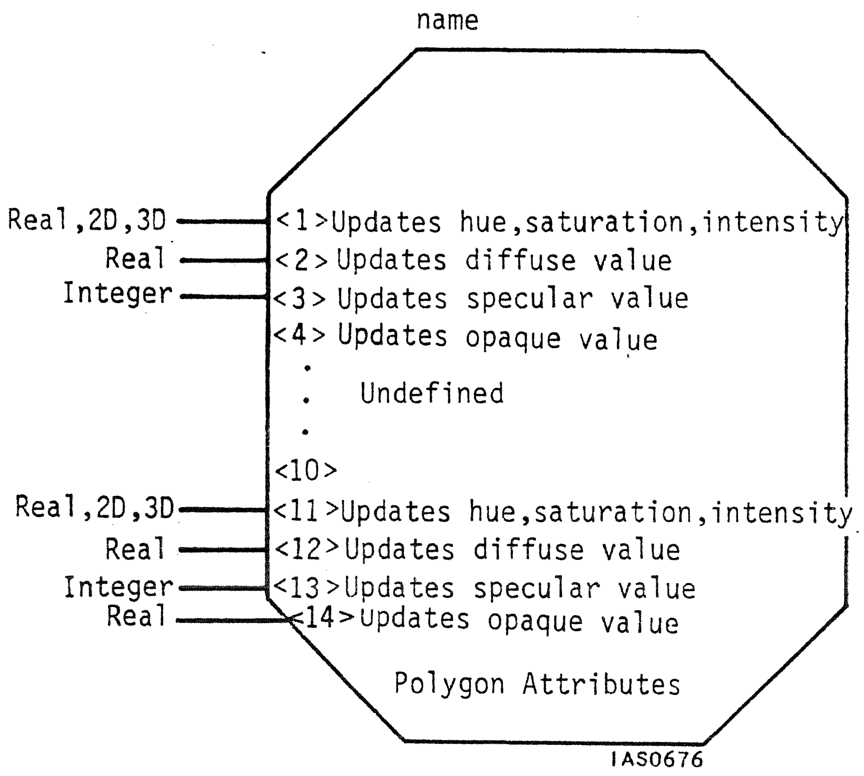
2. The various attributes may be changed from a function network via inputs to an attribute node, but the changes have no effect until a new rendering is created.

3. A second set of attributes may be given after the word AND in the ATTRIBUTES command. These attributes apply to the obverse side of the polygon(s) concerned. In other words, the two sides of an object may have different attributes. The attributes defined in the first attributes pertain to front-facing polygons. Those in the AND attributes clause pertain to backfacing polygons.

NODE CREATED

Polygon ATTRIBUTES definition node. This node resides in mass memory, but is not included in a display tree.

INPUTS FOR UPDATING NODE



NOTES ON INPUTS

1. Inputs <1> and <11> accept a real number as hue, a 2D vector as hue and saturation, and a 3D vector as hue, saturation and intensity.

2. Values sent to inputs <1>, <2>, and <3> specify the color and attributes for shading the front of the polygon(s) or for both sides if no obverse attributes are given. (Values sent to inputs <11>, <12>, and <13> specify the color and attributes for shading the obverse side of the polygon.

3. Inputs <4> and <14> accept a real number to update the opaque value of the polygon's attributes.

4. If anything other than a 3D vector is sent to input <1> or <11>, default values for the other variables are assumed.

FORMAT

```
name := [WITH ATTRIBUTES name1] [WITH OUTLINE h] [COPLANAR]
        POLYGon vertex ... vertex;
```

DESCRIPTION

Allows you to define primitives as solids and surfaces. This command is only used with the PS 340 and PS 390. For a detailed explanation of defining and interacting with polygons, consult the "Using the PS 340 - Rendering Operations For Surfaces and Solids" tutorial in Volume 2B.

PARAMETERS

WITH ATTRIBUTES - An option that assigns the attributes defined by **name1** for all polygons until superseded by another WITH ATTRIBUTES clause.

WITH OUTLINE - An option that specifies the color of the edges of a polygon on the color CSM display, or their intensity on a black and white display as a real number (**h**).

COPLANAR - Declares that the specified polygon and the one immediately preceding it have the same plane equation. This should only be used to define holes in the interior of a polygon.

vertex - A vertex is defined as follows:

```
[ S ] x,y,z [ N x,y,z ] [C h[,s[i]]]
```

where

S - indicates that the edge drawn between the previous vertex and this one represents a soft edge of the polygon. If the S specifier is used for the first vertex in a polygon definition, the edge connecting the last vertex with the first is soft. This option is only necessary to use if soft edges are desired in hidden-line pictures on the calligraphic display. This option has no effect on renderings displayed on the raster screen.

PARAMETERS (continued)

N – Indicates a normal to the surface with each vertex of the polygon. Normals are used only in smooth-shaded renderings. Normals must be specified for all vertices of a polygon or for none of them. If no normals are given for a polygon, they are defaulted to the same as the plane equation for the polygon.

x, y, z – are coordinates in a left-handed Cartesian system.

C – indicates a color to be assigned to the vertex. This color will be interpolated across the polygon to the other vertices. Color must be specified for all vertices of a polygon or for none of them (4K ACP only).

h, s, i – are coordinates of the Hue-Saturation-Intensity color system.

Polygons may be solidly colored by specifying a color through the attributes command or the colors may be assigned to the vertices by giving a color with each vertex specified. The color is specified by giving first the vertex and then the color (h, s, i). If just the hue and saturation are given, the intensity will default to 1. If just the hue value is given, the saturation and intensity will default to 1. If no vertex colors are given, the vertex colors will default to those specified in the ATTRIBUTE clause.

Vertex colors must be specified for all vertices of a polygon or for none of them. However, as with normals, some polygons may have color at their vertices while others polygons do not have color at their vertices. This means that it is possible to have some objects in the picture color interpolated, while others are not.

Although color of polygon vertices is specified h, s, i, the colors are linearly interpolated across the vertices in the Red-Green-Blue color system. If colors are not interpolating the way you would like them to, add more vertices to the polygon, or break up large solid volumes into smaller sub-volumes and assign the desired colors to the new vertices in the object.

You can specify color for a polygon with both the ATTRIBUTES command and the color by vertex specification. A new input to the SHADINGENVIRONMENT function allows you to switch between attribute-defined color and vertex-defined color. Input <10> of SHADINGENVIRONMENT accepts a Boolean to determine how color will be specified. To use vertex colors rather than surface attributes, send TRUE to input <10> of SHADINGENVIRONMENT. To return to using the attributes specified in the ATTRIBUTE command, send FALSE to input <10> of SHADINGENVIRONMENT.

NOTES

1. A polygon declared to be coplanar must lie in the same plane as the previous polygon if correct renderings are to be obtained. The system does not check for this condition. Coplanar polygons may be defined without the COPLANAR specifier, unless outer and inner contours are being associated.

2. To use the COPLANAR specifier to define a hole, the vertices of the hole must be ordered in a counter-clockwise direction, while the vertices of the surrounding polygon must be ordered in a clockwise direction.

3. All members of a set of consecutive coplanar polygons are taken to have the same plane equation, that of the previous polygon not containing the coplanar option. If COPLANAR is specified for the first polygon in a node, it has no effect.

4. If the N (normal) specifier is specified for a vertex in a polygon, it must be specified for all vertices in that polygon. The same is true for the C (color at vertex) specifier.

5. If the S (soft) specifier is used for the first vertex in a polygon definition, the edge connecting the last vertex with the first is soft.

6. No more than 250 vertices per POLYGon may be specified.

7. The last defined vertex in the polygon is assumed to connect to the first defined vertex; that is, polygons are implicitly closed.

8. There is no syntactical limit for the number of POLYGon clauses in a group.

9. The ordering of vertices within each POLYGon has important consequences for rendering operations.

DISPLAY TREE NODE CREATED

POLYGON data node.

INPUTS FOR UPDATING NODE

None.

FORMAT

name := SOLID_rendering APPLied to name1;

DESCRIPTION

Declares a POLYGon object to be a solid and marks the object so that rendering operations can be performed on it. This command creates a rendering node. It is used exclusively with the PS 340 and PS 390.

PARAMETERS

name1 - Either a POLYGon node or an ancestor of one or more POLYGon nodes.

NOTES

1. If non-POLYGon data nodes (VECtor/list, CHARacters, LABELS, POLYnomial, and BSPLINE) are included in name1, these data objects are displayed along with the polygon objects prior to rendering but are omitted from renderings. The rendering operations have no effect on these data nodes. However, special vector lists output from F:XFORMDATA used to display spheres and lines in the static viewport can be used and will be displayed if rendered.

2. IF and SET Conditional_BIT, IF and SET LEVel_of_detail, INCRement LEVel_of_detail, DECrement LEVel_of_detail, IF PHASE, SET RATE, SET RATE EXTernal, SET DEPTH_CLipping, and BEGIN_Structure... END_Structure may be placed between a rendering node and its data. A rendering takes into account any effects of these nodes at the time the request is made; for example, if IF PHASE and SET RATE are being used to blink an object and that object is "off" at the moment the request is made, the object is excluded from the rendering.

   The nodes in the above paragraph may also be placed above the rendering node.

3. The transformations ROTate, TRANslate, SCALE, Matrix_2X2, Matrix_3X3, Matrix_4X3, and LOOK may be placed between a rendering node and its data node(s). However, these nodes should be used with caution, since, like the operation nodes mentioned above, their effects will be incorporated into renderings, and precision problems may result.

NOTES (continued)

Since most vertices in an object usually belong to more than one polygon, each vertex must be defined with the same numerical value in each of its polygons; otherwise, precision discrepancies may cause inaccurate renderings. The transformation nodes mentioned above may also be placed above the rendering node.

4. The five nodes WINDOW, VIEWport, EYE, Field_Of_View, and Matrix_4X4 should not, in general, be made descendants of a rendering node. Like other transformations, these five are incorporated into the output data from a rendering operation. However, this rendered data is generally displayed within a framework that already includes global 4x4-matrix transformations of its own. Including these transformations as part of the rendering, then, usually has the net effect of applying an unwanted double-WINDOW (double-VIEWport, etc.) to the rendered object.

5. SOLID_rendering, SURFACE_rendering, and SECTioning_plane may not be descendants of a rendering node, especially if multiply-instanced rendering nodes are involved. If this rule is not observed, bad renderings or a system crash may result. The system does not check for this condition.

6. Other nodes, including character transformations and the SET nodes (SET RATE, SET COLOR, SET PLOTTER) not mentioned above, are ignored by rendering operations when placed beneath a rendering node. Such nodes must therefore be placed above a rendering node if they are to have their customary effects on renderings. Data nodes other than POLYGon are also ignored.

7. Before an object can be rendered, its rendering node must be part of a structure which is DISPlayed. If the object itself is DISPlayed but its rendering node is not, no renderings can be created.

8. Any input to input<1> of a rendering node causes an output. Inputs sent to input<2> will not cause an output to be sent. If output<1> has not been connected, and an integer, string, or Boolean is sent to input<1>, a message will appear on the screen upon successful completion of the rendering operation. An error message will appear if the rendering was not completed.

NOTES (continued)

9. Input <3> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets the vectors as "moves" and "draws" for raster-line rendering.

10. Input <4> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets each vector as an x,y,z spherical

11. Input <5> of the rendering node accepts the name of the original vector list (sent to F:XFORMDATA with its output <1> sent to input <4> of the rendering node) to enable accurate scaling for rendering raster lines and spheres.

12. Toggling between the current rendering and the original object (sending a fix(0) to input <1> of the SOLID_rendering or SURFACE_rendering node) works only after requesting hidden-line pictures, backface pictures, sectioned pictures, or cross-sectioned pictures. Raster images may not be toggled.

13. Sending a fix(7) to input <1> of the SOLID_rendering or SURFACE_rendering node produces a type of Phong shading. Phong shading is made by interpolating the surface normal between vertices of the polygon and then calculating the correct lighting at each pixel. This is the highest quality of smooth shading currently supported.

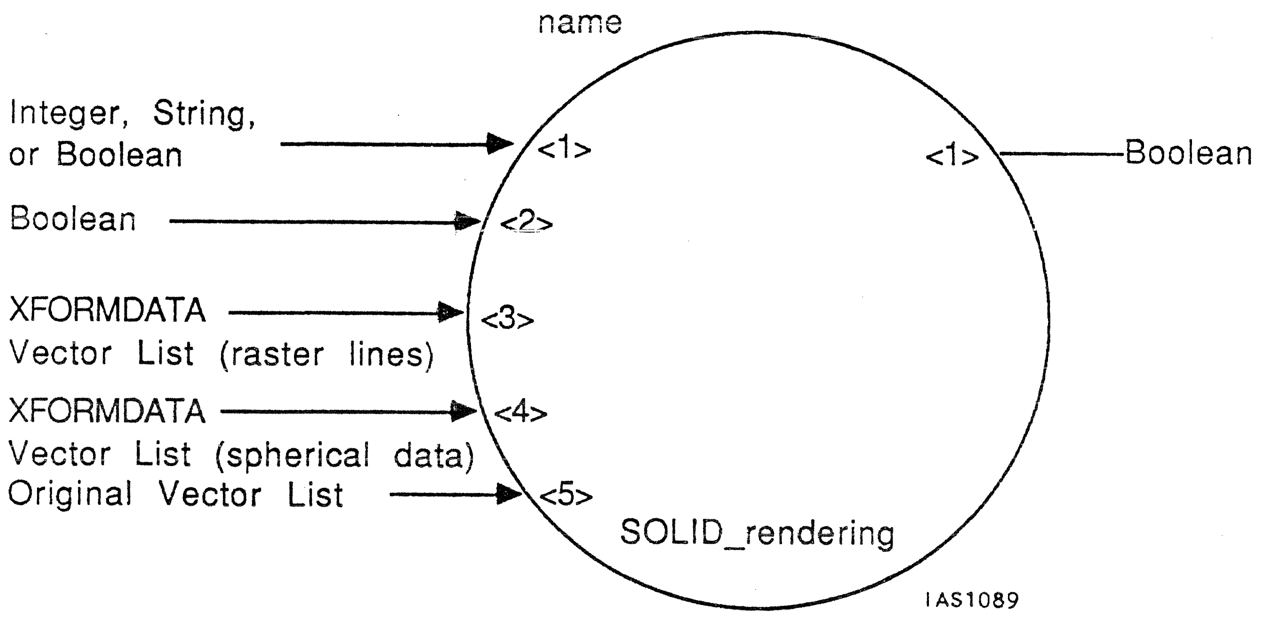
14. Sending a fix(8) to input <1> of the SOLID_rendering or SURFACE_rendering node will produce a type of Gouraud shading. Gouraud shading is made by calculating the correct lighting at the vertices of the polygon only and interpolating the intensity across the polygon to produce a smooth-shaded picture. An image produced with Gouraud shading will not be the quality of an image produced with Phong shading, but the Gouraud-shaded image will be produced at a faster rate. The user must supply normals at each of the polygons for the object to be smooth-shaded.

15. Sending data a non-existent rendering node input, will cause the system to crash.

DISPLAY TREE NODE CREATED

Rendering operation node.

INPUTS FOR UPDATING NODE


name
Integer, String, or Boolean
<1>
Boolean
<2>
XFORMDATA Vector List (raster lines)
<3>
XFORMDATA Vector List (spherical data)
<4>
Original Vector List
<5>
<1>
Boolean
SOLID_rendering
IAS1089

NOTES ON INPUTS

Input <1>

0: Toggles between the current rendering and the original object in the dynamic viewport.
1: Creates and displays a cross-section of an object defined by the sectioning plane (solid only) in the dynamic viewport.
2: Creates and displays a sectioned rendering in the dynamic viewport.
3: Creates and displays a rendering using backface removal (solid only) in the dynamic viewport.
4: Creates and displays a rendering using hidden-line removal in the static viewport.
5: Generates a wash-shaded image in the static viewport.
6: Generates a flat-shaded image in the static viewport.
7: Generates a Phong-shaded image in the static viewport.
8: Generates a Gouraud-shaded image in the static viewport.

NOTES ON INPUTS (continued)

String:	Causes the current rendering to be saved under the name given in the string (dynamic viewport only).
False:	Sets the original view. The original descendent structure of the rendering operate node is displayed.
True:	Sets the rendered view. The rendered view of the original descendent structure of the rendering operate node.

Input <2>

True:	Declares the object to be a solid.
False:	Declares the object to be a surface.

Input <3>
Accepts a transformed vector list from output <1> of F:XFORMDATA to define raster lines.

Input <4>
Accepts a transformed vector list from output <1> of F:XFORMDATA to define spherical centers.

Input <5>
Accepts the original vector list to enable accurate spherical scaling.

Output <1>

True:	Rendering is displayed
False:	Rendering is not displayed

Version A2.V02

FORMAT

name := SURFACE_rendering APPLied to name1;

DESCRIPTION

Declares a POLYGon object to be a surface and marks the object so that rendering operations can be performed on it. This command creates a rendering node. It is used exclusively with the PS 340 and PS 390.

PARAMETERS

name1 - Either a POLYGon node or an ancestor of one or more POLYGon nodes.

NOTES

1. If non-POLYGon data nodes (such as VECtor_list, CHARacters, LABELS, POLYnomial, and BSPLINE) are included in name1, these data objects are displayed along with the polygon objects prior to rendering but are omitted from renderings. The rendering operations have no effect on these data nodes. However, special vector lists output from F:XFORMDATA used to display spheres and lines in the static viewport can be used and will be displayed if rendered.

2. IF and SET conditional_BIT, IF and SET LEVel_of_detail, INCRement LEVel_of_detail, DECrement LEVel_of_detail, IF PHASE, SET RATE, SET RATE EXTernal, SET DEPTH_CLipping, and BEGIN_Structure... END_Structure may be placed between a rendering node and its data. A rendering takes into account any effects of these nodes at the time the request is made; for example, if IF PHASE and SET RATE are being used to blink an object and that object is "off" at the moment the request is made, the object is excluded from the rendering.

   The nodes in the above paragraph may also be placed above the rendering node.

3. The transformations ROTate, TRANslate, SCALE, Matrix_2X2, Matrix_3X3, Matrix_4X3, and LOOK may be placed between a rendering node and its data node(s). However, these nodes should be used with caution, since, like the operation nodes mentioned above, their effects will be incorporated into renderings, and precision problems may result.

NOTES (continued)

Since most vertices in an object usually belong to more than one polygon, each vertex must be defined with the same numerical value in each of its polygons; otherwise, precision discrepancies may cause inaccurate renderings. The transformation nodes mentioned above may also be placed above the rendering node.

4. The five nodes WINDOW, VIEWport, EYE, Field_Of_View, and Matrix_4X4 should not, in general, be made descendants of a rendering node. Like other transformations, these five are incorporated into the output data from a rendering operation. However, this rendered data is generally displayed within a framework that already includes global 4x4-matrix transformations of its own. Including these transformations as part of the rendering, then, usually has the net effect of applying an unwanted double-WINDOW (double-VIEWport, etc.) to the rendered object.

5. SOLID_rendering, SURFACE_rendering, and SECTioning_plane may not be descendants of a rendering node, especially if multiply-instanced rendering nodes are involved. If this rule is not observed, bad renderings or a system crash may result. The system does not check for this condition.

6. Other nodes, including character transformations and the SET nodes (SET RATE, SET COLOR, SET PLOTTER) not mentioned above, are ignored by rendering operations when placed beneath a rendering node. Such nodes must therefore be placed above a rendering node if they are to have their customary effects on renderings. Data nodes other than POLYGon are also ignored.

7. Before an object can be rendered, its rendering node must be part of a structure which is DISPlayed. If the object itself is DISPlayed but its rendering node is not, no renderings can be created.

8. Any input to input<1> of a rendering node causes an output. Inputs sent to input<2> will not cause an output to be sent. If output<1> has not been connected, and an integer, string, or Boolean is sent to input<1>, a message will appear on the screen upon successful completion of the rendering operation. An error message will appear if the rendering was not completed.

NOTES (continued)

9. Input <3> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets the vectors as "moves" and "draws" for raster-line rendering.

10. Input <4> of the rendering node accepts a transformed vector list (from output <1> of F:XFORMDATA) and interprets each vector as an x,y,z spherical

11. Input <5> of the rendering node accepts the name of the original vector list (sent to F:XFORMDATA with its output <1> sent to input <4> of the rendering node) to enable accurate scaling for rendering lines and spheres.

12. Toggling between the current rendering and the original object (sending a fix(0) to input <1> of the SOLID_rendering or SURFACE_rendering node) works only after requesting hidden-line pictures, backface pictures, sectioned pictures, or cross-sectioned pictures. Raster images may not be toggled.

13. Sending a fix(7) to input <1> of the SOLID_rendering or SURFACE_rendering node produces a type of Phong shading. Phong shading is made by interpolating the surface normal between vertices of the polygon and then calculating the correct lighting at each pixel. This is the highest quality of smooth shading currently supported.

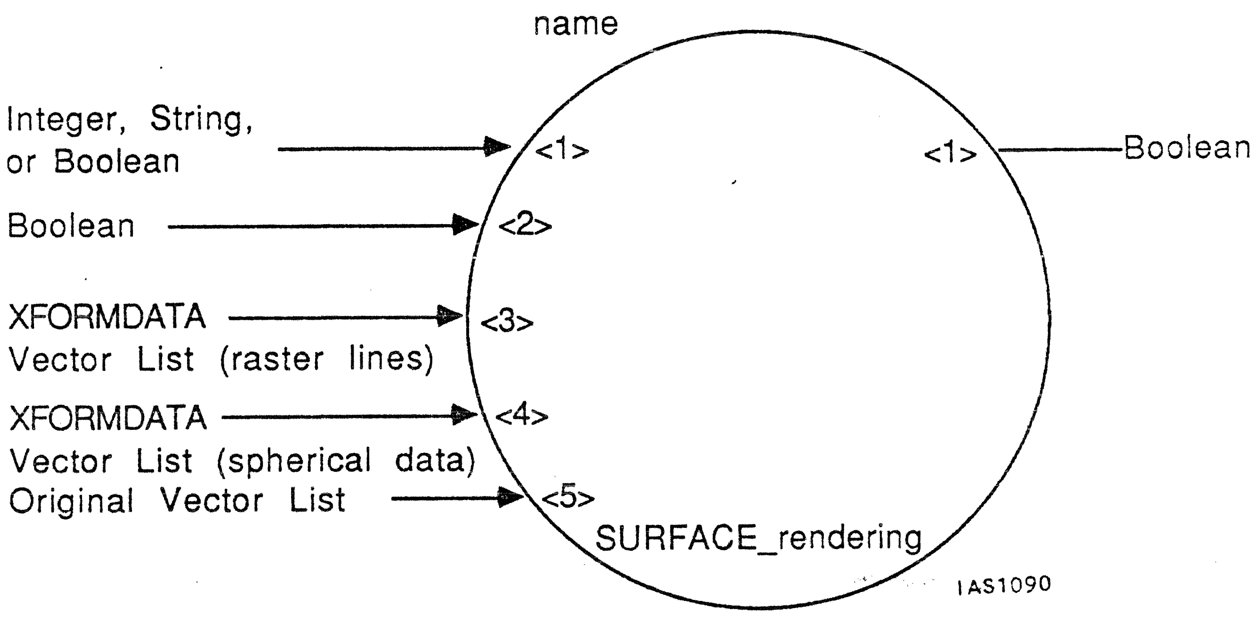
14. Sending a fix(8) to input <1> of the SOLID_rendering or SURFACE_rendering node will produce a type of Gouraud shading. Gouraud shading is made by calculating the correct lighting at the vertices of the polygon only and interpolating the intensity across the polygon to produce a smooth-shaded picture. An image produced with Gouraud shading will not be the quality of an image produced with Phong shading, but the Gouraud-shaded image will be produced at a faster rate. The user must supply normals at each of the polygons for the object to be smooth-shaded.

15. Sending data a non-existent rendering node input, will cause the system to crash.

DISPLAY TREE NODE CREATED

Rendering operation node.

INPUTS FOR UPDATING NODE


name
Integer, String, or Boolean
<1>
Boolean
<2>
XFORMDATA Vector List (raster lines)
<3>
XFORMDATA Vector List (spherical data)
<4>
Original Vector List
<5>
<1>
Boolean
SURFACE_rendering
IAS1090

NOTES ON INPUTS

Input <1>

0: Toggles between the current rendering and the original object in the dynamic viewport.
1: Creates and displays a cross-section of an object defined by the sectioning plane (solid only) in the dynamic viewport.
2: Creates and displays a sectioned rendering in the dynamic viewport.
3: Creates and displays a rendering using backface removal (solid only) in the dynamic viewport.
4: Creates and displays a rendering using hidden-line removal in the static viewport.
5: Generates a wash-shaded image in the static viewport.
6: Generates a flat-shaded image in the static viewport.
7: Generates a Phong-shaded image in the static viewport.
8: Generates a Gouraud-shaded image in the static viewport.

NOTES ON INPUTS (continued)

String: Causes the current rendering to be saved under the name given in the string (dynamic viewport only).

False: Sets the original view. The original descendent structure of the rendering operation node is displayed.
True: Sets the rendered view. The rendered view of the original descendent structure of the rendering operation node is displayed.

Input <2>
True: Declares the object to be a solid.
False: Declares the object to be a surface.

Input <3>
Accepts a transformed vector list from output <1> of F:XFORMDATA to define raster lines.

Input <4>
Accepts a transformed vector list from output <1> of F:XFORMDATA to define a spherical center.

Input <5>
Accepts the original vector list to enable accurate spherical scaling.

Output <1>
True: Rendering is displayed
False: Rendering is not displayed

Version A2.V02

```
                              SHADINGENVIRONMENT
                             _____________________
                            |                     |
      R, 2D, 3D ------->    | <1>             <1> |--------> PS 340 or PS 390
                            |                     |          Raster Display
      R, 2D, 3D ------->    | <2>                 |
                            |                     |
      3D -------------->    | <3>                 |
                            |                     |
      R --------------->    | <4>                 |
                            |                     |
      I --------------->    | <5>                 |
                            |                     |
      R --------------->    | <6>                 |
                            |                     |
      B, I ------------>    | <7>                 |
                            |                     |
   Reserved ----------->    | <8>                 |
                            |                     |
      B --------------->    | <9>                 |
                            |                     |
      B --------------->    | <10>                |
                            |                     |
      B --------------->    | <11>                |
                            |                     |
      B --------------->    | <12>                |
                            |                     |
      B --------------->    | <13>                |
                            |                     |
  string  (name of ----->   | <14>                |
   3D tabulated vec list)   |                     |
     B, R -------------->   | <15>                |
                            |         D C         |
                            |_____________________|
```

PURPOSE

For use with the PS 340 or PS 390 system, this function allows you to control various non-dynamic factors of shaded renderings displayed in the raster viewport.

DESCRIPTION

INPUT
<1> – ambient color
<2> – background color
<3> – raster viewport
<4> – exposure
<5> – edge-smoothing (anti-aliasing) control
<6> – depth cueing
<7> – screen wash
<8> – Reserved
<9> – overlay/refresh control
<10> – color by vertex control
<11> – opaque (transparency) control
<12> – specular highlights control
<13> – special color blending for spheres control
<14> – update attribute table
<15> – line z-value control

OUTPUT
<1> – connected to the PS 340 or PS 390 Raster Display

NOTES

1. Ambient color: input <1> accepts a real number as hue, a 2D vector as hue and saturation, and a 3D vector as hue, saturation, and intensity, to specify the ambient color. The ambient color is combined with the result obtained from the light sources to determine the color of ambient light. The default ambient color is white, with a default intensity of .25. The ambient color is analogous to the color reflected off a wall.

2. Background color: input <2> accepts a real number as hue, a 2D vector as hue and saturation, and or a 3D vector as hue, saturation, and intensity to specify the background color. The raster screen will be colored with the background color prior to any shaded rendering done in the refresh mode (refer to input <9>). The default background color is black (0,0,0).

3. Raster (static) viewport: input <3> accepts a 3D vector which specifies physical pixel locations for the viewport on the raster image buffer where shaded renderings will be displayed. Raster viewports are always square, the lower left corner being given by the X and Y coordinates of the vector, and its size given by the Z coordinate, such that the upper right corner is at (x+z,y+z). Values are rounded to the nearest pixel. The default viewport is V3D(80,0,863).

NOTES

The viewport can be used for rendering multiple images side by side on the raster display. For example, send V3D(0,-80,1023) would be a valid command to specify the largest recommended value for the static viewport. This viewport encompasses the entire displayable screen as well as the undisplayable area in Y that is in excess of 863. Images in this viewport are clipped to the physical raster for which $0 \leq X < 1024$ and $0 \leq Y < 864$.

4. Exposure: input <4> accepts a real number as the exposure, controlling the overall brightness of the picture. The exposure is like that on a camera. If a picture is taken of an object with a very bright specular highlight, it may be so bright that the rest of the object is darkened. If three light sources exist, the object would be about three times brighter, making the object too bright. The exposure should be brought down to control this.

   The exposure is multiplied by the intensity at each pixel and the result clipped to the maximum intensity. This enables the overall brightness of a rendering to be increased without causing bright spots to exceed maximum intensity (instead forming "plateaus" of maximum intensity). Note that this may cause changes in color on a plateau, where color has reached its maximum, but the others have not. Exposure values may vary between .3 and 3, values outside that range being changed to .3 or 3. The default exposure is 1.

5. Edge smoothing (anti-aliasing) control: input <5> accepts an integer which will allows users to choose between having a relatively fast rendering with jagged edges along the edges of polygons and having a slower rendering with smoother edges. Anti-aliasing is accomplished by taking 16 samples per pixel instead of just one. You are given the choice of having no edge smoothing at all, smoothing along the edges only, or sampling 16 times within every pixel for every polygon.

   Sending fix(0) to this input produces no smooth edges, and is fastest. The polygons are rendered with one sample per pixel.

   Sending fix(1) produces smooth edges, but may not correctly resolve obscurity between surfaces that are extremely close in z-values or that are interpenetrating. The 16 samples are taken only where the edges of the polygon touch a pixel. The interior of the polygons are still rendered with one sample per pixel. This has a speed intermediate between a fix(0) and a fix(2).

   Sending fix(2) to input <5> of SHADINGENVIRONMENT produces edge-smoothing, and is slower, but resolves surfaces. The 16 samples are taken in every pixel of every polygon. The default is 0.

NOTES (continued)

6. Depth cueing: input <6> accepts a real number in the range of 0 to 1 to control the effect of depth cueing in the shaded image (1 specifying no depth cueing and 0 specifying maximum depth cueing). As perceived depth from the viewer increases, the colors are mixed with the ambient light color (input <1> of SHADINGENVIRONMENT). Thus, if a 3D vector(0,0,0) (black) is sent to the ambient input <1> and if a 0 is sent to input <6>, the objects will be rendered with a ramp ending in black at the back clipping plane. A 1 sent to input <6> will turn off depth cueing. The default is 0.2 giving a fairly large depth cueing effect.

7. Screen wash: input <7> accepts a Boolean or an integer, and is the only input to cause a visual effect immediately. Sending a True to input <7> clears the entire screen to static and causes a screen wash with the current static background color. Sending False to input <7> clears the currently specified static viewport and causes the viewport to be filled with the current static background color.

   Sending fix(0) to input <7> clears the entire screen to static and causes a screen wash with the current static background color (This has the same effect as sending a True to this input.)

   Sending fix(1) to input <7> clears the currently specified static viewport and causes the viewport to be filled with the current static background color. (This has the same effect as sending a False to this input.) If you are requesting another rendering to be displayed in the same static viewport you just used, you do not need to clear that viewport first, since this is accomplished by the request for a new rendering.

   Sending fix(2) to input <7> clears the entire screen to dynamic and causes a screen wash with the current dynamic background color. You must do this to clear either a shaded image or a dynamic image (from the entire screen or viewport), before displaying a new dynamic image.

   Sending fix(3) to input <7>, clears the currently specified dynamic viewport with the current dynamic background color.

8. Reserved

NOTES (continued)

9. Clear/Overlay Control: input <9> accepts a Boolean which determines whether the screen is to be cleared with the current background color before the rendering is done. Sending a TRUE to this input causes the current object to be rendered on top of the image currently being displayed on the raster screen. Sending a FALSE to this input causes the screen to be washed clean with the current background color. In the PS 340 the object which overlays the image is anti-aliased with the background color so the object is correctly composited into the image with no jagged edges. In the PS 390, the object will have jagged edges because the readback of pixel data is not currently supported. The default is false.

10. Color by Vertex Control: input <10> accepts a Boolean which turns off (or on) the use of vertex colors. Color by vertex is accomplished by defining a color for each vertex in the polygon:

    [S] x,y,z [N x,y,z] [C h [,s [,i]]]

    Refer to the notes on the POLYGon command for more information about how to add color by vertex. To use the vertex colors defined this way rather than the color defined in the ATTRIBUTES, send TRUE to this input. Send FALSE to this input to return to using the color specified by the ATTRIBUTES command. The default is false.

11. Opaque (Transparency) Control: input <11> accepts a Boolean which allows you to turn off (or on) the transparency assigned to the polygon with the OPAQUE clause of the attribute command. Transparent polygons are created by modifying the ATTRIBUTE command in the following manner.

```
Name := ATTRIBUTE  [Color h [,s [,i] ] ] [OPAQUE t]
                   [Diffuse d] [Specular s];
```

    where **t** refers to a value between 0 and 1, with 1 being a fully opaque polygon and 0 being a fully transparent polygon. The default is false.

    As **t** decreases from 1 to 0, more of the color of the obscured object(s) will show through. At **t**=0, the transparent polygon becomes completely invisible. If no opaque attribute is specified, the default is 1 (fully opaque).

NOTES (continued)

12. Specular Highlight Control: input <12> accepts a Boolean which allows you to turn on (or off) specular highlights. Flat, Gouraud, and Phong shading all use the same shading equation. This means that multiple light sources are processed in each case and that specular highlights are calculated. Specular highlights may appear strange in Gouraud or flat shading. In Gouraud shading, the highlights may cause bright horizontal bands to appear inside the polygons. In flat shading, some polygons will appear very bright when viewed from certain angles unless specular highlights are turned off. The default is true.

13. Special Color Blending for Spheres: input <13> accepts a Boolean which turns off (or on) the color blending used for correct spherical rendering. Sending a TRUE to input <13> turns ON this special color blending. Sending a FALSE to input <13> turns OFF special color blending. The default is false.

14. Update Attribute Table: input <14> accepts a string which is the name of a 3D tabulated vector list to update the attribute table that specifies color, radii, diffuse, and specular highlights for spheres and lines. The attribute table has 0 to 127 entries with six table components for each entry. The attribute table can be updated by encoding the table entries into a PS 300 vector list and then sending the name of the vector list to input <14>. The six table components are encoded into two consecutive 3D tabulated vectors of the vector list. Hue, saturation, and intensity are encoded into the first x,y,z respectively. Radius, diffuse, and specular are encoded into the second x,y,z respectively. The table index is encoded into the t field of the second vector.

The table is initialized as follows:

INDEX	Hue	Sat	Intensity	Radius	Diffuse	Specular
0	0	0	0.5	1.8	0.7	4 (Grey)
1	0	0	1	1.2	0.7	4 (White)
2	120	1	1	1.35	0.7	4 (Red)
3	240	1	1	1.8	0.7	4 (Green)
4	0	1	1	1.8	0.7	4 (Blue)
5	180	1	1	1.7	0.7	4 (Yellow)
6	0	0	0.7	1.8	0.7	4 (Grey)
7	300	1	1	2.15	0.7	4 (Cyan)
8	60	1	1	1.8	0.7	4 (Magenta)
9	0	0	0	1.8	0.7	4 (Black)
10-127						Color Wheel

NOTES (continued)

15. Raster Lines Z-value Control: input <15> accepts a Boolean or a real number in the range of 0-1 which is added to the z-values of lines in raster renderings. Sending a 0 to this input will leave lines in their original z position. Sending a 1 to this input will force lines to be in front of everything else in the image. This feature may be desirable when rendering lines exactly along polygon edges. Leaving lines at their original z values will cause obscurity problems with the edges of the the polygons. By adding an offset to the lines' z-values, this obscurity problem is resolved more easily. The default is 0.05.

    If a Boolean is sent to this input, polygon edges are toggled on and off. TRUE causes lines to be drawn along polygon borders and temporarily turns on full antialiasing.

Name := ATTRIBUTES

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PATTR (Name, Hue, Sat, Intens, Opaque, Diffus, Specul, ErrHnd)
```

where

Name is a CHARACTER STRING
Hue is a REAL
Sat is a REAL
Intens is a REAL
Opaque is a REAL
Diffus is a REAL
Specul is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. Hue, Sat and Intens define the color of the polygon. Hue specifies an angle between 0 and 360 indicating the color on a color wheel with pure blue being 0, red being 120 and green being 240. Sat specifies the saturation of the color with 0 being no color and 1 being full saturation. Intens specifies the intensity of the color with 0 being no color (black) and 1 being full intensity. Diffus is the proportion of color contributed by diffuse reflection versus that contributed by specular reflection with a value of 1 eliminating all specular highlighting and a value of 0 eliminating all diffuse reflectivity. Specul adjusts the concentration of specular highlights in the range of 0 to 10. Opaque specifies how transparent the polygon is with 1 being fully opaque and 0 being fully transparent.

PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

Name := ATTRIBUTES ... AND

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PATTR2 (Name, Hue, Sat, Intens, Opaqul, Diffus, Specul,
             Hue2, Sat2, Inten2, Opaqu2, Diffu2, Specul2, ErrHnd)
```

where

Name is a CHARACTER STRING
Hue is a REAL
Sat is a REAL
Intens is a REAL
Opaqul is a REAL
Diffus is a REAL
Specul is an INTEGER*4
Hue2 is a REAL
Sat2 is a REAL
Inten2 is a REAL
Opaqu2 is a REAL
Diffu2 is a REAL
Specul2 is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. This is similar to the PATTR subroutine but allows for a second set of attributes to be defined for the backside of polygons.

PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t]
        AND        [COLOR h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

Name := ATTRIBUTES ... AND

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PATTR2 (Name, Hue, Sat, Intens, Opaqul, Diffus, Specul,
             Hue2, Sat2, Inten2, Opaqu2, Diffu2, Specul2, ErrHnd)
```

where

Name is a CHARACTER STRING
Hue is a REAL
Sat is a REAL
Intens is a REAL
Opaqul is a REAL
Diffus is a REAL
Specul is an INTEGER*4
Hue2 is a REAL
Sat2 is a REAL
Inten2 is a REAL
Opaqu2 is a REAL
Diffu2 is a REAL
Specul2 is an INTEGER*4
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. This is similar to the PATTR subroutine but allows for a second set of attributes to be defined for the backside of polygons.

PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t]
        AND        [COLOR h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PATTRIB ( %DESCR Name       : P_Varying_Type;
                           Hue        : REAL;
                           Saturation : REAL;
                           Intensity  : REAL;
                           Opaque     : REAL;
                           Diffused   : REAL;          {default .75}
                           Specular   : REAL;          {default 4}
                  Procedure Error_Handler (Err : INTEGER));;
```

DEFINITION

This procedure defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. Hue, Saturation, and Intensity define the color of the polygon. Hue specifies an angle between 0 and 360 indicating the color on a color wheel with pure blue being 0, red being 120 and green being 240. Saturation specifies the saturation of the color with 0 being no color and 1 being full saturation. Intensity specifies the intensity of the color with 0 being no color (black) and 1 being full intensity. Diffused is the proportion of color contributed by diffuse reflection versus that contributed by specular reflection with a value of 1 eliminating all specular highlighting and a value of 0 eliminating all diffuse reflectivity. Specular adjusts the concentration of specular highlights in the range of 0 to 10. Opaque specifies how transparent the polygon is with 1 being fully opaque and 0 being fully transparent.

PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PATTRIB ( %DESCR Name       : P_Varying_Type;
                           Hue        : REAL;
                           Saturation : REAL;
                           Intensity  : REAL;
                           Opaque     : REAL;
                           Diffused   : REAL;          {default .75}
                           Specular   : REAL;          {default 4}
                    Procedure Error_Handler (Err : INTEGER));;
```

DEFINITION

This procedure defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. Hue, Saturation, and Intensity define the color of the polygon. Hue specifies an angle between 0 and 360 indicating the color on a color wheel with pure blue being 0, red being 120 and green being 240. Saturation specifies the saturation of the color with 0 being no color and 1 being full saturation. Intensity specifies the intensity of the color with 0 being no color (black) and 1 being full intensity. Diffused is the proportion of color contributed by diffuse reflection versus that contributed by specular reflection with a value of 1 eliminating all specular highlighting and a value of 0 eliminating all diffuse reflectivity. Specular adjusts the concentration of specular highlights in the range of 0 to 10. Opaque specifies how transparent the polygon is with 1 being fully opaque and 0 being fully transparent.

PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

Name := ATTRIBUTES ... AND

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
     PROCEDURE PATTRIB2 ( %DESCR Name         : P_Varying_Type;
                                 Hue1         : REAL;
                                 Saturation1  : REAL;
                                 Intensity1   : REAL;
                                 Opaque1      : REAL;
                                 Diffused1    : REAL;         {default .75}
                                 Specular1    : REAL;         {default 4}
                                 Hue2         : REAL;
                                 Saturation2  : REAL;
                                 Intensity2   : REAL;
                                 Opaque2      : REAL;
                                 Diffused2    : REAL;         {default .75}
                                 Specular2    : REAL;         {default 4}
                      Procedure Error_Handler (Err : INTEGER));;
```

DEFINITION

This procedure defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. This is similar to the PATTRIB procedure but allows for a second set of attributes to be defined for the back side of polygons.

PS 300 COMMAND AND SYNTAX

```
Name := ATTRIBUTES [COLOR h[,s[i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t]
        AND        [COLOR h[,s[,i]]]
                   [DIFFUSE d]
                   [SPECULAR s]
                   [OPAQUE t];
```

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
     PROCEDURE PATTRIB2 ( %DESCR Name         : P_Varying Type;
                                 Hue1         : REAL;
                                 Saturation1  : REAL;
                                 Intensity1   : REAL;
                                 Opaque1      : REAL;
                                 Diffused1    : REAL;          {default .75}
                                 Specular1    : REAL;          {default 4}
                                 Hue2         : REAL;
                                 Saturation2  : REAL;
                                 Intensity2   : REAL;
                                 Opaque2      : REAL;
                                 Diffused2    : REAL;          {default .75}
                                 Specular2    : REAL;          {default 4}
                         Procedure Error_Handler (Err : INTEGER));;
```

DEFINITION

This procedure defines polygon characteristics used by the rendering firmware in the PS 340 to produce shaded renderings. This is similar to the PATTRIB procedure but allows for a second set of attributes to be defined for the back side of polygons.

PS 300 COMMAND AND SYNTAX

```
     Name  := ATTRIBUTES   [COLOR h[,s[i]]]
                           [DIFFUSE d]
                           [SPECULAR s]
                           [OPAQUE t]
              AND          [COLOR h[,s[,i]]]
                           [DIFFUSE d]
                           [SPECULAR s]
                           [OPAQUE t];
```

Name := POLYGON (ATTRIBUTES - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGA (Attr, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies that the attributes named by Attr and specified in a call to PATTR or PATTR2 apply to all subsequent polygons until superseded by another call to PPLYGA.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (ATTRIBUTES - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGA (Attr, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine specifies that the attributes named by Attr and specified in a call to PATTR or PATTR2 apply to all subsequent polygons until superseded by another call to PPLYGA.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :           :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i ];
```

Name := POLYGON (ATTRIBUTES - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGATR ( %DESCR Attr : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure specifies that the attributes named by Attr and specified in a call to PATTRIB or PATTRIB2 apply to all subsequent polygons until superseded by another call to PPLYGATR.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :           :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (ATTRIBUTES - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGATR ( CONST Attr : STRING;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure specifies that the attributes named by Attr and specified in a call to PATTRIB or PATTRIB2 apply to all subsequent polygons until superseded by another call to PPLYGATR.

This procedure is one of the procedures used to implement the PS_340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (BEGIN - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPlygB (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine begins a polygon display list. The parameter (Name) specifies the name to be given to the polygon display list defined by calls to PPLYGA, PPLYGO and PPLYGL.

Defining a polygon list requires that the user call a minimum of three subroutines: PPLYGB, to begin the list, one of the list routines (PPLYGL, PPLYGR, PPLYGH) to build the list, and PPLYGE to end the list. The subroutines PPLYGA and PPLYGO provide many options that can be specified when defining a polygon list.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (BEGIN - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPlygB (Name, ErrHnd)
```

where:

Name is a CHARACTER STRING
ErrHnd is the user-defined error-handler subroutine.

DESCRIPTION

This subroutine begins a polygon display list. The parameter (Name) specifies the name to be given to the polygon display list defined by calls to PPLYGA, PPLYGO and PPLYGL.

Defining a polygon list requires that the user call a minimum of three subroutines: PPLYGB, to begin the list, one of the list routines (PPLYGL, PPLYGR, PPLYGH) to build the list, and PPLYGE to end the list. The subroutines PPLYGA and PPLYGO provide many options that can be specified when defining a polygon list.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (BEGIN - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGBEG ( %DESCR Name  : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure begins a polygon display list. The parameter (Name) specifies the name to be given to the polygon display list defined by PPLYGATR, PPLYGOTL, AND PPLYGLIS.

Defining a polygon list requires that the user call a minimum of three routines: PPLYGBEG, to begin the list, one of the list routines (PPLYGLIS, PPLYGRGB, PPLYGHSI) to build the list, and PPLYGEND to end the list. The routines PPLYGATR and PPLYGOTL provide many options that can be specified when defining a polygon list.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (BEGIN - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGBEG ( %DESCR Name  : P_VaryingType;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DEFINITION

This procedure begins a polygon display list. The parameter (Name) specifies the name to be given to the polygon display list defined by PPLYGATR, PPLYGOTL, AND PPLYGLIS.

Defining a polygon list requires that the user call a minimum of three procedures: PPLYGBEG, to begin the list, one of the list routines (PPLYGLIS, PPLYGRGB, PPLYGHSI) to build the list, and PPLYGEND to end the list. The procedures PPLYGATR and PPLYGOTL provide many options that can be specified when defining a polygon list.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :           :           :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (END - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGE (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine ends the definition of a polygon display list.

This subroutine is one of the subroutines required to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :         :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (END - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGE (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine ends the definition of a polygon display list.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (END - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGE (ErrHnd)
```

where:

ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine ends the definition of a polygon display list.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :         :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (END - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPlygEnd  (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure ends the definition of a polygon display list.

This procedure is one of the procedures required to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :         :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (END - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGEND (PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure ends the definition of a polygon display list.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :        :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (Colors - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

CALL PPLYGH (Coplan, Nverts, Verts, Vedges, NorSpec, Norms, ColSpe, Colors, ErrHnd)

where:

Coplan is a LOGICAL
Nverts is an INTEGER*4
Verts is a REAL*4 (4, Nverts)
Vedges is a LOGICAL*1 (NVerts)
NorSpec is a LOGICAL
Norms is a REAL*4 (4, Nverts)
ColSpe is a LOGICAL
Colors is a REAL*4 (4,Nverts)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines another polygon within the polygon display list currently being constructed. The subroutine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB subroutine call. It has the following parametric definitions:

- Coplan determines whether the polygon is coplanar with the previous polygon or not.
  .TRUE. = coplanar, .FALSE. = not coplanar

- NVert specifies the number of vertices in the polygon

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

(Continued on next page)

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Vertic.

- ColSpe specifies if the colors attached to the polygon vertices are specified.
  ColSpe = .TRUE. if specified, ColSpe = .FALSE. if not specified.

- Colors specifies the colors of the vertices of the polygon. It is of the same form as Verts:
  Colors(1,n) = Hue n
  Colors(2,n) = Saturation n
  Colors(3,n) = Intensity n

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (Colors - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

CALL PPLYGH (Coplan, Nverts, Verts, Vedges, NorSpec, Norms, ColSpe, Colors, ErrHnd)

where:

Coplan is a LOGICAL
Nverts is an INTEGER*4
Verts is a REAL*4 (4, Nverts)
Vedges is a LOGICAL*1 (NVerts)
NorSpec is a LOGICAL
Norms is a REAL*4 (4, Nverts)
ColSpe is a LOGICAL
Colors is a REAL*4 (4,Nverts)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines another polygon within the polygon display list currently being constructed. The subroutine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB subroutine call. It has the following parametric definitions:

- Coplan determines whether the polygon is coplanar with the previous polygon or not.
  .TRUE. = coplanar, .FALSE. = not coplanar

- NVert specifies the number of vertices in the polygon

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

(Continued on next page)

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

- ColSpe specifies if the colors attached to the polygon vertices are specified.
  ColSpe = .TRUE. if specified, ColSpe = .FALSE. if not specified.

- Colors specifies the colors of the vertices of the polygon. It is of the same form as Verts:
  Colors(1,n) = Hue n
  Colors(2,n) = Saturation n
  Colors(3,n) = Intensity n

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (Colors - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPlygHsi    (     Coplanar  : Boolean;
                            NVertices : Integer:
                        VAR Vertices  : P_VectorListType;
                            NormSpec  : Boolean;
                        VAR Normals   : P_VectorListType;
                            Colorspec : Boolean
                        VAR Colors    : P_VectorListType;
 PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines another polygon within the polygon display list currently being constructed. The procedure may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygBeg procedure. It has the following parametric definitions:

- Coplanar determines whether the polygon is coplanar with the previous polygon or not.
  TRUE = coplanar, FALSE = not coplanar
- NVertices specifies the number of vertices in the polygon
- Vertices specifies the vertices of the polygon
  Vertices [n].Draw = False defines the edge as 'soft'
  Vertices [n].Draw = True defines the edge as 'hard'
  Vertices [n].V4[1] = vertex n: x-coordinate;
  Vertices [n].V4[2] = vertex n: y-coordinate;
  Vertices [n].V4[3] = vertex n: z-coordinate;
- NormSpec specifies if the normals to the vectors defining the polygon are specified. It is TRUE if normals are specified in the Normals array. Otherwise NormSpec = FALSE.
- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

- ColorSpec specifies colors of vertices if the colors associated with the defining polygon vertices are present. TRUE = colors are present, FALSE = colors are not present.

- Colors specifies the colors associated with the polygon vertices.
  Colors[n].Draw - Not used
  Colors[n].V4[1] = Hue value mapped to a range 0-360.0;
  Colors[n].V4[2] = Saturation value mapped to range 0-1;
  Colors[n].V4[3] = Intensity value mapped to a range 0-1;

Saturation and intensity values are clamped to the nearest range without warning.

NOTE

Polygon color by vertex capability requires PS 340 Firmware Version A2.V01 or higher and a 4K ACP.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i];
```

Name := POLYGON (Colors - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPlygHsi    (      Coplanar  : Boolean;
                             NVertices : Integer:
                         VAR Vertices  : P_VectorListType;
                             NormSpec  : Boolean;
                         VAR Normals   : P_VectorListType;
                             Colorspec : Boolean
                         VAR Colors    : P_VectorListType;
 PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines another polygon within the polygon display list currently being constructed. The procedure may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygBeg procedure. It has the following parametric definitions:

- Coplanar determines whether the polygon is coplanar with the previous polygon or not.
  TRUE = coplanar, FALSE = not coplanar

- NVertices specifies the number of vertices in the polygon

- Vertices specifies the vertices of the polygon
  Vertices [n].Draw = False defines the edge as 'soft'
  Vertices [n].Draw = True defines the edge as 'hard'
  Vertices [n].V4[1] = vertex n: x-coordinate;
  Vertices [n].V4[2] = vertex n: y-coordinate;
  Vertices [n].V4[3] = vertex n: z-coordinate;

- NormSpec specifies if the normals to the vectors defining the polygon are specified. It is TRUE if normals are specified in the Normals array. Otherwise NormSpec = FALSE.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

- ColorSpec specifies colors of vertices if the colors associated with the defining polygon vertices are present. TRUE = colors are present, FALSE = colors are not present.

- Colors specifies the colors associated with the polygon vertices.
  Colors[n].Draw - Not used
  Colors[n].V4[1] = Hue value mapped to a range 0-360.0;
  Colors[n].V4[2] = Saturation value mapped to range 0-1;
  Colors[n].V4[3] = Intensity value mapped to a range 0-1;

Saturation and intensity values are clamped to the nearest range without warning.

NOTE

Polygon color by vertex capability requires PS 340 Firmware Version A2.V01 or higher and a 4K ACP.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i]
          :         :         :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i];
```

Name := POLYGON (LIST - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGL (Coplan, Nverts, Verts, Vedges, NorSpec, Norms, ErrHnd)
```

where:

Coplan is a LOGICAL
Nverts is an INTEGER*4
Verts is a REAL*4 (4, Nverts)
Vedges is a LOGICAL*1 (NVerts)
NorSpec is a LOGICAL
Norms is a REAL*4 (4, Nverts)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines another polygon within the polygon display list currently being constructed. The subroutine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB subroutine call. It has the following parametric definitions:

- Coplan determines whether the polygon is coplanar with the previous polygon or not.
  .TRUE. = coplanar, .FALSE. = not coplanar

- NVerts specifies the number of vertices in the polygon

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

(Continued on next page)

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (LIST - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGL (Coplan, Nverts, Verts, Vedges, NorSpec, Norms, ErrHnd)
```

where:

Coplan is a LOGICAL
Nverts is an INTEGER*4
Verts is a REAL*4 (4, Nverts)
Vedges is a LOGICAL*1 (NVerts)
NorSpec is a LOGICAL
Norms is a REAL*4 (4, Nverts)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines another polygon within the polygon display list currently being constructed. The subroutine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB subroutine call. It has the following parametric definitions:

- Coplan determines whether the polygon is coplanar with the previous polygon or not.
  .TRUE. = coplanar, .FALSE. = not coplanar
- NVerts specifies the number of vertices in the polygon
- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;
- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".
- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.
- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

(Continued on next page)

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE r]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (LIST - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPlygLis  (        Coplanar  : BOOLEAN;
                             NVertices : INTEGER:
                         VAR Vertices  : P_VectorListType;
                             NormSpec  : BOOLEAN;
                         VAR Normals   : P_VectorListType;
                       PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines another polygon within the polygon display list currently being constructed. The procedure may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygBeg procedure. It has the following parametric definitions:

- Coplanar determines whether the polygon is coplanar with the previous polygon or not.
  TRUE = coplanar, FALSE = not coplanar

- NVertices specifies the number of vertices in the polygon

- Vertices specifies the vertices of the polygon
  Vertices [n].Draw = False defines the edge as 'soft'
  Vertices [n].Draw = True defines the edge as 'hard'
  Vertices [n].V4[1] = vertex n: x-coordinate;
  Vertices [n].V4[2] = vertex n: y-coordinate;
  Vertices [n].V4[3] = vertex n: z-coordinate;

- NormSpec specifies if the normals to the vectors defining the polygon are specified. It is TRUE if normals are specified in the Normals array. Otherwise NormSpec = FALSE.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i]
           :           :           :
        [[WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (LIST - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGLIS (        Coplanar  : BOOLEAN;
                            NVertices : INTEGER:
                      CONST Vertices  : P_VectorListType;
                            NormSpec  : Boolean;
                      CONST Normals   : P_VectorListType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines another polygon within the polygon display list currently being constructed. The procedure may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygBeg procedure. It has the following parametric definitions:

- Coplanar determines whether the polygon is coplanar with the previous polygon or not.
  TRUE = coplanar, FALSE = not coplanar

- NVertices specifies the number of vertices in the polygon
- Vertices specifies the vertices of the polygon
  Vertices (.n.).Draw = False defines the edge as 'soft'
  Vertices (.n.).Draw = True defines the edge as 'hard'
  Vertices (.n.).V4(.1.) = vertex n: x-coordinate;
  Vertices (.n.).V4(.2.) = vertex n: y-coordinate;
  Vertices (.n.).V4(.3.) = vertex n: z-coordinate;

- NormSpec specifies if the normals to the vectors defining the polygon are specified.
  TRUE = specified in the Normals array, FALSE = not specified.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [[WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR]  [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (OUTLINE - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGO (Outlin, ErrHnd)
```

where

Outlin is a REAL
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine specifies that Outln be used as the color (if between 1 and 360) or intensity (if between 0 and 1) of all polygons edges on the calligraphic display until superseded by another call to PPLYGO.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (OUTLINE - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGO (Outlin, ErrHnd)
```

where

Outlin is a REAL
Errhnd is the user-defined error-handler subroutine

DESCRIPTION

This subroutine specifies that Outln be used as the color (if between 1 and 360) or intensity (if between 0 and 1) of all polygons edges on the calligraphic display until superseded by another call to PPLYGO.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (OUTLINE - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGOTL ( VAR Outline : REAL;
                     PROCEDURE Error_Handler ( Err : INTEGER));
```

DESCRIPTION

This procedure specifies that Outline to be used as the color (if between 1 and 360) or intensity (if between 0 and 1) of all polygons edges on the calligraphic display until superseded by another call to PPLYGOTL.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
                 :        :        :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (OUTLINE - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGOTL ( VAR Outline : REAL;
                        PROCEDURE Error_Handler ( Err : INTEGER));
```

DESCRIPTION

This procedure specifies that Outline to be used as the color (if between 1 and 360) or intensity (if between 0 and 1) of all polygons edges on the calligraphic display until superseded by another call to PPLYGOTL.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (RGBVal - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGR (Coplan, Nverts, Verts, Vedges, NorSpec, Norms,
ColSpe, RGBVal, ErrHnd)
```

where:

Coplan is a LOGICAL
Nverts is an INTEGER*4
Verts is a REAL*4 (4, Nverts)
Vedges is a LOGICAL*1 (NVerts)
NorSpec is a LOGICAL
Norms is a REAL*4 (4, Nverts)
ColSpe is a LOGICAL
RGBVal is an INTEGER*4(3,Nverts)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines another polygon within the polygon display list currently being constructed. The subroutine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB subroutine call. It has the following parametric definitions:

- Coplan determines whether the polygon is coplanar with the previous polygon or not.
  .TRUE. = coplanar, .FALSE. = not coplanar

- NVert specifies the number of vertices in the polygon

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

(Continued on next page)

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

- ColSpe specifies if the colors attached to the polygon vertices are specified.
  ColSpe = .TRUE. if specified, ColSpe = .FALSE. if not specified.

- RGBVal specifies the colors of the vertices of the polygon.
  Colors(1,n) = Red intensity n (range 0..255)
  Colors(2,n) = Green intensity n (range 0..255)
  Colors(3,n) = Blue intensity n (range 0..255)
  Out of range values are converted to the nearest range value without warning.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
           :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (RGBVal - no corresponding command)

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PPLYGR (Coplan, Nverts, Verts, Vedges, NorSpec, Norms, ColSpe,
RGBVal, ErrHnd)
```

where:

Coplan is a LOGICAL
Nverts is an INTEGER*4
Verts is a REAL*4 (4, Nverts)
Vedges is a LOGICAL*1 (NVerts)
NorSpec is a LOGICAL
Norms is a REAL*4 (4, Nverts)
ColSpe is a LOGICAL
RGBVal is an INTEGER*4(3,Nverts)
ErrHnd is the user-defined error-handler subroutine.

DEFINITION

This subroutine defines another polygon within the polygon display list currently being constructed. The subroutine may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygB subroutine call. It has the following parametric definitions:

- Coplan determines whether the polygon is coplanar with the previous polygon or not.
  .TRUE. = coplanar, .FALSE. = not coplanar

- NVert specifies the number of vertices in the polygon

- Verts specifies the vertices of the polygon
  Verts (1, n) = vertex n: x-coordinate;
  Verts (2, n) = vertex n: y-coordinate;
  Verts (3, n) = vertex n: z-coordinate;

- Vedges specifies the "soft" versus "hard" nature of each edge specified by: Verts.
  Vedges (n) = .FALSE. if "soft edge", .TRUE. if "hard edge".

- NorSpe specifies if the normals to the vectors defining the polygon are specified.
  NorSpe = .TRUE. if specified, NorSpe = .FALSE. if not specified.

(Continued on next page)

- Norms specifies a normal to correspond to each vertex. This parameter is of the same form as: Verts.

- ColSpe specifies if the colors attached to the polygon vertices are specified.
  ColSpe = .TRUE. if specified, ColSpe = .FALSE. if not specified.

- RGBVal specified the colors of the vertices of the polygon.
  Colors(1,n) = Red intensity n (range 0..255)
  Colors(2,n) = Green intensity n (range 0..255)
  Colors(3,n) = Blue intensity n (range 0..255)
  Out of range values are converted to the nearest range value without warning.

This subroutine is one of the subroutines used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (RGBList - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGRGB (        Coplanar  : Boolean;
                            NVertices : Integer:
                        VAR Vertices  : P_VectorListType;
                            NormSpec  : Boolean;
                        VAR Normals   : P_VectorListType;
                            ColorSpec : Boolean;
                        VAR RGBList   : P_PolyColorType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines another polygon within the polygon display list currently being constructed. The procedure may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygBeg procedure. It has the following parametric definitions:

- Coplanar determines whether the polygon is coplanar with the previous polygon or not.
  TRUE = coplanar, FALSE = not coplanar

- NVertices specifies the number of vertices in the polygon
- Vertices specifies the vertices of the polygon
  Vertices [.n.].Draw = False defines the edge as 'soft'
  Vertices [.n.].Draw = True defines the edge as 'hard'
  Vertices [.n.].V4[.1.] = vertex n: x-coordinate;
  Vertices [.n.].V4[.2.] = vertex n: y-coordinate;
  Vertices [.n.].V4[.3.] = vertex n: z-coordinate;

- NormSpec specifies if the normals to the vectors defining the polygon are specified.
  TRUE = specified in the Normals array, FALSE = not specified.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

- ColorSpec specifies colors of vertices if the colors associated with the defining polygon vertices are present. TRUE = colors are present, FALSE = colors are not present.

- RGBList specifies the colors associated with the polygon vertices.
  RGBList[1,n] = Red
  RGBList[2,n] = Green
  RGBList[3,n] = Blue

  P_PolycolorType is defined as
  P_PolycolorType = ARRAY [1..3, 1..P_MaxpolygonSize] OF INTEGER;

All Red, Green, Blue values are mapped to the range 0-255. Out of range values are clamped to the nearest range without warning.

NOTE

Polygon color by vertex capability requires PS 340 Firmware Version A2.V01 or higher and a 4K ACP.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z] [C h,s,i];
```

Name := POLYGON (RGBList - no corresponding command)

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PPLYGRGB (        Coplanar  : BOOLEAN;
                            NVertices : INTEGER:
                        VAR Vertices  : P_VectorListType;
                            NormSpec  : Boolean;
                        VAR Normals   : P_VectorListType;
                            ColorSpec : Boolean;
                        VAR RGBList   : P_PolyColorType;
                    PROCEDURE Error_Handler (Err : INTEGER));
```

DEFINITION

This procedure defines another polygon within the polygon display list currently being constructed. The procedure may be called many times to specify additional polygons for the polygon display currently under construction as named by the PPlygBeg procedure. It has the following parametric definitions:

- Coplanar determines whether the polygon is coplanar with the previous polygon or not.
  TRUE = coplanar, FALSE = not coplanar

- NVertices specifies the number of vertices in the polygon
- Vertices specifies the vertices of the polygon
  Vertices [.n.].Draw = False defines the edge as 'soft'
  Vertices [.n.].Draw = True defines the edge as 'hard'
  Vertices [.n.].V4[.1.] = vertex n: x-coordinate;
  Vertices [.n.].V4[.2.] = vertex n: y-coordinate;
  Vertices [.n.].V4[.3.] = vertex n: z-coordinate;

- NormSpec specifies if the normals to the vectors defining the polygon are specified.
  TRUE = specified in the Normals array, FALSE = not specified.

- Normals specifies the normals to the corresponding vector and is of the identical form as: Vertices.

- ColorSpec specifies colors of vertices if the colors associated with the defining polygon vertices are present. TRUE = colors are present, FALSE = colors are not present.

- RGBList specifies the colors associated with the polygon vertices.
  RGBList[1,n] = Red
  RGBList[2,n] = Green
  RGBList[3,n] = Blue

  P_PolycolorType is defined as
  P_PolycolorType = ARRAY [1..3, 1..P_MaxpolygonSize] OF INTEGER;

All Red, Green, Blue values are mapped to the range 0-255. Out of range values are clamped to the nearest range without warning.

NOTE

Polygon color by vertex capability requires PS 340 Firmware Version A2.V01 or higher and a 4K ACP.

This procedure is one of the procedures used to implement the PS 340 command:

```
Name := [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i]
          :          :          :
        [WITH [ATTRIBUTES attr] [OUTLINE h]]
        POLYGON [COPLANAR] [S] x,y,z [N x,y,z]  [C h,s,i];
```

Version A2.V01

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSURGB (Red, Green, Blue, Hue, Sat, Intens)
```

where

Red, Green, Blue are INTEGER*4
Hue, Sat, Intens are REAL*4

DESCRIPTION

This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification.

For a given Hue, Saturation, and Intensity the routine returns RGB values as integers between 0 and 255 and uses a color wheel where a Hue of 0 is blue, a Hue of 120 is red, and a Hue of 240 is green.

APPLICATION SUBROUTINE AND PARAMETERS

```
CALL PSURGB (Red, Green, Blue, Hue, Sat, Intens)
```

where

Red, Green, Blue are INTEGER*4
Hue, Sat, Intens are REAL*4

DESCRIPTION

This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification.

For a given Hue, Saturation, and Intensity the routine returns RGB values as integers between 0 and 255 and uses a color wheel where a Hue of 0 is blue, a Hue of 120 is red, and a Hue of 240 is green.

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSUTIL_HSIRGB ( VAR red, green blue:INTEGER;
                          VAR Hue, Saturation, Intensity:REAL);
```

DEFINITION

This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification.

For a given Hue, Saturation, and Intensity the routine returns RGB values as integers between 0 and 255 and uses a color wheel where a Hue of 0 is blue, a Hue of 120 is red, and a Hue of 240 is green.

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Version A2.V01

APPLICATION PROCEDURE AND PARAMETERS

```
PROCEDURE PSUTIL_HSIRGB ( VAR red, green blue:INTEGER;
                          VAR Hue, Saturation, Intensity:REAL);
```

DEFINITION

This procedure converts Hue, Saturation, and Intensity color specifications to Red, Green, and Blue color specification.

For a given Hue, Saturation, and Intensity the routine returns RGB values as integers between 0 and 255 and uses a color wheel where a Hue of 0 is blue, a Hue of 120 is red, and a Hue of 240 is green.

E&S CUSTOMER SERVICE TELEPHONE INFORMATION LIST

Evans & Sutherland Customer Engineering provides a central service numbered staffed by CE representatives who are available to take requests from 9:00 a.m. Eastern Time to 5:00 p.m. Pacific Time (7:00 a.m. to 6:00 p.m. Mountain Time). All calls concerning customer service should be made to one of the following numbers during these hours. Before you call, please have available your customer site number and system tag number. These numbers are on the label attached to your PS 300 display or control unit.

Customers in the continental United States should call toll-free:

1 + 800 + 582-4375

Customers within Utah or outside the continental United States should call Dispatch at:

(801) 582-9412

If problems arise during product installation or you have a question that has not been answered adequately by the customer engineer or the customer service center, contact the regional manager at one of the following Customer Engineering offices:

Eastern Regional Manager
(for Eastern and Central Time Zones)
(518) 885-4639

Western Regional Manager
(for Mountain and Pacific Time Zones)
(916) 448-0355

If the regional office is unable to resolve the problem, you may want to call the appropriate department manager at corporate headquarters:

National Field Operations
(for field service issues)
(801) 582-5847, ext 4843

Software Support
(for sofware issues)
(801) 582-5847, ext 4810

Technical Support
(for hardware issues)
(801) 582-5847, ext 4868

Director of Customer Engineering
(for any unresolved problem)
(801) 582-5847, ext 4840

READER COMMENT FORM

Publication Number ____________________

Title ____

Your comments will help us provide you with more accurate, complete, and useful documentation. After making your comments in the space below, cut and fold this form as indicated, and tape to secure (please do not staple). This form may be mailed free within the United States. Thank you for your help.

How did you use this publication?

[] General information	[] As a reference manual
[] Guide to operating instructions	[] Other ____________________

Please rate the quality of this publication in each of the following areas.

	EXCELLENT	GOOD	FAIR	POOR
Technical Accuracy Is the manual technically accurate?	[]	[]	[]	[]
Completeness Does the manual contain enough information?	[]	[]	[]	[]
Readability Is the manual easy to read and understand?	[]	[]	[]	[]
Clarity Are the instructions easy to follow?	[]	[]	[]	[]
Organization Is it easy to find needed information?	[]	[]	[]	[]
Illustrations and Examples Are they clear and useful?	[]	[]	[]	[]
Physical Attractiveness What do you think of the overall appearance?	[]	[]	[]	[]

What errors did you find in the manual? (Please include page numbers)____________________

____
____
____
____
____
____
____
____

Name ____________________	**Street** ____________________
Title ____________________	**City** ____________________
Department ____________________	**State** ____________________
Company ____________________	**Zip Code** ____________

Fold

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 4632 SALT LAKE CITY, UTAH

POSTAGE WILL BE PAID BY ADDRESSEE

EVANS & SUTHERLAND
580 Arapeen Drive
Salt Lake City, Utah 84108

ATTN: IAS TECHNICAL PUBLICATIONS

Fold

Cut along dotted line