DrawP3D Fortran Language Reference Guide

Joel Welling, Ph.D
Pittsburgh Supercomputing Center

DrawP3D is a general purpose subroutine library for creating and displaying 3D models. Interfaces are available from Fortran and C. This document is the reference guide for the Fortran language interface .

1. Introduction

This document gives the details of the Fortran language interface to DrawP3D. It begins with general usage instructions, and includes detailed descriptions of predefined quantities and the function calling interface .

2. Usage

The Fortran interface to DrawP3D uses a number of constants with special meanings. These constants are defined using PARAMETER statements in the file drawp3d_def.f, which can be included in your code if your Fortran compiler supports file inclusion. If not, you will have to insert the definitions manually or use the parameter values in place of the names. These parameters will be written in upper case; they all begin with P and their values are given elsewhere in this document. In general, when a constant is mentioned in this manual, it will be followed by its value in parenthesis, for example PTRUE (1).

Drawp3d_def.f also provides type statements for the functions of DrawP3D. All are integer-valued functions, returning either PPASS (1) or PFAIL (0) depending on whether the function succeeded or failed for some reason. Unless otherwise specified, all parameters are input only; they are not modified by DrawP3D.

Compile the DrawP3D program as you would any Fortran program, and link it against the DrawP3D object library. On Unix systems this will be done using a switch like -ldrawp3d on the appropriate command; on VMS systems you will need to include drawp3d.olb. These library names are system-dependent, so check to make sure the names are the same at your site.

3. Specifying Colors, Points, and Vectors

The Fortran interface to DrawP3D often requires the user to specify a color as a real array of dimension 4, or to specify a point or vector as a real array of dimension 3. This is done in the obvious fashion. Consider COLOR, POINT, and VECTOR defined as follows:

```
REAL COLOR(4), POINT(3), VECTOR(3)
```

Then COLOR(1) through (4) will be the red, green, blue, and opacity components of the color respectively, and POINT(1) through POINT(3) or VECTOR(1) through VECTOR(3) will be the X, Y, and Z components of the point or vector respectively .

4. Vertex Lists

In DrawP3D, a vertex must include a set of 3D coordinates. It can also include a local normal vector and/or either a color or a scalar value to be mapped into a color by the current color map function. Some special functions provide for vertices with two scalar values (and no colors), although most functions will ignore this second value. Considering all these cases, there are seven different possible collections of data for a vertex.

Further, the color information given could be given in several color formalisms. At the moment, the only one supported by DrawP3D is the red-green-blue-opacity color system, but support for others may be added in the future.

This collection of options means that every time a list of vertex data is given, you must specify what data is given for each vertex and what color system is to be used. (Obviously we could have left out the color system selection, but we felt it would be best to leave it in so that codes written now would still work when more options are available). These choices are needed every time you define a geometrical primitive that requires a list of vertices.

Each primitive creation function includes parameters for a vertex type and a color mode. The possible vertex types are PCVX (0) (for coordinate-only vertices), PCCVX (1) (coordinates and colors), PCNVX (3) (coordinates and normals), PCCNVX(2) (coordinates, colors, and normals), PCVVX (4) (coordinates and values to be mapped to colors), PCVNVX (5) (coordinates, values, and normals), and PCVVVX (6) (coordinates and two values). The option for color mode should be specified as PRGB (0), for the red-green-blue-opacity color specification scheme.

In the FORTAN language interface to DrawP3D, functions requiring vertex data are passed three data arrays. If there are N vertices to be defined, the first of these will be a real array of dimension (3,N) containing the X, Y, and Z coordinates of each vertex. This array is generally called 'coords' in the function specifications in this document.

The second array can take three forms, depending on the vertex type the user passes. If the vertex type requires a color, the array will be of dimension (4,N), and each entry will contain a red, a green, a blue, and an opacity value for that vertex. These values are real between 0.0 and 1.0 inclusive, 1.0 being brightest or most opaque. If the vertex type requires single scalar values to be mapped to colors by the current color map function, the array will be of dimension (1,N), and each real entry will be the value for that vertex. If the vertex type requires two scalar values, the array will be of dimension (2,N), and each pair of real entries will be the values for that vertex. The DrawP3D routines know enough to avoid confusing the two. This array is generally called 'colors' in the function specifications.

The third data array provides normal information for each vertex. It is a real array of dimension (3,N), and each entry includes the X, Y, and Z components of that vertex. This array is generally called norms in the function specifications.

Thus, different vertex list types require different data parameters. The correspondence between type and parameters is as follows:

vertex type	coords param	colors param	normals param
PCVX	real coords(3,N)	none	none
PCCVX	real coords(3,N)	real colors(3,N)	none
PCCNVX	real coords(3,N)	real colors(3,N)	real norms(3,N)
PCNVX	real coords(3,N)	none	real norms(3,N)
PCVVX	real coords(3,N)	real colors(1,N)	none

PCVNVX	real coords(3,N)	real colors(1,N)	real norms(3,N)
PCVVVX	real coords(3,N)	real colors(2,N)	none

Note that some vertex types do not require some of this data. For example, vertices of type PCVX do not require either colors or normals. If the DrawP3D routines are called with vertex types which do not require a certain vertex data parameter, that parameter will never be used, so it is safe to substitute something like '0' for it in the parameter list .

5. Predefined Objects

Object	Purpose
standard_camera	A camera looking toward the origin from a point 20.0 units up the Z axis, with the up direction lying along the Y axis.
standard_lights	A GOB consisting of a positional light at the point (0.0, 2.0, 20.0) and a weak ambient light.

6. Fortran Parameters Used by the DrawP3D Interface

Constant	Value	Use
Version numbers:		
PPGENV	1.0	Current version of P3DGen
PDP3DV	1.0	Current version of DrawP3D
Return codes:		
PPASS	1	Function status return code
PFAIL	0	Function status return code
Booleans:		
PTRUE	1	Boolean

PFALSE	0	Boolean

Vertex types:

PCVX	0	Vertices are coordinate only
PCCVX	1	Vertices have coordinate, color
PCCNVX	2	Vertices have coordinate, color, normal
PCNVX	3	Vertices have coordinate, normal
PCVVX	4	Vertices have coordinate, value
PCVNVX	5	Vertices have coordinate, value, normal
PCVVVX	6	Vertices have coordinates, two values

Color modes:

PRGB	0	Color information is RGBA

Predefined materials:

PDFMAT	0	Default material (renderer dependent)
PDULL	1	Dull material (e.g. paper)
PSHINY	2	Shiny material (e.g. plastic, ceramic)
PMETAL	3	Metallic material (e.g. chrome)
PMATTE	4	Matte material (no reflection, e.g. soot)
PALUM	5	Aluminum material (less shiny than chrome)

Symbol name length:

PNAMELN	64	Maximum length of names of objects

7. Functions

pamblt

Purpose: Create an ambient light primitive GOB

Use: pamblt(ctype, color)

Parameters:

ctype: integer color type (currently must be PRGB (0))

color: real array(4) containing color components

Discussion: This function adds an ambient light source to the current GOB. The opacity component of the input color is ignored.

pascal

Purpose: Add an anisotropic scaling transformation to the current GOB

Use: pascal(xscale, yscale, zscale)

Parameters:

xscale: real scaling factor in the current GOB's X direction

yscale: real scaling factor in the current GOB's Y direction

zscale: real scaling factor in the current GOB's Z direction

Discussion: This function adds an anisotropic scaling transformation to the current GOB. This transformation will be concatenated with the existing transformation by left multiplication; GOBs with no existing transformation effectively have the identity transformation associated with them. To add a scaling transformation which is uniform in all directions, see pscale.

paxis

Purpose: Add an axis gob to the current GOB

Use: paxis(strt, end, up, strtvl, endvl, ntics, lbl, txtsz, precsn)

Parameters:

strt: real array(3) point where axis starts

end: real array(3) point where axis ends

up: real array(3) up direction of text

strtvl: real value to be placed at start

endvl: real value to be placed at end

ntics: integer value for the number of tics to be displayed. Includes end and start.

lbl: character string to be placed below startval

txtsz: real value which gives the size of the text

precsn: integer value for the number of decimal points to be displayed (0..8)

Discussion: This function adds an axis to the Current GOB. The axis itself is in the (end-start) direction with the tics in the negative (up) direction. The front of the axis is the cross product of (end-start) and (up). Startval is placed at start and endval placed at end. Other values are interpolated and placed at the corresponding tics. The number of tics is specified, and this number includes the end and start tic. If num_tics is less than 2, it is set to 2 by default. The label is placed below startval. All text is scaled by text_size, which is exactly the same as in ptxtht(). The variable precision defines the number of decimal places to be displayed. This range is between 0 and 8, and if the range is exceeded, then precision is set to the nearest number that is within the range.

pbezp

Purpose: Create a nameless Bezier Patch primitive GOB

Use: pbezp(vtxtyp, ctype, coords, colors, norms)

Parameters:

vtxtyp: integer vertex type specifier constant

ctype: integer color type (currently only P3D_RGB)

coords: real array of coordinate data (16 vertices)

colors: real array of color data

norms: real array of normal data

Discussion: This produces a simple bicubic Bezier patch, with a four by four pattern of knots specified by the data given by the 16 vertices. See the section on vertex lists in this document for format information. The use of coloring information on the inner four vertices is renderer dependent.

pbkcul

Purpose: Add a backface culling attribute to the current GOB

Use: pbkcul(flag)

Parameters:

flag: integer; turns backface culling on if non-zero, off if zero

Discussion: Backface culling is a useful feature to save time by not drawing hidden faces of closed objects. It causes surfaces facing away from the not to be drawn. See the description of the various primitives in the User's Guide for information on which surface is the front of any given primitive. This function adds the this attribute to the currently open GOB, where it will apply to that GOB and any of its descendents.

pblatt

Purpose: Add an arbitrary boolean-valued attribute to the current GOB

Use: pblatt(attrib, flag)

Parameters:

attrib: character string specifying the attribute name

flag: integer value of the attribute; non-zero means true

Discussion: This procedure can be used to add an arbitrary boolean attribute to a GOB.

pbndbx

Purpose: Create a nameless Bounding Box primitive GOB

Use: pbndbx(crna, crnb)

Parameters:

crna: real array(3) specifies one of the bounding box corners.

crnb: real array(3) specifies the other bounding box corner.

Discussion: This procedure produces a bounding box. The two corners specify opposite corners on the bounding box diagonal.

pcamra

Purpose: Create a camera object

Use: pcamra(name, lookfm, lookat, up, fovea, hither, yon)

Parameters:

name: character string giving the name for the camera

lookfm: real array(3) giving the location for the camera

lookat: real array(3) giving the location at which to point the camera

up: real array(3) giving the vector specify the rotational position of the camera about the line from lookfm to lookat

fovea: real giving the opening angle of the view in degrees

hither: negative real distance to the near clipping plane

yon: negative real distance to the far clipping plane

Discussion: This procedure creates a camera object, which can be used to specify a view of a model. See the discussion of cameras in the DrawP3D User's Guide for details. Note that the hither and yon values are negative numbers, with the magnitude of hither less than that of yon. Those objects which fall within a cone of opening angle fovea with its base at lookfrom and centered on lookat, and which fall between the hither and yon distances, will be rendered by a psnap function using this camera.

pchild

Purpose: Add a child GOB to the current GOB.

Use: pchild(name)

Parameters:

name:	character string giving the name of the previously defined GOB to be added

Discussion: This function causes a previously defined GOB to become a child of the current GOB. Note that it can also be a child of one or more other GOBs, and that it can (and will) have children of its own.

pclatt

Purpose: Add an arbitrary color-valued attribute to the current GOB.

Use: pclatt(name, ctype, r, g, b, a)

Parameters:

name:	character string giving the attribute name
ctype:	integer color type (currently must be PRGB (0))
r:	real red color intensity
g:	real green color intensity
b:	real blue color intensity
a:	real color opacity

Discussion: This procedure can be used to add an arbitrary color attribute to a GOB. Do not confuse this with setting the color of the GOB, which is done with the function pgbclr and involves setting the color attribute named 'color'.

pclose

Purpose: Close the current GOB.

Use: pclose()

Parameters:

Discussion: This function closes the current GOB. If the current GOB is a child of another open GOB, for example if it was opened by calling popen with an empty character string for a name, the parent open GOB becomes the current GOB. If the current GOB was opened by calling popen with a non-empty name, it must be the highest level open GOB. In this case when the GOB is closed there will be no current GOB until the next call to popen.

pclsrn

Purpose: Close a renderer.

Use: pclsrn(name)

Parameters:

name: character string giving the name of the renderer to be closed

Discussion: This function closes the named renderer. It stays closed until reopened with a call to popnrn. While the renderer is closed, it is not informed of snap functions, the creation of new GOBs or attributes, or anything else; as far as the renderer is concerned absolutely nothing happens while it is closed. This means that it is an error to, for example, define a GOB while the renderer is closed and then open the renderer and attempt to render the GOB.

pcmbkg

Purpose: Set the background color of a camera object.

Use: pcmbkg(name, ctype, color)

Parameters:

name: character string giving the name of the camera to be changed

ctype: integer color type (currently must be PRGB (0))

color: real array(4) containing color components

Discussion: This function changes the background color of a camera. All scenes subsequently rendered with the changed camera will have backgrounds of the given color, unless and until the background color is changed again. Cameras are created with a black background color.

pcyl

Purpose: Add a cylinder primitive to the current GOB.

Use: pcyl()

Parameters:

Discussion: This procedure adds a cylinder primitive to the current GOB. The cylinder is of radius 1.0, and extends from the origin to the point (0.0, 0.0, 1.0). It is expected that the user will apply a transformation to it to produce a cylinder of the desired shape in the desired location.

pdebug

Purpose: Toggle the debugging trace on and off.

Use: pdebug()

Parameters:

Discussion: DrawP3D can produce a debugging trace of its internal operation. The trace is written to the standard error output; it can be very verbose but can be useful for debugging and bug reporting. If no trace is being produced pdebug turns it on; if it is on pdebug turns it off.

pfatt

Purpose: Add an arbitrary floating point attribute to the current GOB.

Use: pfatt(name, value)

Parameters:

name: character string giving the attribute name

value: real value to be associated with name

Discussion: This procedure can be used to add an arbitrary floating point attribute to a GOB.

pfree

Purpose: Free a named GOB.

Use: pfree(name)

Parameters:

name: character string giving the name of the GOB to free

Discussion: This function explicitly removes the name from a named GOB. If this GOB is not the descendent of another named GOB, it will be freed, and all of its descendents which do not have names and are not descendents of named GOBs will also be freed. The effect of this mechanism is that a GOB is freed as soon as it becomes impossible to refer to it or any of its ancestors. Note that using popen to create a new gob with the same name as an existing GOB has the effect of applying pfree to the existing GOB.

pgbclr

Purpose: Specify the gob color attribute of the current GOB.

Use: pgbclr(ctype, r, g, b, a)

Parameters:

ctype: integer color type (currently must be PRGB (0)) r: real red color intensity

g: real green color intensity

b: real blue color intensity

a: real color opacity

Discussion: This procedure can be used to set the gob color attribute of a GOB. This call is equivalent to calling pclatt with the attribute name 'color' and the same color parameter.

pgbmat

Purpose: Specify the gob material attribute of the current GOB.

Use: pgbmat(material)

Parameters:

material: integer ID of material

Discussion: This procedure can be used to set the gob material attribute of a GOB. This call is equivalent to calling pmtatt with the attribute name 'material' and the same material parameter. The material ID must be one of the predefined materials, described earlier in this document.

piatt

Purpose: Add an arbitrary integer-valued attribute to the current GOB.

Use: piatt(name, value)

Parameters:

name: character string giving the attribute name

value: integer value to be associated with name

Discussion: This procedure can be used to add an arbitrary integer attribute to a GOB.

pintrn

Purpose: Create, initialize and open a renderer.

Use: pintrn(name, rndrer, device, data)

Parameters:

name:	character string name to be assigned to the new renderer
rndrer:	character string identifying the renderer
device:	character string specifying output device to use (renderer-dependent meaning)
data:	character string specifying renderer data string (renderer-dependent meaning)

Discussion:

This function creates, initializes, and opens a renderer. Three possible renderer types are P3D, Painter, and GL.

A P3D renderer produces output in the form of P3D model files. For this renderer, the rndrer parameter must be "p3d". The device parameter specifies the output file, and the data parameter is ignored. Multiple instances of this renderer may be used simultaneously. The P3D renderer recognizes the device string "-" as meaning that output should be written to the standard output.

When the rndrer parameter string is "painter", a Painter renderer is created. This renderer used Painter's Algorithm to draw three dimensional objects. This means simply that the object is drawn from back to front, so that closer parts of the object are drawn later and hide the parts behind them. This algorithm is fast, but it is not completely accurate- sometimes errors are made in which surfaces which should be hidden are visible and vice versa. Also, for implementation reasons and because of the need for speed, many features of DrawP3D like specular lighting and shading of colors across polygons are not supported by this renderer. Despite all this, it produces quite nice images very fast, and is very useful for scientific models.

In creating a Painter renderer, the device parameter string is used to specify the output device interface to be used. The Painter Renderer uses the output software from the PSC's GPLOT graphics package, and so can write to any device to which GPLOT can write. The parameter strings for some devices are "xws" or "xl" for the X Window System (xl being better for monochrome displays), "cgmb" and "cgmc" for binary and clear text CGM respectively, and "ps" for postscript. The data parameter string specifies the file name to which output is to be written. The special string "-" means to write output to the Unix standard output. This value should also be used for output devices which do not write to a file, such as X.

Because of internal restrictions in the GPLOT device interface, only one Painter renderer can exist at a time. We hope to remove this restriction with a later release of DrawP3D. The output window produced by the X Window System devices "xws" and "xl" will have names appropriate to GPLOT rather than DrawP3D; we hope to improve this in a later release as well.

When the rndrer parameter string to pintrn is "gl", an interface to the Silicon Graphics GL graphics library is created. This allows high-speed rendering of DrawP3D models on displays which support GL. You may create as many GL renderers as you want; they can all operate independently at the same time.

In creating a GL renderer, the device parameter string is currently unused. The data parameter string is composed of a series of keyword entries. Currently supported entries are "title" for the window title and " geometry" for the window size and position (in standard X Window System format). So, the following string would produce a window with the given title and size:

```
'title="my_window",geometry="256x256+0+0"'
```

pisosf

Purpose: Create an iso-valued surface composite GOB

Use: pisosf(vtxtyp, data, vdata, nx, ny, nz, value, crnr1, crnr2, inside)

Parameters:

vtxtyp:	integer vertex type to generate
data:	3D real array of data, of dimensions (nx,ny,nz)
valdata:	3D real array of value data for coloring, of dimensions (nx,ny,nz)
nx, ny, nz:	dimensions of data and valdata
value:	value of the isosurface to extract
crnr1:	real array(3) specifying one corner of the volume in which to draw the surface
crnr2:	real array(3) specifying the other corner
inside:	integer flag to control whether inner or outer surfaces are drawn

Discussion:

The isosurface composite function is used to extract a surface of constant value from a three dimensional array of real data. The best way to understand what this means is to think of a continuous function in three dimensional space. Consider all the points in space for which the function has a given value. Because the function is continuous, the points must group together to form two dimensional closed surfaces in space. These surfaces are iso-valued surfaces, or isosurfaces, of the continuous function.

pisosf assumes that the grid of data that it is passed forms samples of a continuous function in space, and extracts the isosurface of that function at a given function value. Linear interpolation of the data is used to calculate values between grid points. The surfaces formed are closed unless they intersect the boundary of the grid; in that case the parts of the surface that would fall outside are omitted and holes are left in the surface.

The data parameter and its nx, ny, and nz dimensions provide the data for the 3D grid, which is assumed to fill the space defined by the opposite corners corner1 and corner2. The value parameter provides the constant value of the isosurface.

The inside parameter controls whether inner or outer surfaces are drawn. The outer surface is that for which function values above the surface value lie inside the surface; inner surfaces are drawn such that higher function values lie outside. The distinction controls which direction the 'front' sides of the polygons of the surface are facing.

The two remaining parameters, vtxtyp and vdata, control the appearance of the surface. If the vertex type is one which includes normal vectors, normals will be generated; this makes the surface look smoother but substantially increases the memory and file size of the model and takes time to compute. If a vertex type requiring value information is given, the grid of value data is accessed at appropriate points (using linear interpolation as necessary), and the current color map is used to color the surface appropriately. Vertex types implying that explicit colors are to be given for the vertices are converted to the most similar type without explicit colors.

The algorithm used is Lorensen and Cline's Marching Cubes algorithm, as described in the proceedings of Siggraph '87, with several corrections. See the source code for details.

See the routine pbndbx for an easy way to show the boundaries of the volume from which the surface is being extracted.

This routine involves the correct handling of a large number of distinct cases. We believe that all the cases are properly coded, but errors may exist. If you encounter a case where the algorithm seems to create an incorrect surface, particularly one with missing triangles or holes not at the boundary of the volume specified by the corners, please let us know so that we can make the appropriate corrections.

plight

Purpose: Create a positional light source primitive GOB

Use: plight(loc, ctype, color)

Parameters:

- loc: real array (3) giving the position of the light source.
- ctype: integer color type (currently must be PRGB (0))
- color: real array(4) giving the color for the light source.

Discussion: This procedure is used to add a positional light source primitive to the currently open GOB. The opacity component of the input color is ignored. The renderer may retreat the light source to infinity, or otherwise neglect some of the light source data.

pmesh

Purpose: This function adds a generalized mesh geometrical primitive to the current GOB.

Use: pmesh(vtxtyp, ctype, npts, coords, colors, norms, nfacet, faclng, vrtind)

Parameters:

- vtxtyp: integer vertex type
- ctype: integer color type (currently must be PRGB (0))
- npts: integer number of vertices
- coords: real array of coordinate data
- colors: real array of color data
- norms: real array of normal data
- nfacet: integer number of facets
- faclng: integer array of facet length data

vrtind: integer array of facet vertex index data (numbered from 0)

Discussion:

The general mesh is one of the most useful primitives, but also one of the more complex. It consists of a collection of polygonal facets, sharing a common collection of vertices. The first five parameters to pmesh define the vertices (in the usual fashion- see the section on vertex lists in this document), while the remainder of the parameters define how they are grouped to form the facets.

If there are nfacet facets, faclng must be an array of length nfacets containing the number of vertices in each facet. vrtind must be an array of total length equal to the sum of the values in faclng, containing indices into the array of vertices specified by vdata. The first element of facet_lengths specifies a number of entries to be taken from facet_vertices to define the first facet, the second element specifies the number of entries to define the second facet, and so on. NOTE: Vertex indices in vrtind number the vertices from 0, as is the C language custom, rather than from 1, as is the Fortran custom. The front faces of the facets are defined by the right hand rule applied to the vertices in the order in which their indices appear in facet_vertices. This means that the facets of a smooth surface do not necessarily all have their front faces on the same side of the surface.

pmtatt

Purpose: Add a material-valued attribute to the current GOB.

Use: pmtatt(name, material)

Parameters:

name: character string giving the attribute name

material: integer ID of material

Discussion: This procedure can be used to add an arbitrary material attribute to a GOB. Do not confuse this with setting the material of the GOB, which is done with the function pgbmat and involves setting the material attribute named 'material'. The material ID must be one of the predefined materials, described earlier in this document.

popen

Purpose: Create and open a new named or unnamed GOB.

Use: popen(name)

Parameters:

name: character string given the name for the new GOB, or ' ' for a nameless GOB

Discussion: All non-primitive GOBs are created and opened using popen and closed using pclose. If popen is called with a non-empty name, a named GOB is created. This can only happen if no GOB is currently open. If popen is called with an empty character string (") or a character string containing only a single space (' ') as a parameter, a nameless GOB is created. This can only happen if another GOB is already open; the new GOB becomes a child of the previously open GOB.

popnrn

Purpose: Open a previously closed renderer.

Use: popnrn(name)

Parameters:

name: character string giving the name of the renderer to reopen

Discussion: This routine opens a renderer previously closed using dclsrn. It is not necessary to open a renderer immediately after it is created by pinirn, since renderers are created open. Note that any definitions, attributes, snaps, or other changes are ignored by the renderer while it is closed. Thus it is an error to define a GOB while a renderer is closed, open the renderer, and then attempt to render that GOB.

pplygn

Purpose: Add a polygon geometrical primitive to the current GOB.

Use: pplygn(vtxtyp, ctype, npts, coords, colors, norms)

Parameters:

vtxtyp:	integer vertex type specifier
ctype:	integer color type (currently must be PRGB)
npts:	integer number of vertices
coords:	real array of coordinate data
colors:	real array of color data
norms:	real array of normal data

Discussion: This function adds a polygon bounded by the given vertices to the current GOB. See the vertex list section of this document for a description of the vertex data format. It is the user's responsibility to make sure that the vertices given are coplanar. The handling of concave polygons is renderer-dependent. The handling of self-intersecting polygons is very renderer-dependent and should be avoided if at all possible. The front face of the polygon is determined by applying the right hand rule to the circulation direction of the vertices.

pplyln

Purpose: Add a polyline geometrical primitive to the current GOB.

Use: pplyln(vtxtyp, ctype, npts, coords, colors)

Parameters:

vtxtyp:	integer vertex type specifier

ctype: integer color type (currently must be PRGB)

npts: integer number of vertices

coords: real array of coordinate data

colors: real array of color data

Discussion: This function adds a polyline defined by the given vertices to the current GOB. See the vertex list section of this document for a description of the vertex data format. Since normal vectors are not applicable to the drawing of polylines, they are not included in the parameter list. Vertex types including normals are converted to the otherwise similar type without normals.

pplymk

Purpose: Add a polymarker geometrical primitive to the current GOB.

Use: pplymk(vtxtyp, ctype, npts, coords, colors)

Parameters:

vtxtyp: integer vertex type specifier

ctype: integer color type (currently must be PRGB)

npts: integer number of vertices

coords: real array of coordinate data

colors: real array of color data

Discussion: This function adds a polymarker defined by the given vertices to the current GOB. See the vertex list section of this document for a description of the vertex data format. At the moment, DrawP3D only supports point markers. Since normal vectors are not applicable to the drawing of polylines, they are not included in the parameter list. Vertex types including normals are converted to the otherwise similar type without normals.

pprtcm

Purpose: Write a description of a camera to the standard output.

Use: pprtcm(name)

Parameters:

name: character string giving the name of the camera to be written out

Discussion: This routine causes a description of the given camera, including all internal parameters, to be written to the standard output. This can be very useful for debugging purposes.

pprtgb

Purpose: Write a description of a GOB to the standard output.

Use: pprtgb(name)

Parameters:

name: character string giving the name of the GOB to be written out

Discussion: This routine causes a hierarchical description of the given GOB to be written to the standard output. This description includes all of the GOB's attributes, its transformation if any, and all of its children recursively, down to and including any primitive GOBs. This can be very useful for debugging purposes.

pprtrn

Purpose: Write a description of a renderer to the standard output.

Use: pprtrn(name)

Parameters:

name: character string giving the name of the renderer to be written out

Discussion: This routine causes a description of the given renderer, including output file and parameter information, to be written to the standard output. This can be very useful for debugging purposes.

pptatt

Purpose: Add an arbitrary point-valued attribute to the current GOB.

Use: pptatt(name, x, y, z)

Parameters:

name:	character string giving the attribute name
x:	real x coordinate of the point
y:	real y coordinate of the point
z:	real z coordinate of the point

Discussion: This procedure can be used to add an arbitrary point attribute to a GOB.

prniso

Purpose: Create an iso-valued surface composite GOB from randomly distributed data.

Use: prniso(vtxtyp, ctype, npts, coords, vals, norms, value, inside)

Parameters:

vtxtyp:	integer vertex type specifier
ctype:	integer color type (currently must be PRGB)
npts:	integer number of vertices
coords:	real array of coordinate data
vals:	real array of value data (as for vertex list 'colors')
norms:	real array of normal data
value:	real value at which the isosurface is to be drawn

inside: integer flag to control whether inner or outer surfaces are drawn

Discussion:

The random isosurface composite function is used to extract a surface of constant value from a collection of data points in 3D space. The data may be (and in fact should be) ungridded; it need not be defined on a regular array.

The best way to understand isosurfaces is to think of a continuous function in three dimensional space. Consider all the points in space for which the function has a given value. Because the function is continuous, the points must group together to form two dimensional closed surfaces in space. These surfaces are iso-valued surfaces, or isosurfaces, of the continuous function. The data points given provide function values at various points within the space. The random isosurface utility finds a function which has the given values at the given points and is linearly interpolated between them, and extracts an isosurface of that function. The surfaces formed are closed unless they intersect the convex hull of the given data; in that case the parts of the surface that would fall outside are omitted and holes are left in the surface.

The data values are provided to the function in the form of a vertex list. Each point must have three coordinates, and a value to provide the local value of the function defining the isosurface. It is also possible to specify a second value. In this case, the second value will be used to color the surface according to the current color map. Normals may also be provided and will be interpolated, but this is not a very sensible thing to do unless the normals are known to be perpendicular to the surface which is generated. (This will be the case if the normals given correspond to the gradient of the function in question). These combinations of possible vertex quantities mean that the valid vertex types are PCVVX, PCVVVX, and PCVNVX.

The inside parameter controls whether inner or outer surfaces are drawn. The outer surface is that for which function values above the surface value lie inside the surface; inner surfaces are drawn such that higher function values lie outside. The distinction controls which direction the 'front' sides of the polygons of the surface are facing.

The isosurface is generated by first finding the Dirichlet and Delaunay tesselations of the data points, and then using the Delaunay tesselation to define a lattice of tetrahedra, the vertices of which are the given data points. A 'Marching Tets' algorithm is applied to these tetrahedra to extract the isosurface. Value data for coloring or interpolated normal data are calculated by linear interpolation along the edges of the tetrahedra as needed. Note that the result is unique; there are no hidden free parameters the adjustment of which might produce a different isosurface from the same data.

The algorithm used to calculate the Dirichlet and Delaunay structures if that of A. Bowyer (see The Computer Journal, vol. 24, no. 2, 1981). The process is very compute intensive and uses a great deal of memory; generating isosurfaces for large numbers of data points is a very memory-intensive and time-consuming task. Thus it is much better to use pisosf if at all possible.

prnzsf

Purpose: Create a Z surface composite GOB from randomly distributed data

Use: prnzsf(vtxtyp, ctype, npts, coords, colors, norms, tfnull, tfun)

Parameters:

vtxtyp:	integer vertex type specifier constant
ctype:	integer color type (currently only P3D_RGB)
coords:	real array of coordinate data
colors:	real array of color data
norms:	real array of normal data
tfnull:	integer; if PTRUE, then include all points
tfun:	a subroutine that excludes points from the Z surface.

Discussion:

This function connects the given data points to form a surface. The given points must form a continuous, single-valued surface in z as a function of x and y. The surface drawn is bounded in the x-y plane by the convex hull of the data points. All vertex types are supported; normal, color, or value data associated with the vertices is used to properly color or light the surface created. For example, vertices of type P3D_CVVTX could be used to color the surface according to a function different from its height.

The user can cause points to be excluded from the surface by setting tfnull to PFALSE (0), and specifying a subroutine tfun to be used in determining which points to exclude. If tfnull is PTRUE (1), all points will be included in the surface and the parameter tfun will be ignored. Excluding a point causes a hole to appear in the surface where the point would have been.

Tfun is a user supplied subroutine that determines whether a point should be excluded from the zsurface. It should be defined as follows:

```
subroutine tfun( ixclde, x, y, z, index )
integer ixclude, index
real x, y, z
```

This function is called several times for each point in the data, with the x, y, and z parameters being passed the coordinates of the data point and the index being passed the ordinal index of the given point in the data. NOTE THAT THE INDICES ARE NUMBERED FROM 0, rather than from 1 as would be traditional in Fortran. If on return from the function ixclde is PTRUE (1), the point will be excluded and a hole will be left in the surface at the appropriate point. If ixclde is PFALSE (0), the point will be included. This enables the user to remove points on the basis of position or value.

The connections between points which define the surface are found by calculating the Dirichlet tesselation of the points. This process divides the space in the x-y plane into regions closest to each of the given points. These regions are polygons; the collection of polygons is the Dirichlet tesselation. The dual to this set of polygons is a collection of triangles which have the given data points as their vertices. These triangles make up the Delaunay triangulation of the data and are what is actually drawn to create the Z surface. The convex hull of the data is automatically calculated during the generation of the Delaunay triangulation. The triangulation calculated is completely unambiguous; no hidden free parameters are used to choose which data points are connected to which.

The algorithm used to calculate the Dirichlet and Delaunay structures if that of A. Bowyer (see The Computer Journal, vol. 24, no. 2, 1981). The process is very compute intensive and uses a great deal of memory; generating Z surfaces for large numbers of data points is a very memory-intensive and time-consuming task. Thus it is much better to use pzsurf if at all possible.

protat

Purpose: Add a rotation transformation to the current GOB.

Use: protat(x, y, z, angle)

Parameters:

x:	real x component of axis of rotation
y:	real y component of axis of rotation
z:	real z component of axis of rotation
angle:	angle to rotate, in degrees

Discussion: This function adds a rotation transformation to the current GOB. This transformation will be concatenated with the existing transformation by left multiplication; GOBs with no existing transformation effectively have the identity transformation associated with them. The positive direction of rotation is determined by the right hand rule.

pscale

Purpose: Add a scaling transformation to the current GOB.

Use: pscale(factor)

Parameters:

factor: real factor by which to scale the GOB

Discussion: This function adds a scaling transformation to the current GOB. This transformation will be concatenated with the existing transformation by left multiplication; GOBs with no existing transformation effectively have the identity transformation associated with them. Values of scale greater than 1.0 enlarge the GOB; values less than 1.0 reduce it. pscale adds a scaling transformation which is uniform in all directions; to add an anisotropic scaling transformation see pascal.

psdcmp

Purpose: Set the current color map to one of several standard functions.

Use: psdcmp(min, max, which)

Parameters:

min: real value to be mapped to 0.0

max: real value to be mapped to 1.0

which: integer selector for the standard map function desired

Discussion:

When vertex lists of types containing scalar values (for example, PCVVX (4)) are rendered, the values are translated to colors by the current color map function. psdcmp allows the user to easily select that function.

When translation occurs, the value is first rescaled into the interval from 0.0 to 1.0. If the value is beyond the range delimited by min and max, it maps to 0.0 if it is closer to min or 1.0 if it is closer to max. Otherwise, it maps to (value-min)/(max-min), which is in the range 0.0 to 1.0 inclusive. The mapped value is then passed to the a mapping function. whichmap selects which of the mapping functions will be used, from the following list:

```
whichmap mapping function

0  gray scale, 0.0 being black and 1.0 white
1  fades from blue (0.0) to yellow (1.0)
2  fades from red (0.0) to blue (1.0), with
   waves of green to form 'contours'
3  pseudospectral
4  inverted pseudospectral
```

Vertex type PCVVVX contains two scalar values. Most primitives will ignore the second value and apply the color map to the first value. The dp_rand_isosurf function will use the first set of values associated with each vertex to produce its isosurface, and color the surface by interpolation of the second set of values.

pshtdn

Purpose: Close and shut down all renderers and DrawP3D.

Use: pshtdn()

Parameters:

Discussion: This routine is intended to be called at the end of a program using DrawP3D. It closes and shuts down all initialized renderers, and does everything else necessary to terminate the session.

pshtrn

Purpose: Close and shut down a renderer.

Use: pshtrn(name)

Parameters:

name:	character string giving the name of the renderer to be shut down

Discussion: This function closes and shuts down a renderer, destroying the renderer object. Once it has been shut down, the renderer cannot be reopened. Any reference to a GOB known to a renderer which has since been shut down is an error, and may result in a program crash. See also pshtdn, which shuts down all initialized renderers.

psnap

Purpose: Cause a model to be rendered.

Use: psnap(model, lights, camera)

Parameters:

model:	character string giving the name of the GOB to be rendered
lights:	character string giving the name of the lighting GOB to use
camera:	character string giving the name of the camera to use

Discussion: This function causes the renderer(s) to render the given GOB. The given lighting GOB provides the illumination model used, and the given camera provides the view.

psphr

Purpose: Add a sphere primitive to the current GOB.

Use: psphr()

Parameters:

Discussion: This function adds a sphere primitive to the current GOB. The sphere is 1.0 units in radius and is centered at the origin. It is expected that the user will use transformations to move it to the desired location and scale it to the desired size.

pstatt

Purpose: Add an arbitrary string-valued attribute to the current GOB.

Use: pstatt(name, string)

Parameters:

name:	character string giving the attribute name
string:	character string to be associated with name

Discussion: This procedure can be used to add an arbitrary string attribute to a GOB.

pstcmp

Purpose: Set the current color map function.

Use: external mapfun pstcmp(min, max, mapfun)

Parameters:

min:	real value to be mapped to 0.0
max:	real value to be mapped to 1.0
mapfun:	user-supplied color mapping subroutine

Discussion:

When vertex lists of types containing scalar values (for example, PCVVX (4)) are rendered, the values are translated to colors by the current color map function. pstcmp allows the user to specify that function.

When translation occurs, the value is first rescaled into the interval from 0.0 to 1.0. If the value is beyond the range delimited by min and max, it maps to 0.0 if it is closer to min or 1.0 if it is closer to max. Otherwise, it maps to (value-min)/(max-min), which is in the range 0.0 to 1.0 inclusive. The mapped value is then passed to the subroutine mapfun, provided by the user, which is defined as follows:

```
subroutine mapfun( val, r, g, b, a )
real val, r, g, b, a
```

where val is the translated value. mapfun should then calculate red, green, blue, and opacity values, and store them in r, g, b, and a respectively. The subroutine can use any algorithm it wishes to do this, although fast algorithms are preferable as mapfun may be called many times. r, g, b, and a should be assigned values between 0.0 and 1.0 inclusive.

Vertex type PCVVVX contains two scalar values. Most primitives will ignore the second value and apply the color map to the first value. The dp_rand_isosurf function will use the first set of values associated with each vertex to produce its isosurface, and color the surface by interpolation of the second set of values.

ptext

Purpose: Add a text primitive to the current GOB.

Use: ptext(text, loc, u, v)

Parameters:

text:	character string giving text string of the primitive
loc:	real array (3) giving the point where the text is to begin
u:	real array (3) giving the writing direction vector
v:	real array (3) giving the text up direction vector

Discussion: This procedure adds a text primitive to the current GOB. The text is written starting at location, and running in the u direction such that vertical strokes run in the v direction. The height of text is controlled by the text-height attribute. The 'front' face of the text is determined by the cross product of u with v, so it is in the expected direction.

ptnsfm

Purpose: Add an arbitrary transformation to a GOB.

Use: ptnsfm(array)

Parameters:

array: real array(4,4) giving the transformation to be added

Discussion: This function adds an arbitrary transformation to the current GOB. This transformation will be concatenated with the existing transformation by left multiplication; GOBs with no existing transformation effectively have the identity transformation associated with them. See the discussion of transformations in the User's Guide for information on the format of the transformation. Transformations are stored in array in column order, such that a translation by 3 units in the X direction could be created as follows:

```
 real array(4,4)
 data array/1.0, 0.0, 0.0, 0.0,
$           0.0, 1.0, 0.0, 0.0,
$           0.0, 0.0, 1.0, 0.0,
$           3.0, 0.0, 0.0, 1.0/
```

ptorus

Purpose: Add a torus primitive to the current GOB.

Use: int ptorus(major, minor)

Parameters:

major: real major axis radius

minor: real minor axis radius

Discussion: This function adds a torus primitive to the current GOB. The torus lies in the x-y plane, centered at the origin. The two parameters control the major and minor radii.

ptrans

Purpose: Add a translation transformation to the current GOB.

Use: ptrans(x, y, z)

Parameters:

xshift:	real distance to translate in the X direction
yshift:	real distance to translate in the Y direction
zshift:	real distance to translate in the Z direction

Discussion: This function adds a translation to the current GOB. This transformation will be concatenated with the existing transformation by left multiplication; GOBs with no existing transformation effectively have the identity transformation associated with them. See the discussion of transformations in the User's Guide for information on the format of the transformation.

ptratt

Purpose: Add an arbitrary transformation-valued attribute to the current GOB.

Use: ptratt(name, trnsfm)

Parameters:

name:	character string giving the attribute name
transform:	real array(4,4) giving the transformation to be associated with name

Discussion: This procedure can be used to add an arbitrary transformation attribute to a GOB. Transformations are stored in array in column order, such that a translation by 3 units in the X direction could be created as follows:

```
      real array(4,4)
      data array/1.0, 0.0, 0.0, 0.0,
     $           0.0, 1.0, 0.0, 0.0,
```

```
     $          0.0, 0.0, 1.0, 0.0,
     $          3.0, 0.0, 0.0, 1.0/
```

ptrist

Purpose: Add a triangle strip primitive to the current GOB.

Use: ptrist(vtxtyp, ctype, npts, coords, colors, norms)

Parameters:

vtxtyp:	integer vertex type specifier
ctype:	integer color type (currently must be PRGB)
npts:	integer number of vertices
coords:	real array of coordinate data
colors:	real array of color data
norms:	real array of normal data

Discussion: This function adds a triangle strip to the current GOB. There will be nvertices-2 triangles in the triangle strip. The first is defined by vertices 0, 1, and 2, the second (if present) by vertices 1, 2, and 3, and so on. For a description of the vertex data format, see the section on vertex lists in this document.

ptxtht

Purpose: To set the text height attribute of the current GOB.

Use: ptxtht(height)

Parameters:

height:	real giving the character height of text primitives

Discussion: This procedure sets the height for text primitives, by setting the value of the text-height attribute. Since the aspect ratio of the characters is preserved, the length of the text string is also effected.

pvcatt

Purpose: Add an arbitrary vector-valued attribute to the current GOB.

Use: pvcatt(name, x, y, z)

Parameters:

name: character string giving the attribute name

x: real x component of vector to be associated with name

y: real y component of vector to be associated with name

z: real z component of vector to be associated with name

Discussion: This procedure can be used to add an arbitrary vector attribute to a GOB.

pzsurf

Purpose: Create a Z surface composite GOB

Use: pzsurf(vtype, zdata, vdata, nx, ny, crna, crnb, tfnull, tfun)

Parameters:

vtype: an integer vertex type specifier constant

zdata: an nx by ny array of reals specifying the z component

vdata: an nx by ny array of reals specifying values for coloring via the current color map

nx: an integer storing the number of vertices in the x direction

ny: an integer storing the number of vertices in the y direction

crna: real array (3) that specifies the starting corner crnb: real array (3) that specifies the ending corner

tfnull: an integer; if PTRUE, then include all points

tfun: a subroutine that excludes points from the Z surface.

Discussion:

This function draws a surface, rectangular in the x and y directions, the height of which is given by the data in the array zdata. Nx and ny specify the number of vertices in the x and y direction respectively.

If the vertex type requested is one which includes normal data, normals will be generated. This makes the surface look smoother, but increases the size of the model. If the vertex type includes value data, the user must supply a valdata array containing values for color mapping. Zdata and valdate are mapped from crna to crnb, so that the first point is at crna. From crna, the values are mapped in the x-direction, with increasing rows in the y direction. The z values of crna and crnb are ignored.

The user can cause points to be excluded from the surface by setting tfnull to PFALSE (0), and specifying a subroutine tfun to be used in determining which points to exclude. If tfnull is PTRUE (1), all points will be included in the surface and the parameter tfun will be ignored. Excluding a point causes a hole to appear in the surface where the point would have been.

Tfun is a user supplied subroutine that determines whether a point should be excluded from the zsurface. It should be defined as follows:

```
subroutine tfun( ixclde, value, i, j )
integer ixclude, i, j
real value
```

This function is called for each point in the surface, setting value to the z value at that point and i and j to the array indices (between 1 and nx and 1 and ny respectively) of the point in question. If on return from the function ixclde is PTRUE (1), the point will be excluded and a hole will be left in the surface at the appropriate point. If ixclde is PFALSE (0), the point will be included. This enables the user to remove points on the basis of position or value.