NAME

xgl_polygon – draws a general-case polygon

SYNOPSIS

```
#include <xgl/xgl.h>
void  xgl_polygon (
        Xgl_ctx          ctx,
        Xgl_facet_type   facet_type,
        Xgl_facet        *facet,
        Xgl_bbox         *bbox,
        Xgl_usgn32       num_pt_lists,
        Xgl_pt_list      pl[] );
```

DESCRIPTION

Purpose

This operator draws a single polygon according to the parameter list and the graphics state described in the Context *ctx*, which can be either a 2D or 3D Context.

A *general* polygon is defined by one or more point lists. Each point list defines a sequence of connected edges that together form an individual boundary. These boundaries, along with the filling rule specified by the **XGL_CTX_SURF_INTERIOR_RULE**(3) attribute, define the interior of the polygon.

For the boundary consisting of the *n* vertices *v(0)*, *v(1)*, ..., *v(n-1)*, the edges defined are those between vertices *v(0)* to *v(1)*, *v(1)* to *v(2)*, ..., *v(n-2)* to *v(n-1)*, and *v(n-1)* to *v(0)*. The resulting boundary can be self-intersecting or non-planar. The exact appearance of non-planar polygons when projected on the viewing surface is hardware-dependent.

Parameters

ctx The XGL 2D or 3D Context containing the graphics state used when processing **xgl_polygon**(3)**.**

facet_type

Is the data type used to define each facet in the list. *Xgl_facet_type* defines four enumerated values as possible facet data types:

XGL_FACET_NONE
XGL_FACET_COLOR
XGL_FACET_NORMAL
XGL_FACET_COLOR_NORMAL

facet A pointer to an *Xgl_facet* structure, which is defined as:

```
typedef union {
        Xgl_color_facet         color_facet;
        Xgl_normal_facet        normal_facet;
        Xgl_color_normal_facet  color_normal_facet;
} Xgl_facet;
```

The value of *facet_type* determines the interpretation of the union pointed to by *facet*. For example, if *facet_type* is XGL_FACET_COLOR_NORMAL, *facet→color_normal_facet* will contain the facet data for the polygon. If *facet_type* is XGL_FACET_NONE, *facet* can be NULL.

bbox Either NULL or is the bounding box around all of the points in *pl[]*. The bounding box information can be used to optimize clipping and also to simplify rendering when its projection on the view surface is small. (See **XGL_CTX_THRESHOLD**(3)**.**)

If the application does not want to provide bounding box information, *bbox* must be NULL. If the bounding box is given, it is the application's responsibility to ensure that it is correct.

In releases of XGL before 3.0, if an application provided a bounding box, then *bbox* was a pointer to a structure of type *Xgl_bbox*. In XGL 3.0 and subsequent releases, this approach is still supported for backward compatibility, but applications now can store bounding box information in any of six new structures, which are similar to *Xgl_bbox*:

Xgl_bbox_i2d
Xgl_bbox_f2d
Xgl_bbox_f3d
Xgl_bbox_status
Xgl_bbox_d2d
Xgl_bbox_d3d

In place of a structure of type *Xgl_bbox*, an application can use one of the above structures. For example, if an application uses a variable called *bbox_f2d* of type *Xgl_bbox_f2d*, it needs to cast the pointer to this structure using (Xgl_bbox *) (&bbox_f2d) before passing the information to XGL.

num_pt_lists
The number of elements in *pl[]*.

pl A pointer to the first element of an array of *Xgl_pt_list* structures, which is defined as:

```
typedef struct {
        Xgl_pt_type     pt_type;        /* type of point */
        Xgl_bbox        *bbox;          /* optional box containing
                                           all points */
        Xgl_usgn32      num_pts;        /* number of points */
        Xgl_usgn32      num_data_values; /* number of data values at
                                           each point */
        Xgl_pt_ptr_union pts;           /* union of pointers to array
                                           of points */
} Xgl_pt_list;
```

Each *pl[i]* specifies the vertices of a single boundary of the polygon.

pl[i].pt_type specifies the type of the points in *pl[i].pts*. Note that all of the *types* in the list pointed to by *pts* must be of this type.

pl[i].bbox specifies a bounding box for the *i'th* boundary. This bounding box is handled in the same way as *bbox* (above) is handled.

pl[i].num_pts is the number of points in the array *pts*. Specifically, it is the number of vertices in the *i'th* polygon boundary.

pl[i].num_data_values specifies the number of data values. Specifically, it is the number of data values per vertex in the *i'th* polygon boundary. This value is looked into only if the *pl[i].pt_type* is specified as having data values.

Texture mapping can be done by using vertices with data values. The data values, also known as *u* and *v* values for texturing, specify how the texture map should be mapped on a polygon. The coordinate space of the texture map that *u* values correspond to is a space that places 0.0 at left edge of the texture and 1.0 at the right edge of the texture. Similarly for *v*, the coordinate space of the texture map that *v* values use is a space that places 0.0 at the bottom edge of the texture and 1.0 at the top edge of the texture. An application can give several data values and can choose the specific data values to be used from the array by using the attributes **XGL_DMAP_TEXTURE_U_INDEX**(3) and **XGL_DMAP_TEXTURE_V_INDEX**(3)**.**

The data values are used only if the attributes **XGL_3D_CTX_SURF_FRONT_DMAP_-SWITCHES**(3) and **XGL_3D_CTX_SURF_BACK_DMAP_SWITCHES**(3) have at least one switch set to TRUE.

pl[i].pts is a union of pointers, one of which points to the array of points that make up the vertices of the *i'th* boundary. The value of *pl[i].pt_type* determines the interpretation of this union. Refer to the **xgl_pt_list**(3) man page for the valid point types.

If vertices with *flag* information are used, a bit within the flag word at each point determines if specific polygon boundary edges are drawn (1 means draw it). The bit can be manipulated using macros defined in the header file *xgl.h*. XGL_DRAW_-EDGE corresponds with the flag bit (0x1 in the flag word). The macro XGL_SET_-EDGE_FLAG(PT) can be used to set the flag when given a pointer to a point containing flag information. Likewise, a macro is available to clear the flag bit. For more information, see the include file *xgl.h*.

The edge flag bit is used only if the attribute **XGL_CTX_SURF_EDGE_FLAG**(3) is TRUE. It is interpreted as follows:

The XGL_DRAW_EDGE bit (0x1) is used to determine whether particular boundary edges are drawn. For the boundary vertices *v(0)*, *v(1)*, ..., *v(n)*, the flag on vertex *v(i)* determines whether the edge between vertices *v(i-1)* and *v(i)* is drawn. The flag on vertex *v(0)* applies to the edge between vertices *v(n)* and *v(0)*.

FACET RENDERING

Generally, a *facet* refers to a single polygon, which might be either a general or simple polygon. Several Context attributes specify how this facet is to be rendered. The vertex point type *pl[i].pt_type* and the facet type *facet_type* can provide sources of color in addition to the surface color Context attributes **XGL_CTX_SURF_FRONT_COLOR**(3) and **XGL_3D_-CTX_SURF_BACK_COLOR**(3)**.**

When an application supplies more than one source of colors for a polygon, it can select the source — Context, facet, or vertex — to be chosen for rendering by setting the Context attributes **XGL_CTX_SURF_FRONT_COLOR_SELECTOR**(3) and **XGL_3D_CTX_SURF_BACK_-COLOR_SELECTOR**(3)**.** In 3D, the final appearance of the surface depends on the illumination model specified with the attributes **XGL_3D_CTX_SURF_FRONT_ILLUMINATION**(3) and

XGL_3D_CTX_SURF_BACK_ILLUMINATION(3)**,** as well as the depth-cue mode given by the attribute **XGL_3D_CTX_DEPTH_CUE_MODE**(3)**.**

As part of the processing of a facet, XGL may need to determine the orientation of a facet: either front- or back-facing. These situations include enabling illumination, face culling, or face distinguishing. For these cases, an application benefits by supplying a unit normal vector with the facet data, which points in the direction of the front face of the polygon. This saves XGL from calculating it from the vertices with the method given by **XGL_3D_CTX_SURF_GEOM_NORMAL**(3)**.**

RETURN VALUE

Nothing is returned.

SEE ALSO

XGL_3D_CTX_DEPTH_CUE_COLOR(3)
XGL_3D_CTX_DEPTH_CUE_INTERP(3)
XGL_3D_CTX_DEPTH_CUE_MODE(3)
XGL_3D_CTX_DEPTH_CUE_REF_PLANES(3)
XGL_3D_CTX_DEPTH_CUE_SCALE_FACTORS(3)
XGL_3D_CTX_LIGHTS(3)
XGL_3D_CTX_LIGHT_NUM(3)
XGL_3D_CTX_LIGHT_SWITCHES(3)
XGL_3D_CTX_SURF_BACK_AMBIENT(3)
XGL_3D_CTX_SURF_BACK_COLOR(3)
XGL_3D_CTX_SURF_BACK_COLOR_SELECTOR(3)
XGL_3D_CTX_SURF_BACK_DIFFUSE(3)
XGL_3D_CTX_SURF_BACK_DMAP(3)
XGL_3D_CTX_SURF_BACK_DMAP_NUM(3)
XGL_3D_CTX_SURF_BACK_DMAP_SWITCHES(3)
XGL_3D_CTX_SURF_BACK_FILL_STYLE(3)
XGL_3D_CTX_SURF_BACK_FPAT(3)
XGL_3D_CTX_SURF_BACK_FPAT_POSITION(3)
XGL_3D_CTX_SURF_BACK_ILLUMINATION(3)
XGL_3D_CTX_SURF_BACK_LIGHT_COMPONENT(3)
XGL_3D_CTX_SURF_BACK_SPECULAR(3)
XGL_3D_CTX_SURF_BACK_SPECULAR_COLOR(3)
XGL_3D_CTX_SURF_BACK_SPECULAR_POWER(3)
XGL_3D_CTX_SURF_BACK_TRANSP(3)
XGL_3D_CTX_SURF_FACE_CULL(3)
XGL_3D_CTX_SURF_FACE_DISTINGUISH(3)
XGL_3D_CTX_SURF_FRONT_AMBIENT(3)
XGL_3D_CTX_SURF_FRONT_DIFFUSE(3)
XGL_3D_CTX_SURF_FRONT_DMAP(3)
XGL_3D_CTX_SURF_FRONT_DMAP_NUM(3)
XGL_3D_CTX_SURF_FRONT_DMAP_SWITCHES(3)
XGL_3D_CTX_SURF_FRONT_ILLUMINATION(3)
XGL_3D_CTX_SURF_FRONT_LIGHT_COMPONENT(3)
XGL_3D_CTX_SURF_FRONT_SPECULAR(3)

XGL_3D_CTX_SURF_FRONT_SPECULAR_COLOR(3)
XGL_3D_CTX_SURF_FRONT_SPECULAR_POWER(3)
XGL_3D_CTX_SURF_FRONT_TRANSP(3)
XGL_3D_CTX_SURF_GEOM_NORMAL(3)
XGL_3D_CTX_SURF_NORMAL_FLIP(3)
XGL_3D_CTX_SURF_SILHOUETTE_EDGE_FLAG(3)
XGL_3D_CTX_SURF_TRANSP_BLEND_EQ(3)
XGL_3D_CTX_SURF_TRANSP_METHOD(3)
XGL_CTX_EDGE_AA_BLEND_EQ(3)
XGL_CTX_EDGE_AA_FILTER_SHAPE(3)
XGL_CTX_EDGE_AA_FILTER_WIDTH(3)
XGL_CTX_EDGE_ALT_COLOR(3)
XGL_CTX_EDGE_CAP(3)
XGL_CTX_EDGE_COLOR(3)
XGL_CTX_EDGE_JOIN(3)
XGL_CTX_EDGE_MITER_LIMIT(3)
XGL_CTX_EDGE_PATTERN(3)
XGL_CTX_EDGE_STYLE(3)
XGL_CTX_EDGE_WIDTH_SCALE_FACTOR(3)
XGL_CTX_PLANE_MASK(3)
XGL_CTX_ROP(3)
XGL_CTX_SURF_AA_BLEND_EQ(3)
XGL_CTX_SURF_AA_FILTER_SHAPE(3)
XGL_CTX_SURF_AA_FILTER_WIDTH(3)
XGL_CTX_SURF_EDGE_FLAG(3)
XGL_CTX_SURF_FRONT_COLOR(3)
XGL_CTX_SURF_FRONT_COLOR_SELECTOR(3)
XGL_CTX_SURF_FRONT_FILL_STYLE(3)
XGL_CTX_SURF_FRONT_FPAT(3)
XGL_CTX_SURF_FRONT_FPAT_POSITION(3)
XGL_CTX_SURF_INTERIOR_RULE(3)
XGL_CTX_THRESHOLD(3)

NOTES

All points in *pl[]* must have the same Xgl_pt_type.

2D point types do not work with a 3D Context, and vice versa.

When filling the interior of polygons, XGL uses a technique that ensures adjoining polygons do not overlap. XGL assumes that pixels along the top and left edges of a polygon are *inside* the polygon, and those along the right and bottom edges are *outside*. Using this scheme, pixels along the common edge of two adjoining polygons are not drawn twice. However, it also prevents pixels from being drawn along the right and bottom edges of the clipping window. (Note that this is true only for polygons, triangle strips, triangle lists, NURB surfaces, and quadrilateral meshes. It is not true for lines, edges, stroke text, curves, and markers.)

Use **xgl_polygon**(3) to draw self-intersecting or multibounded polygons. For polygons with a simpler definition, use **xgl_multi_simple_polygon**(3)**.**