NAME

xgl_multi_simple_polygon – draws simple polygons from a list of data

SYNOPSIS

#include <xgl/xgl.h>

```
void  xgl_multi_simple_polygon (
          Xgl_ctx            ctx,
          Xgl_facet_flags    flags,
          Xgl_facet_list     *facets,
          Xgl_bbox           *bbox,
          Xgl_usgn32         num_pt_lists,
          Xgl_pt_list        pl[] );
```

DESCRIPTION

Purpose

This operator draws a set of simple polygons according to the parameter list and the graphics state described in the Context *ctx*, which may be either a 2D or 3D Context. All of the Context attributes that directly affect the display of polygons are listed in the **SEE ALSO** section below.

A *simple* polygon is a polygon that is described by a single list of *n* vertices *v(0)*, *v(1)*, ..., *v(n-1)*. These vertices form the edges *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 polygon can be self-intersecting or non-planar. Each polygon described by *pl[]* is drawn separately, starting at *pl[0]* and ending at *pl[num_pt_lists-1]*. This order of drawing is guaranteed.

Parameters

ctx The XGL 2D or 3D Context containing the graphics state used when processing xgl_multi_simple_polygon.

flags Contains bits that specify the kind of polygons being drawn. Xgl_facet_flags is specified as an enumerated type, but it actually contains several values OR'd together. The bits in flags apply to *all* polygons in *pl[]*.

Here are the available *Xgl_facet_flags*:

Bits for number of sides

XGL_FACET_FLAG_SIDES_ARE_3
XGL_FACET_FLAG_SIDES_ARE_4
XGL_FACET_FLAG_SIDES_UNSPECIFIED

At most, one of these can be set. If either XGL_FACET_FLAG_SIDES_ARE_3 or XGL_FACET_FLAG_SIDES_ARE_4 is set, each contour in *pl[]* is assumed to represent a triangle or quadrilateral, respectively. Specifically, if XGL_-FACET_FLAG_SIDES_ARE_3 is set, the first three points of each *pl[]* element are used to draw the polygon. Similarly, if XGL_FACET_FLAG_SIDES_ARE_4 is set, the first four points are used. If XGL_FACET_FLAG_SIDES_-UNSPECIFIED is set, no assumption is made regarding the number of sides.

It is not an error for either the triangles or quadrilaterals to be degenerate. It is an error if there are fewer points in *pl[i]* than specified by this flag. The default value is XGL_FACET_FLAG_SIDES_UNSPECIFIED. To select these bits, use *flags* and XGL_FACET_FLAG_SIDES_MASK.

Bits for convexity

XGL_FACET_FLAG_SHAPE_CONVEX
XGL_FACET_FLAG_SHAPE_NONCONVEX
XGL_FACET_FLAG_SHAPE_UNKNOWN

At most, one of these can be set. In the 3D case, a non-planar polygon is generally not convex when projected to the viewing plane; therefore, it must have XGL_FACET_FLAG_SHAPE_NONCONVEX set. If no information is known about the polygon, XGL_FACET_FLAG_SHAPE_UNKNOWN must be set. The default value is XGL_FACET_FLAG_SHAPE_UNKNOWN. To test these bits, use *flags* and XGL_FACET_FLAG_SHAPE_MASK.

Bits for facet normal and vertex order

XGL_FACET_FLAG_FN_CONSISTENT
XGL_FACET_FLAG_FN_NOT_CONSISTENT

Setting the XGL_FACET_FLAT_FN_CONSISTENT flag indicates whether the direction of facet normal and the order in which the points are given are consistent for all multisimple polygons.

In a right-handed coordinate system, the order of the points implies a facet normal that is calculated as follows:

implied facet normal = v1v2 × v1v3

A user-provided facet normal indicates the front face of a polygon. Call a group of polygons XGL_FACET_FLAG_FN_CONSISTENT if either of the following is true:

- The front of the polygon chosen by the user-provided facet normal always *agrees* with the front chosen by the implied facet normal.
- The front chosen by the user-provided facet normal always *disagrees* with that chosen by the implied facet normal.

Call a group of polygons XGL_FACET_FLAG_FN_NOT_CONSISTENT if neither of the above conditions holds. That is, for some polygons, there is agreement; for others, there is disagreement.

The default is XGL_FACET_FLAG_FN_NOT_CONSISTENT.

Setting this flag with XGL_FACET_FLAG_FN_CONSISTENT gives better performance on some devices. Consult your accelerator guide for details.

Bits for geometry data memory allocation

XGL_FACET_FLAG_DATA_CONTIG
XGL_FACET_FLAG_DATA_NOT_CONTIG

Setting the XGL_FACET_FLAG_DATA_CONTIG indicates that the geometry data being sent is contiguous in memory. With this information some rendering devices will be able to process the geometry data more efficiently.

When this attribute is set, the application guarantees that all geometry data is stored in one contiguous memory location. Setting XGL_FACET_-FLAG_DATA_NOT_CONTIG means the application cannot guarantee that the geometry data is contiguous in memory, resulting in slower performance on some rendering devices. To determine if your rendering device can take advantage of XGL_FACET_FLAG_DATA_CONTIG, consult the accelerator guide for your device.

The default is XGL_FACET_FLAG_DATA_NOT_CONTIG. To test this bit, use *flags* and XGL_FACET_FLAG_DATA_NOT_CONTIG.

facets A pointer to an *Xgl_facet_list* structure, which is defined as:

```
typedef struct {
        Xgl_facet_type          facet_type;        /* type of facet */
        Xgl_usgn32              num_facets;        /* number of facets *
        union {                                    /* pointers to array o
                Xgl_color_facet         *color_facets;
                Xgl_normal_facet        *normal_facets;
                Xgl_color_normal_facet  *color_normal_facets;
        } facets;
} Xgl_facet_list;
```

facets→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,

The value of this field determines the interpretation of the union *facets→facets*, which points to the first element of an array of facet information. For example, if *facets→facet_type* is XGL_FACET_COLOR, then *facets→facets.color[i]* should be the data for the *i'th* Xgl_color_facet.

This array of facet data is assumed to be *facets→num_facets* elements long.

The *facets* information is not used to draw the polygons if either *facets* is NULL, *facets→facet_type* is XGL_FACET_NONE, or *facets→num_facets* is less than *num_pt_lists*. Otherwise, the data in the *i'th* facet is used for the simple polygon whose boundary is described by *pl[i]*. Normal information supplied to 2D Contexts is ignored.

bbox Is either NULL or is the bounding box around all of the points in *pl[]*. The bounding box information is 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* can be NULL. If the bounding box is given, it is the application's responsibility to ensure 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 are 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 simple polygon.

pl[i].pt_type specifies the type of the points in *pl[i].pts*. Note that all points in *pl[]* must have the same *Xgl_pt_type.*

pl[i].bbox specifies a bounding box for the *i'th* polygon. If XGL_FACET_FLAG_-SIDES_ARE_3 or XGL_FACET_FLAG_SIDES_ARE_4 are set, this bounding box is ignored, since it takes longer to process the bounding box data than to clip the triangle or quadrilateral directly.

pl[i].num_pts is the number of points in the array *pts*. Specifically, it is the number of vertices in the *i'th* simple polygon. If XGL_FACET_FLAG_SIDES_ARE_3 or XGL_-FACET_FLAG_SIDES_ARE_4 are set, the number of vertices in each polygon is known. In that case, the value of *pl[i].num_pts* is ignored.

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* simple polygon. This value is looked into only if the *pl[i].pt_type* is specified as having data values. For more information on how the data values are used, see **xgl_polygon**(3).

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* simple polygon. The value of *pl[i].pt_type* determines the interpretation of this union.

For further details about this union of pointers and the effect of *pl[i].pt_type* and *facets→facet_type* on how facets are colored, see **xgl_polygon**(3). These details are the same for all primitives that draw polygons.

If vertices with *flag* information are used, the flag bits are interpreted as specified in **xgl_polygon**(3).

FACET COLORING

Generally, a *facet* refers to a single polygon, which might be either a general or simple polygon. The Context attributes specify how this facet is to be colored. The vertex point type *pl[i].pt_type* and the facet type *facet_type* also interact to affect the color of the polygon. The model is simple: The point and facet types determine how the normal and base colors of each vertex are obtained. Then the Context attributes are applied to color the facet. For more information about facet coloring, see **xgl_polygon**(3).

RETURN VALUE

Nothing is returned.

SEE ALSO

xgl_polygon(3)
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 will not work with a 3D Context, and vice versa.

Only XGL_FACET_NONE or XGL_FACET_COLOR should be used in *facets→facet_type* for 2D Contexts.

The exact appearance of non-planar polygons when projected on the viewing surface is hardware-dependent.

The results of rendering degenerate polygons is hardware-dependent.