NAME

xgl_multicircle – draws a list of circles

SYNOPSIS

```
#include <xgl/xgl.h>
void  xgl_multicircle (
            Xgl_ctx           ctx,
            Xgl_circle_list   *circle_list );
```

DESCRIPTION

Purpose

This operator draws a list of circles according to the graphics state described in the Context *ctx*. The center and radius of each circle in the list are transformed by the current transformation matrix (see **XGL_CTX_VIEW_TRANS**(3) and **XGL_CTX_MODEL_TRANS**(3)) and drawn in the plane of the view surface.

If the composite view transformation has unequal scaling along the *x* and *y* axes, each circle is drawn as an ellipse, in the case of 2D Contexts. When calling xgl_multicircle with a 3D Context, the circles are ''drawn'' in the plane described by the two direction vectors provided with each circle. The result is then projected using the current set of Transforms, so that all the standard 3D pipeline applies. In the case of 3D circles, the circles are approximated as polygons, using the following context attributes:

XGL_CTX_NURBS_CURVE_APPROX(3)
XGL_CTX_NURBS_CURVE_APPROX_VAL(3)
XGL_CTX_MAX_TESSELLATION(3)
XGL_CTX_MIN_TESSELLATION(3)

Parameters

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

circle_list

A pointer to the list of circles to draw. It is a structure defined as:

```
typedef struct {
        Xgl_usgn32              num_circles;   /* number of circles */
        Xgl_multicircle_type    type;          /* data type of multicircle */
        Xgl_bbox                *bbox;         /* optional box contain. all circles */
        union {                                /* union of pointers to circles */
                Xgl_circle_i2d    *i2d;
                Xgl_circle_f2d    *f2d;
                Xgl_circle_d2d    *d2d;
                Xgl_circle_f3d    *f3d;
                Xgl_circle_af3d   *af3d;
                Xgl_circle_d3d    *d3d;
                Xgl_circle_ad3d   *ad3d;
        } circles;
} Xgl_circle_list;
```

circle_list→num_circles is the number of circles in the list.

circle_list→type is the data type of the points used to specify the center and radius

of each circle in the list. Xgl_multicircle_type defines the following enumerated values as possible multicircle data types:

XGL_MULTICIRCLE_I2D
XGL_MULTICIRCLE_F2D
XGL_MULTICIRCLE_D2D
XGL_MULTICIRCLE_F3D
XGL_MULTICIRCLE_AF3D
XGL_MULTICIRCLE_D3D
XGL_MULTICIRCLE_AD3D

Ultimately, this determines which member of the union *circle_list→circles* (see below) points to the circle data.

circle_list→bbox points to a structure describing a bounding box for all of the circles in the list. This may be NULL to indicate that no bounding box information is available.

circle_list→circles is a union of pointers, one of which points to the array of structures that define each circle in the list. For example, the *circle_list→circles* structure for multicircles of type XGL_MULTICIRCLE_I2D looks as follows:

```
typedef struct {
        Xgl_pt_flag_i2d   center;   /* center point of circle */
        Xgl_usgn32        radius;   /* radius of circle */
} Xgl_circle_i2d;
```

The structure for multicircles of type XGL_MULTICIRCLE_F2D is similar except that the data types of the *center* and *radius* members of the structure become *Xgl_pt_flag_f2d* and *float*, respectively.

The *circle_list→circles* structure for multicircles of type XGL_MULTICIRCLE_F3D looks as follows:

```
typedef struct {
        Xgl_pt_flag_f3d   center;           /* center point of circle */
        Xgl_pt_f3d        dir[3];           /* dir. vectors and normal vector */
        Xgl_boolean       dir_normalized;   /* TRUE when dir[] are unit vectors */
        Xgl_boolean       dir_normal;       /* TRUE if dir[2] is a valid vector */
        Xgl_usgn32        radius;           /* radius of circle */
} Xgl_circle_f3d;
```

See **xgl_struct**(3) for structures defining other multicircles in the list.

The first two elements of the field *dir[3]* contain the two orthogonal vectors in model coordinates (MC), used to define the plane on which the circle is going to be built. The third element is used to specify a normal vector for the circle.

dir_normalized should be set to TRUE when the vectors are normalized (that is, unit vectors). *dir_normal* should be set to TRUE when the third element of *dir* is set to a valid vector. Otherwise, XGL computes a normal vector to the circle, using the two vectors specified for the circle plane.

When specifying type XGL_MULTICIRCLE_AF3D for multicircles, XGL generates *annotation circles*. In other words, the circles generated by XGL are guaranteed to be in a plane parallel to the projection plane. In this case, the center of each circle represents the reference point and is specified in model coordinates. The radius is specified in virtual device coordinates (VDC). There is no need to specify any direction vectors, and the primitive is always front-facing.

In all other cases, the *center* and *radius* are specified in the application's coordinate system.

Execution

Edges can be drawn around the circles. To do so, first set the attribute **XGL_CTX_SURF_EDGE_FLAG**(3) to TRUE. Edges are drawn around each circle within the circle list that has the flag associated with its *center* point set. Note that the *center* point of each circle always has a flag associated with it. XGL assumes the application will initialize these flags (to either **0** or **1**) before calling xgl_multicircle. If the flags are not initialized, unpredictable results may occur.

Circles act like 2D or 3D polygons (see **xgl_polygon**(3)) in that the following surface-fill attributes control the appearance of the enclosed area of each circle:

XGL_CTX_SURF_FRONT_FILL_STYLE(3)
XGL_CTX_SURF_FRONT_FPAT(3)
XGL_CTX_SURF_FRONT_FPAT_POSITION(3)
XGL_CTX_SURF_FRONT_COLOR(3)

In 3D, the following equivalent attributes for back-facing polygons also apply:

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_COLOR(3)

The lighting and shading attributes are also in effect.

RETURN VALUE

Nothing is returned.

SEE ALSO

xgl_object_get(3)
xgl_object_set(3)

XGL_CTX_NURBS_CURVE_APPROX(3)
XGL_CTX_NURBS_CURVE_APPROX_VAL(3)
XGL_CTX_MAX_TESSELLATION(3)
XGL_CTX_MIN_TESSELLATION(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_DC_OFFSET(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_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)

NOTES

The *center* point of each circle must have the flag associated with it set to either **0** or **1** (to turn the circle's edge either Off or On). If the flag is not initialized by the application, unpredictable results can occur.