NAME

xgl_multirectangle – draws a list of rectangles

SYNOPSIS

```
#include <xgl/xgl.h>
void  xgl_multirectangle (
            Xgl_ctx          ctx,
            Xgl_rect_list    *rect_list );
```

DESCRIPTION

Purpose

This operator draws a list of rectangles according to the graphics state described in the Context *ctx*. Each rectangle in the list is 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.

Parameters

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

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

```
typedef struct {
        Xgl_usgn32          num_rects;   /* number of rectangles */
        Xgl_multirect_type  rect_type;   /* data type of multirectangle */
        Xgl_bbox            *bbox;       /* optional box contain. all rectangles */
        union {                          /* union of pointers to rectangles */
                Xgl_rect_i2d    *i2d;
                Xgl_rect_f2d    *f2d;
                Xgl_rect_d2d    *d2d;
                Xgl_rect_f3d    *f3d;
                Xgl_rect_af3d   *af3d;
                Xgl_rect_d3d    *d3d;
                Xgl_rect_ad3d   *ad3d;
        } rects;
} Xgl_rect_list;
```

rect_list→num_rects is the number of rectangles in the list.

rect_list→type is the data type of the points used to specify two diagonal corners for each rectangle in the list. Xgl_multirect_type defines the following enumerated values as possible multirectangle data types:

XGL_MULTIRECT_I2D
XGL_MULTIRECT_F2D
XGL_MULTIRECT_D2D
XGL_MULTIRECT_F3D
XGL_MULTIRECT_AF3D
XGL_MULTIRECT_D3D
XGL_MULTIRECT_AD3D

Ultimately, this determines which member of the union *rect_list→rects* (see below) points to the rectangle data.

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

rect_list→rects is a union of pointers, one of which points to the array of structures that defines each rectangle in the list. For example, the *rect_list→rects* structure for multirectangles of type XGL_MULTIRECT_I2D looks as follows:

```
typedef struct {
        Xgl_pt_flag_i2d   corner_min;   /* diagonal corners of rectangle */
        Xgl_pt_i2d        corner_max;
} Xgl_rect_i2d;
```

The structure for multirectangles of type XGL_MULTIRECT_F2D is similar, except that the data types of the two corner points become *Xgl_pt_flag_f2d* and *Xgl_pt_f2d*, respectively.

The *rect_list→rects* structure for multirectangles of type XGL_MULTIRECT_F3D looks as follows:

```
typedef struct {
        Xgl_pt_flag_f3d   corner_min;       /* reference corner of rectangle */
        Xgl_pt_f2d        corner_max;       /* width and height in MC */
        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_rect_f3d;
```

The first two elements of the field *dir* contain the two orthogonal vectors in model coordinates (MC), used to define the plane on which the rectangle is going to be built, using *corner_min* as a reference point. The third element is used to specify a normal vector for the rectangle.

Unlike the 2D case, *corner_max* represents the width and height of the rectangle in MC.

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 rectangle, using the two vectors specified for the rectangle plane.

When specifying type XGL_MULTIRECT_AF3D for multirectangles, XGL generates *annotation rectangles*. In other words, the rectangles generated by XGL are guaranteed to be in a plane that is parallel to the projection plane. In this particular case, the reference corner of each rectangle is specified in model coordinates, and the other corner represents the virtual device coordinates (VDC) dimensions in *x* and *y* of the rectangle. There is no need to specify any direction vectors, and the primitive is always front-facing.

Except when rendering annotation rectangles, both corner points are specified in the application's coordinate system with the *corner_min* point defining the corner

at (*x*-min, *y*-min) and the *corner_max* point the one at (*x*-max, *y*-max). For example, in a standard Cartesian coordinate system (*x* increasing to the right and *y* increasing upward), *corner_min* would be the lower left corner of the rectangle, and *corner_max* would be the upper right corner.

Execution

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

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

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

This operator fills rectangles in an X11-compliant fashion where for filled rectangles, the lower-right row of pixels within the rectangle will not be rendered.