Line Patterns 13

This chapter discusses the Line Pattern object and its associated attributes. It includes information on the following topics:

- Using the line patterns supplied with the XGL library
- Creating new line patterns
- Setting the pattern for surface edges

Introduction to the Line Pattern Object

Line primitives (polylines and B-spline curves) are characterized by their line style and color. Line primitives can be rendered as solid lines, the default line style, or as patterns of on-off segments known as *line patterns*. To define the line pattern of a line or curve, the application can use line patterns supplied by XGL, or it can create its own line patterns using the Line Pattern object. XGL-supplied line patterns are predefined and are available for application use at any time. Line patterns defined by the Line Pattern object are used for rendering when the Line Pattern object is attached to a Context object.

Line patterns are also used to define the appearance of the edges of surface primitives. The default style for a surface edge is solid. The edge pattern can be changed by setting it to one of the XGL-supplied line patterns or to the pattern defined in a Line Pattern object.

Patterns for lines and surface edges can be rendered in a single color or in two colors that alternate by line segments.

13

Using the Predefined Line Patterns

XGL supplies ten line patterns for applications to use. These predefined line patterns are Line Pattern objects that the System State object creates when XGL is initialized. Predefined Line Pattern objects are read-only objects that cannot be modified or destroyed. The predefined line patterns are shown in Figure 13-1.

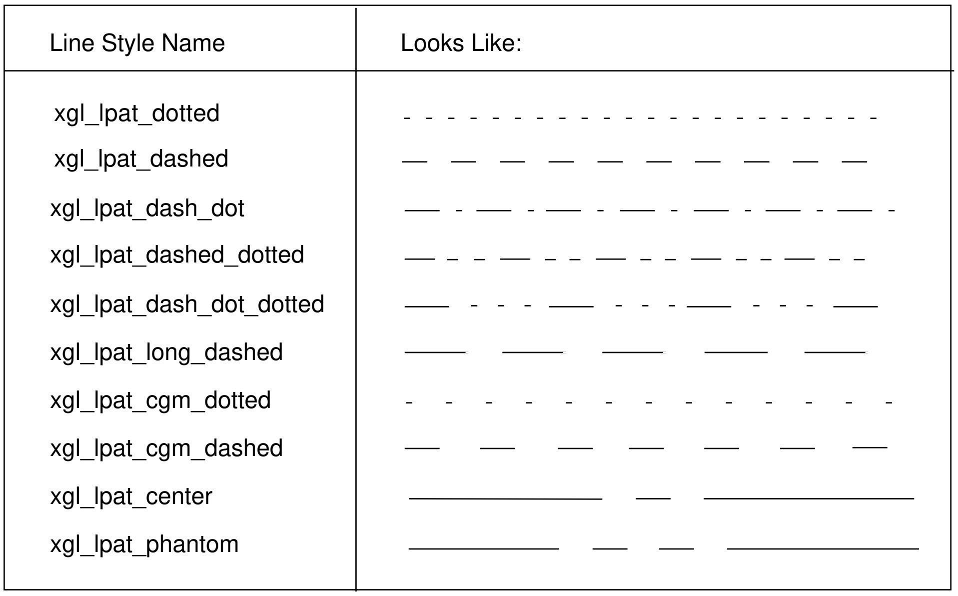

Figure 13-1 Predefined Line Patterns

To use a predefined line pattern, the application sets the Context attribute `XGL_CTX_LINE_STYLE` to `XGL_LINE_PATTERNED` and sets `XGL_CTX_LINE_PATTERN` to the pattern name, as in the following example:

```
xgl_object_set(ctx, XGL_CTX_LINE_STYLE, XGL_LINE_PATTERNED,
                XGL_CTX_LINE_PATTERN, xgl_lpat_dashed, NULL);
```

Similarly, to specify patterned surface edges, the application must set the Context attribute XGL_CTX_EDGE_STYLE to XGL_LINE_PATTERNED and set XGL_CTX_EDGE_PATTERN to the line pattern name, as in the following example:

```
xgl_object_set(ctx, XGL_CTX_EDGE_STYLE, XGL_LINE_PATTERNED,
                   XGL_CTX_EDGE_PATTERN, xgl_lpat_dashed, NULL);
```

The default value for the attributes XGL_CTX_LINE_PATTERN or XGL_CTX_EDGE_PATTERN is NULL.

Creating a Line Pattern Object

A Line Pattern object is created with the xgl_object_create() operator, using XGL_LPAT as the *type* parameter value. The data for the line pattern is attached to the Line Pattern object with the XGL_LPAT_DATA attribute, as in the following example:

```
lpat = xgl_object_create(sys_st, XGL_LPAT, NULL,
                             XGL_LPAT_DATA_SIZE, 4,
                             XGL_LPAT_DATA, lpatdata,
                             NULL);
```

Defining the pattern lpatdata for the new Line Pattern object is discussed in the section below.

To render lines, curves, or surface edges in the new pattern, the Line Pattern object must be attached to a Context object using the XGL_CTX_LINE_PATTERN attribute for lines and curves or the XGL_CTX_EDGE_PATTERN attribute for surface edges.

```
xgl_object_set(ctx, XGL_CTX_LINE_PATTERN, lpat, NULL);
```

Defining a New Line Pattern

A line pattern is defined in a line segment data array that is associated with the Line Pattern object. The line segment data array stores line segment lengths of alternating *on* and *off* pattern segments. If the pattern has an even number of line segments, the segments are used cyclically; when the end of a pattern is reached (that is, when the end of the array is reached), the pattern starts over again from the beginning. If the pattern has an odd number of line segments, XGL concatenates two repetitions of the pattern so that an even-length line pattern is created, inverts the *on* and *off* segments for the second half of the concatenated pattern, and then uses this pattern as the line pattern.

The first segment of a line pattern is assumed to be an *on* segment. The pattern wraps around along lines, curves, and corners until the end of the line is reached. The code fragment below illustrates a line pattern data array with four segments.

```
float           lpatdata[4];

lpatdata[0] = 1.0; lpatdata[1] = 3.0;
lpatdata[2] = 10.0; lpatdata[3] = 5.0;
```

Figure 13-2 illustrates line patterns with even and odd numbers of line segments.

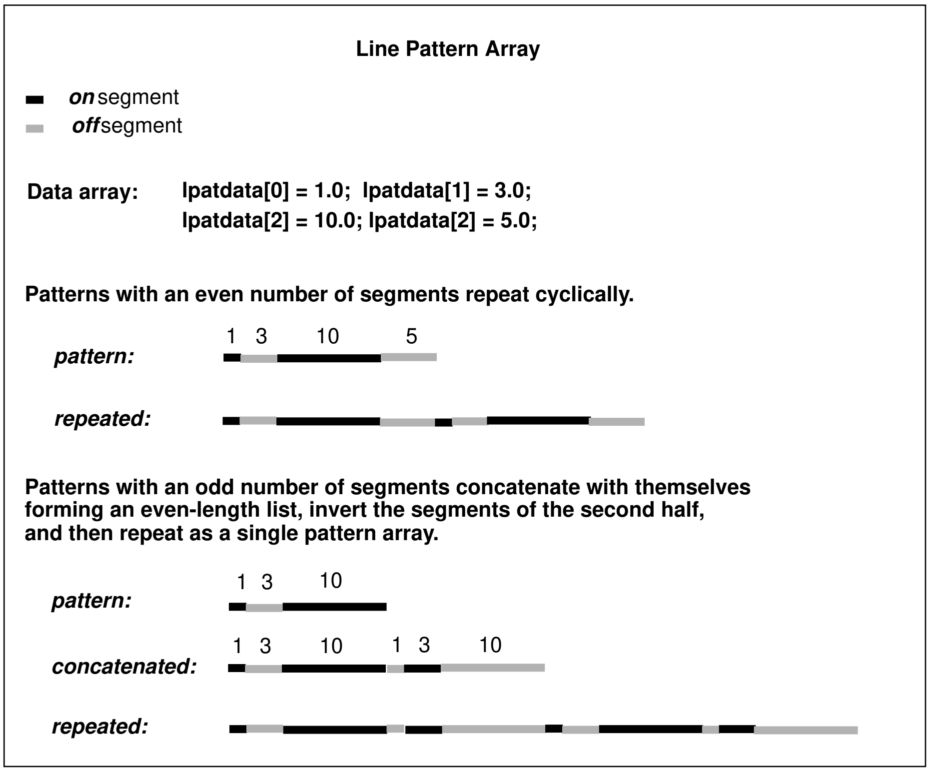

Figure 13-2 Line Pattern Array Formation

Line Pattern Attributes

The pattern of a line or edge pattern is defined with Line Pattern attributes. The Line Pattern attribute `XGL_LPAT_DATA` defines the on and off segments that comprise a line pattern. The on and off segments are stored as an array of either 32-bit integers or floating-point numbers. The data type used to specify the segments is set with `XGL_LPAT_DATA_TYPE`; the values for `XGL_LPAT_DATA_TYPE` are `XGL_DATA_INT` or `XGL_DATA_FLT`. The default data type is float.

The length of the line segment data array is specified by the `XGL_LPAT_DATA_SIZE` attribute, which must be set before passing the data. The default value for `XGL_LPAT_DATA_SIZE` is 0.

The offset into the line segment data array where the line pattern will begin is specified with the XGL_LPAT_OFFSET attribute. The offset into the line pattern is used only at the beginning of the patterned line being rendered. It is interpreted as the distance into the line pattern at which the pattern will be started. The application should ensure that this attribute is less than the size of the data array specified by XGL_LPAT_DATA_SIZE. The default value for XGL_LPAT_OFFSET is 0.

The way in which a line pattern is applied to a polyline can be changed with the attribute XGL_LPAT_STYLE. This attribute has the following values:

XGL_LPAT_FIXED_OFFSET
: The line pattern is applied with the offset defined by XGL_LPAT_OFFSET. The pattern carries over from one line segment to the next. This is the default behavior.

XGL_LPAT_BALANCED_SEGMENT
: For each segment of a polyline, the line pattern is balanced around the midpoint of the segment before clipping. In this case, the line pattern does not carry over to the adjacent segment; the balancing is independent of the adjacent segments. This attribute is only valid with polylines and does not apply to surface edges.

If XGL_LPAT_STYLE is set to XGL_LPAT_BALANCED_SEGMENT, the application can specify whether to balance the line pattern around the first or second dash using the attribute XGL_LPAT_BALANCED_DASH. This attribute enables the application to center the first dash of the pattern at the midpoint of the segment (XGL_LPAT_BAL_DASH_0) or center the second dash of the pattern at the midpoint of the segment (XGL_LPAT_BAL_DASH_1), as shown in the example in Figure 13-3. XGL_LPAT_BAL_DASH_0 is the default value. The midpoint of a line segment is calculated in device coordinates.

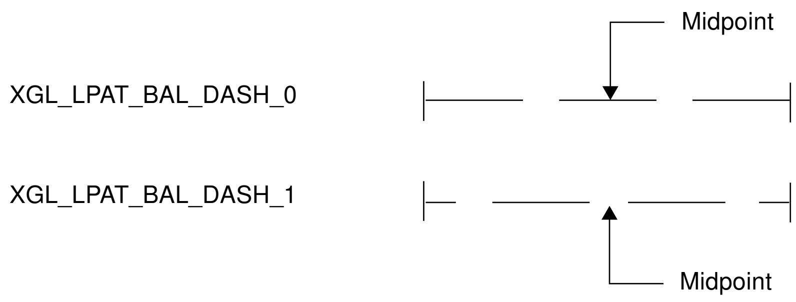

Figure 13-3 Balancing Line Patterns around a Line Segment Midpoint

Context Line Pattern Attributes

The appearance of line and edge patterns when rendered is controlled by Context attributes. The line and edge Context attributes related to the line pattern rendering are discussed below. For more information on the Context line pattern rendering attributes, see the appropriate man pages.

Line Rendering with a Line Pattern

The attribute `XGL_CTX_LINE_STYLE` defines the way in which lines or curves are drawn in a given Context. To render a patterned line, the application must set this attribute to one of the following values:

`XGL_LINE_PATTERNED`
: Lines are patterned using the Line Pattern object given by `XGL_CTX_LINE_PATTERN`, and color is specified by the `XGL_CTX_LINE_COLOR` attribute.

`XGL_LINE_ALT_PATTERNED`
: Lines are patterned using the Line Pattern object given by attribute `XGL_CTX_LINE_PATTERN`. The color is specified by the attributes `XGL_CTX_LINE_COLOR` for odd line segments and `XGL_CTX_LINE_ALT_COLOR` for even line segments.

`XGL_LINE_SOLID`
: Lines are not patterned. A solid line is drawn in the color specified by the `XGL_CTX_LINE_COLOR` attribute.

The default value for the attribute XGL_CTX_LINE_STYLE is XGL_LINE_SOLID.

When the attribute XGL_CTX_LINE_STYLE is set to XGL_LINE_PATTERNED or XGL_LINE_ALT_PATTERNED, the Context attribute XGL_CTX_LINE_PATTERN specifies the line pattern. The attribute XGL_CTX_LINE_PATTERN attaches a Line Pattern object to a Context.

Setting Line Color

The attribute XGL_CTX_LINE_COLOR sets the color of lines and curves when rendering line segments. If the data supplied with polylines has color for the individual vertices, XGL applies these colors to the vertices. If the data does not have color for the vertices, XGL uses the Context line color. The application can use the XGL_CTX_LINE_COLOR_SELECTOR attribute to control whether Context color or vertex color is used for lines with color data at the vertices. XGL_CTX_LINE_COLOR_SELECTOR has the following values:

XGL_LINE_COLOR_CONTEXT
: Line color is rendered as defined by XGL_CTX_LINE_COLOR. Using the Context color to display lines with vertex color may be useful for echoing a picking response. Depth cueing affects the final appearance of the line in 3D.

XGL_LINE_COLOR_VERTEX
: Line color is taken from the vertex data if the vertex data includes color data for individual vertices. If the vertex data does not specify colors for individual vertices, the color defined by XGL_CTX_LINE_COLOR.

The color type (RGB or indexed) must match the color type set by the XGL_RAS_COLOR_TYPE attribute of the Raster associated with the Context. The default value is a color of index 1 for indexed devices, or green for RGB devices. When specifying a color of type XGL_COLOR_INDEX in a 3D Context with depth cueing computations enabled, the color table of the appropriate Color Map object must have consistent color ramps to ensure a correct rendering.

The attribute XGL_CTX_LINE_ALT_COLOR sets the color to fill even (or alternate) line segments when the line style (XGL_CTX_LINE_STYLE) is set to XGL_CTX_LINE_ALT_PATTERNED. The odd line segments are filled using the color specified by XGL_CTX_LINE_COLOR. The color type (RGB or indexed) must match the color type set by the XGL_RAS_COLOR_TYPE attribute of the

Raster associated with the Context. The default value for the attribute XGL_CTX_LINE_ALT_COLOR is a color of index 0 for indexed devices, or black for RGB devices.

Edge Rendering with a Line Pattern

Edges are drawn around the perimeter of surfaces created by primitives capable of edge rendering (such as a polygon). Edges are drawn if the surface edge attribute XGL_CTX_SURF_EDGE_FLAG is set to TRUE. Primitives that accept point data that includes edge flag information can use the edge flag information to turn on or off the rendering of edge boundaries, but this is only applicable if the global edge flag attribute XGL_CTX_SURF_EDGE_FLAG is set to TRUE. The boundaries produced by clipping are not drawn.

The attributes specifying patterned edges are similar to those specifying patterned lines. The attribute XGL_CTX_EDGE_STYLE defines the edge style used for edges of surfaces. The values for this attribute are the same as the values for XGL_CTX_LINE_STYLE:

XGL_LINE_PATTERNED
: Edges are patterned using the Line Pattern object given by XGL_CTX_EDGE_PATTERN. Color is specified by the XGL_CTX_EDGE_COLOR attribute.

XGL_LINE_ALT_PATTERNED
: Edges are patterned using the Line Pattern object given by XGL_CTX_EDGE_PATTERN. The color is specified by the attributes XGL_CTX_EDGE_COLOR for odd line segments and XGL_CTX_EDGE_ALT_COLOR for even line segments.

XGL_LINE_SOLID
: Edges are not patterned. A solid edge is drawn in the color specified by the XGL_CTX_EDGE_COLOR attribute.

The default value for the attribute XGL_CTX_EDGE_STYLE is XGL_LINE_SOLID.

When XGL_CTX_EDGE_STYLE is set to XGL_LINE_PATTERNED or XGL_LINE_ALT_PATTERNED, the attribute XGL_CTX_EDGE_PATTERN attaches a Line Pattern object that specifies the pattern used for edge drawing to a Context. The default value for the attribute XGL_CTX_EDGE_PATTERN is NULL.

Setting Edge Color

The attribute XGL_CTX_EDGE_COLOR sets the color of the surface edges. The color type (RGB or indexed) must match the color type set by the XGL_RAS_COLOR_TYPE attribute of the Raster associated with the Context.

When specifying a color of type XGL_COLOR_INDEX in a 3D Context with depth cueing computations enabled, the color table of the appropriate Color Map object must have consistent color ramps to ensure a correct rendering. The default value is a color of type XGL_COLOR_INDEX with a value of 1.

The attribute XGL_CTX_EDGE_ALT_COLOR sets the color to fill even (or alternate) line segments when the edge style (XGL_CTX_EDGE_STYLE) is set to XGL_CTX_LINE_ALT_PATTERNED. The odd edge segments are filled using the color specified by XGL_CTX_EDGE_COLOR. The color type (RGB or indexed) must match the color type set by the XGL_RAS_COLOR_TYPE attribute of the Raster associated with the Context. The default value is a color of type XGL_COLOR_INDEX with a value of 0.

Line Pattern Examples

The following example programs use the Line Pattern object to create patterned lines and patterned edges. Each example is a fragment of a larger program. The complete program includes ex_utils.c and lpat_main.c, both of which are listed in Appendix B, as well as the two examples listed in this chapter. To compile the complete program, type make lpat in the example program directory. The compiled program allows you to cycle through both line pattern example programs.

Patterned Line Example

Example lpat_lines.c creates three patterns and draws 10 lines for each pattern. The output is shown on Plate 6.

Code Example 13-1 Line Pattern Example

```
/*
 * lpat_lines.c
 */
#include <xview/xview.h>
#include <xgl/xgl.h>
```

```
#include "ex.h"

/*
 * an easy to use macro for setting up an Xgl point list data
 * structure
 */

#define  XGL_SET_PT_LIST_I2D(PL, PT_TYPE, BBOX, NUM_PTS, PTS_I2D)\
          PL.pt_type = PT_TYPE;  \
          PL.bbox = BBOX;        \
          PL.num_pts = NUM_PTS;  \
          PL.pts.i2d = PTS_I2D;

/*
 * this routine draws 10 lines for each pattern in order to
 * emphasize what the pattern looks like...
 */
void
lpat_lines (Xgl_object     ctx)
{
    Xgl_sgn32            i;
    Xgl_color            ln_fg_color,
                         ln_bg_color;
    Xgl_pt_i2d           pts_i2d[2];
    Xgl_pt_list          pl_2d;

    /*
     * set to dashed line pattern object, set the line style to
     * patterned, set the line foreground pixel color (bits
     * in the pattern that have the value of 1 will be MAGENTA),
     * copy the point data into the Xgl data structure,
     * and then draw the lines
     */
    ln_fg_color = magenta_color;
    xgl_object_set (ctx,
                  XGL_CTX_LINE_PATTERN, xgl_lpat_dashed,
                  XGL_CTX_LINE_STYLE, XGL_LINE_PATTERNED,
                  XGL_CTX_LINE_COLOR, &ln_fg_color,
                  NULL);
    XGL_SET_PT_LIST_I2D (pl_2d, XGL_PT_I2D, NULL, 2, pts_i2d);

    pts_i2d[0].x = 10;
    pts_i2d[0].y = 10;
    pts_i2d[1].x = 410;
    pts_i2d[1].y = 10;
```

```
for (i = 0; i < 10; i++) {
       xgl_multipolyline (ctx, NULL, 1, &pl_2d);
       pts_i2d[0].y++;
       pts_i2d[1].y++;
}

/*
 * set to dotted line pattern object, set the line color
 * to cyan (bits in the pattern that have the value of 1
 * will be CYAN), set line style to alternate line pattern,
 * set the line alternate color to green (bits in the
 * pattern that have a value of 0 will be BLUE),
 * and then draw the lines
 */
ln_fg_color = cyan_color;
ln_bg_color = blue_color;
xgl_object_set (ctx,
              XGL_CTX_LINE_PATTERN, xgl_lpat_dotted,
              XGL_CTX_LINE_STYLE, XGL_LINE_ALT_PATTERNED,
              XGL_CTX_LINE_COLOR, &ln_fg_color,
              XGL_CTX_LINE_ALT_COLOR, &ln_bg_color,
              NULL);

pts_i2d[0].x = 10;
pts_i2d[0].y = 25;
pts_i2d[1].x = 410;
pts_i2d[1].y = 25;
for (i = 0; i < 10; i++) {
       xgl_multipolyline (ctx, NULL, 1, &pl_2d);
       pts_i2d[0].y++;
       pts_i2d[1].y++;
}

/*
 * set to Xgl dashed-dotted line pattern object, set
 * the line color to white, the alternate line color to
 * red and then draw the lines
 * NOTE: still doing alternate line colors
 */
ln_fg_color = white_color;
ln_bg_color = red_color;
xgl_object_set (ctx,
              XGL_CTX_LINE_PATTERN, xgl_lpat_dashed_dotted,
              XGL_CTX_LINE_COLOR, &ln_fg_color,
              XGL_CTX_LINE_ALT_COLOR, &ln_bg_color,
              NULL);
```

```
pts_i2d[0].x = 10;
pts_i2d[0].y = 40;
pts_i2d[1].x = 410;
pts_i2d[1].y = 40;
for (i = 0; i < 10; i++) {
        xgl_multipolyline (ctx, NULL, 1, &pl_2d);
        pts_i2d[0].y++;
        pts_i2d[1].y++;
}

/*
 * set our own dashed line pattern object, set the line
 * style to patterned, set the line foreground pixel color
 * (bits in the pattern with the value of 1 will be WHITE),
 * copy the point data into the Xgl data structure,
 * and then draw the lines
 */
ln_fg_color = white_color;
xgl_object_set (ctx,
               XGL_CTX_LINE_STYLE, XGL_LINE_PATTERNED,
               XGL_CTX_LINE_COLOR, &ln_fg_color,
               NULL);

/*
 * create our own line pattern and draw a line with it
 */
{
        Xgl_sgn32            i;
        Xgl_object           lpat;
        float                lpatdata[6];

        lpatdata[0] = lpatdata[1] = 2.;
        lpatdata[2] = lpatdata[3] = 4.;
        lpatdata[4] = lpatdata[5] = 1.;

        lpat = xgl_object_create (sys_st, XGL_LPAT, NULL,
                   XGL_LPAT_DATA_SIZE, 6,
                   XGL_LPAT_DATA, lpatdata,
                   NULL);

        xgl_object_set (ctx, XGL_CTX_LINE_PATTERN, lpat, NULL);

        pts_i2d[0].x = 10;
        pts_i2d[0].y = 55;
        pts_i2d[1].x = 410;
```

```
            pts_i2d[1].y = 55;
            for (i = 0; i < 10; i++) {
                xgl_multipolyline (ctx, NULL, 1, &pl_2d);
                pts_i2d[0].y++;
                pts_i2d[1].y++;
            }

            xgl_object_set (ctx,
                   XGL_CTX_LINE_STYLE, XGL_LINE_SOLID,
                   NULL);
            xgl_object_destroy (lpat);
      }
}
```

Line Pattern Polygon Edge Example

Example `lpat_pgon_edges.c` uses the predefined and custom-patterned lines shown in the previous example as polygon edges. The output is shown on Plate 6.

Code Example 13-2 Edge Pattern Example

```
/*
 * lpat_pgon_edges.c
 */

#include <xview/xview.h>
#include <xgl/xgl.h>

#include "ex.h"

extern Xgl_pt_i2d           pts_i2d_pgon1[];
extern Xgl_pt_i2d           pts_i2d_pgon2[];
extern Xgl_pt_i2d           pts_i2d_pgon3[];
extern Xgl_pt_i2d           pts_i2d_pgon4[];

/*
 * an easy to use macro for setting up an Xgl point list data
structure
 */
#define XGL_SET_PT_LIST_I2D(PL, PT_TYPE, BBOX, NUM_PTS, PTS_I2D)\
          PL.pt_type = PT_TYPE;  \
          PL.bbox = BBOX;        \
          PL.num_pts = NUM_PTS;  \
```

```
            PL.pts.i2d = PTS_I2D;

void
lpat_pgon_edges (Xgl_object   ctx)
{
    Xgl_color             ln_fg_color,
                          ln_bg_color,
                          pgon_front_color;
    Xgl_pt_list           pl_2d;

    /*
     * turn those edges on so we can see our patterned edges
     */
    xgl_object_set (ctx, XGL_CTX_SURF_EDGE_FLAG, TRUE, NULL);

    /*
     * set to dashed line pattern object, set the line style to
     * patterned, set the line foreground pixel color (bits
     * in the pattern with the value of 1 will be MAGENTA),
     * copy the point data into the Xgl data structure,
     * and then draw the line
     */
    ln_fg_color = magenta_color;
    pgon_front_color = green_color;
    xgl_object_set (ctx,
                  XGL_CTX_EDGE_PATTERN, xgl_lpat_dashed,
                  XGL_CTX_EDGE_STYLE, XGL_LINE_PATTERNED,
                  XGL_CTX_EDGE_COLOR, &ln_fg_color,
                  XGL_CTX_SURF_FRONT_COLOR, &pgon_front_color,
                  NULL);

    XGL_SET_PT_LIST_I2D (pl_2d, XGL_PT_I2D, NULL, 6,
                          pts_i2d_pgon1);
    xgl_polygon (ctx, XGL_FACET_NONE, 0, 0, 1, &pl_2d);

    /*
     * set to dotted line pattern object, set the line color
     * to cyan (bits in the pattern with the value of 1
     * will be CYAN), set line style to alternate line pattern,
     * set the line alternate color to green (bits in the
     * pattern with a value of 0 will be BLUE),
     * and then draw the line
     */
    ln_fg_color = cyan_color;
    ln_bg_color = blue_color;
    pgon_front_color = white_color;
```

```
xgl_object_set (ctx,
              XGL_CTX_EDGE_PATTERN, xgl_lpat_dotted,
              XGL_CTX_EDGE_STYLE, XGL_LINE_ALT_PATTERNED,
              XGL_CTX_EDGE_COLOR, &ln_fg_color,
              XGL_CTX_EDGE_ALT_COLOR, &ln_bg_color,
              XGL_CTX_SURF_FRONT_COLOR, &pgon_front_color,
              NULL);

pl_2d.pts.i2d = pts_i2d_pgon2;
xgl_polygon (ctx, XGL_FACET_NONE, 0, 0, 1, &pl_2d);

/*
 * set to Xgl dashed-dotted line pattern object,
 * set the line color to white, the alternate
 * line color to red, and then draw the line
 * NOTE: still doing alternate line colors
 */
ln_fg_color = white_color;
ln_bg_color = red_color;
pgon_front_color = blue_color;
xgl_object_set (ctx,
              XGL_CTX_EDGE_PATTERN, xgl_lpat_dashed_dotted,
              XGL_CTX_EDGE_COLOR, &ln_fg_color,
              XGL_CTX_EDGE_ALT_COLOR, &ln_bg_color,
              XGL_CTX_SURF_FRONT_COLOR, &pgon_front_color,
              NULL);

pl_2d.pts.i2d = pts_i2d_pgon3;
xgl_polygon (ctx, XGL_FACET_NONE, 0, 0, 1, &pl_2d);

/*
 * set our own dashed line pattern object, set the
 * line style to patterned, set the line foreground
 * pixel color ( bits in the pattern with the value
 * of 1 will be WHITE), copy the point data into
 * the Xgl data structure, and then draw the line
 */
ln_fg_color = white_color;
pgon_front_color = yellow_color;
xgl_object_set (ctx,
              XGL_CTX_EDGE_STYLE, XGL_LINE_PATTERNED,
              XGL_CTX_EDGE_COLOR, &ln_fg_color,
              XGL_CTX_SURF_FRONT_COLOR, &pgon_front_color,
              NULL);
pl_2d.pts.i2d = pts_i2d_pgon4;
```

```
/*
 * loop along changing the offset into the line pattern and
 * draw the line
 */
{
        Xgl_sgn32               i;
        Xgl_object              lpat;
        float                   lpatdata[6];

        lpatdata[0] = lpatdata[1] = 2.;
        lpatdata[2] = lpatdata[3] = 4.;
        lpatdata[4] = lpatdata[5] = 1.;

        lpat = xgl_object_create (sys_st, XGL_LPAT, NULL,
                    XGL_LPAT_DATA_SIZE, 6,
                    XGL_LPAT_DATA, lpatdata,
                    NULL);

        xgl_object_set (ctx, XGL_CTX_EDGE_PATTERN, lpat, NULL);

        xgl_polygon (ctx, XGL_FACET_NONE, 0, 0, 1, &pl_2d);

        xgl_object_set (ctx,
                XGL_CTX_EDGE_STYLE, XGL_LINE_SOLID,
                NULL);

        xgl_object_destroy (lpat);
    }
}
```