Markers 14

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

- Using the markers predefined by the XGL library
- Creating new markers
- Example programs

Introduction to the Marker Object

A marker is a two-dimensional symbol that represents a location in 2D or 3D space. For example, when rendering a weather map, markers can show the location of rain and thus provide a quick visual display of rain activity.

XGL provides a set of markers that an application can use, or the application can define its own markers using Marker objects. XGL-supplied markers are defined by the System State object and are available for application use at any time. Markers defined by the Marker object are available for rendering when the Marker object is attached to a Context object.

A marker is drawn centered at each point in a point list passed to the `xgl_multimarker()` operator. Markers can be rendered from a 2D or 3D Context; if the Context is 3D, the Marker location points are transformed, if necessary, on their way through the pipeline, but the Markers themselves are rendered parallel to the display in VDC or in DC.

If a Marker location point is outside the clipped bounds, the Marker is not drawn. If the location is inside the clipping bounds, the Marker is drawn but is clipped at the clipping bounds. A marker can be assigned color, but since the marker is not a surface, its color is not affected by lighting. The marker color is, however, affected by depth cueing when depth cueing is enabled. A marker can be scaled for rendering.

Using the Predefined Markers

XGL supplies eight predefined markers, which markers are shown in Figure 14-1. The predefined markers are Marker objects that the System State object creates when XGL is initialized. Predefined Marker objects are read-only objects that cannot be modified or destroyed.

Marker Name	Looks Like:	Marker Name	Looks Like:
xgl_marker_dot		xgl_marker_cross	
xgl_marker_plus		xgl_marker_square	
xgl_marker_asterisk		xgl_marker_bowtie_ne	
xgl_marker_circle		xgl_marker_bowtie_nw	

Figure 14-1 XGL Predefined Markers

To use a predefined Marker, the application sets the Context attribute `XGL_CTX_MARKER` to the marker name, as in the following example:

```
xgl_object_set(ctx, XGL_CTX_MARKER, xgl_marker_dot, NULL);
```

The default Marker is `xgl_marker_plus`.

Creating a New Marker

Applications can create their own markers using Marker objects. A Marker object is created with the `xgl_object_create()` operator, using `XGL_MARKER` as the *type* parameter value.

```
marker = xgl_object_create(sys_st, XGL_MARKER, NULL, NULL);
```

Marker objects consist of a list of 2D coordinates that define one or more polylines that comprise the marker. Marker object descriptions are limited to 128 float 2D coordinates; the point type can be *Xgl_pt_f2d*, *Xgl_pt_color_f2d*, or *Xgl_pt_flag_f2d*, although color and flag data in the point type is ignored if supplied. The coordinates can be any float values but are normally between -0.5 and 0.5 in both the *x* and *y* directions.

When creating a Marker description, the application fills in an *Xgl_pt_list_list* data structure and sets the Marker object description to this data structure with the Marker attribute `XGL_MARKER_DESCRIPTION`, as shown in thefollowing code fragment.

```
new_marker = xgl_object_create(sys_st, XGL_MARKER, NULL,
                 XGL_MARKER_DESCRIPTION, &marker_pt_list_list,
                 NULL);
```

The application then attaches the new Marker to a Context with the `XGL_CTX_MARKER` attribute.

```
xgl_object_set(ctx, XGL_CTX_MARKER, new_marker, NULL);
```

Note that if the application needs to get a Marker description, the application need only declare the *Xgl_pt_list_list* structure and pass XGL a pointer to it. XGL assumes that the application will not know the size of the point lists and thus will not be able to allocate memory to hold the point list information; therefore, XGL allocates the memory for the *Xgl_pt_list_list* structure on behalf of the application. Once the memory is allocated, however, it is the responsibility of the application to free the memory with a free system call. The following lines of code show how to free the memory for an *Xgl_pt_list_list* data structure:

```
free (point_list_list.pt_lists->pts.f2d);
free (point_list_list.pt_lists);
```

Note – For information on markers for the XGL-provided CGM device, see "CGM Device Object" on page 72.

Context Marker Attributes

The attribute XGL_CTX_MARKER defines the Marker symbol that is drawn by xgl_multimarker() for a Context at each point in the point list passed to xgl_multimarker().

The size of a Marker is controlled by the XGL_CTX_MARKER_SCALE_FACTOR attribute. This attribute specifies a scale factor by which the nominal Marker width of the Device (one pixel for Raster devices) is multiplied to produce the Marker width in device coordinates. The default value of XGL_CTX_MARKER_SCALE_FACTOR is 1.0. A scale factor must be provided for all predefined markers but the dot marker, or the specified marker will be rendered as one pixel. The predefined dot marker is not scaled by XGL_CTX_MARKER_SCALE_FACTOR. The dot marker always draws a 1-pixel dot.

Markers are always drawn parallel to the display. In 3D Contexts, the size of Markers is not affected by perspective transformations; in other words, Markers at different z values are the same size.

Each marker is a single color. Markers in a Context can all have the same color, or each marker's color can be specified individually. The attribute XGL_CTX_MARKER_COLOR sets the color of all Markers within a Context. If the point data supplied to the xgl_multimarker() primitive has color data for individual points (point type *Xgl_pt_color_f2d*), the attribute XGL_CTX_MARKER_COLOR_SELECTOR can specify that the color used for the Marker is taken from the point data. XGL_CTX_MARKER_COLOR_SELECTOR has the following values:

XGL_MARKER_COLOR_CONTEXT
: Marker color is rendered as defined by XGL_CTX_MARKER_COLOR. Depth cueing affects the final appearance of a marker in 3D.

XGL_MARKER_COLOR_POINT
: Marker color is taken from the point data if the point data includes colors for individual points. If the primitive data does not specify colors for individual points, the color defined by XGL_CTX_MARKER_COLOR is used.

The default value for XGL_CTX_MARKER_COLOR_SELECTOR is XGL_MARKER_COLOR_POINT.

The attributes XGL_CTX_MARKER_AA_BLEND_EQ, XGL_CTX_MARKER_AA_FILTER_WIDTH, and XGL_CTX_MARKER_FILTER_SHAPE control whether antialiasing is performed and define how markers are antialiased. For information on antialiasing, see the *XGL Reference Manual.*

Marker Example Programs

The following programs illustrate the use of predefined Markers and show how an application can define its own Marker. Each example is a fragment of a larger program. The complete program includes `ex_utils.c` and `prims_2d_main.c`, both of which are listed in Appendix B, as well as all the examples listed in this section and the primitive examples listed in Chapter 8, "Primitives and Graphics Context Attributes". To compile the complete program, type `make prims_2d` in the example program directory. The compiled program allows you to look at all the primitive example programs.

Predefined Marker Example Program

The program `prims_2d_marker.c` illustrates the predefined Markers. Figure 14-2 on page 386 shows the program output.

14

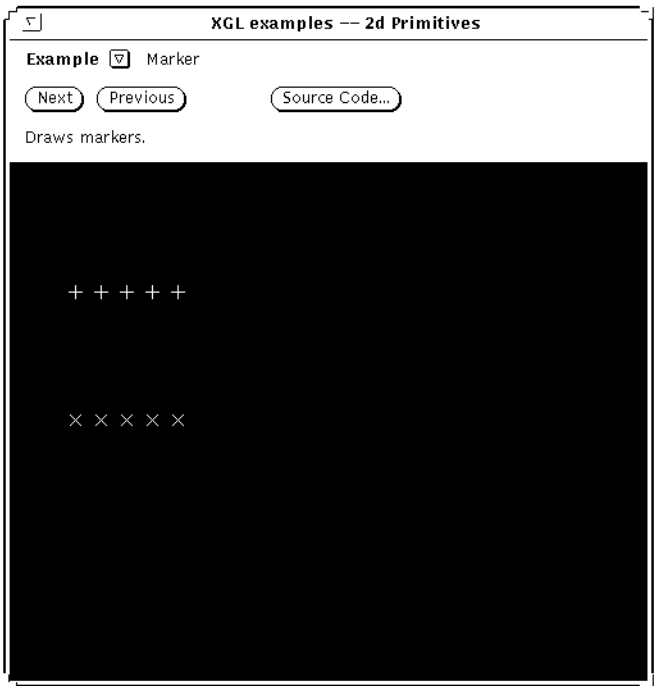

Figure 14-2 Output of `prims_2d_marker.c`

Code Example 14-1 Predefined Marker Example

```
/*
* prims_2d_marker.c
*/
#include "ex.h"

prims_2d_marker (Xgl_object    ctx)
{
    Xgl_pt_i2d           pts[20];
    Xgl_pt_list          pl[1];
    Xgl_color            color;

    xgl_object_set (ctx,
                    XGL_CTX_MARKER_SCALE_FACTOR, 10.0,
                    NULL);

    /* set up the Xgl_pt_list structure */
    pl[0].pt_type = XGL_PT_I2D;
```

```
    pl[0].bbox = NULL;
    pl[0].num_pts = 5;
    pl[0].pts.i2d = pts;

    /* draw markers in white as plus signs */

    color = white_color;
    xgl_object_set (ctx,
                    XGL_CTX_MARKER_COLOR, &color,
                    XGL_CTX_MARKER, xgl_marker_plus,
                    NULL);

    pts[0].x = 50;
    pts[0].y = 100;
    pts[1].x = 70;
    pts[1].y = 100;
    pts[2].x = 90;
    pts[2].y = 100;
    pts[3].x = 110;
    pts[3].y = 100;
    pts[4].x = 130;
    pts[4].y = 100;

    xgl_multimarker (ctx, pl);

    /* draw markers in magenta as crosses */
    color = magenta_color;
    xgl_object_set (ctx,
                    XGL_CTX_MARKER_COLOR, &color,
                    XGL_CTX_MARKER, xgl_marker_cross,
                    NULL);

    pts[0].x = 50;
    pts[0].y = 200;
    pts[1].x = 70;
    pts[1].y = 200;
    pts[2].x = 90;
    pts[2].y = 200;
    pts[3].x = 110;
    pts[3].y = 200;
    pts[4].x = 130;
    pts[4].y = 200;

    xgl_multimarker (ctx, pl);
}
```

14

User-defined Marker Example Program

The program `prims_2d_umarker.c` illustrates user-defined markers. Figure 14-3 shows the output of the program.

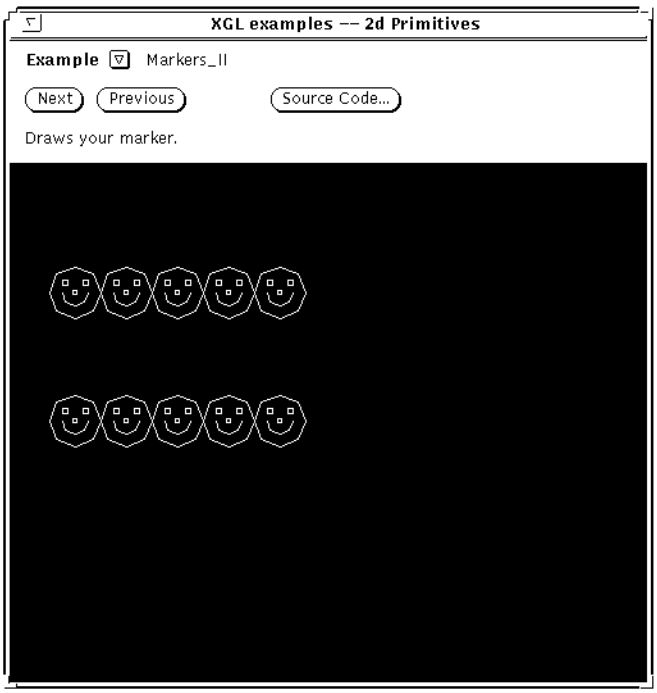

Figure 14-3 Output of `prims_2d_umarker.c`

Code Example 14-2 User-defined Marker Example

```
/*
 * prims_2d_umarker.c
 */

#include "ex.h"

#define    XGL_SET_PT_LIST(arg, argtype, argbb, argcnt, PTS_TYPE,
argpts) do { \
                (arg).pt_type = (argtype);\
                (arg).bbox = (argbb);\
                (arg).num_pts = (argcnt);\
```

```
            (arg).pts.PTS_TYPE = (argpts);\
            } while (0)

/* data for user defined markers */
static     Xgl_pt_f2d     head_circle[9] = {{0.0, -0.5},
                                {0.353553391, -0.353553391},
                              {0.5, 0.0},
                              {0.353553391, 0.353553391},
                              {0.0, 0.5},
                              {-0.353553391, 0.353553391},
                              {-0.5, 0.0},
                              {-0.353553391, -0.353553391},
                              {0.0, -0.5}};
static     Xgl_pt_f2d     mouth_circle[5] = {{0.5/2., 0.0},
                               {0.353553391/2., 0.353553391/2.},
                               {0.0, 0.5/2.},
                               {-0.353553391/2., 0.353553391/2.},
                               {-0.5/2., 0.0}};
static     Xgl_pt_f2d     nose_square[5] = {{-0.03,-0.03},
                              { 0.03,-0.03},
                              { 0.03, 0.03},
                              {-0.03, 0.03},
                              {-0.03,-0.03}};
static     Xgl_pt_f2d     eye_square[5] = {{-0.25,-0.25},
                              {-0.15,-0.25},
                              {-0.15,-0.15},
                              {-0.25,-0.15},
                              {-0.25,-0.25}};
static     Xgl_pt_f2d     eye_square2[5] = {{ 0.25,-0.25},
                              { 0.15,-0.25},
                              { 0.15,-0.15},
                              { 0.25,-0.15},
                              { 0.25,-0.25}};
static     Xgl_pt_list      face_pt_list[5];
static     Xgl_pt_list_list   face_pt_list_list = {0, 5,
                                   &face_pt_list[0]};

prims_2d_umarker (Xgl_object  ctx)

    Xgl_pt_i2d        pts[20];
    Xgl_pt_list       pl[1];
    Xgl_color         color;
    Xgl_Object        face_marker;

    /* create our own marker (the smiley-face marker) and assign
     * its data */
```

```
face_marker = xgl_object_create(sys_st, XGL_MARKER, NULL, 0);

XGL_SET_PT_LIST(face_pt_list[0], XGL_PT_F2D,NULL, 9, f2d,
                &head_circle[0]);
XGL_SET_PT_LIST(face_pt_list[1], XGL_PT_F2D,NULL, 5, f2d,
                &mouth_circle[0]);
XGL_SET_PT_LIST(face_pt_list[2], XGL_PT_F2D,NULL, 5, f2d,
                &eye_square[0]);
XGL_SET_PT_LIST(face_pt_list[3], XGL_PT_F2D,NULL, 5, f2d,
                &eye_square2[0]);
XGL_SET_PT_LIST(face_pt_list[4], XGL_PT_F2D,NULL, 5, f2d,
                &nose_square[0]);

xgl_object_set(face_marker,
          XGL_MARKER_DESCRIPTION, &face_pt_list_list,
          NULL);

xgl_object_set (ctx,
               XGL_CTX_MARKER_SCALE_FACTOR, 40.0,
               NULL);

/* setup the Xgl_pt_list structure */

pl[0].pt_type = XGL_PT_I2D;
pl[0].bbox    = NULL;
pl[0].num_pts = 5;
pl[0].pts.i2d = pts;

/* draw row of smiley-face markers in white */

color = white_color;
xgl_object_set (ctx,
               XGL_CTX_MARKER_COLOR, &color,
               XGL_CTX_MARKER, face_marker,
               NULL);

pts[0].x = 50;
pts[0].y = 100;
pts[1].x = 90;
pts[1].y = 100;
pts[2].x = 130;
pts[2].y = 100;
pts[3].x = 170;
pts[3].y = 100;
pts[4].x = 210;
```

```
    pts[4].y = 100;

    xgl_multimarker (ctx, pl);

    /* draw row of smiley-face markers in magenta */

    color = magenta_color;
    xgl_object_set (ctx,
                    XGL_CTX_MARKER_COLOR, &color,
                    NULL);

    pts[0].x = 50;
    pts[0].y = 200;
    pts[1].x = 90;
    pts[1].y = 200;
    pts[2].x = 130;
    pts[2].y = 200;
    pts[3].x = 170;
    pts[3].y = 200;
    pts[4].x = 210;
    pts[4].y = 200;

    xgl_multimarker (ctx, pl);

    /* destroy the marker we created and reset to default dot
     * marker
     */

    xgl_object_destroy(face_marker);
    xgl_object_set (ctx,
                    XGL_CTX_MARKER, xgl_marker_dot,
                    NULL);
}
```

14