Picking 15

This chapter provides information on XGL picking operators and attributes. It includes an example program illustrating picking.

Introduction to XGL Picking

In interactive graphics applications, the user may need to select graphics primitives from the ones visible on the display. This is often done with the aid of a pointing device (such as a mouse).

Once a picking region is defined and picking is enabled, any primitive that is sent into the XGL pipeline is checked against the picking region. This comparison is done in screen coordinates after the primitives have had their coordinate values transformed. In general, if any part of the transformed primitive is found to be in the pick region, that primitive is picked. The parts of a polygon primitive that can be picked depend on the `XGL_CTX_PICK_SURF_STYLE` attribute. When a primitive is picked, the pick buffer is updated with the pick identifier information in the Context at that time.

Every Context has a *pick buffer* that stores information about pick events. A *pick event* occurs whenever a part of a primitive that is sent down the pipeline after picking is enabled falls within the *pick aperture*. The pick aperture is the area (for a 2D Context) or volume (for a 3D Context) used to test primitives for picking. If a primitive falls within the bounds set by the attribute `XGL_CTX_PICK_APERTURE` (set in Device Coordinates), the entire primitive is picked when any part of the primitive is picked.

A 3D Context requires a range of z values as well as x and y values in order to define a picking volume.

Note – The XGL viewing transform can scale and translate the z values of a primitive to unexpectedly high values. Be sure to set the maximum z value appropriately. If in doubt, or if you just want to pick in x and y, call `xgl_object_get()`, inquiring on `XGL_DEV_MAXIMUM_COORDINATES`, and use the z value as the maximum z picking range (use 0 as the minimum).

The *pick identifier* attributes `XGL_CTX_PICK_ID1` and `XGL_CTX_PICK_ID2` are added as entries to the pick buffer according to certain rules. The application should set the pick identifier attributes to unique values as each primitive is sent down the pipeline. That way, the application can tell which primitive was picked. Only one pick aperture can be active at any one time.

If Z-buffering is enabled during a pick, primitives are checked for visibility against the current Z-buffer values. If the primitive is obscured within the picking region (using the Z-buffer hidden surface algorithm), the object will not be picked even if it lands in the pick region.

Picking by itself does not update the Z-buffer, only rendering does. If the Z-buffer has not been properly updated (by re-rendering the image with Z-buffering enabled whenever a change occurs in the image), the Z-buffered picking values will most likely be incorrect. The application should ensure that the Z-buffer state reflects the primitives it is trying to pick.

Picking Attributes

The attribute `XGL_CTX_PICK_ENABLE` must be `TRUE` for picking to occur. All rendering primitives except `xgl_image()` can be picked. `XGL_CTX_PICK_STYLE` can only be changed when picking is disabled. The size of the pick buffer is determined by the attribute `XGL_CTX_PICK_BUFFER_SIZE`. The default size of the pick buffer is 256. It can be changed to any nonnegative value.

The attributes `XGL_CTX_PICK_ID_1` and `XGL_CTX_PICK_ID_2` identify the primitives that were picked. Having two pick IDs enables the application to associate more information with a picked primitive. The application can identify picked primitives, depending on how the pick IDs were set before the

primitives were sent through the pipeline. The current pick ID must differ from the last entry in the buffer, or it will not be stored again. The default value for the pick ID attributes is 0.

The attribute `XGL_CTX_PICK_STYLE` defines what happens when pick IDs are added to a full pick buffer. It also controls how pick events are returned to the application when the operator `xgl_pick_get_identifiers()` is invoked. The two pick styles are:

`XGL_PICK_FIRST_N`
: The first *n* pick events (*n* being the pick buffer size) are stored in the buffer. Once the buffer is full, later pick events are ignored. The events are returned in the order they were stored, the first event being the first one returned.

`XGL_PICK_LAST_N`
: The pick events are stored in a first-in-first-out (FIFO) queue. When the pick buffer is full, the oldest event is removed to make room for the most recent event. The events are returned in reverse order to that in which they were stored; the most recent event in the buffer is the first one returned.

The attribute `XGL_CTX_PICK_SURF_STYLE` determines which parts of the surface primitives are pickable:

`XGL_PICK_SURF_AS_SOLID`: The entire surface is pickable.

`XGL_PICK_SURF_AS_FILL_STYLE`: The pickable part is device-dependent if the polygon is stipple-filled. (A stipple-filled polygon will either be picked like a solid-filled one, or only be pickable on the visible parts of the fill pattern.) Empty polygons are not pickable, unless their edges are made explicitly visible with the vertex visibility flag. Then only the visible sections of the edges are pickable. Hollow polygons can be picked only at their edges.

15

XGL Picking Operators

The picking operators are `xgl_pick_clear()` and `xgl_pick_get_identifiers()`.

Clearing the Pick Buffer

The operator `xgl_pick_clear()` clears the pick buffer associated with the Context `ctx` to the *nothing-picked* state.

```
void xgl_pick_clear (Xgl_ctxctx);
```

Identifying Picked Primitives

The `xgl_pick_get_identifiers()` operator returns the identifiers (IDs) of picked primitives stored in the Context `ctx`. The returned pick IDs are stored in the array `pick_id_list`, which is passed as an argument to the operator. The application must have allocated `pick_id_list` to hold as many *Xgl_pick_info* structures as are in the Context pick buffer. The number of structures in the Context pick buffer is set by the attribute `XGL_CTX_PICK_BUFFER_SIZE`. The default value is 256. If the pick buffer overflows, the oldest pick information may be dropped, or the most recent information may be ignored, depending on the value of `XGL_CTX_PICK_STYLE`. The order in which the pick events are returned depends on the value of the attribute `XGL_CTX_PICK_STYLE`.

```
void xgl_pick_get_identifiers(
    Xgl_ctx             ctx,
    Xgl_usgn32          *count,
    Xgl_pick_info       pick_id_list[] );
```

Picking Example

The example program `pick_2d_prims.c` draws white plus-sign markers, a blue polygon, and a yellow polyline that can be picked interactively using the right mouse button. This program uses the examples utilities package `ex.utils.c` to pick when the left mouse button is pressed. The application must allocate enough space to hold all possible return *Xgl_pick_info* structures. If anything is picked, information on picked geometry will be returned to the application program. For example, if the polygon is selected, the statement "Polygon picked" is displayed. If nothing is selected, the statement "Nothing picked" is displayed.

To compile this program, type `make pick_2d` in the example program directory. The compiled program includes `ex_utils.c` and `pick_2d_main.c`, which are listed in Appendix B. Figure 15-1 shows the output of the program.

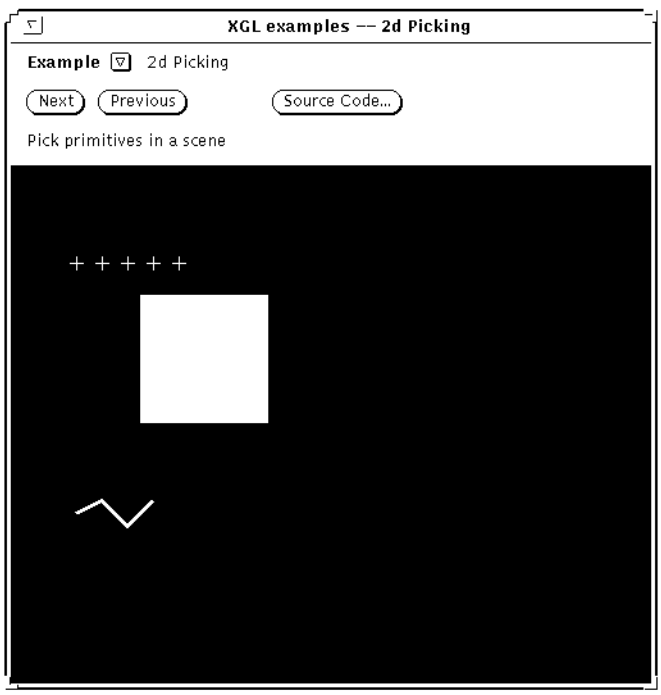

Figure 15-1 Output of `pick_2d_prims.c`

15

Code Example 15-1 Picking Example

```
/*
 * pick_2d_prims.c
 */
#include "ex.h"
#include <xgl/xgl.h>
#include <malloc.h>

extern Xgl_color_typeex_color_type;

/* Pick Identifiers for the primitives */

#define PICK_ID_MARKER1
#define PICK_ID_PLINE2
#define PICK_ID_PGON3

/* XGL provides 2 pick identifiers.  For this example, we are only
 * using one of them: XGL_CTX_PICK_ID_1.  We draw each primitive in
 * a different color.  We set all the colors at once, before the
 * primitives are drawn.
 *
 * The pick identifier is set just before each primitive is drawn
 * because the pick identifier applies to _all_ subsequent
 * primitives.To uniquely
 * identify each primitive, we need to change the pick id for it.
 *
 * A performance tip: if you are drawing lots of primitives, don't
set
 * the pick identifier if you are not picking; the "picking" variable
 * is used for this.
 */


int        picking = TRUE; /* This is TRUE if we ar picking.*/

pick_draw_scene (Xgl_object     ctx)

    Xgl_pt_i2d          pts[20];
    Xgl_pt_list         pl[1];
    Xgl_color           marker_color;
    Xgl_color           pgon_color;
    Xgl_color           pline_color;

    /* draw markers in white as plus signs */
    marker_color = white_color;
```

```
/* draw square polygon in cyan */
pgon_color = cyan_color;
/* draw 3-segment polyline in yellow */
pline_color = yellow_color;
xgl_object_set (ctx,
          XGL_CTX_MARKER_SCALE_FACTOR, 10.0,
          XGL_CTX_MARKER_COLOR, &marker_color,
          XGL_CTX_MARKER, xgl_marker_plus,
          XGL_CTX_SURF_FRONT_COLOR, &pgon_color,
          XGL_CTX_LINE_COLOR, &pline_color,
          XGL_CTX_LINE_WIDTH_SCALE_FACTOR, 3.0,
          NULL);

/* setup the Xgl_pt_list structure */

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

pl[0].num_pts = 5;
pts[0].x = 50;
pts[0].y = 75;
pts[1].x = 70;
pts[1].y = 75;
pts[2].x = 90;
pts[2].y = 75;
pts[3].x = 110;
pts[3].y = 75;
pts[4].x = 130;
pts[4].y = 75;

if (picking)
   xgl_object_set (ctx, XGL_CTX_PICK_ID_1, PICK_ID_MARKER,
                   NULL);
xgl_multimarker (ctx, pl);

/* Draw a square as a polygon */

pl[0].num_pts = 4;

pts[0].x = 100;
pts[0].y = 100;
pts[1].x = 200;
pts[1].y = 100;
pts[2].x = 200;
pts[2].y = 200;
```

```
    pts[3].x = 100;
    pts[3].y = 200;

    if (picking)
       xgl_object_set (ctx, XGL_CTX_PICK_ID_1, PICK_ID_PGON,
                       NULL);
    xgl_polygon (ctx, XGL_FACET_NONE, NULL, NULL, 1, pl);

    /* Draw a polyline */

    pl[0].num_pts = 4;

    pts[0].x = 50;
    pts[0].y = 270;
    pts[1].x = 70;
    pts[1].y = 260;
    pts[2].x = 90;
    pts[2].y = 280;
    pts[3].x = 110;
    pts[3].y = 260;
    pts[4].x = 130;
    pts[4].y = 270;

    if (picking)
       xgl_object_set (ctx, XGL_CTX_PICK_ID_1, PICK_ID_PLINE,
                       NULL);
    xgl_multipolyline (ctx, NULL, 1, pl);
}

/* For this example, the pick aperture is a square
 * (2 * APERTURE_SIZE) pixels
 * in width and height centered at the event X, Y.
 */

#define APERTURE_SIZE  5

/* The example utilities package (ex_utils.c) calls this function
 * when the left (ACTION_SELECT) mouse button goes down.
 */

pick_do_pick (
    Xgl_object          ctx,
    int                 x,
    int                 y)
{
    Xgl_bounds_f2d      pick_aperture;
```

```
Xgl_usgn32           count;
int               i;
static Xgl_pick_info *id_list = NULL;

pick_aperture.xmin = x - APERTURE_SIZE;
pick_aperture.xmax = x + APERTURE_SIZE;
pick_aperture.ymin = y - APERTURE_SIZE;
pick_aperture.ymax = y + APERTURE_SIZE;

/* Set the pick aperture before turning on picking */

xgl_object_set (ctx,
          XGL_CTX_PICK_APERTURE, &pick_aperture,
          XGL_CTX_PICK_ENABLE, TRUE,
          NULL);
xgl_pick_clear (ctx);
picking = TRUE;

/*
 * "Draw" the scene. Note: nothing is really drawn on the
 * screen because we have turned XGL_CTX_PICK_ENABLE on,
 * which turns rendering off.
 * The primitives are just processed for picking.
 */
pick_draw_scene (ctx);

picking = FALSE;
xgl_object_set (ctx,
          XGL_CTX_PICK_ENABLE, FALSE,
          NULL);

/* See what, if anything, we have picked */

/*
 * But, first, we need to allocate enough space to hold all the
 * possible returned pick_info structures.
 */
if (id_list == NULL) {
   Xgl_usgn32           buff_size;

   xgl_object_get (ctx, XGL_CTX_PICK_BUFFER_SIZE,
                   &buff_size);
   id_list = (Xgl_pick_info *) malloc (sizeof (Xgl_pick_info) *
               buff_size);
   if (!id_list) {
       fprintf (stderr, "pick_2d: malloc failed!\n");
```

```
            exit (1);
        }
    }
    xgl_pick_get_identifiers (ctx, &count, id_list);

    if (count == 0)
       printf ("Nothing picked.\n");
    else {
       for (i = 0; i < count; i++) {
           switch (id_list[i].id1) {
           case PICK_ID_MARKER:
               printf ("Markers picked.\n");
               break;
           case PICK_ID_PGON:
               printf ("Polygon picked.\n");
               break;
           case PICK_ID_PLINE:
               printf ("Polyline picked.\n");
               break;
           default:
               printf ("Invalid pick_id.\n");
           break;
           }
       }
    }
}
```