Chapter 12

Picking and Selecting

This chapter discusses how to use the GL *picking* and *selecting* subroutines. Picking and selecting are used to create a "point and click" interface, where the user can position the mouse over an area of the screen and click the mouse button to choose an item on the screen.

- Section 12.1, "Picking," describes how to test for proximity to the cursor.
- Section 12.2, "Selecting," describes how to designate a screen region to be used for picking.

12.1 Picking

Picking identifies items on the screen that are near the cursor. When the system is in picking mode, it does not draw anything on the screen. Instead, drawing subroutines that would have been drawn near the current location of the cursor cause *hits* to be recorded in the picking buffer.

All the standard drawing routines cause hits, except raster operations, such as character strings drawn with `charstr()`. However, because `cmov()` does cause a hit, character strings can be picked because they can appear to cause hits. The cursor must be near the lower-left corner of the string for it to be picked. Because pixel subroutines such as `readpixels()` and `readRGB()` are often preceded by `cmov()`, these routines can also appear to cause hits.

To use picking effectively, your program must be structured in such a way that you can regenerate the picture on the screen whenever picking is required.

To perform picking:

1. Set the system into picking mode
2. Redraw the image on the screen using `pick()`
3. Call `endpick()`.

The results of the pick appear in a buffer specified by `pick()` and `endpick()`.

A *name stack* stores the name of the items that are hit while picking is enabled. When you call a subroutine that alters the name stack or when you exit picking mode, the contents of the name stack are copied to the buffer if a hit occurs. See Section 12.1.2, "Using the Name Stack," for more details about the name stack.

pick

`pick()` puts the system in picking mode:

```
void pick(short buffer[], long numnames)
```

The *numnames* argument specifies the maximum number of values that the picking buffer can store. The graphical items that intersect the picking region are hits and store the contents of the name stack in *buffer*.

endpick

`endpick()` takes the system out of picking mode and returns the number of hits that occurred in the picking session:

```
long endpick(short buffer[])
```

If `endpick()` returns a positive number, the buffer stored all of the name lists. If it returns a negative number, the buffer was too small to store all the name lists; the magnitude of the returned number is the number of name lists that were stored.

buffer contains all of the name lists stored in picking mode, one list for each valid hit. The first value in each name list is the length of a name list. If a name stack is empty when a hit occurs, the first and only entry in the list for that hit is "0."

12.1.1 Defining the Picking Region

Picking loads a projection matrix that makes the picking region fill the entire viewport. This picking matrix replaces the projection transformation matrix that is normally used when drawing routines are called. Therefore, you must restate the original projection transformation after each `pick()` and after each `endpick()`, to ensure that the system maps the objects to be picked to the proper coordinates. If no projection transformation was originally issued, you must specify the default, `ortho2()`. When the transformation routine is restated, the product of the transformation matrix and the picking matrix is placed at the top of the matrix stack.

Note: If you do not restate the projection transformation, picking does not work properly. Instead, the system typically picks every object, regardless of cursor position and pick size.

Specifying or defining the default `ortho2()` parameters brings up the issue of creating a graphics window that has a one-to-one mapping between screen space (viewport) and world space (in this case, `ortho2()`).

In the following example, assume a graphics window that is 4 pixels wide by 5 pixels high. This window runs from coordinates 1 to 4 in *x* and 1 to 5 in *y*. In order to set up a mapping between floating point coordinates and screen space (integer coordinates) that centers pixels at integer coordinates in the `ortho2()` projection, you need to specify:

```
viewport(0, 3, 0, 4);
ortho2(0.5, 4.5, 0.5, 5.5);
```

Extrapolate from this and assume a situation where the graphics window has been resized and you need to redefine a current `ortho2()` based on the new size. To do this, use the following three statements:

```
getsize(&xsize, &ysize);
viewport(0, xsize - 1, 0, ysize - 1);
ortho2  (x1-0.5,(float)(xsize-0.5), y1-0.5,(float)(ysize-0.5);
```

In the call to `viewport()`, you must subtract 1 from the value of *xsize* and *ysize* because they start at 0, not at 1. Likewise, in the call to ortho2, you need to start at -0.5 less than the corner coordinates and subtract -0.5 from *xsize* and *ysize* to create the straddling effect described earlier.

picksize

The default height and width of the picking region is 10 pixels centered at the cursor. You can change the picking region with `picksize()`.

```
picksize(deltax, deltay)
```

deltax and *deltay* specify a rectangle centered at the current cursor position (the origin of the cursor glyph; see Chapter 11, "Drawing Modes," for a discussion of cursors.)

12.1.2 Using the Name Stack

A name stack stores the name of the items that are hit while picking is enabled. The name stack is a stack of 16-bit names whose contents are controlled by `loadname()`, `pushname()`, `popname()`, and `initname()`. You can store up to 1000 names in a name stack. You can intersperse these routines with drawing routines, or you can insert them into object definitions (see Chapter 16, "Graphical Objects," for a discussion of objects).

Note: Because the contents of the name stack are reported only when it changes, one hit is reported no matter how many drawing routines actually draw something near the cursor. If the application requires more accuracy than this, it must modify the name stack more often.

loadname

`loadname()` puts *name a*t the top of the name stack and erases what was there before:

```
void loadname(short name)
```

pushname

`pushname()` puts *name* at the top of the stack and pushes all the other names in the stack one level lower:

```
void pushname(short name)
```

Before the first `loadname()` is called, the current name is unpredictable. Calling `pushname()` before `loadname()` can cause unpredictable results.

popname

`popname()` discards the name at the top of the stack and moves all the other names up one level:

```
void popname(void)
```

initnames

`initnames()` discards all the names in the stack and leaves the stack empty:

```
void initnames(void)
```

The sample program *pick.c* draws an object consisting of three shapes; then it loops, until you press the right mouse button. Each time you press the middle mouse button, the system:

1. enters picking mode
2. calls the object
3. records hits for any routines that draw into the picking region
4. prints out the contents of the picking buffer

Note: When you call an object in picking mode, the screen does not change. Because the picking matrix is recalculated only when you call `pick()`, the system exits and reenters picking mode to obtain new cursor positions.

Position the cursor near one of the shapes on the screen and click the left mouse button. The results of the pick are printed to the screen. Five outcomes are possible for each picking session. The circles can be picked together because they overlap:

- nothing is picked = “hit count: 0 hits:”
- the square is picked = “hit count: 1 hits: (1)”
- the bottom circle is picked = “hit count: 1 hits: (2 21)”
- the top circle is picked = “hit count: 1 hits: (2 22)”
- both the top and bottom circles are picked = “hit count: 2 hits: (2 21) (2 22)”

```
#include <stdio.h>
#include <gl/gl.h>
```

```
#include <gl/device.h>
#define BUFSIZE 50

void drawit()
{
    loadname(1);
    color(BLUE);
    writemask(BLUE);
    sboxfi(20, 20, 100, 100);
    loadname(2);
    pushname(21);
        color(GREEN);
        writemask(GREEN);
        circfi(200, 200, 50);
    popname();
    pushname(22);
        color(RED);
        writemask(RED);
        circi(200, 230, 60);
    popname();
    writemask(0xfff);
}
void printhits(buffer, hits)
short buffer[];
long hits;
{
    int indx, items, h, i;
    char str[20];
    sprintf(str, "%ld hit", hits);
    charstr(str);
    if (hits != 1)
        charstr("s");
    if (hits > 0)
        charstr(": ");
    indx = 0;
    for (h = 0; h < hits; h++) {
        items = buffer[indx++];
        charstr("(");
        for (i = 0; i < items; i++) {
            if (i != 0)
                charstr(" ");
            sprintf(str, "%d", buffer[indx++]);
            charstr(str);
        }
        charstr(") ");
    }
}
```

```
main()
{
    Device dev;
    short val;
    long hits;
    long xsize, ysize;
    short buffer[BUFSIZE];
    Boolean run;
    prefsize(400, 400);
    winopen("pick");
    getsize(&xsize, &ysize);
    mmode(MVIEWING);
    ortho2(-0.5, xsize - 0.5, -0.5, ysize - 0.5);
    color(BLACK);
    clear();
    qdevice(LEFTMOUSE);
    qdevice(ESCKEY);
    drawit();
    run = TRUE;
    while (run) {
        dev = qread(&val);
        if (val == 0) {  /* on upstroke */
            switch (dev) {
            case LEFTMOUSE:
                pick(buffer, BUFSIZE);
                    ortho2(-0.5, xsize - 0.5, -0.5, ysize - 0.5);
                    drawit();/* no actual drawing takes place */
                hits = endpick(buffer);
                ortho2(-0.5, xsize - 0.5, -0.5, ysize - 0.5);
                color(BLACK);
                sboxfi(150, 20-getdescender(), size -1, 20 + getheight());
                color(WHITE);
                cmov2i(150, 20);
                printhits(buffer, hits);
                break;
            case ESCKEY:
                run = FALSE;
                break;
                }
        }
    }
    gexit();
    return 0;
}
```

12.2 Selecting

Selecting is a more general mechanism than picking for identifying the routines that draw to a particular region. A selecting region is a 2-D or 3-D area of world space. When `gselect()` turns on selecting mode, the region represented by the current viewing matrix becomes the selecting region. You can change the selecting region at any time by issuing a new viewing transformation routine. To use selecting mode:

1. Issue a viewing transformation routine that specifies the selecting region.
2. Call `gselect()`.
3. Call the objects or routines of interest.
4. Exit selecting mode and look to see what was selected.

gselect

`gselect()` turns on the selection mode:

```
gselect(short buffer[], long numnames)
```

`gselect()` and `pick()` are identical, except `gselect()` allows you to create a viewing matrix in selection mode.

`numnames()` specifies the maximum number of values that the buffer can store. Names are 16-bit numbers that you store on the name stack. Each drawing routine that intersects the selecting region causes the contents of the name stack to be stored in buffer. The name stack is used in the same way as it is in picking.

endselect

`endselect()` turns off selecting mode:

```
long endselect(short buffer[])
```

b*uffer* stores any hits the drawing routines generated between `gselect()` and `endselect()`. Each name list represents the contents of the name stack when a routine was called that drew into the selecting region. `endselect()` returns the number of name lists in *buffer*. If the number is negative, more routines drew into the selecting region than were specified by *numnames*.

This sample program, *select1.c*, uses selecting to determine if a rocket ship is colliding with a planet. The program calls a simplified version of the planet and draws a box representing the ship each time you press the left mouse button, and beeps when the ship collides with the planet.

```
#include <stdio.h>
#include <gl/gl.h>
#include <gl/device.h>
#define X           0
#define Y           1
#define XY          2
#define BUFSIZE         10
#define PLANET          109
#define SHIPWIDTH       20
#define SHIPHEIGHT      10
void drawplanet()
{
    color(GREEN);
    circfi(200, 200, 20);
}
main()
{
    float ship[XY];
    long org[XY];
    long size[XY];
    Device dev;
    short val;
    Device mdev[XY];
    short mval[XY];
    long nhits;
    short buffer[BUFSIZE];
    Boolean run;
    prefsize(400, 400);
    winopen("select1");
    getorigin(&org[X], &org[Y]);
    getsize(&size[X], &size[Y]);
    mmode(MVIEWING);
    ortho2(-0.5, size[X] - 0.5, -0.5, size[Y] - 0.5);
    qdevice(LEFTMOUSE);
    qdevice(ESCKEY);
    color(BLACK);
    clear();
    mdev[X] = MOUSEX;
    mdev[Y] = MOUSEY;
    drawplanet();
    run = TRUE;
```

```
    while (run) {
    dev = qread(&val);
        if (val == 0) {  /* on upstroke */
            switch (dev) {
            case LEFTMOUSE:
                getdev(XY, mdev, mval);
                ship[X] = mval[X] - org[X];
                ship[Y] = mval[Y] - org[Y];
                color(BLUE);
                sbox(ship[X], ship[Y],
                     ship[X] + SHIPWIDTH, ship[Y] + SHIPHEIGHT);

/*  specify the selecting region to be a box surrounding the rocket ship */
                ortho2(ship[X], ship[X] + SHIPWIDTH,
                        ship[Y], ship[Y] + SHIPHEIGHT);
                initnames();
                gselect(buffer, BUFSIZE);
                    loadname(PLANET);
                    /* no actual drawing takes place */
                    drawplanet();
                nhits = endselect(buffer);

/*  restore the Projection matrix. Can't use push/popmatrix because they only
*   work for the ModelView matrix stack when in MVIEWING mode */
            ortho2(-0.5, size[X] - 0.5, -0.5, size[Y] - 0.5);

/*  check to see if PLANET was selected, nhits is NOT the number
*   of buffer elements written */
                if (nhits < 0) {
                    fprintf(stderr, "gselect buffer overflow\n");
                    run = FALSE;
                }
                else if (nhits >= 1  & buffer[0] == 1 && buffer[1] == PLANET)
                    ringbell();
                break;
            case ESCKEY:
                run = FALSE;
                break;
            }
        }
    }
    gexit();
    return 0;
}
```