Chapter 17

Feedback

This chapter describes methods used to get hardware feedback during the drawing process. Because this is a special topic with limited applications, you may want to skip this chapter on the first reading.

- Section 17.1, “Feedback on IRIS-4D / GT / GTX Systems,” describes feedback on those systems.
- Section 17.2, “Feedback on the Personal IRIS and IRIS Indigo,” describes feedback on those systems.
- Section 17.3, “Feedback on IRIS-4D / VGX, SkyWriter, and RealityEngine Systems,” describes feedback on those systems.
- Section 17.4, “Feedback Example,” demonstrates feedback.
- Section 17.5, “Additional Notes on Feedback,” discusses additional information.

Feedback is a system-dependent mechanism that uses the Geometry Pipeline to do calculations and to return the results of those calculations to the user process. From a hardware point of view, the net result of most Graphics Library calls is to send a series of commands and data down the Geometry Pipeline. In the pipeline, points are first transformed, clipped, and scaled, then lighting calculations are performed and colors are computed. Next, the points, lines, and polygons are scan-converted, and finally, pixels in the bitplanes are set to the appropriate values.

Note: Feedback is different on each IRIS-4D Series system. Avoid using the feedback mechanism unless it is absolutely necessary.

There are, however, a few places where feedback might be valuable. If you have code that draws an object on the screen, and you would like to draw the

same picture on a plotter with a different resolution than that of the screen, you can change only the `viewport()` subroutine (which controls the scaling of coordinates) so that it scales to your plotter coordinates, and then draw the picture in feedback mode. The transformed data returned to your process can often be interpreted and used to drive a plotter. `feedback()` puts the system into feedback mode, and any set of graphics subroutines can then be issued, followed by `endfeedback()`. All the commands and data that come out of the Geometry Engine subsection are as a result stored in a buffer supplied when the initial call to `feedback()` was made.

When the system is put into feedback mode, the Graphics Library commands send exactly the same information into the front of the pipeline, but the pipeline is short-circuited, and the results of some of the calculations are returned before the standard drawing process is complete. The pipeline can be broken down into many distinct stages, the first of which is composed of Geometry Engine™ processors. The Geometry Engines transform, clip, and scale vertices to screen coordinates, and do the basic lighting calculations. In feedback mode, the raw output from the Geometry Engines is sent back to the host process, and no further calculations are done.

The hardware that makes up the Geometry Engine subsection of the pipeline is different on the various IRIS workstation models. The command and data format differs and certain calculations are done on some systems and not on others. In spite of object code compatibility, the results of feedback are not compatible. If you use feedback, your code must be written differently for every system, and each time a new system is introduced, it will probably have to be modified.

Almost all feedback-type calculations can easily be done in portable host software. After a feedback session, the feedback buffer can contain any or all of the following data: points, lines, moves, draws, polygons, character move, `passthrough()`, `z-buffer`, `linestyle()`, `setpattern()`, `linewidth()`, and `lsrepeat()` values.

On the IRIS-4D/VGX, Iris Indigo, and Personal IRIS, `feedback()` returns 32-bit floating point values instead of 16-bit integers. On all other IRIS-4D Series systems, `feedback()` returns 16-bit integers.

In feedback mode, all the graphical subroutines are transformed, clipped, and scaled by the viewport, and all lighting calculations are done. Because of clipping, more or fewer vertices might appear in the feedback buffer than were sent in. A three-sided polygon can come out with up to nine sides. due to

clipping against all six clipping planes—even more side if user -defined arbitrary clipping planes are enabled (see Chapter 8).

Figure 17-1 shows the effect clipping has on feedback.

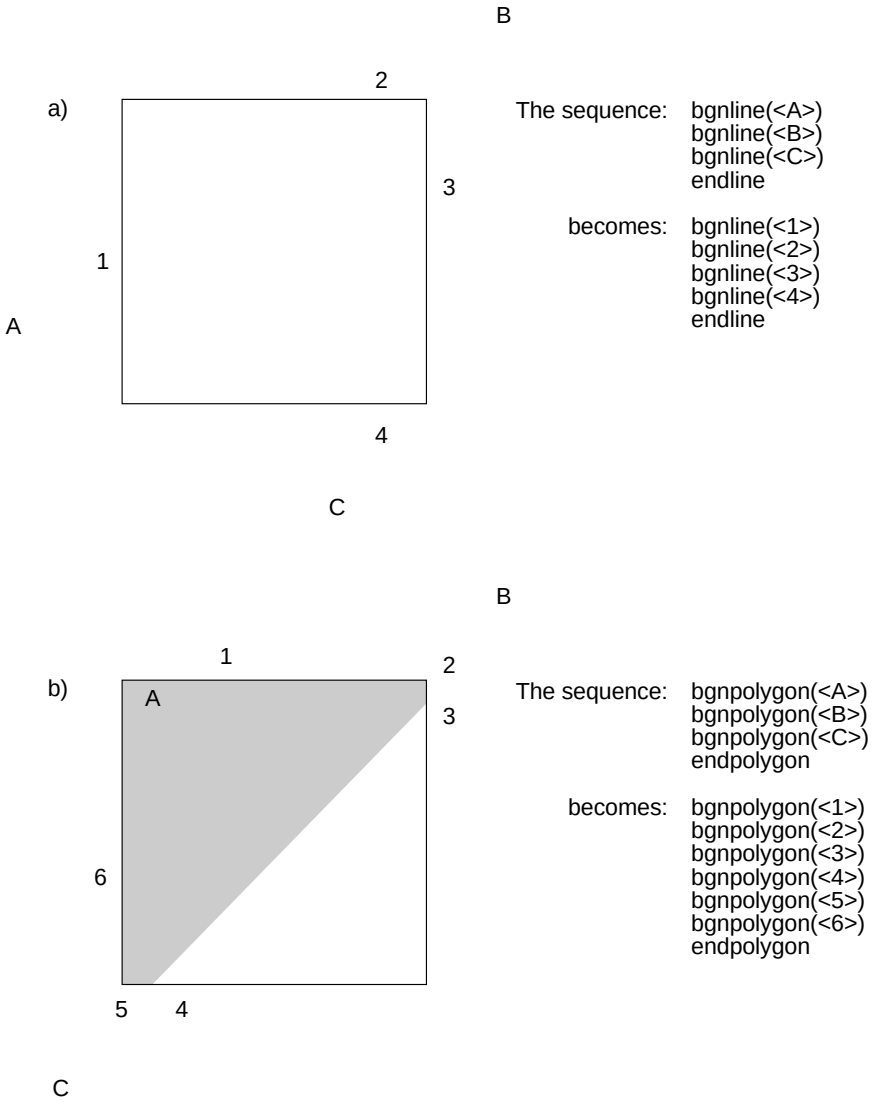

Figure 17-1 Effects of Clipping on Feedback

Because the length of the output is not generally predictable from the input, passthrough marks divisions in the input data. For example if you send this sequence:

```
v3f(A);
passthrough(1);
v3f(B);
passthrough(2);
v3f(C);
passthrough(3);
v3f(D);
```

the parsed information in the feedback buffer might look like this:

```
transformed point (X)
passthrough (1)
passthrough (2)
transformed point (Y)
passthrough (3)
```

Point *X* is the transformed version of point *A*, and point *Y* is the transformed version of point *C*. Points *B* and *D* must have been clipped out.

The feedback data types are in the file *gl/feed.h* for your reference. All returned information is raw and system-specific.

17.1 Feedback on IRIS-4D/GT/GTX Systems

Feedback data occurs in groups of 8n+2 shorts, where *n* is the number of vertices involved, as shown in Table 17-1.

Short #	Data
1	<data type>
2	<count>
3 through (count+2)	<vertex data>

Table 17-1 IRIS-4D / G / GT / GTX Feedback Data

The vertex data is always arranged in groups of 8 (so count is a multiple of 8) and contain the values:

```
x, y, zhigh, zlow, r, g, b, alpha
```

x is the screen (not window) x-coordinate, y-1024 is the screen y-coordinate, (zhigh<<16)+zlow is the z-coordinate, and, r, g, b, and alpha are the red, green, blue, and alpha values.

By the time the data makes it through the geometry hardware, all the transformations have been done to it, including translations to put the data in the proper window. In the IRIS-4D / GT / GTX, the hardware screen *y*-coordinates begin at 1024 (for the bottom of the screen) and increase to 2047. Thus, 1024 must be subtracted to get what you would consider the screen *y*-coordinate.

24 bits of *z*-coordinate data is returned in two 16-bit chunks. The two chunks must be concatenated to get the full 24 bits of data. Finally, the red, green, blue, and alpha values are the colors that would be written into the frame buffer at the vertex. In RGB mode, all values vary between 0 and 255; in color map mode, the color index is sent as the red value and is in the range 0 to 4095. In color map mode, the values in the green, blue and alpha components are meaningless.

There are five possible kinds of data type: `FB_POINT`, `FB_LINE`, `FB_POLYGON`, `FB_CMOV`, and `FB_PASSTHROUGH`.

```
FB_POINT  24
x1, y1, zhigh1, zlow1, r1, g1, b1, alpha1
x2, y2, zhigh2, zlow2, r2, g2, b2, alpha2
x3, y3, zhigh3, zlow3, r3, g3, b3, alpha3
```

`FB_LINE` and `FB_POLYGON` are similar. `FB_CMOV` and `FB_PASSTHROUGH` always have 8 shorts of data as follows:

```
FB_CMOV 8
x1, y1, zhigh1, zlow1, r1, g1, b1, alpha1
value, junk, junk, junk, junk, junk, junk, junk
```

17.2 Feedback on the Personal IRIS and IRIS Indigo

The Personal IRIS and IRIS Indigo has the following feedback tokens defined in *gl/feed.h*:

```
FB_POINT
FB_MOVE
FB_DRAW
FB_POLYGON
FB_CMOV
FB_PASSTHROUGH
FB_ZBUFFER
FB_LINESTYLE
FB_SETPATTERN
FB_LINEWIDTH
FB_LSREPEAT
```

Each group of feedback data begins with one of the above tokens to indicate data type. Vertex data for points, lines, and polygons always appears in groups of six floating-point values:

```
x, y, z, r, g, b
```

x and *y* are screen (not window) coordinates, *z* is the *z* value, and *r, g, b* are the red, green, and blue (RGB) values.

The RGB values are the colors that would be written into the frame buffer at the vertex. In RGB mode, all values vary between 0 and 255. In color map mode, the *r* value is the color index (between 0 and 4095) and the *g* and *b* values are ignored.

If a move, draw, or point (as in this example) comes out of the Geometry Pipeline, the returned data consists of seven floats:

```
FB_POINT
x, y, z, r, g, b
```

For polygons, feedback data includes a count number as well as the data type number. This number indicates how many of the next float values apply to the polygon. There are six for each vertex, so this number is always a multiple of six (6, 12, etc.).

For example, the returned data for a triangle consists of 20 floats:

```
FB_POLYGON   18.0
x1, y1, z1, r1, g1, b1
x2, y2, z2, r2, g2, b2
x3, y3, z3, r3, g3, b3
```

The 18.0 indicates three vertices with six values in each; the 18 values follow.

FB_CMOV returns only three floats of data:

```
FB_CMOV
x, y, z
```

The rest of the commands (FB_PASSTHROUGH, FB_ZBUFFER, FB_LINESTYLE, FB_SETPATTERN, FB_LINEWIDTH, FB_LSREPEAT) return only one float. For example, FB_PASSTHROUGH returns:

```
FB_PASSTHROUGH
value
```

17.3 Feedback on IRIS-4D/VGX, SkyWriter, and RealityEngine Systems

The IRIS-4D / VGX and SkyWriter systems return 32-bit floating point numbers in feedback mode. The feedback data is in the following format:

```
<data type> <count> <count words of data>
```

There are five data types: FB_POINT, FB_LINE, FB_POLYGON, FB_CMOV, and FB_PASSTHROUGH. The actual values of these data types are defined in *gl/feed.h*. Following is the feedback format:

```
FB_POINT, count (9.0), x, y, z, r, g, b, a, s, t.
FB-LINE, count (18.0), x1, y1, z1, r1, g1, b1, a1, s1, t1, x2,
y2, z2, r2, g2, b2, a2, s2, t2.
FB_POLYGON, count (27.0), x1, y1, z1, r1, g1, b1, a1, s1, t1,
x2, y2, z2, r2, g2, b2, a2, s2, t2, x3, y3, z3, r3, g3, b3,
a3, s3, t3.
FB_PASSTHROUGH, count (1.0), passthrough.
FB-CMOV, count (3.0), x, y, z.
```

The *x* and *y* values are in floating point screen coordinates, the *z* value is the floating point transformed *z*. Red, green, blue, and alpha are floating point values ranging from 0.0 to 255.0 in RGB mode. In color map mode, the color

index is stored in the red value and ranges from 0.0 to 4095.0. The green, blue, and alpha values are undefined in color map mode. The *s* and *t* values are in floating point texture coordinates.

RealityEngine returns the following feedback data for points, lines, and triangles:

```
x,y,z     /* position */
r,g,b,a   /* color */
s,t,u,q   /* texture */
```

17.4 Feedback Example

The following program transforms some simple geometric figures and the results are returned in a buffer.

In this example, `feedback()` puts the system to into feedback mode, and tells the system to return all data into the buffer (either *fbuf* or *sbuf*, depending on the machine type). In addition, the 110 indicates that the size of the buffer is 110 data items. If more than 110 items of data are generated, only the first 110 are saved. The geometry is drawn (in feedback mode), and `endfeedback()` ends the feedback session. `endfeedback()` returns the total number of items returned in the buffer. If an overflow occurs, the system returns a error. Finally, the loop at the end prints out the contents of the feedback buffer.

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

#define BUFSIZE 110

float vert[3][2] = {
   {0.1, 0.2},
   {0.7, 0.4},
   {0.2, 0.7}
};

void drawit()
{
   pushmatrix();
   color(WHITE);
   bgnpolygon();
      v2f(vert[0]);
```

```
        v2f(vert[1]);
        v2f(vert[2]);
    endpolygon();
    translate(0.1, 0.1, 0.0);
    color(RED);
    bgnline();
        v2f(vert[0]);
        v2f(vert[1]);
        v2f(vert[2]);
    endline();
    translate(0.1, 0.1, 0.0);
    color(GREEN);
    bgnpoint();
        v2f(vert[0]);
        v2f(vert[1]);
        v2f(vert[2]);
    endpoint();
    popmatrix();
}
/* feedback buffer is an array of floats on Personal Iris and
VGX */
Boolean floatfb()
{
    char model[12];
    Boolean isPI, isVGX;
    gversion(model);
    isPI = (strncmp(&model[4], “PI”, 2) == 0);
    isVGX = (strncmp(&model[4], “VGX”, 3) == 0);
    return (isPI || isVGX);
}

main()
{
    short sbuf[BUFSIZE];
    float fbuf[BUFSIZE];
    void *buf;
    long i, count;
    Boolean hasfloatfb;
    foreground();
    prefsize(400, 400);
    winopen(“feedback”);
    color(BLACK);
    clear();
    ortho2(0.0, 1.0, 0.0, 1.0);
    hasfloatfb = floatfb();
    drawit();
```

```
        if (hasfloatfb)
        buf = fbuf;
    else
        buf = sbuf;
    feedback(buf, BUFSIZE);
    drawit();
    count = endfeedback(buf);
    if (count == BUFSIZE) {
        printf("Feedback buffer overflow\n");
        return 1;
    }
    else
        printf("Got %d items:\n", count);
    for (i = 0; i < count; i++) {
        if (hasfloatfb)
         printf("%.2f", fbuf[i]);
        else
             printf("%d", sbuf[i]);
        if (i % 8 == 7)
            printf("\n");
        else
            printf("\t");
    }
    printf("\n");
    sleep(10);
    gexit();
    return 0;
}
```

17.5 Additional Notes on Feedback

Any graphics subroutines can be called between `feedback()` and `endfeedback()`, but only subroutines generating points, lines, polygons, cmovs, or passthroughs can generate values in the feedback buffer. If, for example, you are writing code to generate both a display and data for a plotter, certain data can be lost (polygon patterning, for example). If it is necessary to use this information in the plotting package, you should encode it somehow into `passthrough()` commands.

Also note that subroutines such as `curve()`, `patch()`, and `mesh()`, generate feedback buffer data, because they are converted in the graphics pipeline into a series of lines or polygons.