Chapter 13

Depth-Cueing and Atmospheric Effects

This chapter introduces you to two techniques that contribute to the perception of 3-D in your scenes: depth-cueing and atmospheric effects.

- Section 13.1, "Depth-Cueing,"describes how to use depth-cueing in RGB mode and in colormap mode.
- Section 13.2, "Atmospheric Effects," shows you how to create atmospheric effects such as fog and haze.

13.1 Depth-Cueing

When you look at objects in the real world, it is your eye's ability to perceive depth—*depth perception*, that makes you aware of the 3-D nature of objects and lets you judge the relative distance of objects. The illusion of depth perception can be created on the 2-D screen by *depth-cueing* images. Depth-cueing modifies an object's color based on its distance from the viewer. Figure 13-1 shows how a cube looks when it is depth-cued.

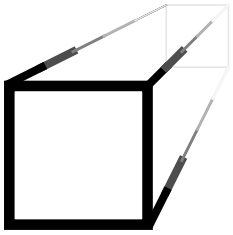

Figure 13-1 Depth-Cued Cube

Two methods of depth-cueing are presented in this chapter: color replacement and color blending.

Color replacement makes objects closer to the viewer brighter than those far away from the viewer. Depth-cueing makes an image appear 3D by replacing the color of all points, lines, and polygons with colors determined by their *z* values.

Color blending blends true object color with another color, where the blend ratio is determined by the depth of the object. You can use color blending for depth-cueing, but color blending is better known as a technique for simulating atmospheric phenomena such as fog and smoke. (See Section 13.2)

Note: Depth-cueing and lighting cannot be used simultaneously.

13.1.1 Setting Up Depth–Cueing

To set up depth-cueing in your GL application:

1. Set the proper modes:

   ```
   shademodel(GOURAUD); — this is the default
   ```

2. Define a range of *z* values that describes the viewing volume that is subject to depth-cueing.
3. Clear the z-buffer to the maximum z value and clear the screen color to the background color.
4. Turn depth–cueing on.
5. Specify a mapping of *z* values to color.
6. Draw objects that are to be depth-cued.

Defining the Boundaries for Depth-Cueing

You need to tell the GL where in your viewing volume you want objects to be depth-cued. Doing so establishes a reference for mapping the maximum and minimum color intensities. The near and far clipping planes establish the reference for the color mapping.

Figure 13-2 shows how the minimum and maximum *z* values are mapped to the brightest and dimmest colors within a viewing volume.

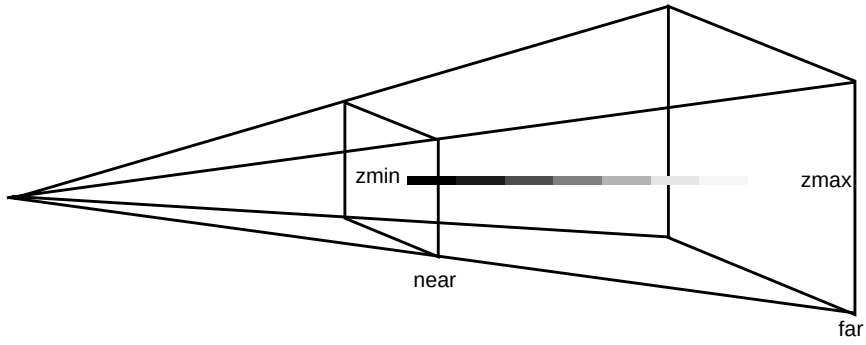

Figure 13-2 Viewing Volume Showing Clipping Planes and a Depth-Cued Line

Use the `lsetdepth()` subroutine to define the near and far *z* values that form the boundary *z* values used for depth-cueing within a viewing volume. The C specification for `lsetdepth()` is:

```
void lsetdepth(long near, long far)
```

The valid range of *near* and *far* depends on the state of the `GLC_ZRANGEMAP` compatibility mode you set. You use the `glcompat()` subroutine with the argument `GLC_ZRANGEMAP` to set the compatibility:

```
glcompat(GLC_ZRANGEMAP)
```

If `glcompat(GLC_ZRANGEMAP)` is set to 0, the valid range for *near* and *far* depends on the graphics hardware: the *z* minimum is the value returned by `getgdesc(GD_ZMIN)` and the *z* maximum is the value returned by `getgdesc(GD_ZMAX)`. If `GLC_ZRANGEMAP` is set to 1, the minimum is 0x0 and the maximum is 0x7FFFFF, and this range is mapped to whatever range the hardware supports. You should always explicitly set `glcompat(GLC_ZRANGEMAP)` because its default state depends on the machine type.

Turning Depth-Cueing On/Off

Use the `depthcue()` subroutine to turn depth-cue mode on and off. The ANSI C specification for depthcue is:

```
void depthcue(Boolean mode)
```

When you specify TRUE, all lines, points, characters, and polygons that the system draws are depth-cued. When you specify FALSE, depth-cue mode is off. Rendering in depth-cue mode may be somewhat slower, so turn off depth-cueing when you don't need it.

Querying the System for Depth-Cueing Mode

Use the subroutine `getdcm()` to query the system about whether depth-cueing is on or off. The ANSI C specification for `getdcm()` is:

```
boolean getdcm(void)
```

`TRUE` means depth-cue mode is on; `FALSE` means depth-cue mode is off.

Specify a mapping of *z* values to color by using either the `lshaderange()` or `lRGBrange()` command. In colormap mode, use `1shaderange()` to describe a mapping from *z* values to color index values. In RGB mode, use `1RGBrange()` to describe a mapping from *z* values to RGB values.

13.1.2 Depth-Cueing in Colormap Mode

When you use depth-cueing in color map mode, the GL uses the colors that you define in the *z* value mapping when it draws the geometry. You need to create a color ramp for depth-cueing in color map mode. Figure 13-3 shows a typical color ramp.

Figure 13-3 Color Ramp

Create a color ramp by specifying the color values at each end of the ramp, and how the colors are incremented.

Use the `mapcolor()` subroutine to load your color ramp into the color map. (See Chapter 4 for information on using `mapcolor()`.

Use the `lshaderange()` subroutine to define the mapping from *z* value to color.

The ANSI C specification for `lshaderange()` is:

```
void lshaderange(Colorindex lowin, Colorindex highin,
                 long znear, long zfar)
```

Specify the low-intensity color map index (*lowin*) and the high-intensity color map index (*highin*) in the `lshaderange()` subroutine. These values are mapped to the near and far *z* values that you specify for *znear* and *zfar*.

`lshaderange()` defines the entire transformation range. The brightest color is mapped to *znear* and the dimmest color is mapped to *zfar*. The color of lines or points extending beyond *znear* and *zfar* are clamped to the brightest and dimmest values respectively. Screen *z* values nearer than *znea*r map to *highin* and screen *z* values farther than *zfar* map to *lowin*.

The values of *znear* and *zfar* should correspond to or lie within the range of *z* values specified by `lsetdepth()`. If *near* < *far*, then *znear* should be less than *zfar*. If *near* > *far*, then *znear* should be greater than *zfar*. In other words, the range [*near, far*] that you define in `lsetdepth()` should bound the range [*znear, zfar*] that you define in `lshaderange()`.

The entries for the color map between *lowin* and the *highin* should reflect the appropriate sequence of intensities for the color being drawn.

When a depth-cued *point* is drawn, its *z* value is used to determine its intensity. When a depth-cued *line* is drawn, the color intensity along the line is linearly interpolated from the intensities of its endpoints, which are determined from their *z* values.

You can achieve higher resolution if the near and far clipping planes bound the object as closely as possible.

The following equation yields the color map index for a point with *a z* coordinate of z. Note that this equation yields a nonlinear mapping when *z* is outside the range of [*znear, zfar*]. Because depth-cued lines are linearly interpolated between endpoints, an endpoint outside the range of [*znear, zfar*] can result in an undesirable image.

(EQ 13-1)

$$color_z = \begin{cases} highin, & \text{if } (z \le znear) \\ \left(\frac{lowin - highin}{zfar - znear}\right)(z - znear) + highin, & \text{if } (znear \le z \le zfar) \\ lowin, & \text{if } (zfar \le z) \end{cases}$$

13.1.3 Depth-Cueing in RGBmode

When you use depth-cueing in RGB mode, the GL uses the colors that you define in the z value mapping when draws the geometry.

You use the `lRGBrange()` subroutine to set the range of colors to use for depth-cueing in RGB mode. The C specification for lRGBrange is:

```
void lRGBrange (short rmin, short gmin, short bmin, short
rmax, short gmax, short bmax, long zmin, long zmax)
```

Specify the minimum and maximum values to be stored in the color bitplanes, and the near and far *z* values to which the colors are mapped. *rmin* and *rmax* are the minimum and maximum values stored in the red bitplanes. Likewise, *gmin*, *gmax*, *bmin*, and *bmax* define the minimum and maximum values stored in the green and blue bitplanes, respectively. *znear* and *zfar* define the *z* values that are mapped linearly into the RGB range. *z* values nearer than *znear* are mapped to *rmax*, *gmax*, *and bmax*; *z* values farther than *zfar* are mapped to *rmin*, *gmin*, *and bmin*.

13.1.4 Sample Depth-Cueing Program

This sample program, *depthcue.c*, draws a cube filled with points that rotates as you move the mouse. Because the image is drawn in depth-cue mode, the edges of the cube and the points inside the cube that are closer to the viewer are brighter than the edges and points farther away.

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

#define RGB_BLACK 0x000000

#define X       0
#define Y       1
#define Z       2
#define XY      2
#define XYZ     3

#define CORNERDIST 1.8           /* center to furthest corner of cube */
#define EYEDIST 3*CORNERDIST     /* center to eye */

#define NUMPOINTS 100

float points[NUMPOINTS][XYZ];
long corner[8][XYZ] = {
    {-1, -1, -1},
    {-1, 1, -1},
    { 1, -1, -1},
    { 1, 1, -1},
    {-1, -1, 1},
    {-1, 1, 1},
    { 1, -1, 1},
    { 1, 1, 1}
};
int edge[12][2] = {
    {0, 1}, {1, 3}, {3, 2}, {2, 0},
    {4, 5}, {5, 7}, {7, 6}, {6, 4},
    {0, 4}, {1, 5}, {2, 6}, {3, 7},
};

void drawcube()
{
    int i;
```

```
    for (i = 0; i < 12; i++) {
        bgnline();
            v3i(corner[edge[i][0]]);
            v3i(corner[edge[i][1]]);
        endline();
    }
}

void drawpoints()
{
    int i;

    bgnpoint();
        for (i = 0; i < NUMPOINTS; i++)
            v3f(points[i]);
    endpoint();
    drawcube();
}
main()
{
    long maxscreen[XY];
    Device mdev[XY];
    short mval[XY];
    float rotang[XY];
    short val;
    int i;
    if (getgdesc(GD_BITS_NORM_DBL_RED) == 0) {
    fprintf(stderr, "Double buffered RGB not available on this machine \n");
    return 1;
    }
    prefsize(400, 400);
    winopen("depthcue");
    doublebuffer();
    RGBmode();
    gconfig();
    cpack(RGB_BLACK);
    clear();
    swapbuffers();
    qdevice(ESCKEY);
    maxscreen[X] = getgdesc(GD_XPMAX) - 1;
    maxscreen[Y] = getgdesc(GD_YPMAX) - 1;
    mdev[X] = MOUSEX;
    mdev[Y] = MOUSEY;
    mmode(MVIEWING);
    window(-CORNERDIST, CORNERDIST, -CORNERDIST, CORNERDIST,
             EYEDIST, EYEDIST + 2*CORNERDIST);
```

```
    lookat(0.0, 0.0, EYEDIST + CORNERDIST, 0.0, 0.0, 0.0, 0);
    /* map the current machine's z range to 0x0 -> 0x7fffff */
    glcompat(GLC_ZRANGEMAP, 1);
    /* set up the mapping of screen z values to cyan intensity */
    lRGBrange(0, 15, 15, 0, 255, 255, 0x0, 0x7fffff);
    /* have screen z values control the color */
    depthcue(TRUE);
    /* generate random points */
    for (i = 0; i < NUMPOINTS; i++) {
        points[i][X] = 2.0 * (drand48() - 0.5);
        points[i][Y] = 2.0 * (drand48() - 0.5);
        points[i][Z] = 2.0 * (drand48() - 0.5);
 }
    while (!(qtest() && qread(&val) == ESCKEY && val == 0)) {
    cpack(RGB_BLACK);
    clear();
    getdev(XY, mdev, mval);
    rotang[X] = 4.0 * (mval[Y] - 0.5 * maxscreen[Y]) / maxscreen[Y];
    rotang[Y] = 4.0 * (mval[X] - 0.5 * maxscreen[X]) / maxscreen[X];
    rot(rotang[X], 'x');
    rot(rotang[Y], 'y');
    drawpoints();
    swapbuffers();
 }
 gexit();
 return 0;
}
```

13.2 Atmospheric Effects

You can create a variety of atmospheric effects on IRIS Indigo, IRIS-4D / VGX, SkyWriter, and RealityEngine systems. Atmospheric detail adds realism to visual simulations and provides interesting depth perception effects. You can simulate fog, smoke, haze and air pollution by varying the color and density of the atmosphere in your scene. In this section, the term "fog" is used to mean any of these atmospheric conditions.

The GL creates atmospheric effects by modifying the color of the objects in your scene based on their distance from the viewer. Object color is blended with fog color to determine the apparent color perceived by the viewer.

When fog is present, objects at close range to the viewer appear in true color. True color is the color an object would be in the absence of fog—that is, the color of the object computed after lighting, shading, and texture mapping (if any). Distant objects look "washed out" as the object color is blended with the fog color, which gives the appearance of seeing the object "through" the fog. There is a certain point in the distance where the fog completely obscures the object.

As an example, consider looking down the runway at an approaching airplane. On a clear sunny day, the airplane is seen in full detail, limited only by your visual acuity (the resolution of your eye). On a foggy day, your view of the airplane is impaired and its apparent color is a combination of its true color and the fog color. As the airplane approaches, your eye begins to detect its true color.

Depth-cueing is another effect possible with fog. When you blend an object's true color with the scene background color, an object moving away from the viewer appears to fade into the background. An advantage of this method over using `depthcue()`/`lshaderange()` is that it is independent of the color of the viewed objects—you have to call `lshaderange()` each time the object color changes, whereas `fogvertex()` need only be called when the background color changes.

13.2.1 Fog

You define and enable fog with the `fogvertex()` subroutine. You call `fogvertex()` once to set up the fog parameters, then you turn the fog on with another call to `fogvertex()`. Fog is currently available only on Indigo, VGX, SkyWriter, and RealityEngine systems, so you should call `getgdesc(GD_FOGVERTEX)` to determine the fog capability of the machine when writing fog applications.

To use fog in your application:

1. Use `getgdesc()` to determine the fog capabilities of your machine.
2. Set up the proper modes for fog:

   ```
   RGBmode()
   mmode(MVIEWING)
   ```

   Note: Remember to call `gconfig()` if you change to RGB mode from colormap mode.

3. Define the fog characteristics.

   ```
   fogvertex(mode, params)
   ```

   Note: Defining a fog effect does not enable it. This must be done with a separate call.

4. Turn fog on.

   ```
   fogvertex(FG_ON, dummy)
   ```

The ANSI C specification for `fogvertex()` is:

```
void fogvertex(long mode,float *params);
```

You use the mode argument to indicate whether you are defining, enabling, or disabling fog effects. You specify the fog characteristics in the *params* array. *params* is an array of floating point values.

13.2.2 Fog Characteristics

Fog characteristics define the color and density of the fog. You can specify *uniformly distributed fog*, where the fog has a uniform density throughout, or you can specify *linearly blended (interpolated) fog*, where the fog begins at a certain point and becomes so dense that it is opaque in the distance.

For uniformly distributed fog, you specify a fog density between 0.0 and 1.0 and the fog color. A fog density of 0.0 represents no fog at all—the object's apparent color is the same as its true color. Increasing positive values increase the fog density. The maximum fog density is 1.0. For a fog density of 1.0, fog totally obscures the true color of the viewed object at a distance of one unit in eye coordinates. This is the reference to which fog density values are normalized.

The proportion of the object's true color that contributes to the apparent color is called the *blend factor*. When you enable fog effects, the blend factor is computed at the vertices of each graphics primitive. The vertex blend factors are then interpolated to determine the blend factor at the interior pixels of the graphics primitive.

SkyWriter and RealityEngine systems allow you to specify per-pixel fog calculations, which is more accurate than interpolation. To maximize the accuracy of the fog, minimize the ratio of the distance to the far clipping plane to the distance to the near clipping plane. In other words, you specify near and far such that the near and far clipping panes are as close together as possible. In addition, you need to specify the maximum `lsetdepth()` range for your machine by calling `lsetdepth(getgdesc(GD_ZMIN), getgdesc(GD_ZMAX))`.

13.2.3 Fog Calculations

The blend factor (*fog*) is calculated according to one of the three equations that follow. The first two equations calculate *fog* in eye coordinates for uniformly distributed fog. The third equation calculates fog varying linearly with distance, given the distance at which the fog begins and the distance at which the fog becomes opaque. Each of these methods allow you to indicate per-vertex or per-pixel fog calculations.

Exponential fog: (EQ 13-2)

$$fog = (1 - e)^{(5.5 \cdot density \cdot Z_{eye})}$$

Exponential-squared fog: (EQ 13-3)

$$fog = (1 - e)^{(-5.5 \cdot (density \cdot Z_{eye})^2)}$$

Linear fog: (EQ 13-4)

$$fog = 1 - \frac{(end\_fog + Z_{eye})}{(end\_fog - start\_fog)}$$

where:

fog	is the computed fog blending factor ($0 \le fog \le 1$).
density	is the fog density.
Z_{eye}	is the eye space Z coordinate (always negative) of the pixel or vertex being fogged.
start_fog	is the distance from the viewer where the fog begins to appear.
end_fog	is the distance from the viewer where the fog becomes opaque.

The pixel color, C_p, is combined with fog color, C_f, to give apparent color, C:

(EQ 13-5)

$$C = C_p \cdot (1 - fog) + C_f \cdot fog$$

where:

fog	is the computed fog blending factor, ($0 \le fog \le 1$).
C	is the resultant color.
C_p	is the incoming pixel color, which may be Gouraud or flat shaded and possibly textured.
C_f	is the fog color.

13.2.4 Fog Parameters

Based on the effect you want to achieve, you enter one of the following symbolic constants for *mode* in `fogvertex()`:

`FG_VTX_EXP` Fog is computed at each vertex of the primitive (EQ 13-2).

`FG_PIX_EXP` Fog is computed at each pixel of the primitive (EQ 13-2).

`FG_VTX_EXP2` Fog is computed at each vertex of the primitive (EQ 13-3).

`FG_PIX_EXP2` Fog is computed at each pixel of the primitive, (EQ 13-3).

`FG_VTX_LIN` Fog is computed at each vertex of the primitive (EQ 13-4).

`FG_PIX_LIN` Fog is computed at each pixel of the primitive (EQ 13-4).

To enable or disable fog effects, use the following:

`FG_ON` Enable the previously defined fog effect.

`FG_OFF` Disable fog effects. This is the default.

You specify four floating point values in the *params* array for uniformly distributed fog (`FG_VTX_EXP`, `FG_PIX_EXP`, `FG_VTX_EXP2`, or `FG_PIX_EXP2`):

density Density(thickness) of fog ($0.0 \leq density \leq 1.0$).

r Red component of fog ($0.0 \leq r \leq 1.0$).

g Green component of fog ($0.0 \leq g \leq 1.0$).

b Blue component of fog ($0.0 \leq b \leq 1.0$).

You specify five values in the params array for linearly blended fog (`FG_VTX_LIN`, or `FG_PIX_LIN`):

start_fog Distance in eye coordinates to start of fog.

end_fog Distance in eye coordinates where fog becomes completely opaque.

r Red component of fog ($0.0 \leq r \leq 1.0$).

g Green component of fog ($0.0 \leq g \leq 1.0$).

b Blue component of fog ($0.0 \leq b \leq 1.0$).