

**CS-4470**  
**Image Synthesis**

**Part 5**

**Shadows**

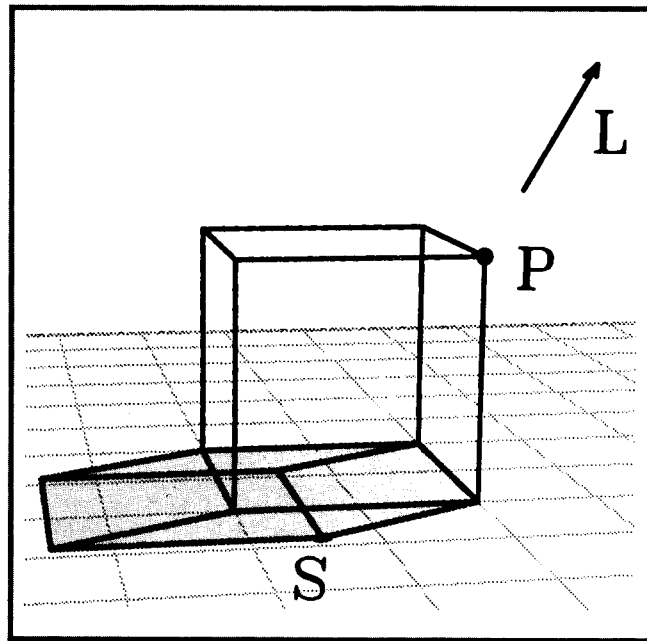
**Notes and Programs by:**

**Dr. Michael J. Zyda**

Fake Shadows 3  
Fake Shadows continued 4  
Fake Shadows continued 5  
Sample Code Sketch for Fake Shadows 6  
A Two-Pass Z-Buffer Shadow Algorithm 13  
A Two-Pass Z-Buffer Shadow Algorithm continued 14  
A Two-Pass Z-Buffer Shadow Algorithm continued 15  
Sample IRISGL Code for the Two-Pass Shadow Algorithm 16

## Fake Shadows

-- What we would like to be able to do is cast a shadow on a ground plane, i.e. cast a shadow of our cube rotating onto our checkerboard.



**Figure 1. Light source at infinity.**

-- The most common trick is to redraw the objects of the scene scaled by zero in the vertical direction (y coordinate).

```
buildnonmovingviewmatrix();  
drawthefloor();  
buildmovingviewmatrix();  
drawcube();  
buildshadowviewmatrix();    /* Moving matrix • M as below */  
drawcube(Half-toned and Grey);
```

## Fake Shadows continued

-- Let's look at the mathematics necessary for Simple Shadows...

### Single Point Source Infinitely Far Away

-- we have  $\mathbf{L} = (x_l, y_l, z_l)$  in the direction of the light source.

-- a point  $P(x_p, y_p, z_p)$  on some object of the scene will cast a shadow on the ground plane at  $(x_s, 0, z_s)$ .

-- the cast shadow starts at  $P$  and moves away from the light source in the opposite direction until it hits  $y = 0$  (the ground).

i.e.  $S_{\text{shadow}} = P_{\text{point}} - \frac{y_p}{y_l} \mathbf{L}_{\text{light}}$

Solve for  $S_{\text{shadow}}$  with the requirement that  $y_s = 0$ .

$$0 = y_p - \frac{y_p}{y_l} y_l$$

or

$$= y_p / y_l$$

## Fake Shadows continued

-- How do we compute the x and z coordinates?

$$x_s = x_p - (y_p/y_1) \cdot x_1$$

$$z_s = z_p - (y_p/y_1) \cdot z_1$$

-- How do we write this as a matrix operation?

$$P' = P \cdot M$$

$$P' = [x_s \ 0 \ z_s \ 1]$$

$$P = [x_p \ y_p \ z_p \ 1]$$

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ X & 0 & Z & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where

$$X = -x_1/y_1$$

and

$$Z = -z_1/y_1$$

## Sample Code Sketch for Fake Shadows

```
//
// draw_the_scene
// This function draws the picture.
//

void draw_the_scene()
{
    // Set the background color to cyan (RGBA) and clear the z-buffer.
    glClearColor(0.0, 1.0, 1.0, 0.0);
    glClear(GL_DEPTH_BUFFER_BIT | GL_COLOR_BUFFER_BIT);

    // Build the accumulative rotation matrices.
    // This computation stomps the top of the ModelView stack.
    build_accumulative_matrices();

    // Set the projection matrix.
    // The near and far values are distances from the viewer
    // to the near (1.0) and far (10000.0) clipping planes.
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0, 1.25, 1.0, 10000.0);

    // Load the ModelView matrix with a unit matrix.
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // We are at the viewpoint and looking towards
    // the reference point of the object.
    // Up vector is the vector orienting our view volume around
    // the line of sight.
    gluLookAt(0.0, 0.0, 200.0, // View point.
              0.0, 0.0, -511.0, // Ref point, point we are looking towards.
              0.0, 1.0, 0.0 ); // Up vector.

    // The floor does not move so draw it now.
    // x (0.0) and z (-511.0) are the center of the floor.
    // y (-200.0) is the plane for the floor.
    // The last value is the width of the floor (500.0).
    drawthefloor(0.0, -200.0, -511.0, 500.0);

    // Concatenate onto the ModelView matrix the results of our
    // accumulative transformations.
    // The coordinate passed in is the point about which
    // we would like our operations (rotations/scales) to occur.
    // Think of this point as the center of the displayed object.
    concatenate_accumulative_matrices(0.0, 0.0, -511.0);
}
```

```

/*****
/***** NEW SHADOW CODE START *****/
/*****

// Disable z-buffering
glDepthMask(GL_FALSE);

// Enable blending for shadow
glEnable(GL_BLEND);

// Set blending parameters for source (shadow)
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);

// Set shadow color to black
glColor4f(0.0, 0.0, 0.0, 0.4);

// Push a copy of the top matrix
glPushMatrix();

// Set the projection matrix.
// The near and far values are distances from the viewer
// to the near (1.0) and far (10000.0) clipping planes.
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluPerspective(45.0, 1.25, 1.0, 10000.0);

// Load the ModelView matrix with a unit matrix.
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();

// We are at the viewpoint and looking towards
// the reference point of the object.
// Up vector is the vector orienting our view volume around
// the line of sight.
gluLookAt(0.0, 0.0, 200.0, // View point.
          0.0, 0.0, -511.0, // Ref point, point we are looking towards.
          0.0, 1.0, 0.0 ); // Up vector.

// Translate the shadow down to the floor level
glTranslatef(0.0, -199.9, 0.0);

// Draw a flat object to simulate the shadow. Since we are scaling
// y to 0, we also get a translation to the x-z plane automatically.
glScalef(1.0, 0.0, 1.0);

// Translate the center of cube back to original location
glTranslatef(REFX, REFY, REFZ);

glMultMatrixf(transacc_ptr);
glMultMatrixf(rxacc_ptr);
glMultMatrixf(ryacc_ptr);
glMultMatrixf(rzacc_ptr);
glMultMatrixf(scaleacc_ptr);

```

```

// Translate cube to the origin
glTranslatef(-(REFX), -(REFY), -(REFZ));

// Draw the shadow cube
drawshadowcube(0.0, 0.0, -511.0, 100.0);

// Restore the top of the matrix
glPopMatrix();

// Disable blending for shadow
glDisable(GL_BLEND);

// Enable z-buffering
glDepthMask(GL_TRUE);

/*****
/***** NEW SHADOW CODE END *****/
/*****

// Draw the cube.
// Anything drawn after the concatenate moves with the accumulative
// transformations.
drawcube(0.0, 0.0, -511.0, 100.0);

// Swap the buffers as we are doing double buffering.
glXSwapBuffers(global_display, global_window);

} // end of draw_the_scene

```

```

/*****
/***** NEW SHADOW CODE START *****/
/*****

// Draw a cube shadow with center (x,y,z), having the designated sidelength.
// Function same as drawcube, but without any glColor() commands.

void drawshadowcube(float x, float y, float z, float sidelength)

// float x,y,z;      center of the cube in 3-space.

// float sidelength; length of cube's side.

{

    float halfside; // length of half the side of the cube.

    float p[4][3]; // array to hold coords for the cube faces.

    // Compute the halfside.
    halfside=sidelength/2.0;

    // Back face.
    p[0][0]=x-halfside;
    p[0][1]=y+halfside;
    p[0][2]=z-halfside;

    p[1][0]=x+halfside;
    p[1][1]=y+halfside;
    p[1][2]=z-halfside;

    p[2][0]=x+halfside;
    p[2][1]=y-halfside;
    p[2][2]=z-halfside;

    p[3][0]=x-halfside;
    p[3][1]=y-halfside;
    p[3][2]=z-halfside;

    glBegin(GL_QUADS);
        glVertex3fv(p[0]);
        glVertex3fv(p[1]);
        glVertex3fv(p[2]);
        glVertex3fv(p[3]);
    glEnd();

    // draw the front face.
    p[0][0]=x+halfside;
    p[0][1]=y+halfside;
    p[0][2]=z+halfside;

    p[1][0]=x-halfside;

```

```

p[1][1]=y+halfside;
p[1][2]=z+halfside;

p[2][0]=x-halfside;
p[2][1]=y-halfside;
p[2][2]=z+halfside;

p[3][0]=x+halfside;
p[3][1]=y-halfside;
p[3][2]=z+halfside;

glBegin(GL_QUADS);
    glVertex3fv(p[0]);
    glVertex3fv(p[1]);
    glVertex3fv(p[2]);
    glVertex3fv(p[3]);
glEnd();

// Draw the top.
p[0][0]=x-halfside;
p[0][1]=y+halfside;
p[0][2]=z-halfside;

p[1][0]=x-halfside;
p[1][1]=y+halfside;
p[1][2]=z+halfside;

p[2][0]=x+halfside;
p[2][1]=y+halfside;
p[2][2]=z+halfside;

p[3][0]=x+halfside;
p[3][1]=y+halfside;
p[3][2]=z-halfside;

glBegin(GL_QUADS);
    glVertex3fv(p[0]);
    glVertex3fv(p[1]);
    glVertex3fv(p[2]);
    glVertex3fv(p[3]);
glEnd();

// Draw the bottom.
p[0][0]=x-halfside;
p[0][1]=y-halfside;
p[0][2]=z-halfside;

p[1][0]=x+halfside;
p[1][1]=y-halfside;
p[1][2]=z-halfside;

p[2][0]=x+halfside;

```

```

p[2][1]=y-halfside;
p[2][2]=z+halfside;

p[3][0]=x-halfside;
p[3][1]=y-halfside;
p[3][2]=z+halfside;

glBegin(GL_QUADS);
    glVertex3fv(p[0]);
    glVertex3fv(p[1]);
    glVertex3fv(p[2]);
    glVertex3fv(p[3]);
glEnd();

// Draw the left side.
p[0][0]=x-halfside;
p[0][1]=y-halfside;
p[0][2]=z-halfside;

p[1][0]=x-halfside;
p[1][1]=y-halfside;
p[1][2]=z+halfside;

p[2][0]=x-halfside;
p[2][1]=y+halfside;
p[2][2]=z+halfside;

p[3][0]=x-halfside;
p[3][1]=y+halfside;
p[3][2]=z-halfside;

glBegin(GL_QUADS);
    glVertex3fv(p[0]);
    glVertex3fv(p[1]);
    glVertex3fv(p[2]);
    glVertex3fv(p[3]);
glEnd();

// Draw the right side.
p[0][0]=x+halfside;
p[0][1]=y-halfside;
p[0][2]=z-halfside;

p[1][0]=x+halfside;
p[1][1]=y+halfside;
p[1][2]=z-halfside;

p[2][0]=x+halfside;
p[2][1]=y+halfside;
p[2][2]=z+halfside;

p[3][0]=x+halfside;

```

```
p[3][1]=y-halfside;
p[3][2]=z+halfside;

glBegin(GL_QUADS);
    glVertex3fv(p[0]);
    glVertex3fv(p[1]);
    glVertex3fv(p[2]);
    glVertex3fv(p[3]);
glEnd();

// the cube is drawn.

}

/*****
/***** NEW SHADOW CODE END *****/
/*****/
```

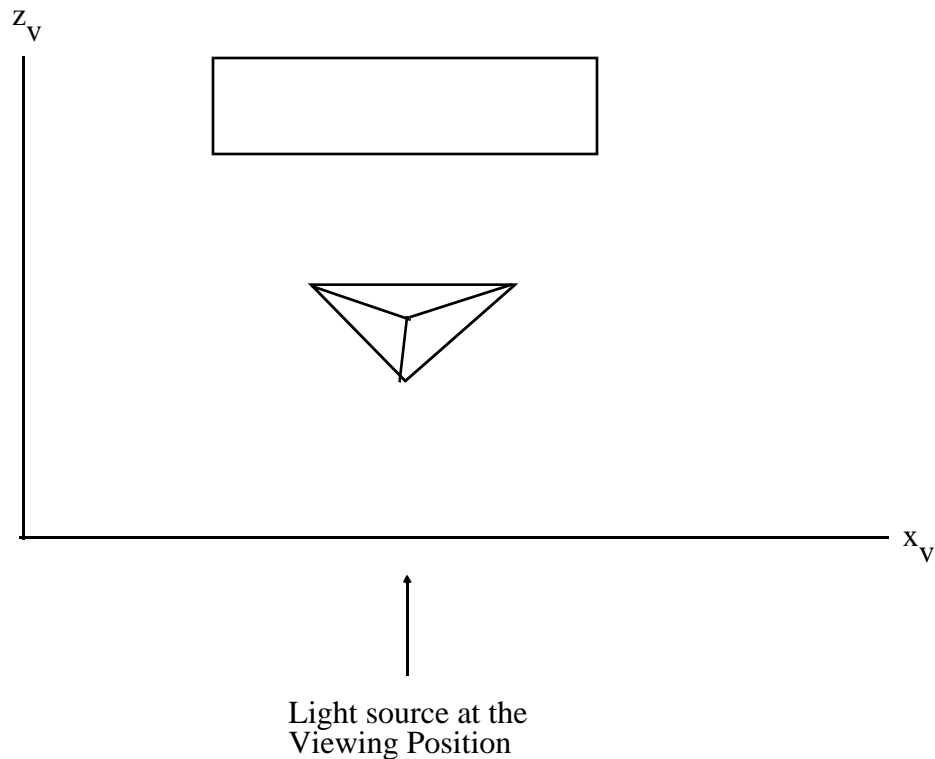
## A Two-Pass Z-Buffer Shadow Algorithm

-- Lance Williams (SIGGRAPH '78, pg. 270-274) developed a shadow-generation method based on two passes through a z-buffer algorithm, one for the viewer and one for the light source.

-- His algorithm determines whether a surface is shadowed by using image precision calculations.

**Step 1:** A view of the scene is constructed from the point of view of the light source.

-- Only the z-values and not the shading values need to be computed and stored.



## A Two-Pass Z-Buffer Shadow Algorithm continued

**Step 2:** A view of the scene is then constructed from the point of view of the observer's eye using a z-buffer with the following modification.

-- Whenever a pixel is determined to be visible, its object-precision coordinates in the observer's view  $(x_0, y_0, z_0)$  are transformed into coordinates in the light source's view  $(x_0', y_0', z_0')$ .

-- The transformed coordinates  $x_0'$  and  $y_0'$  are used to select the value  $z_1$  in the light source's z-buffer to be compared with the transformed value  $z_0'$ .

-- If  $z_1$  is closer to the light than  $z_0'$ , then there is something blocking the light from the point, and the pixel is shaded as being in shadow. Otherwise the point is visible from the light and it is shaded as lit.

-- We can think of the light's z-buffer as a shadow map (similar to a texture map). Multiple light sources can be accommodated by use of a separate shadow map for each light source.

-- Here is the same information again, with the math spelled out...

-- A linear transformation exists which maps  $(x, y, z)_{\text{observer}}$  into  $(x, y, z)_{\text{light source view}}$

$$\text{i.e. } P_{\text{light\_source\_view}} = P_{\text{observer}} \cdot M$$

-- As each  $P_{\text{observer}}$  is generated, it is transformed by  $M$  and tested for visibility to the light source using the z-values<sub>light\_source</sub>.

-- if  $P_{\text{observer}}$  as transformed is not visible to the light source, it is in shadow and should be shaded accordingly.

-- i.e. given the ambient coloring.

### **A Two-Pass Z-Buffer Shadow Algorithm continued**

-- Like the regular z-buffer visible surface algorithm, this algorithm requires that each rendered pixel be shaded.

-- Here this means that shadow calculations must be performed for the pixel, even if its ultimately painted over by closer objects.

## Sample IRISGL Code for the Two-Pass Shadow Algorithm

```
/**                                                    **/  
/** shad.c -                                          **/  
/**     A simple demo of real-time shadows.         **/  
/**                                                    **/  
/**         Rolf van Widenfelt - July 1992          **/  
/**             also hacked by                      **/  
/**         Paul Haeberli - 1992                   **/  
/**             thanks for help from                **/  
/**         Mark Segal - 1992                       **/  
/**                                                    **/  
/** $Revision: 1.9 $                                  **/  
/**                                                    **/  
/** COMMENTED BY:  Matthew Johnson & Christopher Upson **/  
/**                                                    **/  
/** Methodology:  Texture Mapping.  See 'Fast Shadows and Lighting Effects **/  
/**                Using Texture Mapping' by Segal, Korobkin, van Widenfelt, **/  
/**                Foran, Haeberli, COMPUTER GRAPHICS, 2 July 1992.      **/  
/**                                                    **/  
/** Original Code Located In:  ~upsonc/fastshadows  **/  
/**                                                    **/  
/  
*****  
*/  
/  
*****  
*/  
  
#include "stdio.h"  
#include "string.h"  
#include "math.h"  
#include "gl.h"  
#include "gl/device.h"  
#include "gl/image.h"  
#include "vect.h"  
#include "sgiobj.h"  
  
#define NLIGHTS 3          /* Number of moveable lights */  
#define SSIZE 512/* Screen size in pixels square; */  
                /* must be power of 2 */  
  
#define MENU_LIGHTVIEW1  
#define MENU_EYEVIEW 2  
#define MENU_TOGGLECOLOR3  
#define MENU_DISCO4  
#define MENU_QUIT 5  
  
#define SHADZNEAR(0x0000)    /* The near clipping plane */  
#define SHADZFAR(0xffff)    /* The far clipping plane */  
#define BACKCOLOR(0x002020) /* The background color */  
  
#define WINTITLESTR "shad v2"      /* The window title */
```

```

/* Function definitions */
void shadowdisplacez(int mode);
float *maketexprops(int do bicubic,int do fastread);
float *maketevprops(int do alpha);
void fromlight(float mat[4][4]);
void myrectf(float x1,float y1,float x2,float y2,float z);
char *computetitle();

/* Mouse and trackball functions */
typedef struct {
    float ztrans;           /* for zooming in and out          */
    float trotx, troty;    /* for rotating about the x and y axes */
    float wiggle;         /* only used for disco mode        */
} TrackState;           /* the state of the mouse          */

/* Function definitions */
float fgetmousex();      /* Get the mouse's x coord location */
float fgetmousey();     /* Get the mouse's y coord location */
void mytrackztrans(TrackState *ts,float z); /* sets the z translation */
void mytrackrot(TrackState *ts,float x,float y); /* sets the x & y rotations */
void mytrackclick(TrackState *ts); /* gets mouse x & y values and checks for
/* middle mouse button status */
void mytrackpoll(TrackState *ts); /* computes rotations and translation */

/* computes transformation & inverse transformation matrices for normal mode */
void mygettracktransform(TrackState *ts,float mat[4][4],float invmat[4][4]);

/* computes transformation & inverse transformation matrices for disco mode */
void mygetdiscotransform(TrackState *ts,TrackState *ts2,float mat[4][4],
    float invmat[4][4]);

int nobjs;           /* Number of objects in the scene */
sgiobj *obj[100];    /* Array allocated to store objects */
long wxsize;        /* Window size */
long wysize;
long wxorg;         /* Window origin */
long wyorg;
long menu, lmenu;

long ZNear;         /* Near clipping plane */
long ZFar;         /* Far clipping plane */

int dolightview = 0; /* Eyeview or lightview toggle */
int do bicubic = 0; /* For slower, but more accurate, appealing shadows */
int do zdisplace = 1; /* Flag for 'displacepolygon' command use for using
/* z-buffer as a texture for generating shadows. */
int do fastread = 1; /* Flag that is reset to 0 unless user inputs -f */
int coloredlights = 0; /* Colored lights toggle */
int do tracklight = 0; /* Mouse button moves light */
int curlightnum = 0; /* By default, light 1 is edited w/left shift key hack */
int do drawlights = 1; /* If set, the lights are drawn. */
int do disco = 0; /* "disco" mode toggle */
float disco_time; /* Total time in disco mode. Reset when = 10,000 */

```

```

float disco_step = .03; /* Time step per frame - should be a lot less than 1 */

float light_fov = 20.0; /* Field of view angle of the light */
float light_near = 0.1; /* Distance from light to closest clipping plane */
float light_far = 10.0; /* Distance from light to farthest clipping plane */

float eyemat[4][4]; /* Transformation matrix for the eyeview */
float inveymat[4][4]; /* Inverse transformation matrix for the eyeview */
float eye_dz = -0.0015; /* Eye view z displacement */
float eye_fov = 20.0; /* Field of view angle of the viewer */
float eye_near = 0.1; /* Distance from viewer to closest clipping plane */
float eye_far = 10.0; /* Distance from viewer to farthest clipping plane */

TrackState eyetrack[1]; /* Mouse location values for the eye view. */

typedef struct { /* Holds shadow map for indicated light source */
    int texno; /* Texture number */
    float r, g, b; /* The light's color */
    TrackState track[1]; /* Mouse values in normal mode */
    float mat[4][4]; /* Transformation matrix */
    float invmat[4][4]; /* Inverse transformation matrix */
    float texm[4][4]; /* Texture transformation matrix */
    TrackState disco_track[1]; /* Mouse values in disco mode */
    int disco_changed; /* Disco mode indicator */
    unsigned long smap[(SSIZE*SSIZE)/2]; /* Shadow map */
} LightBuffer;

void newshadowmap(LightBuffer *lightsource); /* Create shadow map for
/* indicated light. */

void drawlight(LightBuffer *lt); /* Draw the indicated light */

LightBuffer thelights[8]; /* Array for the 4 lights; colored & uncolored */
LightBuffer *curlight; /* The light being moved around */

float Identity[4][4] = { /* Identity Matrix */
    1, 0, 0, 0,
    0, 1, 0, 0,
    0, 0, 1, 0,
    0, 0, 0, 1,
};

```

```

/
*****
*/
/* setuplight - */
/* Set up the lights for the scene. Control latitude with rotx and longitude */
/* with roty. Translate away from object center with a negative tz. */
/
*****
*/
void
setuplight(LightBuffer *lightsource,float rotx,float roty,float tz)
{
    pushmatrix(); /* Put another copy of top matrix on top of stack. */
    loadmatrix(Identity); /* Replace top copy with identity matrix.
*/
    pushmatrix(); /* Another copy of identity matrix on top of stack. */
    translate(0.0,0.0,tz); /* Translate to 'tz'. */
    rot(rotx,'x'); /* Rotate 'rotx' radians around the x axis. */
    rot(roty,'y'); /* Rotate 'roty' radians around the y axis. */
    getmatrix(lightsource->mat); /*Place above generated matrix in light-
source.*/
    popmatrix(); /* Pop the above generated matrix off of the stack. */
    rot(-roty,'y'); /* Now, with second identity matrix on top of stack,*/
    rot(-rotx,'x'); /* perform inverse x and y rotations and z trans. */
    translate(0.0,0.0,-tz);
    getmatrix(lightsource->invmat); /* Place inverse matrix in lightsource. */
    popmatrix(); /* Pop the above inverse matrix off of stack. */
    lightsource->track->ztrans = tz; /* Store z translation and the x and y */
    lightsource->track->trotx = rotx; /* rotations in the lightsource. */
    lightsource->track->troty = roty;
    setlightcolor(lightsource,1.0,1.0,1.0); /* Store full white as the color */
} /* emitted by the light source */

/
*****
*/
/* setlightcolor - */
/* Set the color emitted by the lightsource to the indicated values. */
/
*****
*/
setlightcolor(lightsource,r,g,b)
LightBuffer *lightsource;
float r, g, b;
{
    lightsource->r = r;
    lightsource->g = g;
    lightsource->b = b;
}

```

```

/
*****
*/
/* positionlight -                                     */
/* Position the light according to its inverse matrix. */
/
*****
*/
positionlight(invmat)
float invmat[4][4];
{
    pushmatrix();          /* Puts another copy of top matrix on stack. */
    multmatrix(invmat);    /* Multiply top matrix by light's inverse matrix*/
    mymakelight(0,0.0,0.0,0.0); /* Assign lighting properties to light 0. */
    popmatrix();          /* Pop the above generated matrix off of stack. */
}

/
*****
*/
/* mymakelight -                                     */
/* Define the lighting properties of the given light. */
/
*****
*/
mymakelight(i,x,y,z)
int i;
float x, y, z;
{
    float li_desc[14];

    li_desc[0] = AMBIENT; /* Ambient light associated with this source */
    li_desc[1] = 0.1;     /* In this case, the ambience is a dull white */
    li_desc[2] = 0.1;
    li_desc[3] = 0.1;
    li_desc[4] = LCOLOR; /* Specifies the color and intensity of the */
    li_desc[5] = 1.0;     /* light that is emitted form the light source*/
    li_desc[6] = 1.0;     /* In this case, it is a full white light. */
    li_desc[7] = 1.0;
    li_desc[8] = POSITION; /* The positon of the light in the scene is at*/
    li_desc[9] = x;       /* the indicated x,y and z coordinates. The */
    li_desc[10] = y;      /* light is a local light. */
    li_desc[11] = z;
    li_desc[12] = 1.0;
    li_desc[13] = LMNULL;
    lmdef(DEFLIGHT,i+1,14,li_desc); /* Define this light source */
    lmbind(LIGHT0+i,i+1); /* Enable this light (turn it on) */
}

```

```

/
*****
*/
/* lfunc - */
/* */
/
*****
*/
lfunc(n)
{
    n = n-1;
    if(n>=0 && n<NLIGHTS) {
        if(curlight)
            newshadowmap(curlight);
        curlight = &thelights[n];
        dolightview = 1; /* Set view from light toggle */
        curlightnum = n; /* Set current light for editing */
        wintitle(computetitle());
    }
    return 0;
}

```

```

#define LSCALE (0.25) /* Scale for normal lights */
#define CLSCALE(0.75) /* Scale for colored lights */
#define SPECSCALE (0.50) /* Scale for specular component of the light. */

/
*****
*/
/* setlcolors - */
/* Depending on value of coloredlights toggle, change the emission colors of */
/* the lights accordingly. */
/
*****
*/
setlcolors()
{
    if(coloredlights) {
        setlightcolor(thelights+0,1.0*CLSCALE,0.0*CLSCALE,0.0*CLSCALE);
        setlightcolor(thelights+1,0.0*CLSCALE,0.6*CLSCALE,0.0*CLSCALE);
        setlightcolor(thelights+2,0.0*CLSCALE,0.0*CLSCALE,1.0*CLSCALE);
        setlightcolor(thelights+3,1.0*CLSCALE,0.5*CLSCALE,0.0*CLSCALE);
    } else {
        setlightcolor(thelights+0,1.0*LSCALE,1.0*LSCALE,0.6*LSCALE);
        setlightcolor(thelights+1,1.0*LSCALE,0.6*LSCALE,1.0*LSCALE);
        setlightcolor(thelights+2,0.6*LSCALE,1.0*LSCALE,1.0*LSCALE);
        setlightcolor(thelights+3,1.0*LSCALE,1.0*LSCALE,1.0*LSCALE);
    }
}

```

```

/
*****
*/
/***** MAIN
*****/
/
*****
*/

main(argc, argv)
int argc;
char *argv[];
{
    short val;
    int dev, i;          /* dev = identifier for input device read.  */
    int firstobj = 1;   /* Counter for objects input on command line.*/
    float *tevps;      /* Texture environment properties.          */
    int needredraw;    /* Flag indication need to redraw the scene. */
    TrackState *track; /* Stores mouse positions for movements.     */

    if(argc<2) { /* If user forgets to input objects for the scene */
        fprintf(stderr,"usage: shadow obj1.sgo [obj2.sgi obj3.sgo . . .\n");
        exit(1);
    }
    if (!strcmp(argv[1],"-f")) {
        dofastread = !dofastread;
        firstobj++; /* hack for cmd line parsing */
        printf("dofastread %d\n",dofastread);
    }
    minsize(SSIZE,SSIZE); /* Sets minimum size of graphics window.      */
    keepaspect(5,4);      /* Sets x to y aspect ratio for graphics window.*/
    winopen("shad");      /* Create a graphics window called 'shad'.      */
    wintitle(computetitle()); /* Add a title bar to graphics window (GW).    */
    getorigin(&wxorg, &wyorg); /* Returns position of lower left corner of GW. */
    getsize(&wxsize, &wysize); /* Returns dimensions of the graphics window.  */
    matrixinit();         /* Places identity matrix on top of ModelView. */
    shadeinit();          /* Initializes obj material & lights for scene. */

    ZNear = getgdesc(GD_ZMIN); /* Returns the min and max depth values that */
    ZFar = getgdesc(GD_ZMAX); /* can be stored in the z-buffer of the      */
                               /* normal framebuffer                        */

    /* Queue devices to be used for input so that events occurring within the */
    /* device are entered into the event queue, a time ordered list of input */
    /* events.                                                                    */
    qdevice(LEFTMOUSE);
    qdevice(MIDDLEMOUSE);
    qdevice(RIGHTMOUSE);
    qdevice(LEFTSHIFTKEY);
    qdevice(LEFTCTRLKEY);
    qdevice(LEFTALTKEY);
    qdevice(TABKEY);
    qdevice(SPACEKEY);
    qdevice(ESCKEY);

```

```

qdevice(QKEY);
qdevice(WKEY);

/* Define two popup menus */
lmenu = defpup("Light Views %t %F|light 1|light 2|light 3",lfunc);
menu = defpup("Options %t|light views %m|eye view|colored lights toggle
             |disco mode|quit",lmenu);

RGBmode();          /* Set rendering and display mode to bypass color map */
doublebuffer();    /* Set display mode to double buffer mode          */
gconfig();         /* Reconfigures the GL modes of the current window          */

subpixel(TRUE);    /* Positions screen vertices exactly; does not force          */
                  /* them to the center of pixels.                                */
zbuffer(TRUE);     /* Enable z-buffer operation to store depth values            */
                  /* for pixels in the framebuffer and for depth maps.          */
zfunction(ZF_EQUAL); /* Function passes if incoming pixel z value is              */
                  /* < or = to z value currently stored.                          */
pixmode(PM_SIZE,16); /* Set # of bits per pixel to 16 for reads & writes          */
readsource(SRC_ZBUFFER); /* Sets source for pixels to the z-buffer                    */

cpack(0);          /* Set RGBA color to black with full transparency for        */
                  /* the framebuffer.                                            */
clear();           /* Clear the screen.                                          */
swapbuffers();    /* Exchange front & back buffers so back will be drawn      */
nobjs = 0;
for(i=firstobj; i<argc; i++) { /* Reads in the objects to be shadowed */
    obj[nobjs] = readsgiobj(argv[i]); /* from the command line.              */
    nobjs++;
}

mytrackztrans(eyetrack,-2.0); /* Eyeview z-trans set to -2.0          */
mytrackrot(eyetrack,0.0,0.0); /* Eyeview x and y rotation set to 0.0 */

setuplight(thelights+0, 30.0, 0.0,-1.5);/* Set up the lights at the given */
setuplight(thelights+1, 40.0,120.0,-1.5);/* location and orientation        */
setuplight(thelights+2, 50.0,240.0,-1.5);
setuplight(thelights+3, 60.0,270.0,-1.5);/* Choose z to avoid clipping      */
setlcolors(); /* Set colors of light to dull initially */
/*frontbuffer(1);*/

tevps = maketevprops(0);/* Define the properties of the texture environment*/
tevdef(1,0,tevps);      /* Define texture mapping environment w/above props*/
tevbind(0,1);          /* Bind/activate the above defined texture env      */

gennewshadowmaps();    /* Generate shadow maps for each light */

curlight = 0;          /* None of the 3 moveable lights is being moved */
drawit();              /* Draw initial scene */

```

```

while (1) {
    /* main loop */
    while (qtest()) {
        /* while there are events in the event queue */
        dev = qread(&val); /* get the event at the head of the queue */
        switch (dev) {
            /* do the following depending on the event */
            case QKEY: /* Reduce the z-displacement of the eye view.*/
                if(val) {
                    eye_dz -= 0.0001;
                    fprintf(stderr,"eye_dz is %g\n",eye_dz);
                    needredraw = 1;
                }
                break;
            case WKEY: /* Increase the z-displacement of the eye view.*/
                if(val) {
                    eye_dz += 0.0001;
                    fprintf(stderr,"eye_dz is %g\n",eye_dz);
                    needredraw = 1;
                }
                break;
            case LEFTMOUSE:
                if(val && (!dodisco || !dolightview)) {
                    /* Move the last light selected in lightview around */
                    /* the object with the mouse while LEFTSHIFTKEY is */
                    /* depressed. */
                    /*
                    if (!dolightview && getbutton(LEFTSHIFTKEY)) {
                        dotracklight = 1; /* Track light's position.*/
                        wintitle(computetitle()); /* Display window title. */
                        curlight = &thelights[curlightnum]; /* Current light */
                        /* is last light selected.*/
                        mytrackclick(curlight->track); /* Get mouse x & y positions. */
                        while(getbutton(LEFTMOUSE)) { /* While left mouse button is */
                            mytrackpoll(curlight->track); /* depressed, update light's */
                            drawit(); /* position and draw scene. */
                        }
                        dotracklight = 0; /* Cease tracking the light, */
                        wintitle(computetitle());
                        curlight = 0; /* set current light to default */
                        needredraw = 1; /* and set redraw the scene flag. */
                    } else {
                        /* Rotate viewpoint around the object using mouse. */
                        /* Viewpoint moved is either the eyeview or one */
                        /* of the light viewpoints, if selected. */
                        if (dolightview) /* If viewing from a light. */
                            track = curlight->track;
                        else /* If viewing from the eyeview. */
                            track = eyetrack;
                        mytrackclick(track);
                        while(getbutton(LEFTMOUSE)) {
                            mytrackpoll(track);
                            drawit(); /* Draw the scene. */
                        }
                    }
                }
                break;
        }
    }
}
break;
*/

```

```

case MIDDLEMOUSE:
    if(val && (!dodisco || !dolightview)) {
        /* Move last light selected in lightview around object with */
        /* object with mouse while LEFTSHIFTKEY is depressed.      */
        if (!dolightview && getbutton(LEFTSHIFTKEY)) {
            dotracklight = 1;          /* Track light's position.    */
            wintitle(computetitle()); /* Display window title.     */
            curlight = &thelights[curlightnum]; /* Current light is */
                                                /* last light selected. */

            mytrackclick(curlight->track);
            while(getbutton(MIDDLEMOUSE)) { /* While middle mouse button */
                mytrackpoll(curlight->track); /* depressed, track light's */
                drawit();                    /* position & draw the scene.*/
            }
            dotracklight = 0;          /* Cease tracking the light, */
            wintitle(computetitle());
            curlight = 0;             /* set current light to default */
            needredraw = 1;          /* and set redraw the scene flag.*/
        } else {
            /* Move closer to or farther from object using mouse. */
            /* Viewpoint moved is either eyeview or from a light. */
            if (dolightview)         /* If viewing from a light.   */
                track = curlight->track;
            else                      /* If viewing from the eyeview. */
                track = eyetrack;
            mytrackclick(track);
            while(getbutton(MIDDLEMOUSE)) {
                mytrackpoll(track);
                drawit();            /* Draw the scene. */
            }
        }
    }
}
break;
case RIGHTMOUSE: /* Bring up popup menu */
    if (val) {
        switch(dopup(menu)) {
            case MENU_EYEVIEW:
                if(curlight)         /* We are moving a light around,*/
                    newshadowmap(curlight); /* generate a new shadow map for*/
                curlight = 0;        /* it. Then reset current light*/
                dolightview = 0;     /* to the default and stop      */
                wintitle(computetitle()); /* viewing from a light.      */
                break;
            case MENU_TOGGLECOLOR: /* Toggle colored lights on or off. */
                coloredlights = coloredlights ? 0 : 1;
                setlcolors();        /* Of lights based on toggle value. */
                break;
        }
    }
}

```

```

        case MENU_DISCO:          /* Toggle disco mode on or off.      */
            dodisco = dodisco ? 0 : 1;
            if (dodisco) {        /* If toggled on, initialize    */
                initdiscomode(); /* disco track position.      */
            } else {              /* If not on,                  */
                gennewtransforms(); /* Generate new transform matrices */
                gennewshadowmaps(); /* and shadow maps for each light. */
                needredraw = 1;    /* Set the redraw the scene flag. */
            }
            wintitle(computetitle()); /* Display proper window title. */
            break;
        case MENU_QUIT:           /* Exit the program.          */
            exit(0);
    }
    needredraw = 1; /* Set the redraw the scene flag. */
}
break;

case REDRAW:                    /* Redraw window if user resizes it. */
    reshapeviewport(); /* Reset viewport to new window size. */
    getorigin(&wxorg, &wyorg);
    getsize(&wxsize, &wysize);
    needredraw = 1; /* Set the redraw the scene flag. */
    break;
case ESCKEY:                    /* Exit the program.          */
    exit(0);
    break;
default:                         /* In case any other keys are pressed. */
    break;
}
} /* end while(qttest()) */
if (dodisco || needredraw) { /* If in disco mode or redraw flag is set.
*/
    drawit(); /* Redraw the scene if user selected option */
    needredraw = 0; /* requires it. i.e. changing viewpoints. */
    if (dodisco) discotimestep();
}
} /* end while(1) */
}

```

```

/*****
/* initdisco -
/* Initializes the values in the disco track for each moveable light.
/*****
initdiscomode()
{
int i;

    disco_time = 0.0; /* reset time */
    for(i=0; i<NLIGHTS; i++) {
        thelights[i].disco_track->ztrans = 0.0;
        thelights[i].disco_track->trotx = 0.0;
        thelights[i].disco_track->troty = 0.0;
        thelights[i].disco_track->>wiggle = 0.0;
        thelights[i].disco_changed = 1;
    }
}

/*****
/* discotimestep -
/* Calculates changes in light locations during each disco time step.
/*****
discotimestep()
{
int n,lnum,cycle2,cycle3;
float t;
LightBuffer *lamp;

    n = (int)disco_time;
    t = (disco_time-n)/(1.0-disco_step);
    lnum = n % NLIGHTS;
    if (lnum < NLIGHTS)
        lamp = &thelights[lnum];
    else
        lamp = &thelights[0];
    cycle2 = (n % (2*NLIGHTS)) / NLIGHTS;
    cycle3 = (n % (3*NLIGHTS)) / NLIGHTS;
    switch (lnum) {
        case 0:
            lamp->disco_track->>wiggle = 25.0*fsin(2.0*M_PI*t);
            lamp->disco_changed = 1;
            break;
        case 1:
            lamp->disco_track->troty = 120.0*(t+cycle3);
            lamp->disco_changed = 1;
            break;
        case 2:
            lamp->disco_track->trotx = (180.0-2.0*lamp->track->trotx)
                *(cycle2?1.0-t:t);
            lamp->disco_changed = 1;
            break;
        default:

```

```

        break;
    }

    /* advance eyemat */
    eyetrack->trotty += .5;
    if (eyetrack->trotty > 360.0)
        eyetrack->trotty -= 360.0;
    mygettracktransform(eyetrack,eyemat,invemat);

    disco_time += disco_step;
    if (disco_time > 100000)
        disco_time = 0.0;
}

/*****
/* gennewtransforms -
/* Results in the generation of new transformations and inverse
/* transformations for each moveable light.
*****/
gennewtransforms()
{
LightBuffer *lamp;
int i;
    for(i=0,lamp=thelights; i<NLIGHTS; i++,lamp++) {
        mygettracktransform(lamp->track,lamp->mat,lamp->invmat);
    }
}

/*****
/* gennewshadowmaps -
/* Results in the generation of new shadow maps for each moveable light.
*****/
gennewshadowmaps()
{
int i;
    for(i=0; i<NLIGHTS; i++) {
        thelights[i].texno = i+1;
        newshadowmap(thelights+i);
    }
}

```

```

/
*****
**/
/* newshadowmap - */
/* Creates shadow map for the light source. This is done by drawing the scene */
/* from the viewpoint of the light passed in in order to get the depth values */
/* for the scene from the light's viewpoint. The z-values for each pixel of */
/* the rendered scene are loaded into the light's texture map array - smap. */
/* These values are later used to determine whether or not a portion of the */
/* objects in a scene are in shadow when viewed from some other point. */
/
*****
**/
void
newshadowmap(LightBuffer *lightsource)
{
    float *texps;

    shadeoff(); /* Turn lighting model off */
    mmode(MVIEWING); /* ModelView matrix is current matrix stack */
    loadmatrix(Identity); /* Replace top of stack with identity matrix */
    lsetdepth(SHADZNEAR,SHADZFAR); /*Set depth range: near & far clipp planes. */
    viewport(0,SSIZE-1,0,SSIZE-1); /* Set window image area dimensions. */
    czclear(0,SHADZFAR); /* Clears color bit planes to black and */
    /* z-buffer to far clipping plane. */
    blendfunction(BF_ONE,BF_ZERO); /* Sets blended color value for a pixel to */
    /* the source pixel value. */
    zfunction(ZF_LEQUAL);
    ortho(-.5,SSIZE-.5,-.5,SSIZE-.5,-1,1); /* Sets clipping planes for 3D
ortho*/
    /* projection. */
    loadmatrix(Identity);
    cpack(0xff00); /* Set transparency to opaque and color to black */
    myrectf(0,0,SSIZE-1,SSIZE-1,1); /*Draw 2-pixel wide border with
z=SHADZNEAR*/
    myrectf(1,1,SSIZE-2,SSIZE-2,1);
    viewport(2,SSIZE-3,2,SSIZE-3); /* Render real shadow map inside */
    fromlight(lightsource->mat); /* Computes transform matrix to view scene */
    /* from the lightsource in question. */
    getmatrix(lightsource->texm); /* Transformation matrix copied into */
    /* lightsource->texm. */
    if (dozdisplace)
        shadowdisplacez(1); /* So no shadows will be drawn, only the object
*/
    drawmodel(); /* Draw the sgo objects in the scene. */
    if (dozdisplace)
        shadowdisplacez(0); /*Disable 'displacepolygon' so shadows can be drawn*/

    texps = maketexprops(dobicubic,dofastread);/*Set up shadow texture props */

```

```

if (!dofastread) {
    lrectread(0,0,SSIZE-1,SSIZE-1,lightsource->smap); /* Reads pixel values */
                                                    /* of the scene into the light's smap */
    texdef2d(lightsource->texno,1,SSIZE,SSIZE, /* Converts the 2d image */
             (unsigned long *)lightsource->smap,0,texps); /* in smap into */
} else {
                                                    /* a texture. */
    texdef2d(lightsource->texno,1,SSIZE,SSIZE,NULL,0,texps);
    texbind(0,lightsource->texno);
}
viewport(0,wxsize-1,0,wysize-1);
}

/*****
/* maps clip space x,y from a [-1,1] range to [0,1] range, which */
/* is the range s,t should be. */
*****/
float specialmat[4][4] = {
    0.5, 0.0, 0.0, 0.0,
    0.0, 0.5, 0.0, 0.0,
    0.0, 0.0, 1.0, 0.0,
    0.5, 0.5, 0.0, 1.0,
};

```

```

/
*****/
/* texsetup - */
/* Sets up the texture matrix for the lightsource passed in. */
/
*****/
texsetup(lightsource,inveye)
LightBuffer *lightsource;
float inveye[4][4];
{
    float p[4];

    /* Set up texture matrix, but first, make sure ModelView is Identity */
    /* so S,T,R,Q plane equations are the same. Remember, texgen(TG_CONTOUR) */
    /* will transform the plane equations by that ModelView matrix. */
    pushmatrix(); /* Pushes copy of current matrix on top of stack */
    loadmatrix(Identity); /* Replace top of stack with Identity matrix */
    mmode(MTEXTURE); /* Texture matrix is current matrix stack */
    loadmatrix(Identity); /* Replace top of texture stack with Identity matrix */
    translate(0.0,0.0,32768.0); /* Translate current matrix to max z */
    scale(1.0,1.0,32767.5); /* Scale current matrix to max z */
    translate(0.0,0.0,eye_dz); /* Translate current matrix back to eye depth */

    multmatrix(specialmat); /* Multiply current matrix by the special matrix */
    multmatrix(lightsource->texm); /* Multiply result by light's texture ma-
    trix*/
    multmatrix(inveye); /* Multiply result by eyeview's inverse matrix */

    p[0] = 1.0; p[1] = 0; p[2] = 0; p[3] = 0; /* Automatic generation of */
    texgen(TX_S,TG_CONTOUR,p); /* specified texture coordinates */
    p[0] = 0.0; p[1] = 1; p[2] = 0; p[3] = 0; /* using a plane defined in EYE */
    texgen(TX_T,TG_CONTOUR,p); /* coordinates with the specified*/
    p[0] = 0.0; p[1] = 0; p[2] = 1; p[3] = 0; /* plane equation. */
    texgen(TX_R,TG_CONTOUR,p);
    p[0] = 0.0; p[1] = 0; p[2] = 0; p[3] = 1;
    texgen(TX_Q,TG_CONTOUR,p);
    texgen(TX_S,TG_ON,0); /* Enable previously defined */
    texgen(TX_T,TG_ON,0); /* replacement for specified texture */
    texgen(TX_R,TG_ON,0); /* coordinate. */
    texgen(TX_Q,TG_ON,0);
    mmode(MVIEWING); /* Switch current matrix stack back to ModelView stack */
    popmatrix(); /* Pop top matrix off the stack - remove Identity matrix */
}

```

```

/*****/
/* drawit - */
/* Draw the scene using texture mapping for shadow generation. First, any */
/* new shadow maps are generated for moving lights. Then, if viewing from */
/* one of the lights, the scene is simply drawn from that point of view as */
/* no shadow calculations are done. However, if viewing from the eye point, */
/* the object is drawn three times, one for each light's textures until is */
/* has been shadowed (textured) by all three lights. */
/*****/
drawit()
{
    LightBuffer *lamp;
    int i,j;
    float *tevps;

    if (!dolightview) { /* If not viewing the scene from a light's point of */
        if (dotracklight) { /* view, but we are moving a light around, then */
            /* compute transformation matrices for the light and */
            /* generate a new shadow map because its position has */
            /* changed. Thus, so has the way it casts shadows. */
            mygettracktransform(curlight->track, curlight->mat, curlight->invmat);
            newshadowmap(curlight);
        } else { /* If not moving a light around, generate transformation */
            /* matrices for eye point of view. We don't need to generate */
            /* new shadow maps because no lights are moving. Therefore, */
            /* the current shadow maps are still valid. */
            mygettracktransform(eyetrack, eyemat, inveyemat);
        }
    }

    if (dodisco) { /* If in the disco mode, generate transformation matrices and */
        /* shadow maps for each light because their positions are */
        /* changing. Therefore, the shadows cast will change too. */
        for(i=0, lamp=thelights; i<NLIGHTS; i++, lamp++) {
            if (lamp->disco_changed) {
                mygetdiscotransform(lamp->track, lamp->disco_track, lamp->mat,
                    lamp->invmat);
                newshadowmap(lamp);
                lamp->disco_changed = 0;
            }
        }
    }

    lsetdepth(ZNear, ZFar); /* Set depth range to maximum allowable by system */
    czclear(BACKCOLOR, ZFar); /* Clear screen to background color and z-buffer to */
    /* maximum depth value for far clipping plane. */
    blendfunction(BF_ONE, BF_ZERO); /* Sets blended color value for a pixel to */
    /* source (incoming) pixel value. */
    zfunction(ZF_LEQUAL); /* Function passes if incoming pixel z value is < */
    /* or = to z value currently stored. If it passes, */
    /* the incoming pixel's color and depth are written */
    /* into the framebuffer and z-buffer. */
}

```

```

if (dolightview) {          /* If viewing from a light's point of view, set */
mmode(MVIEWING);          /* current matrix stack to model view stack. */
/* Define perspective projection transformation w/light's field of view,*/
/* aspect ratio, and distances from light to near & far clipping planes.*/
perspective((int)(light_fov*20),5.0/4.0,light_near,light_far);
loadmatrix(Identity); /* Replace top matrix with identity matrix. */
pushmatrix();          /* Push another identity matrix on top of stack. */
if (!dodisco)          /* If not in the disco mode, get transformation */
                        /* matrices for the light we're viewing from. */
    mygettracktransform(curlight->track,curlight->mat,curlight->invmat);

multmatrix(curlight->mat); /* Multiply current matrix by light's matrix */
positionlight(curlight->invmat); /* Move light to position 0,0,0 */
shadeon();              /* Turn on the lighting model. */
setspecular(SPECSCALE,SPECSCALE,SPECSCALE);/* Set the specular com-
ponant*/
                                /* of the lighting model to .5 r g b. */
setdiffuse(.6,.6,.6);      /* Set the diffuse component of the */
                                /* lighting model to .6 r g b. */
drawmodel();             /* Draw the object input by the user. */
shadeoff();             /* Turn the lighting model off. */
popmatrix();            /* Pop the top matrix off the stack. */
} else {                  /* If viewing from the eye point of view, set the */
mmode(MVIEWING);        /* current matrix stack to model view stack. */
/* Define perspective projection transformation with eye's field of view*/
/* aspect ratio, and distances from light to near & far clipping planes.*/
perspective((int)(eye_fov*20),5.0/4.0,eye_near,eye_far);
loadmatrix(eyemat); /* Replace top matrix with eye point matrix */
lamp = thelights;

/* Now, we draw the object(s) with the shadows caused by each of the */
/* lights. For each light, its texture or shadow map is computed and
*/

/* then the model is drawn with the shadows caused by the light. */
for(i=0; i<NLIGHTS; i++) { /* For each of the three local lights: */
    texsetup(lamp,invemat); /* Set up the texture matrix. */
    if(i==0) {              /* For the first light, */
        zfunction(ZF_LEQUAL); /* Function passes if incoming pixel */
                                /* z value is < or = to z value currently stored. */
        blendfunction(BF_ONE,BF_ZERO); /* Sets blended color value for */
                                /* a pixel to the source pixel value. */
    } else {                /* For the rest of the lights, */
        zfunction(ZF_EQUAL); /* Function passes if incoming pixel */
                                /* z value is = to z value currently stored. */
        blendfunction(BF_ONE,BF_ONE); /* Sets blended color value based */
                                /* on equal contributions from both the */
                                /* source and destination pixels. */
    }
    pushmatrix();          /* Push copy of current matrix onto the stack. */
    positionlight(lamp->invmat); /* Move light to position 0,0,0 */
    mysettexture(lamp->texno); /* Bind this light's texture. */
    shadeon();            /* Turn on the lighting model. */
}

```

```

        drawmodel(); /* Draw the object input by the user WITH textures. */
        shadeoff(); /* Turn the lighting model off. */
        mysettexture(0); /* Turns texturing off. */
        popmatrix(); /* Pop the top matrix off the stack. */
        lamp++; /* Go to the next light. */
    }
}
swapbuffers(); /* Swap front and back framebuffers. */
}

```

```

/*****/
/* shadowdisplace - */
/* In effect, turns shadowing on and off by enabling and */
/* disabling 'displacepolygon'. */
/*****/
void
shadowdisplacez(int mode)
{
    if (mode)
        displacepolygon(1.0); /* Offsets the z values of rendered pixels for */
    else /* polygons by the maximum amount. */
        displacepolygon(0.0); /* Disables 'displacepolygon' */
}

```

```

/*****/
/* fromlight - */
/* Computes the transformation matrix for viewing */
/* from the light whose original transformation matrix is passed in. */
/*****/
void
fromlight(float mat[4][4])
{
    float m[4][4];

    mmode(MSINGLE); /* All modeling, viewing and projection transformations are*/
                  /* made using a single matrix. */
    pushmatrix(); /* Push a copy on top of stack to work on. */
    perspective((int)(light_fov*20),1.0,light_near,light_far);
    multmatrix(mat); /* Multiplies top of stack w/lightsource's position ma-
trix.*/
    getmatrix(m); /* Places transformation matrix from top of stack into m. */
    popmatrix(); /* Pop top matrix off of stack. */
    multmatrix(m); /* Multiplies top of stack by 'm' so view is from light.*/
}

```

```

/*****
/* drawmodel -
/* Draw the sgo objects, and the lights if 'dodrawlights' is set. */
/*****
drawmodel()
{
int i;
LightBuffer *lt;

    cpack(0xffffffff); /* Current color is set to full white with full opacity.*/

    if (dodrawlights) { /* If flag is set, draw the lights too. */
        lt = thelights;
        for (i=0; i<NLIGHTS; i++) {
            drawlight(lt++);
        }
    }

    for(i=0; i<nobjs; i++) { /* Draw the objects for the scene read in earlier */
        /* from the command line. */
        drawsgiobj(obj[i],DRAW_POINTS|DRAW_NORMALS);
    }
    lmcolor(LMC_COLOR); /* RGB color commands will set current color. If color*/
        /* is last thing sent before a vertex, the vertex will */
        /* be colored. Otherwise it will be lit. */
}

/*****
/* pack3f -
/* Creates the xyz point vector for the vertex of a polygon. */
/*****
float *pack3f(float x,float y,float z)
{
static float v[3];
    v[0] = x;
    v[1] = y;
    v[2] = z;
    return v;
}

```

```

/*****
/* drawlight -
/* Draws 6 faces of a box. The face that looks down the -z axis is
/* colored according to the light's color.
/*****
void
drawlight(LightBuffer *lt)
{
#define DEG2RAD (M_PI/180.0)
float x,y,z;
float imat[4][4];

    x = light_near*fsin(DEG2RAD*light_fov);
    y = x;
    z = -(light_near-.01);
    pushmatrix();
    invertmat(lt->mat,imat);
    multmatrix(imat);
    bgntmesh();
    n3f(pack3f(0,0,1));
    v3f(pack3f(-x,-y,0)); /* v0 */
    v3f(pack3f(x,-y,0)); /* v1 */
    v3f(pack3f(-x,y,0)); /* v3 */
    v3f(pack3f(x,y,0)); /* v2 */
    endtmesh();
    bgntmesh();
    n3f(pack3f(1,0,0));
    v3f(pack3f(x,-y,0)); /* v1 */
    v3f(pack3f(x,-y,z)); /* v5 */
    v3f(pack3f(x,y,0)); /* v2 */
    v3f(pack3f(x,y,z)); /* v6 */
    endtmesh();
    bgntmesh();
    n3f(pack3f(-1,0,0));
    v3f(pack3f(-x,-y,z)); /* v4 */
    v3f(pack3f(-x,-y,0)); /* v0 */
    v3f(pack3f(-x,y,z)); /* v7 */
    v3f(pack3f(-x,y,0)); /* v3 */
    endtmesh();
    bgntmesh();
    n3f(pack3f(0,-1,0));
    v3f(pack3f(-x,-y,0)); /* v0 */
    v3f(pack3f(-x,-y,z)); /* v4 */
    v3f(pack3f(x,-y,0)); /* v1 */
    v3f(pack3f(x,-y,z)); /* v5 */
    endtmesh();
    bgntmesh();
    n3f(pack3f(0,1,0));
    v3f(pack3f(x,y,0)); /* v2 */
    v3f(pack3f(x,y,z)); /* v6 */
    v3f(pack3f(-x,y,0)); /* v3 */
    v3f(pack3f(-x,y,z)); /* v7 */
    endtmesh();
    bgntmesh();
}

```

```

n3f(pack3f(0,0,-1));
c3f(pack3f(lt->r,lt->g,lt->b));
v3f(pack3f(x,-y,z)); /* v5 */
v3f(pack3f(-x,-y,z)); /* v4 */
v3f(pack3f(x,y,z)); /* v6 */
v3f(pack3f(-x,y,z)); /* v7 */
endtmesh();
popmatrix();
}

/*****
/* myrectf - */
/* Generates a rectangular border. */
/*****
void
myrectf(float x1,float y1,float x2,float y2,float z)
{
    bgnline();
    v3f(pack3f(x1,y1,z));
    v3f(pack3f(x2+1,y1,z));
    endlne();
    bgnline();
    v3f(pack3f(x2,y1,z));
    v3f(pack3f(x2,y2+1,z));
    endlne();
    bgnline();
    v3f(pack3f(x1,y1,z));
    v3f(pack3f(x1,y2+1,z));
    endlne();
    bgnline();
    v3f(pack3f(x1,y2,z));
    v3f(pack3f(x2+1,y2,z));
    endlne();
}

```

```

/*****
/* maketexprops -
/* Sets up an array with the shadow texture properties.
/*****
float *maketexprops(int do bicubic,int do fastread)
{
    static float texps[32];
    int i = 0;
    if (do bicubic) { /* Not used for real-time shadows. */
        texps[i++] = TX_MAGFILTER;
        texps[i++] = TX_BICUBIC_GEQUAL; /* Uses smooth weighting of 4x4 region of*/
                                        /* image pixels around s,t mapping-slower*/

        texps[i++] = TX_MINFILTER;
        texps[i++] = TX_BICUBIC_GEQUAL;
    } else { /* For real-time shadows. */
        texps[i++] = TX_MAGFILTER; /* Mapping an image to a larger surface. */
        texps[i++] = TX_BILINEAR_GEQUAL; /* Used for real-time shadow computa-
tions*/
        texps[i++] = TX_MINFILTER; /* Mapping an image to smaller surface. */
        texps[i++] = TX_BILINEAR_GEQUAL; /* Avg point hit with 4 nearest neighbors*/
    }
    /* after each point compared with r coord. */
    texps[i++] = TX_WRAP; /* How do we handle texture coords outside.*/
    texps[i++] = TX_CLAMP; /* 0-1 boundry: clamp coords to 0-1 limits.*/
    texps[i++] = TX_INTERNAL_FORMAT; /* Hint to trade image quality for speed. */
    texps[i++] = TX_I_16 /* Use texture as shadow map for real-time.*/
                        /* shadow comps. Image pixel size: 16 bits.*/

    if (do fastread) { /* Used if user includes -f in command line. */
        texps[i++] = TX_FRAMEBUFFER_SRC;
        texps[i++] = 0.0;
        texps[i++] = 0.0;
        texps[i++] = wxsize-1;
        texps[i++] = wysize-1;
    } else { /* What is normally used. */
        texps[i++] = TX_EXTERNAL_FORMAT; /* What size components are being used */
        texps[i++] = TX_PACK_16; /* in image: 16 bit components. */
        texps[i++] = TX_NOCOPY; /* UNDOCUMENTED, but makes it fast. */
    }
    texps[i++] = TX_NULL;
    return texps;
}

```

```

/*****
/* maketevprops -
/* Sets up an array with the texture environment properties.
/*****
float *maketevprops(int doalpha)
{
    static float tevps[32];
    int i = 0;

    if (doalpha) {
        tevps[i++] = TV_ALPHA;
    } else {
        tevps[i++] = TV_MODULATE; /* Incoming color components are */
                                /* multiplied by texture values. */
        tevps[i++] = TV_COLOR; /* Color used by TV_BLEND environment. */
        tevps[i++] = 1.0; /* In this case, the color is full white */
        tevps[i++] = 1.0; /* and there is no transparency. */
        tevps[i++] = 1.0;
        tevps[i++] = 0.0;
    }
    tevps[i++] = TX_NULL;

    return tevps;
}

/*****
/* mysettexture -
/* Either binds the texture passed in or turns texturing off.
/*****
mysettexture(n)
int n;
{
    if(n==0) {
        texbind(0,0);
    } else {
        texbind(0,n);
    }
}

```

```

/* Static variables used to keep track of object locations. */
static float oztrans;
static float otrotx, otroty;
static float trx, try;
static int dotrans;

/*****
/* fgetmousex -
/* Returns the current x location of the mouse.
*****/
float fgetmousex()
{
    return ((float)getvaluator(MOUSEX)-wxorg)/(float)wxsize;
}

/*****
/* fgetmousey -
/* Returns the current y location of the mouse
*****/
float fgetmousey()
{
    return ((float)getvaluator(MOUSEY)-wyorg)/(float)ysize;
}

/*****
/* mytracktrans -
/* Only used to initialize eye point z translation to 2.0.
*****/
void
mytrackztrans(TrackState *ts,float z)
{
    ts->ztrans = z;
}

/*****
/* mytrackrot -
/* Only used to initialize eye point x & y rotations to 0.0 and 0.0.
*****/
void
mytrackrot(TrackState *ts,float x,float y)
{
    ts->trotx = x;
    ts->troty = y;
}

```

```

/*****/
/* mytrackclick - */
/* For the passed in track state, obtain the x and y locations, toggle */
/* the z translation switch based on middle mouse button status, and */
/* assign the state values to the static variables. */
/*****/
void
mytrackclick(TrackState *ts)
{
    trx = fgetmousex();          /* Get mouse x position. */
    try = fgetmousey();          /* Get mouse y position. */
    if(getbutton(MIDDLEMOUSE)) /* If middle mouse button is pressed, */
        dotrans = 1;            /* set flag for z translations. */
    else
        dotrans = 0;            /* If not, set flag for x & y rotations. */
    oztrans = ts->ztrans;        /* Assign values in passed in track state*/
    otrotx = ts->trotx;          /* to the static variables. */
    otroty = ts->troty;
}

/*****/
/* mytrackpoll - */
/* Take values assigned in 'mytrackclick' and determine new values for the */
/* passed in track state based on whether or not we're currently tracking */
/* the z translation or the x & y rotations. i.e. whether the middle or */
/* left mouse button is pressed. */
/*****/
void
mytrackpoll(TrackState *ts)
{
    if(dotrans) { /* Handle the z translations. */
        ts->ztrans = oztrans+3.0*(fgetmousey()-try);
    } else { /* Handle the x and y rotations. */
        ts->trotx = otrotx+90.0*(fgetmousey()-try);
        ts->troty = otroty+360.0*(fgetmousex()-trx);
    }
}

```

```

/*****
/* mygettracktransform -
/* Generates a transformation and inverse transformation matrix for the
/* light passed in with the current values for its location based on user
/* inputs that are stored in the track state for the light.
/*****
void
mygettracktransform(TrackState *ts,float mat[4][4],float invmat[4][4])
{
    pushmatrix();          /* Put another copy of top matrix on top of stack. */
    loadmatrix(Identity); /* Replace top copy with identity matrix. */
    pushmatrix();          /* Another copy of identity matrix on top of stack. */
    translate(0.0,0.0,ts->ztrans); /* Z Translate to 'ts->ztrans'. */
    rot(ts->trotx,'x');      /* Rotate 'trotx' radians around the x axis. */
    rot(ts->troty,'y');      /* Rotate 'troty' radians around the y axis. */
    getmatrix(mat);         /* Return above generated matrix. */

    /* compute inverse by reversing transforms */
    popmatrix();           /* Pop the above generated matrix off of the stack. */
    rot(-ts->troty,'y');    /* Now, with second identity matrix on top of stack, */
    rot(-ts->trotx,'x');    /* perform the inverse x and y rotations and z trans.*/
    translate(0.0,0.0,-ts->ztrans);
    getmatrix(invmat);     /* Return the inverse matrix. */
    popmatrix();           /* Pop the above inverse matrix off of the stack. */
}

```

```

/*****
/* mygetdiscotransform -
/* Generates a transformation and inverse transformation matrix for the
/* light passed in when the disco mode is operating.
/*****
void
mygetdiscotransform(TrackState *ts,TrackState *ts2,float mat[4][4],
                    float invmat[4][4])
{
    pushmatrix();          /* Put another copy of top matrix on top of stack. */
    loadmatrix(Identity); /* Replace top copy with identity matrix. */
    pushmatrix();          /* Another copy of identity matrix on top of stack. */
    rot(ts2->wiggle,'y');  /* Perform disco mode y wiggle with a y rotation. */
    translate(0.0,0.0,ts->ztrans+ts2->ztrans); /* Translate to:
                                                /* 'ts->ztrans+ztrans+ts2->ztrans'. */
    rot(ts->trotx+ts2->trotx,'x'); /* Rotate 'ts->trotx+ts2->trotx'
                                    /* radians around the x axis. */
    rot(ts->troty+ts2->troty,'y'); /* Rotate 'ts->troty+ts2->troty'
                                    /* radians around the y axis. */
    getmatrix(mat);        /* Return above generated matrix. */

    /* compute inverse by reversing transforms */
    popmatrix();           /* Pop the above generated matrix off of the stack */
    rot(-(ts->troty+ts2->troty),'y'); /* With second identity matrix on top,
    rot(-(ts->trotx+ts2->trotx),'x'); /* perform the inverse x and y
    translate(0.0,0.0,-(ts->ztrans+ts2->ztrans)); /* rotations and z trans.
    rot(-ts2->wiggle,'y'); /* Inverse of disco mode y wiggle.
    getmatrix(invmat);     /* Return the inverse matrix.
    popmatrix();           /* Pop the above inverse matrix off of the stack.
}

```

```

/*****
/* computetitle -
/* Displays the proper menu title depending upon what actions are taking
/* place in the scene.
/*****
char *computetitle() /* Display proper menu title for action being taken. */
{
static char s[100];
static char ls[100];
char *p;

    ls[0] = 0;
    p = ls;
    if (dodisco) {
        sprintf(p,"- disco mode ");
        p += strlen(p);
    }
    if (dolightview) {
        sprintf(p,"- view from light %d",curlightnum+1);
    }
    else if (dotracklight) {
        sprintf(p,"- tracking light %d",curlightnum+1);
    }
    sprintf(s,"%s %s",WINTITLESTR,ls);
    return s;
}

```