PUBLISHED IN THE
JANUARY/FEBRUARY 1995 ISSUE OF *The X Journal*.

OPENGL^{TM} AND X, COLUMN 2: LET THERE BE LIGHT!

Mark J. Kilgard *
Silicon Graphics Inc.
$Revision$: 1.9

November 1, 1995

Welcome to The X Journal's second OpenGL column. For readers who are unfamiliar with OpenGL, it is a programming interface for generating 3D graphics. OpenGL is provided as an extension to the X Window System (though OpenGL is window system independent and works with other window systems like Windows NT and OS/2). To learn more about OpenGL, refer to previous issues of *The X Journal* to find my original series on OpenGL and X.

In the first column, I introduced an OpenGL toolkit known as GLUT (the OpenGL Utility Toolkit). In this column, I use GLUT to demonstrate OpenGL's support for lighting as a technique to improve the realism of OpenGL generated graphics.

When generating 3D graphics on a computer, realism is almost always an underlying goal. If the 3D image is to be effective, it needs to convey a sense of realism to the viewer. Lighting is a means to improve the realism of a 3D scene.

The OpenGL Lighting Model

Anyone who studied optics knows that the physical behavior of light is quite complicated. OpenGL cannot and does not try to capture the phenomenon of light completely. Computer generated graphics is often a trade-off between realism and interactivity. OpenGL is designed primarily for interactive graphics and therefore does not go to extremes to simulate the effects of lighting.

The *lighting model* used by OpenGL approximates real world lighting effects by breaking colors into red, green, and blue components. Of course, this is not an accurate representation of light's physical nature. The point is not to simulate light precisely but to achieve realistic effects without too much computational burden.

Light sources may be described by how much red, green, and blue light they emit. The material of surfaces described by OpenGL geometric primitives can be described by the percentage of red, green, and blue light they reflect in various directions. OpenGL uses a set of lighting equations to approximate the effect of lighting. The equations are relatively easy to compute and provide a fairly reasonable lighting effect.

OpenGL allows multiple light sources to be established. They can be separately enabled and disabled. The effects of the different (enabled) light sources interact with each other. For example, if you had two lights, a red light and a blue light, shining on a white object, the object should appear magenta.

OpenGL supports four distinct types of lighting:

Emissive lighting. This is the simplest type of lighting. The object itself emits a certain color of light. In the real world, this might correspond to something like pixels on your computer monitor which emit light of a given color independent of the lighting in the room. Using emissive light is simple: assign an emissive color to a surface.

Ambient lighting. Ambient lighting corresponds to light so scattered that it can be considered equally uniform and coming from every direction. An example of ambient lighting is the effect of back lighting in a room. Back lighting keeps the light from directly shining on most objects in the room, but objects are still visible from the indirect or ambient light in the room.

Diffuse lighting. Unlike ambient lighting, diffuse lighting comes from a given direction. How much of the light is reflected is proportional to the light's incidence to the surface. Light shining directly on the surface will be reflected brighter than light hitting the surface at an angle.

*Mark graduated with B.A. in Computer Science from Rice University and is a Member of the Technical Staff at Silicon Graphics. He can be reached by electronic mail addressed to `mjk@sgi.com`

But once the light encounters the surface, it is reflected evenly in all directions.

Specular lighting. Specular lighting is like diffuse lighting, but the light tends to reflected from the surface in a preferred, reflected direction. Think of a silver ball held up outside with the sun shining. When you look at the ball such that the sun reflects directly upon it, you can see a bright shiny spot. This is called a *specular highlight*. Specularity corresponds well to the everyday notion of "shininess." For example, chalk has almost no specular component, while a shiny metal or plastic has a high specular component.

A single light source may actually be composed of several different types of lighting. For example, a flashlight will have both diffuse and specular components (corresponding to the flashlight's directed light) but there is also a small amount of ambient light (corresponding to the environmental light added by the flashlight).

Lights in OpenGL have an effect on the scene only when they interact with surfaces in the scene. If you enable a red light, but do not render any geometric primitives, do not expect anything to happen. A light source is not itself rendered; it affects only how the scene is rendered.

OpenGL's lighting model provides a reasonable lighting model in that if used effectively, OpenGL lighting can enhance the illusion of 3D. For example, a moving positional light source in an animated scene will change the coloration of the objects in the scene in a reasonably realistic manner.

But there are limitations to OpenGL's lighting model. For example, shadows and reflective surfaces cannot be generated automatically. This does not mean you cannot generate shadows or reflective surfaces in OpenGL, just that you have to do much of the work yourself. But to OpenGL's advantage, the machinery for such effects is a standard part of OpenGL (features like blending and environment mapping of textures). It is just that OpenGL does not build support for shadows or reflection into its lighting model.

Using OpenGL's Lighting Model

OpenGL's lighting is not enabled by default. To enable lighting, call:

```
glEnable(GL_LIGHTING);
```

Without lighting enabled, color is assigned by the *current color* established by the `glColor3f` family of calls. With lighting disabled, the color you assign to geometric primitives is the color you get (though options like smooth shading or fog can affect the color). But the more involved case is when lighting is enabled.

Defining Light Sources

OpenGL supports eight simultaneously active light sources. Light sources can be enabled and disabled individually. For example, to enable light source number zero, a program would call:

```
glEnable(GL_LIGHT0);
```

Because each light you enable complicates the lighting equations, performance is likely to be better with fewer lights enabled.

Each light source has an ambient, diffuse, and specular intensity to it. You can think of each intensity as the color that the light source emits for each type of light. The intensity is specified as color with red, green, blue, and alpha components between zero and one. (Alpha is used in blending operations and will be discussed in detail in a future column; for now ignore it.) For example, a light with a bright white diffuse and specular intensity and a dimmer ambient intensity would be specified like this:

```
GLfloat amb[] =  /* dim white */
                 { 0.3, 0.3, 0.3, 1.0 };
GLfloat dif[] =  /* bright white */
                 { 1.0, 1.0, 1.0, 1.0 };
GLfloat spec[] = /* bright white */
                 { 1.0, 1.0, 1.0, 1.0 };

glLightfv(GL_LIGHT1, GL_AMBIENT, amb);
glLightfv(GL_LIGHT1, GL_DIFFUSE, dif);
glLightfv(GL_LIGHT1, GL_SPECULAR, spec);
```

White light is generated by suppling equal amount of red, green, and blue. The closer each component is to 1.0, the more intense the light source. Unless alpha is being explicitly used, alpha is usually specified as 1.0.

Light sources also have a position. Sometimes the position can be "at infinity." In such a case, the light source is called *infinite* or *directional*. The sun can be considered a directional or infinite light source because of how far away it is from the earthly objects it illuminates.

In a number of cases, you will find "simplifications" exist in the OpenGL lighting model to make the mathematics of calculating lighting effects easier. For example, the effect of a directional light (positioned "at infinity") is easier to calculate compared with a directional light at some absolute position. For a distant light, you can get better performance by setting up the light as an infinite, directional light with little affect on the rendered scene.

You can specify a light source's position like this:

```
GLfloat pos = { 2.0, 3.0, 5.0, 1.0 };

glLightfv(GL_LIGHT1, GL_POSITION, pos);
```

Notice a vector of four values (x, y, z, w) is specified. The x, y, and z parameters are divided by w to determine the actual (x, y, z) location of light source. If w is zero, this indicates the light is an infinite light (notice the division by zero!). A light with a zero w would have a direction, but not a position. Readers with a mathematical background may recognize these as *homogeneous coordinates*.

Defining the Surface to Light

Lights are useless without surfaces to shine on. Two types of information about the surface are used in OpenGL lighting calculations:

Material properties. These are colors specified as red, green, blue, and alpha components between zero and one for each type of light: ambient, diffuse, specular, and emissive. Each color type is combined with the light source's color type intensity to determine what color of light to reflect off the surface. A *specular exponent* is also part of the material properties and is roughly a measure of the surface's shininess.

Normal information. The normal to a surface at a point is a 3D vector at that point determining the orientation of the surface. The diffuse and specular lighting calculations use the angle of incidence between the light and the surface normal to determine the amount of light reflected. For specular lighting, the amount of reflection also depends on the viewer's "eye" position.

The material properties are set using the `glMaterialfv` family of calls. For example:

```
GLfloat amb[] = {0.33, 0.22, 0.03, 1.0};
GLfloat dif[] = {0.78, 0.57, 0.11, 1.0};
GLfloat spc[] = {0.99, 0.91, 0.81, 1.0};

glMaterialfv(GL_FRONT, GL_AMBIENT, amb);
glMaterialfv(GL_FRONT, GL_DIFFUSE, dif);
glMaterialfv(GL_FRONT, GL_SPECULAR, spc);
glMaterialf(GL_FRONT, GL_SHININESS, 27.8);
```

The values in the above example are chosen to give the appearance of brass. Notice the blue color components are lower than the red and green components to give a yellowish tint and the specular color is rather bright.

Per-vertex normals for geometric primitives are specified using the `glNormal3f` family of calls. For the lighting calculations to work as expected, the normals should be unit vectors. Even when you specify unit vectors in your calls to `glNormal3f`, modeling operations like scaling can denormalize your normal vectors. To avoid this problem (at a slight performance penalty), the following call will assure that your normal vectors are always properly normalized before lighting calculations are performed:

```
glEnable(GL_NORMALIZE);
```

Other Lighting Concepts

OpenGL's lighting model has a number of other interesting features that are beyond the scope of this article to describe in detail, but the features are worth mentioning for completeness.

Spotlights. OpenGL allows positional lights to be restricted so the shape of the light generated is a cone. The *spotlight direction* and *cutoff* angle can be specified.

Two-sided lighting. OpenGL can support distinct lighting for the front and back of polygons. This explains the `GL_FRONT` parameter in the example calls to `glMaterialfv`.

Attenuation. In the real world, the intensity of a light source drops off with distance. OpenGL allows a similar attenuation effect to be specified for positional light sources.

Global ambient light. A global ambient light source can be established.

Color index mode lighting. OpenGL supports for lighting in color index mode (the equivalent of an X11 *Psuedo-Color* visual). In general, OpenGL lighting is much more straightforward in RGBA mode.

The lighting parameters for a scene do not have to be static. For example, you may want to move a positional light source for each frame of an animated scene (perhaps the light is on a swinging pendulum). Or you could render part of a scene with lights 1 and 2 enabled, then later disable light 2 for the remainder of the frame. While normally you will want to keep your light sources fixed for the duration of the frame you are are rendering, this is by no means required.

Abstractly, you can think of OpenGL lighting as a large set of equations for determining the color of rendered geometric primitives. It is beyond the scope of this article to examine the mathematical equations for the OpenGL lighting model in details, but mathematically inclined readers are invited to read the discussion of OpenGL lighting in the "red" book [1] or the OpenGL specification itself [2].

A GLUT Example

The code in Appendix A is for a GLUT program named `lightlab` that allows you to change the material and lighting characteristics of a scene containing a teapot, a torus, and an icosahedron. The objects in the scene can be rendered to appear to be made of brass, red plastic, emerald, or slate. Two directional lights coming from the left and right can shine red, white, green, or not at all.

Figures 1, 2, 3, and 4 show various lighting configurations using `lightlab`. GLUT menus are used to control the various lighting parameters. The program's source code is a good start for exploring OpenGL lighting features. Like the GLUT library itself, you can obtain `lightlab` via anonymous `ftp` to `sgigate.sgi.com` and looking in the `pub/opengl/xjournal` directory.

Next issue: Texturing in OpenGL.

References

[1] Jackie Neider, Tom Davis, Mason Woo, *OpenGL Programming Guide: The official guide to learning OpenGL, Release 1*, Addison Wesley, 1993.

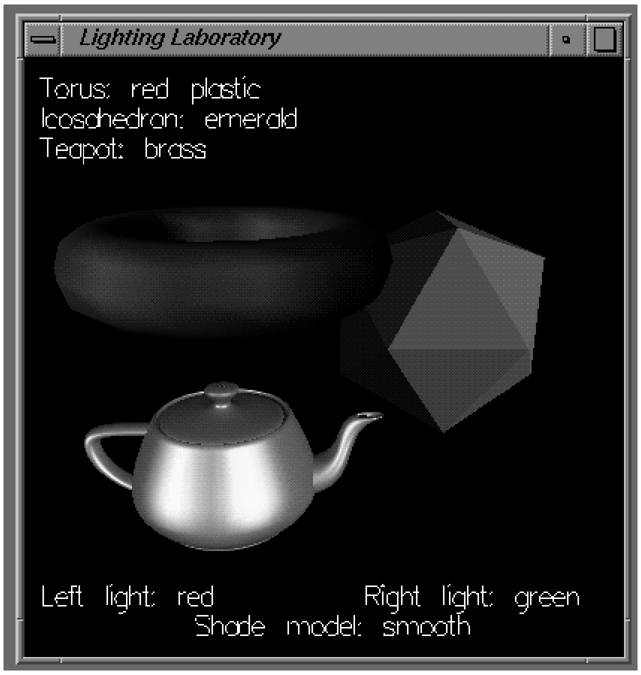

Figure 1: `lightlab`'s initial appearance. Both lights enabled so the teapot has two specular highlights. The left side of the teapot catches the left side's red light; the right side catches the green from the right side's green light.

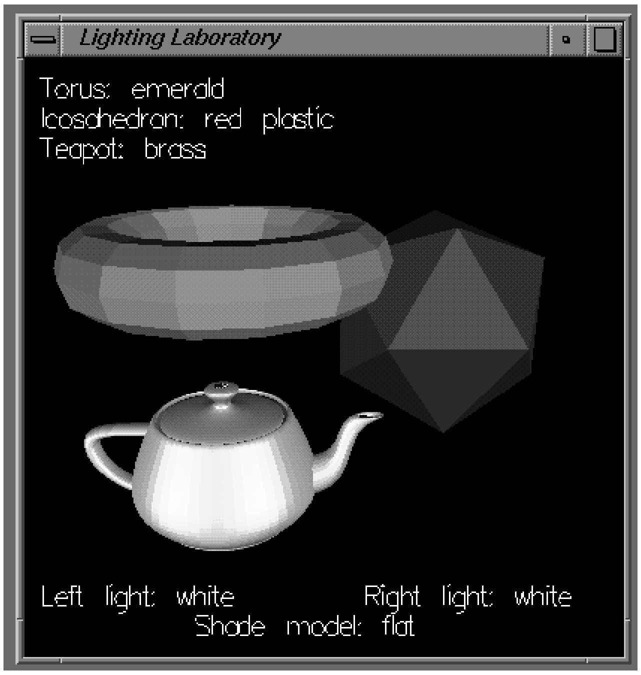

Figure 3: Both lights are shining white and the picture is bright. Flat shading is being used instead of smooth shading. This makes it apparent the torus is not tesselated that finely. Lighting with smooth shading gives it a rounder appearance.

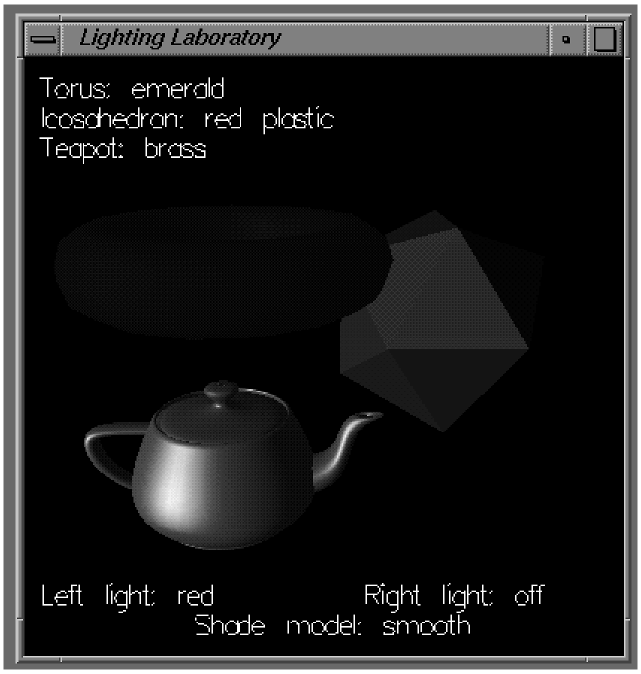

Figure 2: The right light is off; the left light shines red. Notice the emerald torus is hardly visible because there is no green light in the scene. Also because there is only one light shining, the teapot has a single specular highlight.

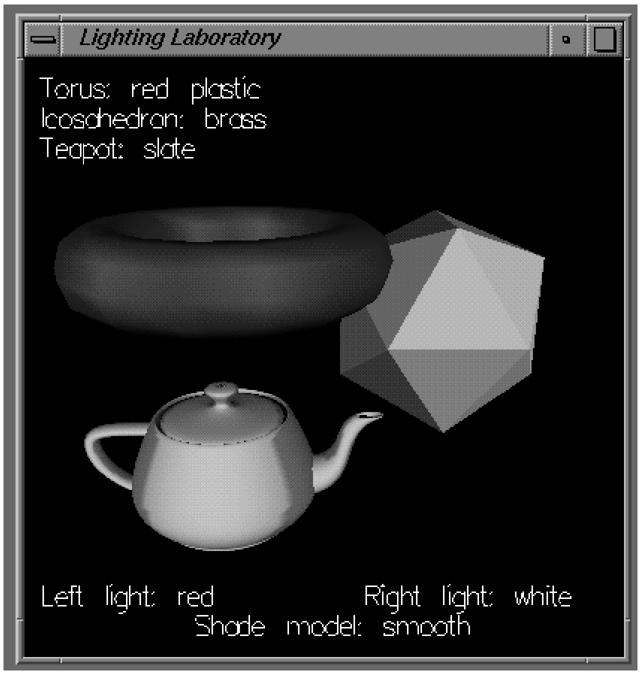

Figure 4: Slate is not very shiny so the teapot's specular highlights are spread widely across the teapot's surface. The specular highlight for the icosahedron is caught by the entire front triangle. The left side of the icosahedron appears redder because of the left red light.

[2] Mark Segal, Kurt Akeley, *The OpenGL^{TM} Graphics System: A Specification*, Version 1.0, Silicon Graphics, June 30, 1992.

A lightlab.c

```
/* Copyright (c) Mark J. Kilgard, 1994. */

/* This program is freely distributable without licensing fees
   and is provided without guarantee or warrantee expressed or
   implied. This program is -not- in the public domain. */

#include <stdlib.h>
#include <stdarg.h>
#include <stdio.h>
#include <GL/glut.h>

enum {
  BRASS, RED_PLASTIC, EMERALD, SLATE
} MaterialType;
enum {
  TORUS_MATERIAL = 1, TEAPOT_MATERIAL = 2, ICO_MATERIAL = 3
} MaterialDisplayList;
enum {
  LIGHT_OFF, LIGHT_RED, LIGHT_WHITE, LIGHT_GREEN
} LightValues;

GLfloat red_light[] =
{1.0, 0.0, 0.0, 1.0}, green_light[] =
{0.0, 1.0, 0.0, 1.0}, white_light[] =
{1.0, 1.0, 1.0, 1.0};
GLfloat left_light_position[] =
{-1.0, 0.0, 1.0, 0.0}, right_light_position[] =
{1.0, 0.0, 1.0, 0.0};
GLfloat brass_ambient[] =
{0.33, 0.22, 0.03, 1.0}, brass_diffuse[] =
{0.78, 0.57, 0.11, 1.0}, brass_specular[] =
{0.99, 0.91, 0.81, 1.0}, brass_shininess = 27.8;
GLfloat red_plastic_ambient[] =
{0.0, 0.0, 0.0}, red_plastic_diffuse[] =
{0.5, 0.0, 0.0}, red_plastic_specular[] =
{0.7, 0.6, 0.6}, red_plastic_shininess = 32.0;
GLfloat emerald_ambient[] =
{0.0215, 0.1745, 0.0215}, emerald_diffuse[] =
{0.07568, 0.61424, 0.07568}, emerald_specular[] =
{0.633, 0.727811, 0.633}, emerald_shininess = 76.8;
GLfloat slate_ambient[] =
{0.02, 0.02, 0.02}, slate_diffuse[] =
{0.02, 0.01, 0.01}, slate_specular[] =
{0.4, 0.4, 0.4}, slate_shininess = .78125;
int shade_model = GL_SMOOTH;
char *left_light, *right_light;
char *ico_material, *teapot_material, *torus_material;

void
output(GLfloat x, GLfloat y, char *format,...)
{
  va_list args;
  char buffer[200], *p;

  va_start(args, format);
  vsprintf(buffer, format, args);
  va_end(args);
  glPushMatrix();
```

```
  glTranslatef(x, y, 0);
  for (p = buffer; *p; p++)
    glutStrokeCharacter(GLUT_STROKE_ROMAN, *p);
  glPopMatrix();
}

void
display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glScalef(1.3, 1.3, 1.3);
  glRotatef(20.0, 1.0, 0.0, 0.0);
  glPushMatrix();
  glTranslatef(-0.65, 0.7, 0.0);
  glRotatef(90.0, 1.0, 0.0, 0.0);
  glCallList(TORUS_MATERIAL);
  glutSolidTorus(0.275, 0.85, 10, 15);
  glPopMatrix();
  glPushMatrix();
  glTranslatef(-0.75, -0.8, 0.0);
  glCallList(TEAPOT_MATERIAL);
  glutSolidTeapot(0.7);
  glPopMatrix();
  glPushMatrix();
  glTranslatef(1.0, 0.0, -1.0);
  glCallList(ICO_MATERIAL);
  glutSolidIcosahedron();
  glPopMatrix();
  glPopMatrix();
  glPushAttrib(GL_ENABLE_BIT);
  glDisable(GL_DEPTH_TEST);
  glDisable(GL_LIGHTING);
  glMatrixMode(GL_PROJECTION);
  glPushMatrix();
  glLoadIdentity();
  gluOrtho2D(0, 3000, 0, 3000);
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glLoadIdentity();
  output(80, 2800, "Torus: %s", torus_material);
  output(80, 2650, "Icosahedron: %s", ico_material);
  output(80, 2500, "Teapot: %s", teapot_material);
  output(80, 250, "Left light: %s", left_light);
  output(1700, 250, "Right light: %s", right_light);
  output(850, 100, "Shade model: %s",
    shade_model == GL_SMOOTH ? "smooth" : "flat");
  glPopMatrix();
  glMatrixMode(GL_PROJECTION);
  glPopMatrix();
  glPopAttrib();
  glutSwapBuffers();
}

void
light_select(GLenum which, int value, char **label)
{
  glEnable(which);
  switch (value) {
```

```
  case LIGHT_OFF:
    *label = "off";
    glDisable(which);
    break;
  case LIGHT_RED:
    *label = "red";
    glLightfv(which, GL_DIFFUSE, red_light);
    break;
  case LIGHT_WHITE:
    *label = "white";
    glLightfv(which, GL_DIFFUSE, white_light);
    break;
  case LIGHT_GREEN:
    *label = "green";
    glLightfv(which, GL_DIFFUSE, green_light);
    break;
  }
  glutPostRedisplay();
}

void
left_light_select(int value)
{
  light_select(GL_LIGHT0, value, &left_light);
}

void
right_light_select(int value)
{
  light_select(GL_LIGHT1, value, &right_light);
}

void
material(int dlist, GLfloat * ambient, GLfloat * diffuse,
  GLfloat * specular, GLfloat shininess)
{
  glNewList(dlist, GL_COMPILE);
  glMaterialfv(GL_FRONT, GL_AMBIENT, ambient);
  glMaterialfv(GL_FRONT, GL_DIFFUSE, diffuse);
  glMaterialfv(GL_FRONT, GL_SPECULAR, specular);
  glMaterialf(GL_FRONT, GL_SHININESS, shininess);
  glEndList();
}

char *
material_select(int object, int value)
{
  glutPostRedisplay();
  switch (value) {
  case BRASS:
    material(object, brass_ambient,
      brass_diffuse, brass_specular, brass_shininess);
    return "brass";
  case RED_PLASTIC:
    material(object, red_plastic_ambient, red_plastic_diffuse,
      red_plastic_specular, red_plastic_shininess);
    return "red plastic";
  case EMERALD:
    material(object, emerald_ambient, emerald_diffuse,
      emerald_specular, emerald_shininess);
```

```
    return "emerald";
  case SLATE:
    material(object, slate_ambient, slate_diffuse,
      slate_specular, slate_shininess);
    return "slate";
  }
  return NULL; /* avoid bogus warning! */
}

void
torus_select(int value)
{
  torus_material = material_select(TORUS_MATERIAL, value);
}

void
teapot_select(int value)
{
  teapot_material = material_select(TEAPOT_MATERIAL, value);
}

void
ico_select(int value)
{
  ico_material = material_select(ICO_MATERIAL, value);
}

void
main_menu_select(int value)
{
  if (value == 666)
    exit(0);
  glShadeModel(shade_model = value);
  glutPostRedisplay();
}

void
main(int argc, char **argv)
{
  int left_light_m, right_light_m, torus_m, teapot_m, ico_m;

  glutInitWindowSize(400, 400);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("Lighting Laboratory");
  glutDisplayFunc(display);

#define LIGHT_MENU_ENTRIES() \
    glutAddMenuEntry("Disable", LIGHT_OFF); \
    glutAddMenuEntry("Red", LIGHT_RED); \
    glutAddMenuEntry("White", LIGHT_WHITE); \
    glutAddMenuEntry("Green", LIGHT_GREEN);
#define MATERIAL_MENU_ENTRIES() \
    glutAddMenuEntry("Brass", BRASS); \
    glutAddMenuEntry("Red plastic", RED_PLASTIC); \
    glutAddMenuEntry("Emerald", EMERALD); \
    glutAddMenuEntry("Slate", SLATE);

  left_light_m = glutCreateMenu(left_light_select);
  LIGHT_MENU_ENTRIES();
```

```
  right_light_m = glutCreateMenu(right_light_select);
  LIGHT_MENU_ENTRIES();
  torus_m = glutCreateMenu(torus_select);
  MATERIAL_MENU_ENTRIES();
  teapot_m = glutCreateMenu(teapot_select);
  MATERIAL_MENU_ENTRIES();
  ico_m = glutCreateMenu(ico_select);
  MATERIAL_MENU_ENTRIES();

  glutCreateMenu(main_menu_select);
  glutAddMenuEntry("Smooth shading", GL_SMOOTH);
  glutAddMenuEntry("Flat shading", GL_FLAT);
  glutAddSubMenu("Left light", left_light_m);
  glutAddSubMenu("Right light", right_light_m);
  glutAddSubMenu("Torus", torus_m);
  glutAddSubMenu("Teapot", teapot_m);
  glutAddSubMenu("Icosahedron", ico_m);
  glutAddMenuEntry("Quit", 666);
  glutAttachMenu(GLUT_RIGHT_BUTTON);

  glLightfv(GL_LIGHT0, GL_POSITION, left_light_position);
  glLightfv(GL_LIGHT0, GL_SPECULAR, white_light);
  glLightfv(GL_LIGHT1, GL_POSITION, right_light_position);
  glLightfv(GL_LIGHT1, GL_SPECULAR, white_light);
  left_light_select(LIGHT_RED);
  right_light_select(LIGHT_GREEN);
  torus_select(RED_PLASTIC);
  teapot_select(BRASS);
  ico_select(EMERALD);
  glEnable(GL_LIGHTING);
  glEnable(GL_DEPTH_TEST);
  glEnable(GL_NORMALIZE);
  glLineWidth(1.0);
  glMatrixMode(GL_PROJECTION);
  gluPerspective( /* degrees field of view */ 50.0,
    /* aspect ratio */ 1.0, /* Z near */ 1.0, /* Z far */ 10.0);
  glMatrixMode(GL_MODELVIEW);
  gluLookAt(0.0, 0.0, 5.0,  /* eye is at (0,0,5) */
    0.0, 0.0, 0.0,      /* center is at (0,0,0) */
    0.0, 1.0, 0.);      /* up is in positive Y direction */
  glTranslatef(0.0, 0.0, -1.0);

  glutMainLoop();
}
```