```
/* texturelab.c --- Experimentation with lighting and textures in
 * OpenGL.
 * Tom Kelliher 4/25/05.
 *
 * This program requires gltx.c and gltx.h.  Make sure to add them
 * to the project.  You can find them on the class Web site.
 *
 * This version uses the  moving light and spotlight anchored to the
 * viewer.  Use l to turn the spot off and L to turn it back on.
 * The spotlight is now actually a point source, so that we can
 * see better --- textures attenuate the lights somewhat.
 *
 * Four different textures are read and applied.  The torus is not
 * textured, so as to demonstrate how to enable/disable texturing.
 *
 * Automatic texture coordinate generation is used.  Note that it
 * does not work well for cubes --- it is used for the left cube.
 * Manual texture coordinates are used for the right cube.  This
 * illustrates how to disable and enable automatic texture
 * coordinate generation.
 *
 * Movement is via x/X, y/Y, and z/Z keys.  Use space bar to exit.
 * Mouse buttons rotate the right cube.
 *
 * The four texture files, imgfile[1-4].rgb should be in the main
 * project directory.
 *
 *
 * Experiments:
 *
 *    Experiment with BREAK_TEXTURE and OBJECT_COORDS.
 *
 *    In textureInit(), vary the use of TEX_IMAGE and MIP_MAP.
 *    Can you see any difference?
 *
 *    Re-configure the viewer-anchored light to be a spotlight,
 *    rather than a point light source.  Notice much of a
 *    difference?
 *
 *    Fix the texture coordinate generation problems of the left
 *    cube.
 *
 *    Create a fifth texture and apply it to the torus.
 */


#include <stdio.h>
#include <stdlib.h>
#include <GL/glut.h>
// The IRIS RGB "library."  gltx.c must be added to the project.
#include "gltx.h"


// Uncomment the following to see the right cube's texture
// applied in a manner similar to the left cube.  If you study
// polygon(), below, you will get a hint as to what's going on.
//#define BREAK_TEXTURE


// Comment-out the following to have automatic texture coordinates
// generated in eye-coordinates rather than object coordinates.
#define OBJECT_COORDS


#define TEX_IMAGE 0
#define MIP_MAP 1


// Base of the display lists.
GLuint lists;


// The right cube can be rotated.  Whoopee.
static GLfloat theta[] = {45.0,0.0,0.0};
static GLint axis = 2;
/* initial viewer location */
static GLdouble viewer[]= {20.0, 6.0, 6.0};
//static GLdouble viewer[]= {0.0, 0.0, 10.0};


// Light 0 is the moving light.  Light 1 is the spotlight that moves
// with us.  Note that these are positional lights.
GLfloat light0_position[] = { 0.0, 0.0, 5.0, 1.0 };
GLfloat light1_position[] = { 0.0, 0.0, 0.0, 1.0 };

// Angle of the moving light.
GLint light_angle = 0;


// For the textures.
GLuint texName1;
GLuint texName2;
GLuint texName3;
GLuint texName4;


// Vertices for right cube.
GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
{1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
{1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};


// These are the real normals for the right cube, 1 per face.
GLfloat normals[][3] = {{0.0,0.0,-1.0},{0.0,1.0,0.0},
{-1.0,0.0,0.0},{1.0,0.0,0.0},{0.0,0.0,1.0},{0.0,-1.0,0.0}};


// Code for rendering the right cube.
void polygon(int a, int b, int c , int d, GLfloat normal[])
{
   glBegin(GL_POLYGON);
      glNormal3fv(normal);
      glTexCoord2d(0.0, 0.0);
      glVertex3fv(vertices[a]);
      glTexCoord2d(0.0, 1.0);
      glVertex3fv(vertices[b]);
#ifndef BREAK_TEXTURE
      glTexCoord2d(1.0, 1.0);
#endif
      glVertex3fv(vertices[c]);
```

```
#ifndef BREAK_TEXTURE
      glTexCoord2d(1.0, 0.0);
#else
      glTexCoord2d(0.0, 0.0);
#endif
      glVertex3fv(vertices[d]);
   glEnd();
}


void colorcube()
{
        polygon(0,3,2,1,normals[0]);
        polygon(2,3,7,6,normals[1]);
        polygon(0,4,7,3,normals[2]);
        polygon(1,2,6,5,normals[3]);
        polygon(4,5,6,7,normals[4]);
        polygon(0,1,5,4,normals[5]);
}


// Using gltx, read the IRIS file and bind the image to a texture.

void getTexture(int *tName, char *fName, int mode)
{
        GLTXimage *image = gltxReadRGB(fName);

        if (!image)
        {
                printf("Problems with the image file: %s.\n",
             fName);
                exit(1);
        }

   // Similar to glGenLists().
   glGenTextures(1, tName);

   // Similar to glNewList(), but there's nothing corresponding to
   // glEndList().
   glBindTexture(GL_TEXTURE_2D, *tName);

   // Tile the 2-D texture in both dimenstions.
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);

   // Use quick and dirty magnification and minification.
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                   GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                   GL_NEAREST);

   // glTexImage2D() only generates a single version of the texture.
   // gluBuild2dMipmaps() generates several scaled versions of the
   // texture.  It's preferable because it has fewer aliasing
   // problems.  On the other hand, it uses more texture memory.

   if (mode == TEX_IMAGE)
      glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width,
                        image->height, 0, GL_RGB, GL_UNSIGNED_BYTE,
                   image->data);
   else if (mode == MIP_MAP)
      gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->width,
                        image->height, GL_RGB, GL_UNSIGNED_BYTE,
                        image->data);
   else
   {
      printf("Invalid texture mode.\n");
      exit(1);
   }

   gltxDelete(image);
}


void textureInit(void)
{
    /* Set texel storage format. */
    glPixelStorei(GL_UNPACK_ALIGNMENT, 1);

   /* Get the textures. */
   getTexture(&texName1, "imgfile1.rgb", TEX_IMAGE);
   getTexture(&texName2, "imgfile2.rgb", MIP_MAP);
   getTexture(&texName3, "imgfile3.rgb", MIP_MAP);
   getTexture(&texName4, "imgfile4.rgb", MIP_MAP);

   /* Use modulated application to work with lighting, enable
    * automatic texture coordinate generation, and enable textures.
    */
   glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
   glEnable(GL_TEXTURE_GEN_S);
   glEnable(GL_TEXTURE_GEN_T);
#ifdef OBJECT_COORDS
   glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
   glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
#endif
   glEnable(GL_TEXTURE_2D);
}


// Paint the scene.
void display(void)
{

   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

   glEnable(GL_TEXTURE_2D);

   /* Update viewer position in modelview matrix */
   glLoadIdentity();

   gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0,
             0.0);

   // Position the moving light.  It will be positioned in world
   // coordinates.
   glPushMatrix();
   glRotatef(light_angle, 1.0, 0.0, 0.0);
   glLightfv(GL_LIGHT0, GL_POSITION, light0_position);

   // Render a solid sphere indicating the light's position.  Notice
```

```
241:    // how it's neither lit nor textured.
242:
243:    glTranslatef(light0_position[0], light0_position[1],
244:                    light0_position[2]);
245:    glDisable(GL_LIGHTING);
246:    glDisable(GL_TEXTURE_2D);
247:    glCallList(lists);
248:    glEnable(GL_TEXTURE_2D);
249:    glEnable(GL_LIGHTING);
250:    glPopMatrix();
251:
252:    // Notice how we indicate which texture to use.  This texture
253:    // will be in use until we specify another or disable 2-D
254:    // textures.
255:
256:    glBindTexture(GL_TEXTURE_2D, texName2);
257:
258:    glPushMatrix();
259:
260:    // Left cube.
261:    glTranslatef(-2.0, 0.0, 0.0);
262:    glCallList(lists + 1);
263:
264:    glBindTexture(GL_TEXTURE_2D, texName1);
265:
266:    // Sphere centered at origin.
267:    glTranslatef(2.0, 0.0, 0.0);
268:    glCallList(lists + 2);
269:
270:    glBindTexture(GL_TEXTURE_2D, texName4);
271:
272:    // Disable automatic texture coordinate generation.
273:    glDisable(GL_TEXTURE_GEN_S);
274:    glDisable(GL_TEXTURE_GEN_T);
275:
276:    // Right cube.
277:    glTranslatef(2.0, 0.0, 0.0);
278:    // Rotate cube.
279:    glRotatef(theta[0], 1.0, 0.0, 0.0);
280:    glRotatef(theta[1], 0.0, 1.0, 0.0);
281:    glRotatef(theta[2], 0.0, 0.0, 1.0);
282:    colorcube();
283:
284:    // Re-enable automatic texture coordinate generation.
285:    glEnable(GL_TEXTURE_GEN_S);
286:    glEnable(GL_TEXTURE_GEN_T);
287:
288:    glBindTexture(GL_TEXTURE_2D, texName3);
289:
290:    // Now that I'm thoroughly confused, let's re-establish the
291:    // origin.
292:    glPopMatrix();
293:
294:    // The cone.
295:    // Up y-axis by 1.
296:    glTranslatef(0.0, 1.0, 0.0);
297:    // Rotate z-axis up to y-axis.
298:    glRotatef(-90.0, 1.0, 0.0, 0.0);
299:    glCallList(lists + 3);
300:
301:    glDisable(GL_TEXTURE_2D);
302:
303:    // The torus.
304:    // Translate up the cone.
305:    glTranslatef(0.0, 0.0, 1.0);
306:    glCallList(lists + 4);
307:
308:    glutSwapBuffers();
309: }
310:
311:
312: // Rotate the right cube.  Looks kinda neat when it cuts into the
313: // sphere.
314: void mouse(int btn, int state, int x, int y)
315: {
316:         if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
317:         if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
318:         if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
319:         theta[axis] += 2.0;
320:         if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
321:         glutPostRedisplay();
322: }
323:
324:
325: void keys(unsigned char key, int x, int y)
326: {
327:
328: /* Use x, X, y, Y, z, and Z keys to move viewer */
329:
330:    if(key == 'x') viewer[0]-= 1.0;
331:    if(key == 'X') viewer[0]+= 1.0;
332:    if(key == 'y') viewer[1]-= 1.0;
333:    if(key == 'Y') viewer[1]+= 1.0;
334:    if(key == 'z') viewer[2]-= 1.0;
335:    if(key == 'Z') viewer[2]+= 1.0;
336:    if(key == 'l') glDisable(GL_LIGHT1);
337:    if(key == 'L') glEnable(GL_LIGHT1);
338:    if(key == ' ') exit(0);
339:
340:    printf("v[x]: %f, v[y]: %f, v[z]: %f.\n", viewer[0], viewer[1],
341:           viewer[2]);
342:
343:    glutPostRedisplay();
344: }
345:
346:
347: void myReshape(int w, int h)
348: {
349:    glViewport(0, 0, w, h);
350:
351:    /* Use a perspective view */
352:
353:    glMatrixMode(GL_PROJECTION);
354:    glLoadIdentity();
355:    gluPerspective(35.0, (GLfloat) w/h, 1.0, 100.0);
356:    glMatrixMode(GL_MODELVIEW);
357: }
358:
359:
360: // Rotate the moving light.
```

```
361: void idle(void)
362: {
363:         light_angle = (light_angle + 3) % 360;
364:         glutPostRedisplay();
365: }
366:
367:
368: // Set material and light values.
369: void init(void)
370: {
371:    // No ambient light whatsoever.
372:    GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
373:    GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
374:    GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
375:
376:    GLfloat light1_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
377:    GLfloat light1_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
378:    GLfloat light1_specular[] = { 1.0, 1.0, 1.0, 1.0 };
379:
380:    //glClearColor (0.2f, 0.3f, 0.4f, 1.0);
381:    glClearColor (0.0f, 0.0f, 0.0f, 1.0);
382:    glShadeModel (GL_SMOOTH);
383:
384:    // Note use of linear attenuation values, the spot cutoff
385:    // half-angle, and the spot exponent (intensity dropoff
386:    // within the cone).
387:
388:    glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
389:    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
390:    glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
391:    //glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.1f);
392:
393:    // Specifying the light position here causes the position to be
394:    // interpreted in eye coordinates.  The default direction of the
395:    // spotlight is (0, 0, -1), which is also interpreted in eye
396:    // coordinates in this case.
397:
398:    glLoadIdentity();
399:    glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
400:
401:    glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
402:    glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
403:    glLightfv(GL_LIGHT1, GL_SPECULAR, light1_specular);
404:    //glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.1f);
405:    //glLighti(GL_LIGHT1, GL_SPOT_CUTOFF, 5);
406:    //glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 0.2f);
407:
408:    // Declare the viewer to be local.
409:
410:    glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
411:    glEnable(GL_LIGHTING);
412:    glEnable(GL_LIGHT0);
413:    glEnable(GL_LIGHT1);
414:    glEnable(GL_DEPTH_TEST);
415:
416:    textureInit();
417: }
418:
419:
420: void
```

```
421: main(int argc, char **argv)
422: {
423:    // Material properties for the torus.
424:    GLfloat mat_ambient[] = { 0.5, 0.0, 0.0, 1.0 };
425:    GLfloat mat_diffuse[] = { 1.0, 0.5, 0.5, 1.0 };
426:    GLfloat mat_specular[] = { 1.0, 0.5, 0.5, 1.0 };
427:    GLfloat mat_shininess[] = { 100.0 };
428:
429:    glutInit(&argc, argv);
430:    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
431:    glutInitWindowSize(500, 500);
432:    glutCreateWindow("Textures Lab");
433:    glutReshapeFunc(myReshape);
434:    glutDisplayFunc(display);
435:    glutMouseFunc(mouse);
436:    glutKeyboardFunc(keys);
437:    glutIdleFunc(idle);
438:    init();
439:
440:    // Get five list handles.
441:    lists = glGenLists(5);
442:
443:    // For the moving light.
444:    glNewList(lists, GL_COMPILE);
445:       glutSolidCube(0.1);
446:    glEndList();
447:
448:    // Default material values are used with the texture mapped
449:    // objects since that seems to look best.
450:
451:    // For the left cube.
452:    glNewList(lists + 1, GL_COMPILE);
453:       glutSolidCube(2.0);
454:    glEndList();
455:
456:    // The sphere at the origin.
457:    glNewList(lists + 2, GL_COMPILE);
458:       glutSolidSphere(1.0, 32, 32);
459:    glEndList();
460:
461:    // The cone.
462:    glNewList(lists + 3, GL_COMPILE);
463:    glutSolidCone(1.0, 2.0, 32, 32);
464:    glEndList();
465:
466:    // The torus.  Note how we restore and save the default material
467:    // values.
468:
469:    glNewList(lists + 4, GL_COMPILE);
470:    glPushAttrib(GL_COLOR_MATERIAL);
471:    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
472:    glMaterialfv(GL_FRONT, GL_DIFFUSE, mat_diffuse);
473:    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
474:    glMaterialfv(GL_FRONT, GL_SHININESS, mat_shininess);
475:    glutSolidTorus(0.25, 1.0, 32, 32);
476:    glPopAttrib();
477:    glEndList();
478:
479:    glutMainLoop();
480: }
```