

```
1: /* texturelab.c --- Experimentation with lighting and textures in
2:  * OpenGL.
3:  * Tom Kelliher 4/26/11.
4:  *
5:  * This program requires gltx.c and gltx.h. Make sure to add them
6:  * to the project. You can find them on the class Web site.
7:  *
8:  * This version uses the moving light and spotlight anchored to the
9:  * viewer. Use l to turn the spot off and L to turn it back on.
10: * The spotlight is now actually a point source, so that we can
11: * see better --- textures attenuate the lights somewhat.
12: *
13: * Four different textures are read and applied. The torus is not
14: * textured, so as to demonstrate how to enable/disable texturing.
15: *
16: * Automatic texture coordinate generation is used. Note that it
17: * does not work well for cubes --- it is used for the left cube.
18: * Manual texture coordinates are used for the right cube. This
19: * illustrates how to disable and enable automatic texture
20: * coordinate generation.
21: *
22: * Movement is via x/X, y/Y, and z/Z keys. Use space bar to exit.
23: * Mouse buttons rotate the right cube.
24: *
25: * The four texture files, imgfile[1-4].rgb should be in the main
26: * project directory.
27: *
28: *
29: * Experiments:
30: *
31: *     Experiment with BREAK_TEXTURE and OBJECT_COORDS.
32: *
33: *     In textureInit(), vary the use of TEX_IMAGE and MIP_MAP.
34: *     Can you see any difference?
35: *
36: *     Re-configure the viewer-anchored light to be a spotlight,
37: *     rather than a point light source. Notice much of a
38: *     difference?
39: *
40: *     Fix the texture coordinate generation problems of the left
41: *     cube.
42: *
43: *     Create a fifth texture and apply it to the torus.
44: */
45:
46:
47: #include <stdio.h>
48: #include <stdlib.h>
49: #include <GL/glut.h>
50: // The IRIS RGB "library." gltx.c must be added to the project.
51: #include "gltx.h"
52:
53:
54: // Uncomment the following to see the right cube's texture
55: // applied in a manner similar to the left cube. If you study
56: // polygon(), below, you will get a hint as to what's going on.
57: // #define BREAK_TEXTURE
58:
59:
60: // Comment-out the following to have automatic texture coordinates
61: // generated in eye-coordinates rather than object coordinates.
62: #define OBJECT_COORDS
63:
```

```
64:
65: #define TEX_IMAGE 0
66: #define MIP_MAP 1
67:
68:
69: // Base of the display lists.
70: GLuint lists;
71:
72:
73: // The right cube can be rotated.  Whoopee.
74: static GLfloat theta[] = {45.0,0.0,0.0};
75: static GLint axis = 2;
76: /* initial viewer location */
77: static GLdouble viewer[] = {20.0, 6.0, 6.0};
78: //static GLdouble viewer[] = {0.0, 0.0, 10.0};
79:
80:
81: // Light 0 is the moving light.  Light 1 is the spotlight that moves
82: // with us.  Note that these are positional lights.
83: GLfloat light0_position[] = { 0.0, 0.0, 5.0, 1.0 };
84: GLfloat light1_position[] = { 0.0, 0.0, 0.0, 1.0 };
85:
86: // Angle of the moving light.
87: GLint light_angle = 0;
88:
89:
90: // For the textures.
91: GLuint texNames[4];
92:
93:
94: // Vertices for right cube.
95: GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
96: {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
97: {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};
98:
99:
100: // These are the real normals for the right cube, 1 per face.
101: GLfloat normals[][3] = {{0.0,0.0,-1.0},{0.0,1.0,0.0},
102: {-1.0,0.0,0.0},{1.0,0.0,0.0},{0.0,0.0,1.0},{0.0,-1.0,0.0}};
103:
104:
105: // Code for rendering the right cube.
106: void polygon(int a, int b, int c , int d, GLfloat normal[])
107: {
108:     glBegin(GL_POLYGON);
109:         glNormal3fv(normal);
110:         glTexCoord2d(0.0, 0.0);
111:         glVertex3fv(vertices[a]);
112:         glTexCoord2d(0.0, 1.0);
113:         glVertex3fv(vertices[b]);
114: #ifndef BREAK_TEXTURE
115:         glTexCoord2d(1.0, 1.0);
116: #endif
117:         glVertex3fv(vertices[c]);
118: #ifndef BREAK_TEXTURE
119:         glTexCoord2d(1.0, 0.0);
120: #else
121:         glTexCoord2d(0.0, 0.0);
122: #endif
123:         glVertex3fv(vertices[d]);
124:     glEnd();
125: }
```

```
127:
128: void colorcube()
129: {
130:     polygon(0,3,2,1,normals[0]);
131:     polygon(2,3,7,6,normals[1]);
132:     polygon(0,4,7,3,normals[2]);
133:     polygon(1,2,6,5,normals[3]);
134:     polygon(4,5,6,7,normals[4]);
135:     polygon(0,1,5,4,normals[5]);
136: }
137:
138:
139: // Using gltx, read the IRIS file and bind the image to a texture.
140:
141: void getTexture(int *tName, char *fName, int mode)
142: {
143:     GLTXimage *image = gltxReadRGB(fName);
144:
145:     if (!image)
146:     {
147:         printf("Problems with the image file: %s.\n",
148:             fName);
149:         exit(1);
150:     }
151:
152:     // Similar to glGenLists().
153:     glGenTextures(1, tName);
154:
155:     // Similar to glNewList(), but there's nothing corresponding to
156:     // glEndList().
157:     glBindTexture(GL_TEXTURE_2D, *tName);
158:
159:     // Tile the 2-D texture in both dimensions.
160:     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
161:     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
162:
163:     // Use quick and dirty magnification and minification.
164:     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
165:         GL_NEAREST);
166:     glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
167:         GL_NEAREST);
168:
169:     // glTexImage2D() only generates a single version of the texture.
170:     // gluBuild2dMipmaps() generates several scaled versions of the
171:     // texture. It's preferable because it has fewer aliasing
172:     // problems. On the other hand, it uses more texture memory.
173:
174:     if (mode == TEX_IMAGE)
175:         glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image->width,
176:             image->height, 0, GL_RGB, GL_UNSIGNED_BYTE,
177:             image->data);
178:     else if (mode == MIP_MAP)
179:         gluBuild2DMipmaps(GL_TEXTURE_2D, 3, image->width,
180:             image->height, GL_RGB, GL_UNSIGNED_BYTE,
181:             image->data);
182:     else
183:     {
184:         printf("Invalid texture mode.\n");
185:         exit(1);
186:     }
187:
188:     gltxDelete(image);
189: }
```

```
190:
191:
192: void textureInit(void)
193: {
194:     /* Set texel storage format. */
195:     glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
196:
197:     /* Get the textures. */
198:     getTexture(texNames, "imgfile1.rgb", TEX_IMAGE);
199:     getTexture(texNames + 1, "imgfile2.rgb", MIP_MAP);
200:     getTexture(texNames + 2, "imgfile3.rgb", MIP_MAP);
201:     getTexture(texNames + 3, "imgfile4.rgb", MIP_MAP);
202:
203:     /* Use modulated application to work with lighting, enable
204:      * automatic texture coordinate generation, and enable textures.
205:      */
206:     glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE);
207:     glEnable(GL_TEXTURE_GEN_S);
208:     glEnable(GL_TEXTURE_GEN_T);
209: #ifdef OBJECT_COORDS
210:     glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
211:     glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
212: #endif
213:     glEnable(GL_TEXTURE_2D);
214: }
215:
216:
217: // Paint the scene.
218: void display(void)
219: {
220:
221:     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
222:
223:     glEnable(GL_TEXTURE_2D);
224:
225:     /* Update viewer position in modelview matrix */
226:     glLoadIdentity();
227:
228:     gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0,
229:               0.0);
230:
231:     // Position the moving light. It will be positioned in world
232:     // coordinates.
233:     glPushMatrix();
234:     glRotatef(light_angle, 1.0, 0.0, 0.0);
235:     glLightfv(GL_LIGHT0, GL_POSITION, light0_position);
236:
237:     // Render a solid sphere indicating the light's position. Notice
238:     // how it's neither lit nor textured.
239:
240:     glTranslatef(light0_position[0], light0_position[1],
241:                 light0_position[2]);
242:     glDisable(GL_LIGHTING);
243:     glDisable(GL_TEXTURE_2D);
244:     glCallList(lists);
245:     glEnable(GL_TEXTURE_2D);
246:     glEnable(GL_LIGHTING);
247:     glPopMatrix();
248:
249:     // Notice how we indicate which texture to use. This texture
250:     // will be in use until we specify another or disable 2-D
251:     // textures.
252:
```

```
253:     glBindTexture(GL_TEXTURE_2D, texNames[1]);
254:
255:     glPushMatrix();
256:
257:     // Left cube.
258:     glTranslatef(-2.0, 0.0, 0.0);
259:     glCallList(lists + 1);
260:
261:     glBindTexture(GL_TEXTURE_2D, texNames[0]);
262:
263:     // Sphere centered at origin.
264:     glTranslatef(2.0, 0.0, 0.0);
265:     glCallList(lists + 2);
266:
267:     glBindTexture(GL_TEXTURE_2D, texNames[3]);
268:
269:     // Disable automatic texture coordinate generation.
270:     glDisable(GL_TEXTURE_GEN_S);
271:     glDisable(GL_TEXTURE_GEN_T);
272:
273:     // Right cube.
274:     glTranslatef(2.0, 0.0, 0.0);
275:     // Rotate cube.
276:     glRotatef(theta[0], 1.0, 0.0, 0.0);
277:     glRotatef(theta[1], 0.0, 1.0, 0.0);
278:     glRotatef(theta[2], 0.0, 0.0, 1.0);
279:     colorcube();
280:
281:     // Re-enable automatic texture coordinate generation.
282:     glEnable(GL_TEXTURE_GEN_S);
283:     glEnable(GL_TEXTURE_GEN_T);
284:
285:     glBindTexture(GL_TEXTURE_2D, texNames[2]);
286:
287:     // Now that I'm thoroughly confused, let's re-establish the
288:     // origin.
289:     glPopMatrix();
290:
291:     // The cone.
292:     // Up y-axis by 1.
293:     glTranslatef(0.0, 1.0, 0.0);
294:     // Rotate z-axis up to y-axis.
295:     glRotatef(-90.0, 1.0, 0.0, 0.0);
296:     glCallList(lists + 3);
297:
298:     glDisable(GL_TEXTURE_2D);
299:
300:     // The torus.
301:     // Translate up the cone.
302:     glTranslatef(0.0, 0.0, 1.0);
303:     glCallList(lists + 4);
304:
305:     glutSwapBuffers();
306: }
307:
308:
309: // Rotate the right cube. Looks kinda neat when it cuts into the
310: // sphere.
311: void mouse(int btn, int state, int x, int y)
312: {
313:     if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
314:     if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
315:     if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
```

```
316:         theta[axis] += 2.0;
317:         if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
318:         glutPostRedisplay();
319:     }
320:
321:
322: void keys(unsigned char key, int x, int y)
323: {
324:
325:     /* Use x, X, y, Y, z, and Z keys to move viewer */
326:
327:     if(key == 'x') viewer[0]-= 1.0;
328:     if(key == 'X') viewer[0]+= 1.0;
329:     if(key == 'y') viewer[1]-= 1.0;
330:     if(key == 'Y') viewer[1]+= 1.0;
331:     if(key == 'z') viewer[2]-= 1.0;
332:     if(key == 'Z') viewer[2]+= 1.0;
333:     if(key == 'l') glDisable(GL_LIGHT1);
334:     if(key == 'L') glEnable(GL_LIGHT1);
335:     if(key == ' ') exit(0);
336:
337:     printf("v[x]: %f, v[y]: %f, v[z]: %f.\n", viewer[0], viewer[1],
338:           viewer[2]);
339:
340:     glutPostRedisplay();
341: }
342:
343:
344: void myReshape(int w, int h)
345: {
346:     glViewport(0, 0, w, h);
347:
348:     /* Use a perspective view */
349:
350:     glMatrixMode(GL_PROJECTION);
351:     glLoadIdentity();
352:     gluPerspective(35.0, (GLfloat) w/h, 1.0, 100.0);
353:     glMatrixMode(GL_MODELVIEW);
354: }
355:
356:
357: // Rotate the moving light.
358: void idle(void)
359: {
360:     light_angle = (light_angle + 1) % 360;
361:     glutPostRedisplay();
362: }
363:
364:
365: // Set material and light values.
366: void init(void)
367: {
368:     // No ambient light whatsoever.
369:     GLfloat light_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
370:     GLfloat light_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
371:     GLfloat light_specular[] = { 1.0, 1.0, 1.0, 1.0 };
372:
373:     GLfloat light1_ambient[] = { 0.0, 0.0, 0.0, 1.0 };
374:     GLfloat light1_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
375:     GLfloat light1_specular[] = { 1.0, 1.0, 1.0, 1.0 };
376:
377:     //glClearColor (0.2f, 0.3f, 0.4f, 1.0);
378:     glClearColor (0.0f, 0.0f, 0.0f, 1.0);
```

```
379:     glShadeModel (GL_SMOOTH);
380:
381:     // Note use of linear attenuation values, the spot cutoff
382:     // half-angle, and the spot exponent (intensity dropoff
383:     // within the cone).
384:
385:     glLightfv(GL_LIGHT0, GL_AMBIENT, light_ambient);
386:     glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
387:     glLightfv(GL_LIGHT0, GL_SPECULAR, light_specular);
388:     //glLightf(GL_LIGHT0, GL_LINEAR_ATTENUATION, 0.1f);
389:
390:     // Specifying the light position here causes the position to be
391:     // interpreted in eye coordinates. The default direction of the
392:     // spotlight is (0, 0, -1), which is also interpreted in eye
393:     // coordinates in this case.
394:
395:     glLoadIdentity();
396:     glLightfv(GL_LIGHT1, GL_POSITION, light1_position);
397:
398:     glLightfv(GL_LIGHT1, GL_AMBIENT, light1_ambient);
399:     glLightfv(GL_LIGHT1, GL_DIFFUSE, light1_diffuse);
400:     glLightfv(GL_LIGHT1, GL_SPECULAR, light1_specular);
401:     //glLightf(GL_LIGHT1, GL_LINEAR_ATTENUATION, 0.1f);
402:     //glLightf(GL_LIGHT1, GL_SPOT_CUTOFF, 5);
403:     //glLightf(GL_LIGHT1, GL_SPOT_EXPONENT, 0.2f);
404:
405:     // Declare the viewer to be local.
406:
407:     glLightModeli(GL_LIGHT_MODEL_LOCAL_VIEWER, GL_TRUE);
408:     glEnable(GL_LIGHTING);
409:     glEnable(GL_LIGHT0);
410:     glEnable(GL_LIGHT1);
411:     glEnable(GL_DEPTH_TEST);
412:
413:     textureInit();
414: }
415:
416:
417: int
418: main(int argc, char **argv)
419: {
420:     // Material properties for the torus.
421:     GLfloat mat_ambient[] = { 0.5, 0.0, 0.0, 1.0 };
422:     GLfloat mat_diffuse[] = { 1.0, 0.5, 0.5, 1.0 };
423:     GLfloat mat_specular[] = { 1.0, 0.5, 0.5, 1.0 };
424:     GLfloat mat_shininess[] = { 100.0 };
425:
426:     setvbuf(stdout, (char *)NULL, _IONBF, 0);
427:     setvbuf(stderr, (char *)NULL, _IONBF, 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:
481:     return 0;
482: }
```