```
// roomView.c
// Tom Kelliher, 4/4/01.
//
// This program demonstrates viewer movement.  It uses two viewports with
// entirely different views: an overhead view using an orthographic projection
// and an immersed view using perspective projection.
//
// This program models a 20x20 room with two internal dividing walls.


#include <math.h>
#include <GL/glut.h>


// This wall section is 10 feet long (y) , 2 feet wide (x), and 8 feet
// high.  It is centered at the origin, in x and y and its base sits on
// the z = 0 plane.  All walls are constructed from this one section.

GLfloat wall[][3] =
{ {-1.0, -5.0, 0.0}, { 1.0, -5.0, 0.0},
  { 1.0,  5.0, 0.0}, {-1.0,  5.0, 0.0},
  {-1.0, -5.0, 8.0}, { 1.0, -5.0, 8.0},
  { 1.0,  5.0, 8.0}, {-1.0,  5.0, 8.0}
};


// Various colors.

GLfloat red[] = {1.0, 0.0, 0.0};
GLfloat green[] = {0.0, 1.0, 0.0};
GLfloat blue[] = {0.0, 0.0, 1.0};
GLfloat magenta[] = {1.0, 0.0, 1.0};
GLfloat white[] = {1.0, 1.0, 1.0};
GLfloat black[] = {0.0, 0.0, 0.0};


// This is a floor/ceiling section.  It's 24x24 and centered at the
//  origin.

GLfloat Floor[][3] =
{ {-12.0, -12.0, 0.0}, { 12.0, -12.0, 0.0},
  { 12.0,  12.0, 0.0}, {-12.0,  12.0, 0.0}
};


// This is our roving viewer.  It's 1x1 and centered at the origin.

GLfloat viewer[][3] =
{ {-0.5, -0.5, 0.0}, { 0.5, -0.5, 0.0},
  { 0.5,  0.5, 0.0}, {-0.5,  0.5, 0.0}
};


// Initial window dimensions.

int width = 500;
int height = 500;


// Constants necessary for rotating the viewer.  The trig functions take
// radian arguments.  (Sigh.)

const float PI = 3.14159f;
const float ROTATE =   0.7853975f;   // 1/8th of a complete revolution in
                                     // radians.


// Viewer state.

GLfloat viewerTheta = 1.570795f;     // View direction.  Initially looking
                                     // up positive y axis.
GLfloat viewerPosition[] = {0.0, 0.0, 6.0};   // Initial view position.


//
// polygon renders a polygon:
//    polygon: name of the vertex list.
//    a, b, c, d: indices of the vertex list vertices to be rendered.
//    color: the color vector.
//
```

```
  81:
  82: void polygon(GLfloat polygon[][3], int a, int b, int c , int d,
  83:              GLfloat *color)
  84: {
  85:    glPushAttrib(GL_ALL_ATTRIB_BITS);
  86:    glBegin(GL_POLYGON);
  87:       glColor3fv(color);
  88:       glVertex3fv(polygon[a]);
  89:       glVertex3fv(polygon[b]);
  90:       glVertex3fv(polygon[c]);
  91:       glVertex3fv(polygon[d]);
  92:    glEnd();
  93:    glPopAttrib();
  94: }
  95:
  96:
  97: //
  98: // colorCube renders a cube.
  99: //    cube: the vertex list specifying the cube.
 100: //    color: the color vector.
 101: //
 102: // Assumptions regarding the vertex list:
 103: // Index   Vertex
 104: //   0     Lower left vertex of back face.
 105: //   1     Lower right vertex of back face.
 106: //   2     Upper right vertex of back face.
 107: //   3     Upper left vertex of back face.
 108: // 4--7    Similar for front face.
 109: // (Assumes we are looking at the origin from the +z axis with the +y axis
 110: // being "up."
 111: //
 112:
 113: void colorCube(GLfloat cube[][3], GLfloat *color)
 114: {
 115:    polygon(cube,0,3,2,1,color);
 116:    polygon(cube,2,3,7,6,color);
 117:    polygon(cube,0,4,7,3,color);
 118:    polygon(cube,1,2,6,5,color);
 119:    polygon(cube,4,5,6,7,color);
 120:    polygon(cube,0,1,5,4,color);
 121: }
 122:
 123:
 124: //
 125: // room renders the room for me.  Because of the needs of the two views, it
 126: // renders neither the ceiling nor the viewer.
 127: //
 128:
 129: void room()
 130: {
 131:         int i;
 132:
 133:         // Render floor.
 134:         polygon(Floor, 0, 1, 2, 3, white);
 135:
 136:         // Render middle walls.
 137:         glPushMatrix();
 138:         glTranslatef(-4.0, 0.0, 0.0);
 139:         colorCube(wall, red);
 140:         glTranslatef(8.0, 0.0, 0.0);
 141:         colorCube(wall, red);
 142:         glPopMatrix();
 143:
 144:         // Render walls defining room boundaries.
 145:         glPushMatrix();
 146:         glTranslatef(-11.0, -5.0, 0.0);
 147:         for (i = 0; i < 4; ++i)
 148:         {
 149:                 colorCube(wall, (i == 1) ? green : blue);
 150:                 glTranslatef(0.0, 10.0, 0.0);
 151:                 colorCube(wall, (i == 1) ? green : blue);
 152:                 glTranslatef(6.0, 6.0, 0.0);
 153:                 glRotatef(-90.0, 0.0, 0.0, 1.0);
 154:         }
 155:         glPopMatrix();
 156: }
 157:
 158:
 159: //
 160: // display is where the work gets done.
```

```
161: //
162:
163: void display(void)
164: {
165:         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
166:
167:         // Set-up to do the overhead view.
168:         glMatrixMode(GL_PROJECTION);
169:         glLoadIdentity();
170:         glOrtho(-13.0, 13.0, -13.0, 13.0, -10.0, 1.0);
171:         glMatrixMode(GL_MODELVIEW);
172:
173:         glViewport(0, 0, 100, 100);
174:
175:         glLoadIdentity();
176:
177:         // Render viewer.
178:         glPushMatrix();
179:         glTranslatef(viewerPosition[0], viewerPosition[1], 10.0);
180:         polygon(viewer, 0, 1, 2, 3, black);
181:         glPopMatrix();
182:
183:         // Render the room.
184:         room();
185:
186:         glFlush();
187:
188:         // Set-up the do the "immersed" view.
189:         glMatrixMode(GL_PROJECTION);
190:         glLoadIdentity();
191:         gluPerspective(90.0, 1.0, 0.1, 100.0);
192:         glMatrixMode(GL_MODELVIEW);
193:
194:         glViewport(100, 100, width - 100, height - 100);
195:
196:         glPushMatrix();
197:
198:         // Position the camera.
199:         gluLookAt(viewerPosition[0], viewerPosition[1], viewerPosition[2],
200:                 viewerPosition[0] + cos(viewerTheta),
201:       viewerPosition[1] + sin(viewerTheta), viewerPosition[2], 0.0, 0.0, 1.0);
202:
203:     room();
204:
205:         // Render ceiling.
206:         glPushMatrix();
207:         glTranslatef(0.0, 0.0, 8.0);
208:         polygon(Floor, 0, 1, 2, 3, magenta);
209:         glPopMatrix();
210:
211:         glutSwapBuffers();
212: }
213:
214:
215: //
216: // reshape doesn't do too much right now.  It should really make sure we handle
217: // aspect ratios correctly.
218: //
219:
220: void reshape(int w, int h)
221: {
222:         width = w;
223:         height = h;
224:
225:         glutPostRedisplay();
226: }
227:
228:
229: //
230: // Callback for the non-ASCII keys.  I.e., the arrow keys.
231: //
232:
233: void special(int key, int x, int y)
234: {
235:         switch (key)
236:         {
237:         case GLUT_KEY_LEFT:
238:                 --viewerPosition[0];
239:                 break;
240:
```

```
241:         case GLUT_KEY_RIGHT:
242:                 ++viewerPosition[0];
243:                 break;
244:
245:         case GLUT_KEY_UP:
246:                 ++viewerPosition[1];
247:                 break;
248:
249:         case GLUT_KEY_DOWN:
250:                 --viewerPosition[1];
251:                 break;
252:
253:         default:
254:                 return;
255:         }
256:
257:         glutPostRedisplay();
258: }
259:
260:
261: //
262: // Callback for the ASCII keys.  "A" is rotate left and "S" is rotate right.
263: //
264:
265: void keyboard(unsigned char key, int x, int y)
266: {
267:         switch (key)
268:         {
269:         case 'a':
270:         case 'A':
271:                 break;
272:
273:         case 's':
274:         case 'S':
275:                 break;
276:
277:         default:
278:                 return;
279:         }
280:
281:         if (viewerTheta >= 2 * PI)
282:                 viewerTheta -= 2 * PI;
283:
284:         glutPostRedisplay();
285: }
286:
287:
288: //
289: // main just initializes the display modes, creates the window, and registers
290: // various callbacks.
291: //
292:
293: int main(int argc, char *argv[])
294: {
295:    glutInit(&argc, argv);
296:    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
297:    glutInitWindowSize(width, height);
298:    glutCreateWindow("Room with a View");
299:    glutReshapeFunc(reshape);
300:    glutDisplayFunc(display);
301:    glutSpecialFunc(special);
302:    glutKeyboardFunc(keyboard);
303:    glEnable(GL_DEPTH_TEST);
304:
305:    glutMainLoop();
306:
307:    return 0;
308: }
```