```
 1: /*
 2:  *  pong.cpp
 3:  *  This is a simple double buffered program.
 4:  *  Use the left and right arrow keys to move the paddle.
 5:  *  Use the home key to re-center the paddle and return the ball "home."
 6:  *  The beginnings of the classic video game.  I say "The beginnings"
 7:  *  because this program doesn't perform any collision detection or response.
 8:  */
 9:
10:
11: #include <unistd.h>
12: #include "Angel.h"
13:
14:
15: const int SLICES = 72;  // For generating the circle's vertices.
16:
17:
18: /* When drawing, we need to switch between the VAO for the paddle and the
19:  * ball, hence we need global variables for holding the VAO handles.
20:  */
21:
22: vec3 paddlePos(0.0, -47.5, 0.0);      // Initial position.
23: vec3 paddleVel(5.0, 0.0, 0.0);        /* Magnitude of velocity when moving
24:                                         * the paddle left or right.
25:                                         */
26: vec4 paddleCol(1.0, 1.0, 1.0, 1.0);   // Color.
27: GLuint paddleVAO;
28:
29:
30: const vec3 START_POS(0.0, 0.0, 0.0);
31: const vec3 START_VEL(0.1, 0.5, 0.0);
32: vec3 ballPos = START_POS;
33: vec3 ballVel = START_VEL;
34: vec4 ballCol(1.0, 1.0, 1.0, 1.0);
35: GLuint ballVAO;
36:
37:
38: /* Handles for uniform variables in the vertex shader. */
39:
40: GLuint projection;
41: GLuint model_view;
42: GLuint vColor;
43:
44:
45: /* Creates a VAO and buffers vertices for a circle centered at the origin
46:  * in the z = 0.0 plane.  loc is the handle for vPosition in the vertex
47:  * shader.  Returns the VAO handle.
48:  */
49:
50: GLuint createCircle(GLfloat radius, GLuint loc)
51: {
52:    GLuint vao, buffer;
53:
54:    vec3 points[SLICES];
55:    GLfloat angle = 0.0;
56:    GLfloat sliceAngle = 2.0 * M_PI / (GLfloat) SLICES;
57:
58:    for (int i = 0; i < SLICES; i++)
59:    {
60:       points[i] = radius * vec3(cos(angle), sin(angle), 0.0);
61:       angle += sliceAngle;
62:    }
63:
```

```
   64:    glGenVertexArrays(1, &vao);
   65:    glBindVertexArray(vao);
   66:
   67:    glGenBuffers(1, &buffer);
   68:    glBindBuffer(GL_ARRAY_BUFFER, buffer);
   69:
   70:    glBufferData(GL_ARRAY_BUFFER, sizeof(points),
   71:                 points, GL_STATIC_DRAW);
   72:
   73:    glEnableVertexAttribArray(loc);
   74:    glVertexAttribPointer(loc, 3, GL_FLOAT, GL_FALSE, 0,
   75:                          BUFFER_OFFSET(0));
   76:
   77:    return vao;
   78: }
   79:
   80:
   81: void display(void)
   82: {
   83:    mat4 mv;   /* Model view matrix */
   84:
   85:    glClear(GL_COLOR_BUFFER_BIT);
   86:
   87:    // Draw the paddle.
   88:
   89:    glBindVertexArray(paddleVAO);
   90:    glUniform4fv(vColor, 1, paddleCol);
   91:    mv = Translate(paddlePos);
   92:    glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);
   93:    glDrawArrays(GL_TRIANGLE_FAN, 0, 4);
   94:
   95:    // Draw the ball.
   96:
   97:    glBindVertexArray(ballVAO);
   98:    glUniform4fv(vColor, 1, ballCol);
   99:    mv = Translate(ballPos);
  100:    glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);
  101:    glDrawArrays(GL_TRIANGLE_FAN, 0, SLICES);
  102:
  103:    glutSwapBuffers();
  104: }
  105:
  106:
  107: void init(void)
  108: {
  109:    /* Vertices for the paddle, centered at the origin.  Ultimately,
  110:     * these will be buffered, hence we use a local array.
  111:     */
  112:
  113:    vec3 points[4] = { vec3(-10.0, -2.5, 0.0), vec3(10.0, -2.5, 0.0),
  114:                       vec3(10.0, 2.5, 0.0), vec3(-10.0, 2.5, 0.0)
  115:                     };
  116:
  117:    glClearColor (0.0, 0.0, 0.0, 1.0);
  118:    glShadeModel (GL_FLAT);
  119:
  120:    /* Compile, link, and bind the shader programs. */
  121:
  122:    GLuint program = InitShader("vshader.glsl", "fshader.glsl");
  123:    glUseProgram(program);
  124:
  125:    /* Get the locations of the uniform variables in the vertex shader.
  126:     * Because we'll be updating these values throughout the execution of
```

```
127:     * the program, these variables are all global.  Because we're not
128:     * loading differing projection matrices into the vertex shader,
129:     * we really don't need a global for the projection variable, but
130:     * setting it up this way would allow us to adjust to aspect ratio
131:     * changes at a later time.
132:     */
133:
134:    vColor = glGetUniformLocation(program, "vColor");
135:    model_view = glGetUniformLocation(program, "model_view");
136:    projection = glGetUniformLocation(program, "projection");
137:
138:    // Get the location of the vertex shader's vPosition input.
139:
140:    GLuint vPosition = glGetAttribLocation(program, "vPosition");
141:
142:    // Create a VAO for the paddle and buffer its vertices.
143:
144:    glGenVertexArrays(1, &paddleVAO);
145:    glBindVertexArray(paddleVAO);
146:
147:    GLuint buffer;
148:    glGenBuffers(1, &buffer);
149:    glBindBuffer(GL_ARRAY_BUFFER, buffer);
150:    glBufferData(GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW);
151:
152:    glEnableVertexAttribArray(vPosition);
153:    glVertexAttribPointer(vPosition, 3, GL_FLOAT, GL_FALSE, 0,
154:                          BUFFER_OFFSET(0));
155:
156:    /* Create a VAO for the ball, buffering its vertices. */
157:
158:    ballVAO = createCircle(5.0, vPosition);
159: }
160:
161:
162: void reshape(int w, int h)
163: {
164:    mat4 p;    /* Projection matrix */
165:
166:    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
167:
168:    p = Ortho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
169:    glUniformMatrix4fv(projection, 1, GL_TRUE, p);
170: }
171:
172:
173: void keyboard(int key, int x, int y)
174: {
175:    // Limited here by keyboard repeat rate.
176:
177:         switch (key)
178:         {
179:         case GLUT_KEY_LEFT:
180:                 paddlePos -= paddleVel;
181:                 break;
182:
183:         case GLUT_KEY_RIGHT:
184:                 paddlePos += paddleVel;
185:                 break;
186:
187:         case GLUT_KEY_HOME:
188:                 paddlePos = vec3(0.0, -47.5, 0.0);
189:                 ballPos = START_POS;
```

```
  190:       ballVel = START_VEL;
  191:                 break;
  192:
  193:    default:
  194:       break;
  195:         }
  196:
  197:    /* Presumably, we've moved the paddle, so redraw the scene. */
  198:
  199:         glutPostRedisplay();
  200: }
  201:
  202:
  203: /* Check for collisions, and respond, and update the ball's position.
  204:  * Collision response is naive --- simply negate the affected velocity
  205:  * component.  When the paddle is struck, the velocity is increased by
  206:  * 15%, and then the velocity's x component is increased by another 10%.
  207:  */
  208:
  209: void idle(void)
  210: {
  211:    /* Handle paddle collisions. */
  212:    if (ballPos.y - 5.0 <= -45.0
  213:        && ballPos.x >= paddlePos.x - 10.0
  214:        && ballPos.x <= paddlePos.x + 10.0)
  215:    {
  216:       ballVel.y = -ballVel.y;
  217:       ballVel *= 1.15;
  218:       ballVel.x *= 1.1;
  219:    }
  220:
  221:    /* Handle collisions with side walls. */
  222:
  223:    if (ballPos.x + 5.0 >= 50.0 || ballPos.x - 5.0 <= -50.0)
  224:       ballVel.x = -ballVel.x;
  225:
  226:    /* Handle collisions with top wall. */
  227:
  228:    if (ballPos.y + 5.0 >= 50.0)
  229:       ballVel.y = -ballVel.y;
  230:
  231:    /* Handle the ball exiting the field of play via the screen bottom. */
  232:
  233:    if (ballPos.y + 5.0 <= -50.0)
  234:    {
  235: #ifdef WIN32
  236:       Sleep(1000);
  237: #else
  238:       sleep(1);
  239: #endif
  240:       ballPos = START_POS;
  241:       ballVel = START_VEL;
  242:    }
  243:
  244:    ballPos += ballVel;
  245:         glutPostRedisplay();
  246: }
  247:
  248:
  249: /*
  250:  *  Request double buffer display mode.
  251:  *  Register "special" input callback functions for arrow and home keys.
  252:  */
```

```
253:
254: int main(int argc, char** argv)
255: {
256:    glutInit(&argc, argv);
257:    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
258:    glutInitWindowSize (500, 500);
259:    glutInitWindowPosition (100, 100);
260:    glutInitContextVersion(3, 2);
261:    glutInitContextProfile(GLUT_CORE_PROFILE);
262:    glutCreateWindow ("Pong");
263:
264:    glewExperimental = GL_TRUE;
265:    glewInit();
266:
267:    init ();
268:
269:    glutDisplayFunc(display);
270:    glutReshapeFunc(reshape);
271:    glutSpecialFunc(keyboard);
272:    glutIdleFunc(idle);
273:
274:    glutMainLoop();
275:
276:    return 0;
277: }
```