```
/*
 *  double.cpp
 *  This is a simple double buffered program.
 *  Pressing the left mouse button rotates the square.
 *  Pressing the middle or right mouse button stops the rotation.
 */


#include "Angel.h"


/* spin is the square's rotation angle.  It's updated by spinDisplay(),
 * which is registered as the idle callback by the mouse callback with
 * a left mouse click.  A middle or right mouse click de-registers the
 * idle callback.  Look at mouse() to see how this is accomplished.
 * spin's units are degress.
 */

GLfloat spin = 0.0;


/* The amount, in degrees, to increase spin by each time spinDisplay()
 * is called.
 */

GLfloat spinDelta = 0.5;


/* Handles for the projection and model view uniform variables in the
 * vertex shader.
 */

GLuint projection;
GLuint model_view;


void display(void)
{
   mat4 mv;   /* Model view matrix */

   glClear(GL_COLOR_BUFFER_BIT);

   /* RotateZ() constructs a rotation matrix using the Z axis as the
    * axis of rotation.  Note that the units of the rotation angle are
    * degrees, not radians.
    */

   mv = RotateZ(spin);
   glUniformMatrix4fv(model_view, 1, GL_TRUE, mv);

   glDrawArrays(GL_TRIANGLE_FAN, 0, 4);

   glutSwapBuffers();
}


void spinDisplay(void)
{
   spin = spin + spinDelta;
   if (spin > 360.0)
      spin = spin - 360.0;

   glutPostRedisplay();
```

```
}


void init(void)
{
   /* The square's color.  It will be passed to the vertex shader as a
    * uniform variable.  Once passed to the vertex shader, it is no
    * longer needed, so we only need a local variable.
    */

   vec4 color = vec4(1.0, 1.0, 1.0, 1.0);

   /* The square's vertices.  This data will be buffered in the GL
    * server, so we only need a local variable for this data.
    */

   vec2 points[4] = { vec2(-25.0, -25.0), vec2(25.0, -25.0),
                      vec2(25.0, 25.0), vec2(-25.0, 25.0)
                    };

   glClearColor (0.0, 0.0, 0.0, 1.0);
   glShadeModel (GL_FLAT);

   /* Compile and link the shader programs. */

   GLuint program = InitShader("vshader.glsl", "fshader.glsl");

   /* Bind the shader programs. */

   glUseProgram(program);

   /* Get the locations of the uniform variables in the vertex shader.
    *
    * IMPORTANT NOTE: Observe that color is declared as a vec4 above,
    * vColor is declared in the vertex shader as a vec4, and that color
    * is assigned to vColor by a call to glUniform4fv().  The source,
    * color, is a 4-tuple, the target, vColor, is a 4-tuple, and we're
    * copying a 4-tuple (the "4fv" in glUniform4fv()) from source to
    * target.  Missing any of these details will likely cause color
    * data to not be drawn.
    */

   GLuint vColor = glGetUniformLocation(program, "vColor");
   glUniform4fv(vColor, 1, color);

   /* The model_view uniform variable receives its value in display(). */

   model_view = glGetUniformLocation(program, "model_view");

   /* The projection uniform variable receives its value in reshape(). */

   projection = glGetUniformLocation(program, "projection");

   /* This VAO remains bound throughout the execution of the program,
    * hence we only need a local variable to perform the one and only
    * VAO binding.
    */

   GLuint vao;
   glGenVertexArrays(1, &vao);
   glBindVertexArray(vao);

   GLuint buffer;
```

```
  127:    glGenBuffers(1, &buffer);
  128:    glBindBuffer(GL_ARRAY_BUFFER, buffer);
  129:    glBufferData(GL_ARRAY_BUFFER, sizeof(points), points, GL_STATIC_DRAW);
  130:
  131:    GLuint loc = glGetAttribLocation(program, "vPosition");
  132:    glEnableVertexAttribArray(loc);
  133:    glVertexAttribPointer(loc, 2, GL_FLOAT, GL_FALSE, 0,
  134:                          BUFFER_OFFSET(0));
  135: }
  136:
  137:
  138: void reshape(int w, int h)
  139: {
  140:    mat4 p;    /* Projection matrix */
  141:
  142:    glViewport (0, 0, (GLsizei) w, (GLsizei) h);
  143:
  144:    /* It strikes me that this is overkill, since we aren't doing anything
  145:     * to correct aspect ratio mismatches.
  146:     */
  147:
  148:    p = Ortho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
  149:    glUniformMatrix4fv(projection, 1, GL_TRUE, p);
  150: }
  151:
  152:
  153: void mouse(int button, int state, int x, int y)
  154: {
  155:    switch (button) {
  156:
  157:       case GLUT_LEFT_BUTTON:
  158:          if (state == GLUT_DOWN)
  159:             glutIdleFunc(spinDisplay);
  160:
  161:          break;
  162:
  163:       case GLUT_MIDDLE_BUTTON:
  164:       case GLUT_RIGHT_BUTTON:
  165:          if (state == GLUT_DOWN)
  166:             glutIdleFunc(NULL);
  167:
  168:          break;
  169:
  170:       default:
  171:          break;
  172:    }
  173: }
  174:
  175:
  176: /*
  177:  *  Request double buffer display mode.
  178:  *  Register mouse input callback functions
  179:  */
  180:
  181: int main(int argc, char** argv)
  182: {
  183:    glutInit(&argc, argv);
  184:    glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB);
  185:    glutInitWindowSize (250, 250);
  186:    glutInitWindowPosition (100, 100);
  187:    glutInitContextVersion(3, 2);
  188:    glutInitContextProfile(GLUT_CORE_PROFILE);
  189:    glutCreateWindow ("Double");
```

```
190:
191:    glewExperimental = GL_TRUE;
192:    glewInit();
193:
194:    init ();
195:
196:    glutDisplayFunc(display);
197:    glutReshapeFunc(reshape);
198:    glutMouseFunc(mouse);
199:
200:    glutMainLoop();
201:
202:    return 0;
203: }
```