```
    1: /* Primitive "Paint" program --- paints up to MAXOBJECTS red triangles. */
    2:
    3: #include "Angel.h"
    4:
    5: /* Width and height of display window */
    6: const int w = 512;
    7: const int h = 512;
    8:
    9: /* Basic color data */
   10: int currentColor = 0;
   11: const vec3  base_colors[] = {
   12:    vec3( 1.0, 0.0, 0.0 ), // red
   13:    vec3( 0.0, 1.0, 0.0 ), // green
   14:    vec3( 0.0, 0.0, 1.0 ), // blue
   15:    vec3( 0.0, 0.0, 0.0 )  // black
   16: };
   17:
   18: /* Max vertices per object */
   19: const int MAXVERTICES = 10;
   20:
   21: /* points and colors are used to collect vertex data while we're "building"
   22:  * an object
   23:  */
   24: vec2 points[MAXVERTICES];
   25: vec3 colors[MAXVERTICES];
   26:
   27: /* Next available location in points and colors */
   28: int  Index = 0;
   29:
   30: /* Maximum number of "paintable" objects */
   31: const int MAXOBJECTS = 10;
   32:
   33: /* Data structure for storing an object --- the vao and the number of
   34:  * stored in the vao.
   35:  */
   36: typedef struct object{
   37:    GLuint vao;
   38:    GLuint numVertices;
   39: } object;
   40:
   41: /* Our array of objects */
   42: object objects[MAXOBJECTS];
   43:
   44: /* Index of next available object */
   45: int nextObject = 0;
   46:
   47: /* Identifier for the shader programs */
   48: GLuint program;
   49:
   50: //----------------------------------------------------------------------------
   51:
   52: /* Create and return a vao using the current vertex information in points
   53:  * and colors, and the current (unchanging) shader programs.
   54:  */
   55:
   56: GLuint
   57: createVao( void )
   58: {
   59:    GLuint vao;
   60:    GLuint buffer;
   61:
   62:    // Create a vertex array object
   63:    glGenVertexArrays( 1, &vao );
```

```
   64:    glBindVertexArray( vao );
   65:
   66:    // Create and initialize a buffer object
   67:    glGenBuffers( 1, &buffer );
   68:    glBindBuffer( GL_ARRAY_BUFFER, buffer );
   69:
   70:    // First, we create an empty buffer of the size we need by passing
   71:    //   a NULL pointer for the data values
   72:    glBufferData( GL_ARRAY_BUFFER, sizeof(points) + sizeof(colors),
   73:                  NULL, GL_STATIC_DRAW );
   74:
   75:    // Next, we load the real data in parts.  We need to specify the
   76:    //   correct byte offset for placing the color data after the point
   77:    //   data in the buffer.  Conveniently, the byte offset we need is
   78:    //   the same as the size (in bytes) of the points array, which is
   79:    //   returned from "sizeof(points)".
   80:    glBufferSubData( GL_ARRAY_BUFFER, 0, sizeof(points), points );
   81:    glBufferSubData( GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors );
   82:
   83:    glUseProgram( program );
   84:
   85:    // Initialize the vertex position attribute from the vertex shader
   86:    GLuint vPosition = glGetAttribLocation( program, "vPosition" );
   87:    glEnableVertexAttribArray( vPosition );
   88:    glVertexAttribPointer( vPosition, 2, GL_FLOAT, GL_FALSE, 0,
   89:                           BUFFER_OFFSET(0) );
   90:
   91:    // Likewise, initialize the vertex color attribute.  Once again, we
   92:    //    need to specify the starting offset (in bytes) for the color
   93:    //    data.  Just like loading the array, we use "sizeof(points)"
   94:    //    to determine the correct value.
   95:    GLuint vColor = glGetAttribLocation( program, "vColor" );
   96:    glEnableVertexAttribArray( vColor );
   97:    glVertexAttribPointer( vColor, 3, GL_FLOAT, GL_FALSE, 0,
   98:                           BUFFER_OFFSET(sizeof(points)) );
   99:
  100:    return vao;
  101: }
  102: //----------------------------------------------------------------------------
  103:
  104: void
  105: init( void )
  106: {
  107:    // Load shaders and use the resulting shader program
  108:    program = InitShader( "vshader24.glsl", "fshader24.glsl" );
  109:
  110:    glClearColor( 1.0, 1.0, 1.0, 1.0 ); /* white background */
  111: }
  112:
  113: //----------------------------------------------------------------------------
  114:
  115: /* Paint the current objects */
  116:
  117: void
  118: display( void )
  119: {
  120:    glClear( GL_COLOR_BUFFER_BIT );
  121:
  122:    for (int i = 0; i < nextObject; i++)
  123:    {
  124:       glBindVertexArray( objects[i].vao );
  125:       glDrawArrays( GL_TRIANGLES, 0, objects[i].numVertices );
  126:    }
```

```
127:
128:    glFlush();
129: }
130:
131: //----------------------------------------------------------------------------
132:
133: void
134: keyboard( unsigned char key, int x, int y )
135: {
136:    switch ( key ) {
137:    case 033:
138:       exit( EXIT_SUCCESS );
139:       break;
140:    }
141: }
142:
143: //----------------------------------------------------------------------------
144:
145: /* Convert window coordinates, in pixels, back to world coordinates */
146:
147: vec2 convert(int x, int y)
148: {
149:    return vec2((GLfloat) x / (GLfloat) w,
150:                (GLfloat) y / (GLfloat) h);
151: }
152:
153: //----------------------------------------------------------------------------
154:
155: /* Mouse callback function */
156:
157: void
158: mouse( int button, int state, int x, int y )
159: {
160:    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
161:    {
162:       std::cout << x << ", " << y << std::endl;
163:       points[Index] = convert(x, y);
164:       std::cout << points[Index].x << ", " << points[Index].y << std::endl;
165:       colors[Index++] = base_colors[currentColor];
166:
167:       if (Index == 3 && nextObject < MAXOBJECTS)
168:       {
169:          objects[nextObject].numVertices = Index;
170:          Index = 0;
171:          objects[nextObject++].vao = createVao();
172:          glutPostRedisplay();
173:
174:       }
175:    }
176: }
177:
178: //----------------------------------------------------------------------------
179:
180: int
181: main( int argc, char **argv )
182: {
183:    glutInit( &argc, argv );
184:    glutInitDisplayMode( GLUT_RGBA );
185:    glutInitWindowSize( w, h );
186:    glutInitContextVersion( 3, 2 );
187:    glutInitContextProfile( GLUT_CORE_PROFILE );
188:    glutCreateWindow( "Simple Paint Program" );
189:
```

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