```
 1: // $Id: ellipses.cpp,v 1.8 2014-07-22 16:38:06-07 - - $
 2:
 3: // Draw several ellipses in window.
 4:
 5: #include <cmath>
 6: #include <iostream>
 7: #include <string>
 8: using namespace std;
 9:
10: #include <GL/freeglut.h>
11: #include <libgen.h>
12:
13: using uchar = unsigned char;
14:
15: // Characteristics of the window.
16: struct window {
17:    string name;
18:    int width {512};
19:    int height {384};
20: } window;
21:
22: const uchar RED[] = {0xFF, 0x00, 0x00};
23: const uchar CYAN[] = {0x00, 0xFF, 0xFF};
24: const uchar BLUE[] = {0x00, 0x00, 0xFF};
25: const uchar YELLOW[] = {0xFF, 0xFF, 0x00};
26:
27: void draw_ellipse (int kind, const uchar* color, float scale) {
28:    glBegin (kind);
29:    glEnable (GL_LINE_SMOOTH);
30:    glColor3ubv (color);
31:    const float delta = 2 * M_PI / 32;
32:    float width = window.width / 3 * scale;
33:    float height = window.height / 3 * scale;
34:    for (float theta = 0; theta < 2 * M_PI; theta += delta) {
35:       float xpos = width * cos (theta) + window.width / 2;
36:       float ypos = height * sin (theta) + window.height / 2;
37:       glVertex2f (xpos, ypos);
38:    }
39:    glEnd();
40: }
41:
42: // Called by glutMainLoop to display window contents.
43: void display() {
44:    cout << __func__ << "()" << endl;
45:    glClearColor (0.25, 0.25, 0.25, 1.0);
46:    glClear (GL_COLOR_BUFFER_BIT);
47:    glLineWidth (4);
48:    draw_ellipse (GL_POLYGON, CYAN, 1.0);
49:    draw_ellipse (GL_LINE_LOOP, RED, 1.0);
50:    draw_ellipse (GL_POLYGON, YELLOW, 0.5);
51:    draw_ellipse (GL_LINE_LOOP, BLUE, 0.5);
52:    glutSwapBuffers();
53: }
54:
```

```
55:
56: void reshape (int width, int height) {
57:    cout << __func__ << "(" << width << "," << height << ")" << endl;
58:    window.width = width;
59:    window.height = height;
60:    glMatrixMode (GL_PROJECTION);
61:    glLoadIdentity();
62:    glOrtho (0, window.width, 0, window.height, -1, +1);
63:    glMatrixMode (GL_MODELVIEW);
64:    glViewport (0, 0, window.width, window.height);
65:    glutPostRedisplay();
66: }
67:
68: void close() {
69:    cout << __func__ << "()" << endl;
70: }
71:
72: void entry (int state) {
73:    cout << __func__ << "(";
74:    switch (state) {
75:       case GLUT_LEFT: cout << "GLUT_LEFT"; break;
76:       case GLUT_ENTERED: cout << "GLUT_ENTERED"; break;
77:       default: cout << state; break;
78:    }
79:    cout << ")" << endl;
80: }
81:
82: int main (int argc, char** argv) {
83:    window.name = basename (argv[0]);
84:    glutInit (&argc, argv);
85:    glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
86:    glutInitWindowSize (window.width, window.height);
87:    glutInitWindowPosition (128, 128);
88:    glutCreateWindow (window.name.c_str());
89:    glutDisplayFunc (display);
90:    glutReshapeFunc (reshape);
91:    glutEntryFunc (entry);
92:    glutCloseFunc (close);
93:    glutMainLoop();
94:    return 0;
95: }
96:
97: //TEST// mkpspdf hello-gl.ps hello-gl.cpp*
```

```
 1: // $Id: glclock.cpp,v 1.13 2014-07-22 16:38:07-07 - - $
 2:
 3: // Show a real-time analog clock.
 4:
 5: #include <cmath>
 6: #include <iostream>
 7: using namespace std;
 8:
 9: #include <GL/freeglut.h>
10: #include <libgen.h>
11: #include <time.h>
12:
13: int window_width = 128;
14: int window_height = 128;
15: string program_name;
16: float radius = 0.9;
17:
18: struct calend {
19:    time_t clock;
20:    struct tm localtime;
21:    char sdate[64];
22:    char stime[64];
23:    void set() {
24:       clock = time (NULL);
25:       localtime_r (&clock, &localtime);
26:       strftime (sdate, sizeof sdate, "%a %b %e", &localtime);
27:       strftime (stime, sizeof stime, "%T", &localtime);
28:    }
29: } calend;
30:
31: void show_time() {
32:    void* font = GLUT_BITMAP_TIMES_ROMAN_10;
33:    glRasterPos2f (-0.95, -0.95);
34:    glutBitmapString (font, (GLubyte*) calend.sdate);
35:    float timewidth = glutBitmapLength (font, (GLubyte*) calend.stime);
36:    float timexpos = 0.95 - 2 * timewidth / window_width;
37:    glRasterPos2f (timexpos, -.95);
38:    glutBitmapString (font, (GLubyte*) calend.stime);
39: }
40:
41: void draw_dots (int points, int count) {
42:    glEnable (GL_POINT_SMOOTH);
43:    glPointSize (points);
44:    glBegin(GL_POINTS);
45:    for (float theta = 0; theta < 2 * M_PI; theta += 2 * M_PI / count) {
46:       float xdot = 0.9 * radius * cos (theta);
47:       float ydot = 0.9 * radius * sin (theta);
48:       glVertex2f (xdot, ydot);
49:    }
50:    glEnd();
51: }
52:
```

```
53:
54: void draw_hand (GLfloat width, GLfloat length, GLfloat clock) {
55:    glEnable (GL_LINE_SMOOTH);
56:    glEnable (GL_POLYGON_SMOOTH);
57:    glPushMatrix();
58:    glRotatef (-clock * 6, 0, 0, 1);
59:    glColor3ub (0x2F, 0xFF, 0x2F);
60:    glBegin (GL_POLYGON);
61:    glVertex2f (-width / 2 * radius, 0);
62:    glVertex2f (+width / 2 * radius, 0);
63:    glVertex2f (+width / 8, length * radius);
64:    glVertex2f (-width / 8, length * radius);
65:    glEnd();
66:    glPopMatrix();
67: }
68:
69: void display() {
70:    glClear (GL_COLOR_BUFFER_BIT);
71:    glColor3ub (0x2F, 0xFF, 0x2F);
72:    draw_dots (2, 60);
73:    draw_dots (5, 12);
74:    calend.set();
75:    float second = calend.localtime.tm_sec;
76:    float minute = calend.localtime.tm_min + second / 60;
77:    float hour = calend.localtime.tm_hour + minute / 60;
78:    draw_hand (0.2, 0.5, hour * 5);
79:    draw_hand (0.1, 0.75, minute);
80:    draw_hand (0.05, 0.95, second);
81:    show_time();
82:    glutSwapBuffers();
83: }
84:
85: const float frequency = 500;
86: void timer (int) {
87:    glutTimerFunc (frequency, timer, 100);
88:    glutPostRedisplay();
89: }
90:
```

```
 91:
 92: void reshape (int width, int height) {
 93:    cout << "reshape(width=" << width << ", height=" << height << endl;
 94:    window_width = width;
 95:    window_height = height;
 96:    glMatrixMode (GL_PROJECTION);
 97:    glLoadIdentity();
 98:    gluOrtho2D (-1, +1, -1, +1);
 99:    glMatrixMode (GL_MODELVIEW);
100:    glHint (GL_POINT_SMOOTH_HINT, GL_NICEST);
101:    glHint (GL_LINE_SMOOTH_HINT, GL_NICEST);
102:    glHint (GL_POLYGON_SMOOTH_HINT, GL_NICEST);
103:    radius = 0.9;
104:    glViewport (0, 0, window_width, window_height);
105:    float gray = 0x2Fp0 / 0xFFp0;
106:    glClearColor (gray, gray, gray, 1.0);
107: }
108:
109: int main (int argc, char** argv) {
110:    program_name = basename (argv[0]);
111:    glutInit (&argc, argv);
112:    glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
113:    glutInitWindowSize (window_width, window_height);
114:    glutCreateWindow (program_name.c_str());
115:    glutDisplayFunc (display);
116:    glutReshapeFunc (reshape);
117:    glutTimerFunc (frequency, timer, 100);
118:    glutMainLoop();
119:    return 0;
120: }
```

```
 1: // $Id: hello-gl.cpp,v 1.29 2014-07-22 16:38:07-07 - - $
 2:
 3: // Display text "Hello World" in a window.
 4:
 5: #include <iostream>
 6: #include <string>
 7: using namespace std;
 8:
 9: #include <GL/freeglut.h>
10: #include <libgen.h>
11:
12: // Characteristics of the window.
13: struct window {
14:    string name;
15:    int width {256};
16:    int height {192};
17: } window;
18:
19: // Called by glutMainLoop to display window contents.
20: void display() {
21:    cout << __func__ << "()" << endl;
22:
23:    // Glut strings use unsigned char instead of signed char.
24:    string hello {"Hello, World"};
25:
26:    // Pointer to one of the bitmap fonts.
27:    void* font = GLUT_BITMAP_TIMES_ROMAN_24;
28:
29:    // Width and height in pixels of the bitmap string.
30:    int str_width = glutBitmapLength (font, (GLubyte*) hello.c_str());
31:    int str_height = glutBitmapHeight (font);
32:
33:    // Set the background default color and clear the window.
34:    glClearColor (0.25, 0.25, 0.25, 1.0);
35:    glClear (GL_COLOR_BUFFER_BIT);
36:
37:    // Set the color of the letters in the message.
38:    const GLubyte GREEN[] = {0x00, 0xFF, 0x00};
39:    glColor3ubv (GREEN);
40:
41:    // Position (x,y) of the left end and base of the string.
42:    float xpos = window.width / 2.0 - str_width / 2.0;
43:    float ypos = window.height / 2.0 - str_height / 4.0;
44:    glRasterPos2f (xpos, ypos);
45:
46:    // Draw the bitmap in the window.
47:    //glutBitmapString (font, hello.c_str());
48:    for (auto ch: hello) glutBitmapCharacter (font, ch);
49:
50:    // Swap the passive and active buffers to display the window.
51:    glutSwapBuffers();
52: }
53:
```

```
54:
55: void reshape (int width, int height) {
56:    cout << __func__ << "(" << width << "," << height << ")" << endl;
57:    window.width = width;
58:    window.height = height;
59:    glMatrixMode (GL_PROJECTION);
60:    glLoadIdentity();
61:    glOrtho (0, window.width, 0, window.height, -1, +1);
62:    glMatrixMode (GL_MODELVIEW);
63:    glViewport (0, 0, window.width, window.height);
64:    glutPostRedisplay();
65: }
66:
67: void close() {
68:    cout << __func__ << "()" << endl;
69: }
70:
71: void entry (int state) {
72:    cout << __func__ << "(";
73:    switch (state) {
74:       case GLUT_LEFT: cout << "GLUT_LEFT"; break;
75:       case GLUT_ENTERED: cout << "GLUT_ENTERED"; break;
76:       default: cout << state; break;
77:    }
78:    cout << ")" << endl;
79: }
80:
81: int main (int argc, char** argv) {
82:    window.name = basename (argv[0]);
83:    glutInit (&argc, argv);
84:    glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
85:    glutInitWindowSize (window.width, window.height);
86:    glutCreateWindow (window.name.c_str());
87:    glutDisplayFunc (display);
88:    glutReshapeFunc (reshape);
89:    glutEntryFunc (entry);
90:    glutCloseFunc (close);
91:    glutMainLoop();
92:    return 0;
93: }
94:
```

```
 1: // $Id: keyboard.cpp,v 1.12 2014-07-22 16:38:08-07 - - $
 2:
 3: // Respond to keystrokes.
 4:
 5: #include <iostream>
 6: #include <string>
 7: #include <unordered_map>
 8: using namespace std;
 9:
10: #include <GL/freeglut.h>
11: #include <libgen.h>
12:
13: unordered_map<int,string> special_keys {
14:    {GLUT_KEY_F1       , "GLUT_KEY_F1"       },
15:    {GLUT_KEY_F2       , "GLUT_KEY_F2"       },
16:    {GLUT_KEY_F3       , "GLUT_KEY_F3"       },
17:    {GLUT_KEY_F4       , "GLUT_KEY_F4"       },
18:    {GLUT_KEY_F5       , "GLUT_KEY_F5"       },
19:    {GLUT_KEY_F6       , "GLUT_KEY_F6"       },
20:    {GLUT_KEY_F7       , "GLUT_KEY_F7"       },
21:    {GLUT_KEY_F8       , "GLUT_KEY_F8"       },
22:    {GLUT_KEY_F9       , "GLUT_KEY_F9"       },
23:    {GLUT_KEY_F10      , "GLUT_KEY_F10"      },
24:    {GLUT_KEY_F11      , "GLUT_KEY_F11"      },
25:    {GLUT_KEY_F12      , "GLUT_KEY_F12"      },
26:    {GLUT_KEY_LEFT     , "GLUT_KEY_LEFT"     },
27:    {GLUT_KEY_UP       , "GLUT_KEY_UP"       },
28:    {GLUT_KEY_RIGHT    , "GLUT_KEY_RIGHT"    },
29:    {GLUT_KEY_DOWN     , "GLUT_KEY_DOWN"     },
30:    {GLUT_KEY_PAGE_UP  , "GLUT_KEY_PAGE_UP"  },
31:    {GLUT_KEY_PAGE_DOWN, "GLUT_KEY_PAGE_DOWN"},
32:    {GLUT_KEY_HOME     , "GLUT_KEY_HOME"     },
33:    {GLUT_KEY_END      , "GLUT_KEY_END"      },
34:    {GLUT_KEY_INSERT   , "GLUT_KEY_INSERT "  },
35: };
36:
37: unordered_map<int,string> control_chars {
38:    {  0, "NUL"}, {  1, "SOH"}, {  2, "STX"}, {  3, "ETX"},
39:    {  4, "EOT"}, {  5, "ENQ"}, {  6, "ACK"}, {  7, "BEL"},
40:    {  8, "BS" }, {  9, "HT" }, { 10, "LF" }, { 11, "VT" },
41:    { 12, "FF" }, { 13, "CR" }, { 14, "SO" }, { 15, "SI" },
42:    { 16, "DLE"}, { 17, "DC1"}, { 18, "DC2"}, { 19, "DC3"},
43:    { 20, "DC4"}, { 21, "NAK"}, { 22, "SYN"}, { 23, "ETB"},
44:    { 24, "CAN"}, { 25, "EM" }, { 26, "SUB"}, { 27, "ESC"},
45:    { 28, "FS" }, { 29, "GS" }, { 30, "RS" }, { 31, "US" },
46:    {127, "DEL"},
47: };
48:
```

```
 49:
 50: using uchar = unsigned char;
 51:
 52: // Characteristics of the window.
 53: struct window {
 54:    string name;
 55:    int width {256};
 56:    int height {192};
 57: } window;
 58:
 59: // Called by glutMainLoop to display window contents.
 60: void display() {
 61:    cout << __func__ << "()" << endl;
 62:    glClearColor (0.25, 0.25, 0.25, 1.0);
 63:    glClear (GL_COLOR_BUFFER_BIT);
 64:    glutSwapBuffers();
 65: }
 66:
 67: void print_special_key (int key) {
 68:    const auto& keyname = special_keys.find (key);
 69:    if (keyname == special_keys.end()) cout << "Unknown GLUT_KEY";
 70:                                  else cout << keyname->second;
 71: }
 72:
 73: void print_keyboard_key (int key) {
 74:    if (isgraph (key)) cout << "'" << (uchar)key << "'";
 75:    else {
 76:       const auto& control = control_chars.find (key);
 77:       if (control != control_chars.end()) cout << control->second;
 78:    }
 79: }
 80:
 81: void special (int key, int x, int y) {
 82:    cout << __func__ << "(" << key << "," << x << "," << y << "): ";
 83:    print_special_key (key);
 84:    cout << endl;
 85: }
 86:
 87: void specialup (int key, int x, int y) {
 88:    cout << __func__ << "(" << key << "," << x << "," << y << "): ";
 89:    print_special_key (key);
 90:    cout << endl;
 91: }
 92:
 93: void keyboard (uchar key, int x, int y) {
 94:    cout << __func__ << "(" << (int)key << "," << x << "," << y << "): ";
 95:    print_keyboard_key (key);
 96:    cout << endl;
 97: }
 98:
 99: void keyboardup (uchar key, int x, int y) {
100:    cout << __func__ << "(" << (int)key << "," << x << "," << y << "): ";
101:    print_keyboard_key (key);
102:    cout << endl;
103: }
104:
```

```
105:
106: void reshape (int width, int height) {
107:    cout << __func__ << "(" << width << "," << height << ")" << endl;
108:    window.width = width;
109:    window.height = height;
110:    glMatrixMode (GL_PROJECTION);
111:    glLoadIdentity();
112:    glOrtho (0, window.width, 0, window.height, -1, +1);
113:    glMatrixMode (GL_MODELVIEW);
114:    glViewport (0, 0, window.width, window.height);
115:    glutPostRedisplay();
116: }
117:
118: void close() {
119:    cout << __func__ << "()" << endl;
120: }
121:
122: void entry (int state) {
123:    cout << __func__ << "(";
124:    switch (state) {
125:       case GLUT_LEFT: cout << "GLUT_LEFT"; break;
126:       case GLUT_ENTERED: cout << "GLUT_ENTERED"; break;
127:       default: cout << state; break;
128:    }
129:    cout << ")" << endl;
130: }
131:
132: int main (int argc, char** argv) {
133:    window.name = basename (argv[0]);
134:    glutInit (&argc, argv);
135:    glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
136:    glutInitWindowSize (window.width, window.height);
137:    glutCreateWindow (window.name.c_str());
138:    glutDisplayFunc (display);
139:    glutReshapeFunc (reshape);
140:    glutEntryFunc (entry);
141:    glutCloseFunc (close);
142:    glutKeyboardFunc (keyboard);
143:    glutKeyboardUpFunc (keyboardup);
144:    glutSpecialFunc (special);
145:    glutSpecialUpFunc (specialup);
146:    glutMainLoop();
147:    return 0;
148: }
149:
```

```
 1: // $Id: menus.cpp,v 1.4 2014-05-09 16:54:10-07 - - $
 2:
 3: #include <cmath>
 4: #include <map>
 5: #include <iostream>
 6: #include <string>
 7: using namespace std;
 8:
 9: #include <GL/freeglut.h>
10: #include <libgen.h>
11:
12: int window_width  = 256;
13: int window_height = 192;
14: string program_name;
15:
16: enum shape_type {RECTANGLE, SQUARE, DIAMOND, ELLIPSE, CIRCLE};
17: map<shape_type,string> shape_map{
18:    {RECTANGLE, "Rectangle"},
19:    {SQUARE, "Square"},
20:    {DIAMOND, "Diamond"},
21:    {ELLIPSE, "Ellipse"},
22:    {CIRCLE, "Circle"},
23: };
24:
25: void main_menu (int value) {
26:    cout << "main_menu(" << value << ")" << endl;
27: }
28:
29: void shape_menu (int value) {
30:    shape_type shape = static_cast<shape_type> (value);
31:    cout << __func__ << "(" << value << ") [" << shape_map[shape]
32:         << "]" << endl;
33: }
34:
35: void quit_menu (int value) {
36:     if (value) exit (0);
37: }
38:
```

```
39:
40: void createmenu() {
41:    int shape_menu_id = glutCreateMenu (shape_menu);
42:    cout << __func__ << ": shape_menu_id=" << shape_menu_id << endl;
43:    glutAddMenuEntry ("Rectangle", RECTANGLE);
44:    glutAddMenuEntry ("Square", SQUARE);
45:    glutAddMenuEntry ("Diamond", DIAMOND);
46:    glutAddMenuEntry ("Ellipse", ELLIPSE);
47:    glutAddMenuEntry ("Circle", CIRCLE);
48:    int quit_menu_id = glutCreateMenu (quit_menu);
49:    glutAddMenuEntry ("Confirm", true);
50:    glutAddMenuEntry ("Cancel", false);
51:    int main_menu_id = glutCreateMenu (main_menu);
52:    cout << __func__ << ": main_menu_id=" << main_menu_id << endl;
53:    glutAddSubMenu ("Draw", shape_menu_id);
54:    glutAddSubMenu ("Quit", quit_menu_id);
55:    glutAttachMenu (GLUT_LEFT_BUTTON);
56: }
57:
58: void display() {
59:    glClear (GL_COLOR_BUFFER_BIT);
60:    glutSwapBuffers();
61: }
62:
63: void reshape (int width, int height) {
64:    cout << width << "x" << height << endl;
65:    glMatrixMode (GL_PROJECTION);
66:    glLoadIdentity();
67:    gluOrtho2D (0, window_width, 0, window_height);
68:    glClearColor (0.2, 0.2, 0.2, 1.0);
69:    glMatrixMode (GL_MODELVIEW);
70: }
71:
72: int main (int argc, char** argv) {
73:    program_name = basename (argv[0]);
74:    glutInit (&argc, argv);
75:    glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
76:    glutInitWindowSize (window_width, window_height);
77:    glutCreateWindow (program_name.c_str());
78:    glutDisplayFunc (display);
79:    glutReshapeFunc (reshape);
80:    createmenu();
81:    glutMainLoop();
82:    return 0;
83: }
84:
```

```
 1: // $Id: mousedepth.cpp,v 1.24 2014-05-12 18:51:37-07 - - $
 2:
 3: #include <cmath>
 4: #include <iostream>
 5: using namespace std;
 6:
 7: #include <GL/freeglut.h>
 8: #include <libgen.h>
 9:
10: struct {
11:    string name;
12:    int width {256};
13:    int height {192};
14:    int depth {64};
15:    int left_state {GLUT_UP};
16: } window;
17:
18: struct color {
19:    GLubyte rgb[3];
20: };
21: const color red    {0xFF, 0x00, 0x00};
22: const color yellow {0xFF, 0xFF, 0x00};
23: const color green  {0x00, 0xFF, 0x00};
24:
25: struct object {
26:    float xpos {0};
27:    float ypos {0};
28:    float zpos {0};
29:    color rgb = green;
30:    bool selected {false};
31:    void draw() {
32:       glPushMatrix();
33:       glBegin (GL_POLYGON);
34:       glColor3ubv (rgb.rgb);
35:       float wid = window.width / 10;
36:       float hgt = window.height / 10;
37:       float delta = 2 * M_PI / 64;
38:       for (float theta = 0; theta < 2 * M_PI; theta += delta) {
39:          float x = wid * cos (theta) + xpos;
40:          float y = hgt * sin (theta) + ypos;
41:          glVertex3f (x, y, zpos);
42:       }
43:       glEnd();
44:       glPopMatrix();
45:    }
46: } object;
47:
48: void display() {
49: // cout << __func__ << "()" << endl;
50:    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
51:    object.draw();
52:    glutSwapBuffers();
53: }
54:
55: void reshape (int width, int height) {
56: // cout << __func__ << "(" << width << "," << height << ")" << endl;
57:    window.width = width;
58:    window.height = height;
```

```
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity();
   glOrtho (0, window.width, 0, window.height, 0, window.depth);
   glMatrixMode (GL_MODELVIEW);
   glViewport (0, 0, window.width, window.height);
   glClearColor (0.25, 0.25, 0.25, 1.0);
   object.xpos = window.width / 2;
   object.ypos = window.height / 2;
   object.rgb = green;
   glutPostRedisplay();
}

GLuint depth (int mouse_x, int mouse_y) {
// cout << __func__ << "(" << mouse_x << "," << mouse_y << ")" << endl;
   glEnable (GL_DEPTH_TEST);
   GLfloat scale, bias, depth;
   glGetFloatv (GL_DEPTH_SCALE, &scale);
   glGetFloatv (GL_DEPTH_BIAS, &bias);
   glReadPixels (mouse_x, window.height - mouse_y, 1, 1,
                 GL_DEPTH_COMPONENT, GL_FLOAT, &depth);
   GLuint intdepth = lrintf (depth * window.depth);
// cout << "scale=" << scale
//      << ", bias=" << bias
//      << ", depth=" << depth
//      << ", intdepth=" << intdepth << endl;
   return intdepth;
}

void mouse (int button, int state, int mouse_x, int mouse_y) {
   (void) mouse_x; (void) mouse_y;
// cout << __func__ << "(" << button << "," << state << "," << mouse_x
//      << "," << mouse_y << ")" << endl;
   switch (button) {
      case GLUT_LEFT:
         window.left_state = state;
         break;
   }
   glutPostRedisplay();
}

void motion (int mouse_x, int mouse_y) {
   if (depth (mouse_x, mouse_y) != 0) object.rgb = green;
   else {
      object.rgb = red;
      object.xpos = mouse_x;
      object.ypos = window.height - mouse_y;
   }
   glutPostRedisplay();
}

void passivemotion (int mouse_x, int mouse_y) {
   if (depth (mouse_x, mouse_y) != 0) object.rgb = green;
                                 else object.rgb = yellow;
   glutPostRedisplay();
}

int main (int argc, char** argv) {
   window.name = basename (argv[0]);
```

```
117:    glutInit (&argc, argv);
118:    glutInitDisplayMode (GLUT_RGBA | GLUT_DOUBLE);
119:    glutInitWindowSize (window.width, window.height);
120:    glutCreateWindow (window.name.c_str());
121:    glutDisplayFunc (display);
122:    glutReshapeFunc (reshape);
123:    glutMouseFunc (mouse);
124:    glutMotionFunc (motion);
125:    glutPassiveMotionFunc (passivemotion);
126:    glutMainLoop();
127:    return 0;
128: }
129:
```

```
 1: // $Id: teapot.cpp,v 1.7 2014-05-09 16:55:06-07 - - $
 2:
 3: #include <iostream>
 4: using namespace std;
 5:
 6: #include <GL/freeglut.h>
 7: #include <libgen.h>
 8:
 9: bool wantsolid = true;
10:
11: void display() {
12:
13:     /* clear window */
14:     glClear (GL_COLOR_BUFFER_BIT);
15:
16:     /* draw scene */
17:     if (wantsolid) glutSolidTeapot (.5);
18:               else glutWireTeapot (.5);
19:
20:     /* flush drawing routines to the window */
21:     glFlush();
22:
23: }
24:
25: void mouse (int button, int state, int x, int y) {
26:    if (state == GLUT_DOWN) wantsolid = not wantsolid;
27:    cout << boolalpha;
28:    cout << "button=" << button << ", state=" << state
29:         << ", x=" << x << ", y=" << y
30:         << ", wantsolid=" << wantsolid << endl;
31:    glutPostRedisplay();
32: }
33:
34: int main (int argc, char** argv) {
35:
36:     // Initialize GLUT, using any commandline parameters passed.
37:     glutInit (&argc, argv);
38:
39:     // Setup the size, position, and display mode for new windows.
40:     glutInitWindowSize (500, 500);
41:     glutInitWindowPosition (0, 0);
42:     glutInitDisplayMode (GLUT_RGB);
43:
44:     // Create and set up a window.
45:     glutCreateWindow (basename (argv[0]));
46:     glutDisplayFunc (display);
47:     glutMouseFunc (mouse);
48:
49:     // Tell GLUT to wait for events.
50:     glutMainLoop();
51: }
52:
```

```
 1: // $Id: translate.cpp,v 1.32 2014-05-08 22:00:26-07 - - $
 2:
 3: #include <iomanip>
 4: #include <iostream>
 5: #include <sstream>
 6: #include <string>
 7: using namespace std;
 8:
 9: #include <GL/freeglut.h>
10: #include <libgen.h>
11:
12: struct {
13:    string name;
14:    int width;
15:    int height;
16: } window;
17:
18: struct rgbcolor {
19:    union {
20:       GLubyte ubvec[3];
21:       struct {
22:          GLubyte red;
23:          GLubyte green;
24:          GLubyte blue;
25:       };
26:    };
27: };
28: const rgbcolor Red     {0xFF, 0x00, 0x00};
29: const rgbcolor Green   {0x00, 0xFF, 0x00};
30: const rgbcolor Blue    {0x00, 0x00, 0xFF};
31: const rgbcolor Cyan    {0x00, 0xFF, 0xFF};
32: const rgbcolor Magenta {0xFF, 0x00, 0xFF};
33: const rgbcolor Yellow  {0xFF, 0xFF, 0x00};
34: const rgbcolor White   {0xFF, 0xFF, 0xFF};
35: const rgbcolor Black   {0x00, 0x00, 0x00};
36:
37: string to_string (const rgbcolor& color) {
38:    ostringstream result;
39:    result << "0x"
40:           << hex << setiosflags (ios::uppercase) << setfill ('0')
41:           << setw(2) << (unsigned) color.red
42:           << setw(2) << (unsigned) color.green
43:           << setw(2) << (unsigned) color.blue;
44:    return result.str();
45: }
46:
```

```
47:
48: void draw_rectangle (const rgbcolor& color, const string& name,
49:                      GLfloat xcenter, GLfloat ycenter) {
50:    cout << __func__ << "(" << to_string (color) << ", "
51:         << xcenter << ", " << ycenter << ")" << endl;
52:    GLfloat delta_x = window.width / 8;
53:    GLfloat delta_y = window.height / 4;
54:    glPushMatrix();
55:    glTranslatef (xcenter,  ycenter, 0);
56:    glBegin (GL_POLYGON);
57:    glColor3ubv (color.ubvec);
58:    glVertex2f (-delta_x, -delta_y);
59:    glVertex2f (+delta_x, -delta_y);
60:    glVertex2f (+delta_x, +delta_y);
61:    glVertex2f (-delta_x, +delta_y);
62:    glEnd();
63:    rgbcolor inverse = {(GLubyte) (0xFF - color.red),
64:                        (GLubyte) (0xFF - color.green),
65:                        (GLubyte) (0xFF - color.blue)};
66:    glColor3ubv (inverse.ubvec);
67:    void* font = GLUT_BITMAP_TIMES_ROMAN_24;
68:    float xpos = - glutBitmapLength (font, (GLubyte*) name.c_str()) / 2;
69:    float ypos = - glutBitmapHeight (font) / 2;
70:    glRasterPos2f (xpos, ypos);
71:    glutBitmapString (font, (GLubyte*) name.c_str());
72:    glPopMatrix();
73:    glutSwapBuffers();
74: }
75:
76: void display() {
77:    GLfloat width = window.width;
78:    GLfloat height = window.height;
79:    glClear (GL_COLOR_BUFFER_BIT);
80:    draw_rectangle (Red    , "Red"    ,  width * 0.125, height * 0.75);
81:    draw_rectangle (Green  , "Green"  ,  width * 0.375, height * 0.75);
82:    draw_rectangle (Blue   , "Blue"   ,  width * 0.625, height * 0.75);
83:    draw_rectangle (White  , "White"  ,  width * 0.875, height * 0.75);
84:    draw_rectangle (Cyan   , "Cyan"   ,  width * 0.125, height * 0.25);
85:    draw_rectangle (Magenta, "Magenta",  width * 0.375, height * 0.25);
86:    draw_rectangle (Yellow , "Yellow" ,  width * 0.625, height * 0.25);
87:    draw_rectangle (Black  , "Black"  ,  width * 0.875, height * 0.25);
88: }
89:
```

```
 90:
 91: void reshape (int width, int height) {
 92:    cout << __func__ << "(" << width << ", " << height << ")" << endl;
 93:    window.width = width;
 94:    window.height = height;
 95:    ostringstream title;
 96:    title << window.name << "(" << window.width << ","
 97:          << window.height << ")";
 98:    glutSetWindowTitle (title.str().c_str());
 99:    glutSetIconTitle (title.str().c_str());
100:    glMatrixMode (GL_PROJECTION);
101:    glLoadIdentity();
102:    gluOrtho2D (0, window.width, 0, window.height);
103:    glViewport (0, 0, window.width, window.height);
104:    glClearColor (0.5, 0.5, 0.5, 1.0);
105: }
106:
107: int main (int argc, char** argv) {
108:    window.name = basename (argv[0]);
109:    glutInit (&argc, argv);
110:    glutInitWindowSize (480, 360);
111:    glutCreateWindow (window.name.c_str());
112:    glutDisplayFunc (display);
113:    glutReshapeFunc (reshape);
114:    glutMainLoop();
115:    return 0;
116: }
```

```
 1: // $Id: triangle.cpp,v 1.9 2014-05-08 19:42:13-07 - - $
 2:
 3: #include <GL/freeglut.h>
 4: #include <libgen.h>
 5:
 6: bool flipflop = true;
 7:
 8: void draw_rgb_triangle() {
 9:    glBegin(GL_TRIANGLES);
10:    glColor3ub (0xFF, 0x00, 0x00);
11:    glVertex2f (0, 1);
12:    glColor3ub (0x00, 0xFF, 0x00);
13:    glVertex2f (-1, -1);
14:    glColor3ub (0x00, 0x00, 0xFF);
15:    glVertex2f (1, -1);
16:    glEnd();
17: }
18:
19: void draw_cmy_triangle() {
20:    glBegin(GL_TRIANGLES);
21:    glColor3ub (0x00, 0xFF, 0xFF);
22:    glVertex2f (0, -1);
23:    glColor3ub (0xFF, 0x00, 0xFF);
24:    glVertex2f (-1, 1);
25:    glColor3ub (0xFF, 0xFF, 0x00);
26:    glVertex2f (1, 1);
27:    glEnd();
28: }
29:
30: void display() {
31:    glClearColor (0.2, 0.2, 0.2, 0.0);
32:    glClear (GL_COLOR_BUFFER_BIT);
33:    if (flipflop) draw_rgb_triangle();
34:             else draw_cmy_triangle();
35:    glFlush();
36: }
37:
38: void mouse (int, int state, int, int) {
39:    if (state == GLUT_DOWN) flipflop =  not flipflop;
40:    glutPostRedisplay();
41: }
42:
43: int main (int argc, char** argv) {
44:    glutInit (&argc, argv);
45:    glutInitWindowSize (640, 480);
46:    glutCreateWindow (basename (argv[0]));
47:    glutDisplayFunc (display);
48:    glutMouseFunc (mouse);
49:    glutMainLoop();
50:    return 0;
51: }
```