```
#include <stdlib.h>
#include <stdio.h>
#include <GL/glut.h>

/* Uses EXT_polygon_offset extension if available to better
   render the fold outlines. */

#if GL_EXT_polygon_offset
int polygon_offset;
#endif

enum {
  FLAT,              /* completely flat sheet of paper */
  FLAP1,             /* left flap being folded in */
  FLAP2,             /* right flap being folded int */
  CENTER2,           /* right side folded up at center */
  WING2,             /* right wing folded down */
  CENTER1,           /* left side folded up at center */
  WING1,             /* left wing folded down */
  FOLDED             /* fully folded paper airplane */
} States;

int motion = 1;
int spinning = 1;
int state = FLAT;
int click = 0;
int delay = 0;
int direction;
float flap1_angle = 0;
float flap2_angle = 0;
float center1_angle = 0;
```

```
float center2_angle = 0;
float wing1_angle = 0;
float wing2_angle = 0;

/**

These correspond to the polygons for the paper sections:

 +----------+----------+
 |       /|\       |
 |  2    / | \   3   |
 |     /  |  \      |
 +------/  |   \------+
 |    /|   |   |\    |
 | 1 / |   |   | \ 4 |
 |  /  |   |   |  \  |
 | /   |   |   |   \ |
 | /   | 5 | 6 |   \ |
 |/    |   |   |    \|
 +     |   |   |     +
 |  7  |   |   |  8  |
 |     |   |   |     |
 |     |   |   |     |
 |     |   |   |     |
 |     |   |   |     |
 |     |   |   |     |
 |     |   |   |     |
 +------+---+---+------+

*/
```

```
typedef GLfloat Point[2];

Point poly1[] =
{
  {-1, 0},
  {-1 / 3.0, 2 / 3.0},
  {-1, 2 / 3.0}
};

Point poly2[] =
{
  {-1, 1},
  {-1, 2 / 3.0},
  {-1 / 3.0, 2 / 3.0},
  {0, 1}
};

Point poly3[] =
{
  {0, 1},
  {1, 1},
  {1, 2 / 3.0},
  {1 / 3.0, 2 / 3.0}
};

Point poly4[] =
{
  {1 / 3.0, 2 / 3.0},
  {1, 2 / 3.0},
  {1, 0}
};
```

```
Point poly5[] =
{
  {-1 / 3.0, 2 / 3.0},
  {0, 1},
  {0, -1.5},
  {-1 / 3.0, -1.5}
};

Point poly6[] =
{
  {0, 1},
  {1 / 3.0, 2 / 3.0},
  {1 / 3.0, -1.5},
  {0, -1.5}
};

Point poly7[] =
{
  {-1, 0},
  {-1 / 3.0, 2 / 3.0},
  {-1 / 3.0, -1.5},
  {-1, -1.5}
};

Point poly8[] =
{
  {1, 0},
  {1 / 3.0, 2 / 3.0},
  {1 / 3.0, -1.5},
  {1, -1.5}
```

```
};

void polydlist(int dlist, int num, Point points[])
{
  int i;

  glNewList(dlist, GL_COMPILE);
  glBegin(GL_POLYGON);
  for (i = 0; i < num; i++) {
    glVertex2fv(&points[i][0]);
  }
  glEnd();
  glEndList();
}

void idle(void)
{
  if (spinning)
    click++;
  switch (state) {
  case FLAT:
    delay++;
    if (delay >= 80) {
      delay = 0;
      state = FLAP1;
      glutSetWindowTitle("paper (folding)");
      direction = 1;
    }
    break;
  case FLAP1:
    flap1_angle += 2 * direction;
```

```
  if (flap1_angle >= 180) {
    state = FLAP2;
  } else if (flap1_angle <= 0) {
    state = FLAT;
  }
  break;
case FLAP2:
  flap2_angle += 2 * direction;
  if (flap2_angle >= 180) {
    state = CENTER2;
  } else if (flap2_angle <= 0) {
    state = FLAP1;
  }
  break;
case CENTER2:
  center2_angle += 2 * direction;
  if (center2_angle >= 84) {
    state = WING2;
  } else if (center2_angle <= 0) {
    state = FLAP2;
  }
  break;
case WING2:
  wing2_angle += 2 * direction;
  if (wing2_angle >= 84) {
    state = CENTER1;
  } else if (wing2_angle <= 0) {
    state = CENTER2;
  }
  break;
case CENTER1:
```

```
    center1_angle += 2 * direction;
    if (center1_angle >= 84) {
      state = WING1;
    } else if (center1_angle <= 0) {
      state = WING2;
    }
    break;
  case WING1:
    wing1_angle += 2 * direction;
    if (wing1_angle >= 84) {
      state = FOLDED;
    } else if (wing1_angle <= 0) {
      state = CENTER1;
    }
    break;
  case FOLDED:
    delay++;
    if (delay >= 80) {
      delay = 0;
      glutSetWindowTitle("paper (unfolding)");
      direction = -1;
      state = WING1;
    }
    break;
  }
  glutPostRedisplay();
}

void draw_folded_plane(void)
{
  /* *INDENT-OFF* */
```

```
glPushMatrix();
  glRotatef(click, 0, 0, 1);
  glRotatef(click / 3.0, 0, 1, 0);
  glTranslatef(0, .25, 0);
  glPushMatrix();
    glRotatef(center1_angle, 0, 1, 0);
    glPushMatrix();
      glTranslatef(-.5, .5, 0);
      glRotatef(flap1_angle, 1, 1, 0);
      glTranslatef(.5, -.5, 0);
      glCallList(2);
    glPopMatrix();
    glCallList(5);

    glPushMatrix();
      glTranslatef(-1 / 3.0, 0, 0);
      glRotatef(-wing1_angle, 0, 1, 0);
      glTranslatef(1 / 3.0, 0, 0);

      glCallList(7);
      glPushMatrix();
        glTranslatef(-.5, .5, 0);
        glRotatef(flap1_angle, 1, 1, 0);
        glTranslatef(.5, -.5, 0);
        glCallList(1);
      glPopMatrix();
    glPopMatrix();
  glPopMatrix();

  glPushMatrix();
    glRotatef(-center2_angle, 0, 1, 0);
```

```
      glPushMatrix();
        glTranslatef(.5, .5, 0);
        glRotatef(-flap2_angle, -1, 1, 0);
        glTranslatef(-.5, -.5, 0);
        glCallList(3);
      glPopMatrix();
      glCallList(6);

      glPushMatrix();
        glTranslatef(1 / 3.0, 0, 0);
        glRotatef(wing2_angle, 0, 1, 0);
        glTranslatef(-1 / 3.0, 0, 0);

        glCallList(8);
        glPushMatrix();
          glTranslatef(.5, .5, 0);
          glRotatef(-flap2_angle, -1, 1, 0);
          glTranslatef(-.5, -.5, 0);
          glCallList(4);
        glPopMatrix();
      glPopMatrix();
    glPopMatrix();
  glPopMatrix();
  /* *INDENT-ON* */

}
void draw()
{
            glColor3f(1.0, 1.0, 1.0);
            glRasterPos3f(0, 585, 0.0);
            char c[10]="SCORE:";
```

```
                int i;
        for (i=0; c[i] != '\0'; i++)
        {

                glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, c[i]);
    }
    }

void display(void)
{


  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
  glColor3ub(232, 53, 202);
#if GL_EXT_polygon_offset
  if (polygon_offset) {
    glPolygonOffset(0.5,0.0);
    glEnable(GL_POLYGON_OFFSET_EXT);
  }
#endif
  draw_folded_plane();
  glPolygonMode(GL_FRONT_AND_BACK, GL_LINE);
  glColor3ub(255, 255, 255);
#if GL_EXT_polygon_offset
  if (polygon_offset) {
    glPolygonOffset(0.0, 0.0);
    /* XXX a bug in the unpatched IRIX 5.3 OpenGL posts
       GL_INVALID_ENUM when GL_POLYGON_OFFSET_EXT is disabled;
       please ignore it. */
```

```
    glDisable(GL_POLYGON_OFFSET_EXT);
  } else {
    glPushMatrix();
    glTranslatef(0, 0, .05);
  }
#else
  glPushMatrix();
  glTranslatef(0, 0, .05);
#endif
  draw_folded_plane();
#if GL_EXT_polygon_offset
  if (!polygon_offset) {
    glPopMatrix();
  }
#else
  glPopMatrix();
#endif
  glutSwapBuffers();

}

void
visible(int state)
{
  if (state == GLUT_VISIBLE) {
    if (motion)
      glutIdleFunc(idle);
  } else {
    glutIdleFunc(NULL);
  }
}
```

```
void asd(unsigned char key,int x,int y)
{
if(key=='a'||key=='A')
{
   direction = -direction;
   if (direction > 0) {
     glutSetWindowTitle("paper (folding)");
   } else {
     glutSetWindowTitle("paper (unfolding)");
   }
   }
 if(key=='s'||key=='S')
 {
   motion = 1 - motion;
   if (motion) {
     glutIdleFunc(idle);
   } else {
     glutIdleFunc(NULL);
   }
   }
if(key=='d'||key=='D')
{
   spinning = 1 - spinning;
   }
 if(key=='f'||key=='F')
 {
   exit(0);
 }
}
```

```
int main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_DEPTH | GLUT_DOUBLE);
  glutCreateWindow("paper folding");
  glutDisplayFunc(display);
  glutVisibilityFunc(visible);
  glClearColor(.488, .617, .75, 1.0);
  glMatrixMode(GL_PROJECTION);
  gluPerspective(40.0, 1.0, 0.1, 10.0);
  glMatrixMode(GL_MODELVIEW);
  gluLookAt(0, 0, 5.5,
    0, 0, 0,
    0, 1, 0);
  glEnable(GL_DEPTH_TEST);
  glDepthFunc(GL_LEQUAL);
  glLineWidth(2.0);
  polydlist(1, sizeof(poly1) / sizeof(Point), poly1);
  polydlist(2, sizeof(poly2) / sizeof(Point), poly2);
  polydlist(3, sizeof(poly3) / sizeof(Point), poly3);
  polydlist(4, sizeof(poly4) / sizeof(Point), poly4);
  polydlist(5, sizeof(poly5) / sizeof(Point), poly5);
  polydlist(6, sizeof(poly6) / sizeof(Point), poly6);
  polydlist(7, sizeof(poly7) / sizeof(Point), poly7);
  polydlist(8, sizeof(poly8) / sizeof(Point), poly8);
 glutKeyboardFunc(asd);
```

```
#if GL_EXT_polygon_offset
  polygon_offset = glutExtensionSupported("GL_EXT_polygon_offset");
#endif
  glutMainLoop();
  return 0;             /* ANSI C requires main to return int. */
}
```