Appendix B

```
import processing.serial.*;
import processing.opengl.*;
import toxi.geom.*;
import toxi.processing.*;

ToxiclibsSupport gfx;

Serial port;                         // The serial port
char[] teapotPacket = new char[14];  // InvenSense Teapot packet
int serialCount = 0;                 // current packet byte position
int aligned = 0;
int interval = 0;

float[] q = new float[4];
Quaternion quat = new Quaternion(1, 0, 0, 0);

float[] gravity = new float[3];
float[] euler = new float[3];
float[] ypr = new float[3];

void setup() {
    // 300px square viewport using OpenGL rendering
    size(300, 300, OPENGL);
    gfx = new ToxiclibsSupport(this);

    // setup lights and antialiasing
    lights();
    smooth();

    // display serial port list for debugging/clarity
    println(Serial.list());

    // get the first available port (use EITHER this OR the specific port code below)
    String portName = "/dev/ttyUSB1";

    // get a specific serial port (use EITHER this OR the first-available code above)
    //String portName = "COM4";

    // open the serial port
    port = new Serial(this, portName, 115200);

    // send single character to trigger DMP init/start
    // (expected by MPU6050_DMP6 example Arduino sketch)
    port.write('r');
}

void draw() {
    if (millis() - interval > 1000) {
        // resend single character to trigger DMP init/start
        // in case the MPU is halted/reset while applet is running
        port.write('r');
        interval = millis();
    }

    // black background
    background(0);

    // translate everything to the middle of the viewport
    pushMatrix();
    translate(width / 2, height / 2);

    // 3-step rotation from yaw/pitch/roll angles (gimbal lock!)
    // ...and other weirdness I haven't figured out yet
    //rotateY(-ypr[0]);
    //rotateZ(-ypr[1]);
    //rotateX(-ypr[2]);

    // toxiclibs direct angle/axis rotation from quaternion (NO gimbal lock!)
    // (axis order [1, 3, 2] and inversion [-1, +1, +1] is a consequence of
    // different coordinate system orientation assumptions between Processing
```

```
    // and InvenSense DMP)
    float[] axis = quat.toAxisAngle();
    rotate(axis[0], -axis[1], axis[3], axis[2]);

    // draw main body in red
    fill(255, 0, 0, 200);
    box(10, 10, 200);

    // draw front-facing tip in blue
    fill(0, 0, 255, 200);
    pushMatrix();
    translate(0, 0, -120);
    rotateX(PI/2);
    drawCylinder(0, 20, 20, 8);
    popMatrix();

    // draw wings and tail fin in green
    fill(0, 255, 0, 200);
    beginShape(TRIANGLES);
    vertex(-100,  2, 30); vertex(0,  2, -80); vertex(100,  2, 30);  // wing top layer
    vertex(-100, -2, 30); vertex(0, -2, -80); vertex(100, -2, 30);  // wing bottom layer
    vertex(-2, 0, 98); vertex(-2, -30, 98); vertex(-2, 0, 70);  // tail left layer
    vertex( 2, 0, 98); vertex( 2, -30, 98); vertex( 2, 0, 70);  // tail right layer
    endShape();
    beginShape(QUADS);
    vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(  0, -2, -80); vertex(  0, 2, -80);
    vertex( 100, 2, 30); vertex( 100, -2, 30); vertex(  0, -2, -80); vertex(  0, 2, -80);
    vertex(-100, 2, 30); vertex(-100, -2, 30); vertex(100, -2,  30); vertex(100, 2,  30);
    vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2, -30, 98); vertex(-2, -30, 98);
    vertex(-2,   0, 98); vertex(2,   0, 98); vertex(2,   0, 70); vertex(-2,   0, 70);
    vertex(-2, -30, 98); vertex(2, -30, 98); vertex(2,   0, 70); vertex(-2,   0, 70);
    endShape();

    popMatrix();}

void serialEvent(Serial port) {
    interval = millis();
    while (port.available() > 0) {
        int ch = port.read();
        print((char)ch);
        if (ch == '$') {serialCount = 0;} // this will help with alignment
        if (aligned < 4) {
            // make sure we are properly aligned on a 14-byte packet
            if (serialCount == 0) {
                if (ch == '$') aligned++; else aligned = 0;
            } else if (serialCount == 1) {
                if (ch == 2) aligned++; else aligned = 0;
            } else if (serialCount == 12) {
                if (ch == '\r') aligned++; else aligned = 0;
            } else if (serialCount == 13) {
                if (ch == '\n') aligned++; else aligned = 0;
            }
            //println(ch + " " + aligned + " " + serialCount);
            serialCount++;
            if (serialCount == 14) serialCount = 0;
        } else {
            if (serialCount > 0 || ch == '$') {
                teapotPacket[serialCount++] = (char)ch;
                if (serialCount == 14) {
                    serialCount = 0; // restart packet byte position

                    // get quaternion from data packet
                    q[0] = ((teapotPacket[2] << 8) | teapotPacket[3]) / 16384.0f;
                    q[1] = ((teapotPacket[4] << 8) | teapotPacket[5]) / 16384.0f;
                    q[2] = ((teapotPacket[6] << 8) | teapotPacket[7]) / 16384.0f;
                    q[3] = ((teapotPacket[8] << 8) | teapotPacket[9]) / 16384.0f;
                    for (int i = 0; i < 4; i++) if (q[i] >= 2) q[i] = -4 + q[i];

                    // set our toxilibs quaternion to new data
                    quat.set(q[0], q[1], q[2], q[3]);

                    /*
```

```
            // calculate gravity vector
            gravity[0] = 2 * (q[1]*q[3] - q[0]*q[2]);
            gravity[1] = 2 * (q[0]*q[1] + q[2]*q[3]);
            gravity[2] = q[0]*q[0] - q[1]*q[1] - q[2]*q[2] + q[3]*q[3];

            // calculate Euler angles
            euler[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
            euler[1] = -asin(2*q[1]*q[3] + 2*q[0]*q[2]);
            euler[2] = atan2(2*q[2]*q[3] - 2*q[0]*q[1], 2*q[0]*q[0] + 2*q[3]*q[3] - 1);

            // calculate yaw/pitch/roll angles
            ypr[0] = atan2(2*q[1]*q[2] - 2*q[0]*q[3], 2*q[0]*q[0] + 2*q[1]*q[1] - 1);
            ypr[1] = atan(gravity[0] / sqrt(gravity[1]*gravity[1] + gravity[2]*gravity[2]));
            ypr[2] = atan(gravity[1] / sqrt(gravity[0]*gravity[0] + gravity[2]*gravity[2]));

            // output various components for debugging
            //println("q:\t" + round(q[0]*100.0f)/100.0f + "\t" + round(q[1]*100.0f)/100.0f + "\t" + round(q[2]*100.0f)/100.0f + "\t" +
round(q[3]*100.0f)/100.0f);
            //println("euler:\t" + euler[0]*180.0f/PI + "\t" + euler[1]*180.0f/PI + "\t" + euler[2]*180.0f/PI);
            //println("ypr:\t" + ypr[0]*180.0f/PI + "\t" + ypr[1]*180.0f/PI + "\t" + ypr[2]*180.0f/PI);
            */
          }
        }
      }
    }
}

void drawCylinder(float topRadius, float bottomRadius, float tall, int sides) {
  float angle = 0;
  float angleIncrement = TWO_PI / sides;
  beginShape(QUAD_STRIP);
  for (int i = 0; i < sides + 1; ++i) {
    vertex(topRadius*cos(angle), 0, topRadius*sin(angle));
    vertex(bottomRadius*cos(angle), tall, bottomRadius*sin(angle));
    angle += angleIncrement;
  }
  endShape();

  // If it is not a cone, draw the circular top cap
  if (topRadius != 0) {
    angle = 0;
    beginShape(TRIANGLE_FAN);

    // Center point
    vertex(0, 0, 0);
    for (int i = 0; i < sides + 1; i++) {
      vertex(topRadius * cos(angle), 0, topRadius * sin(angle));
      angle += angleIncrement;
    }
    endShape();
  }

  // If it is not a cone, draw the circular bottom cap
  if (bottomRadius != 0) {
    angle = 0;
    beginShape(TRIANGLE_FAN);

    // Center point
    vertex(0, tall, 0);
    for (int i = 0; i < sides + 1; i++) {
      vertex(bottomRadius * cos(angle), tall, bottomRadius * sin(angle));
      angle += angleIncrement;
    }
    endShape();
  }
}
```