```
package flobee.accelerometer;

/*
 * Copyright (C) 2010 The Android Open Source Project
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *      http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

import android.app.Activity;
import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.BitmapFactory.Options;
import android.graphics.Canvas;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.PowerManager;
import android.os.PowerManager.WakeLock;
import android.util.DisplayMetrics;
import android.view.Display;
import android.view.Surface;
import android.view.View;
import android.view.WindowManager;
/**
 * This is an example of using the accelerometer to integrate the device's
 * acceleration to a position using the Verlet method. This is illustrated with
 * a very simple particle system comprised of a few iron balls freely moving on
 * an inclined wooden table. The inclination of the virtual table is controlled
 * by the device's accelerometer.
 *
 * @see SensorManager
 * @see SensorEvent
 * @see Sensor
 */
public class AccelerometerPlayActivity extends Activity {
  private SimulationView mSimulationView;
  private SensorManager mSensorManager;
  private PowerManager mPowerManager;
  private WindowManager mWindowManager;
  private Display mDisplay;
  private WakeLock mWakeLock;
   /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
     // Get an instance of the SensorManager
    mSensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
     // Get an instance of the PowerManager
    mPowerManager = (PowerManager) getSystemService(POWER_SERVICE);
     // Get an instance of the WindowManager
    mWindowManager = (WindowManager) getSystemService(WINDOW_SERVICE);
```

```
  mDisplay = mWindowManager.getDefaultDisplay();
   // Create a bright wake lock
  mWakeLock = mPowerManager.newWakeLock(
    PowerManager.SCREEN_BRIGHT_WAKE_LOCK, getClass().getName());
   // instantiate our simulation view and set it as the activity's content
  mSimulationView = new SimulationView(this);
  setContentView(mSimulationView);
}
@Override
protected void onResume() {
  super.onResume();
/*
 * when the activity is resumed, we acquire a wake-lock so that the
 * screen stays on, since the user will likely not be fiddling with the
 * screen or buttons.
 */
  mWakeLock.acquire();
   // Start the simulation
  mSimulationView.startSimulation();
}
@Override
protected void onPause() {
  super.onPause();
/*
 * When the activity is paused, we make sure to stop the simulation,
 * release our sensor resources and wake locks
 */
  // Stop the simulation
  mSimulationView.stopSimulation();
   // and release our wake-lock
  mWakeLock.release();
}
class SimulationView extends View implements SensorEventListener {
   // diameter of the balls in meters
  private static final float sBallDiameter = 0.004f;
  private Sensor  mAccelerometer;
   private float   mXDpi;
   private float   mYDpi;
   private float   mMetersToPixelsX;
   private float   mMetersToPixelsY;
  private Bitmap  mBitmap;
  private Bitmap  mWood;
   private float   mXOrigin;
   private float   mYOrigin;
   private float   mSensorX;
   private float   mSensorY;
   private long    mSensorTimeStamp;
   private long    mCpuTimeStamp;
   private float   mHorizontalBound;
   private float   mVerticalBound;
  private ParticleSystem mParticleSystem;

  public void startSimulation() {
  /*
  * It is not necessary to get accelerometer events at a very high
  * rate, by using a slower rate (SENSOR_DELAY_UI), we get an
  * automatic low-pass filter, which "extracts" the gravity component
  * of the acceleration. As an added benefit, we use less power and
  * CPU resources.
  */
    mSensorManager.registerListener(this, mAccelerometer,
      SensorManager.SENSOR_DELAY_UI);
```

```
}
public void stopSimulation() {
  mSensorManager.unregisterListener(this);
}

public SimulationView(Context context) {
  super(context);
   mAccelerometer = mSensorManager
    .getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
  DisplayMetrics metrics = new DisplayMetrics();
  getWindowManager().getDefaultDisplay().getMetrics(metrics);
  mXDpi = metrics.xdpi;
  mYDpi = metrics.ydpi;
  mMetersToPixelsX = mXDpi / 0.0254f;
  mMetersToPixelsY = mYDpi / 0.0254f;
   // rescale the ball so it's about 0.5 cm on screen
  Bitmap ball = BitmapFactory.decodeResource(getResources(),
    R.drawable.ball);
  final int dstWidth = (int) (sBallDiameter * mMetersToPixelsX + 0.5f);
  final int dstHeight = (int) (sBallDiameter * mMetersToPixelsY + 0.5f);
  mBitmap = Bitmap
    .createScaledBitmap(ball, dstWidth, dstHeight, true);
  Options opts = new Options();
  opts.inDither = true;
  opts.inPreferredConfig = Bitmap.Config.RGB_565;
  mWood = BitmapFactory.decodeResource(getResources(),
    R.drawable.wood, opts);
}

 @Override
protected void onSizeChanged(int w, int h, int oldw, int oldh) {
   // compute the origin of the screen relative to the origin of
  // the bitmap
  mXOrigin = (w - mBitmap.getWidth()) * 0.5f;
  mYOrigin = (h - mBitmap.getHeight()) * 0.5f;
  mHorizontalBound = ((w / mMetersToPixelsX - sBallDiameter) * 0.5f);
  mVerticalBound = ((h / mMetersToPixelsY - sBallDiameter) * 0.5f);
  if (null == mParticleSystem) {
    mParticleSystem =
      new ParticleSystem(sBallDiameter, mHorizontalBound, mVerticalBound);
  }
}

 @Override
public void onSensorChanged(SensorEvent event) {
  if (event.sensor.getType() != Sensor.TYPE_ACCELEROMETER)
    return;
/*
* record the accelerometer data, the event's timestamp as well as
* the current time. The latter is needed so we can calculate the
* "present" time during rendering.
*
* In this application, we need to take into account how the
* screen is rotated with respect to the sensors (which always
* return data in a coordinate space aligned to with the screen
* in its native orientation).
*
*/
  switch (mDisplay.getRotation()) {
    case Surface.ROTATION_0:
      mSensorX = event.values[0];
      mSensorY = event.values[1];
      break;
    case Surface.ROTATION_90:
```

```
          mSensorX = -event.values[1];
          mSensorY =  event.values[0];
          break;
        case Surface.ROTATION_180:
          mSensorX = -event.values[0];
          mSensorY = -event.values[1];
          break;
        case Surface.ROTATION_270:
          mSensorX =  event.values[1];
          mSensorY = -event.values[0];
          break;
      }
      mSensorTimeStamp = event.timestamp;
      mCpuTimeStamp = System.nanoTime();
    }

     @Override
    protected void onDraw(Canvas canvas) {
    //draw the background
      canvas.drawBitmap(mWood, 0, 0, null);
   //compute the new position of our object, based on accelerometer
   //data and present time.
      final ParticleSystem particleSystem = mParticleSystem;
       final long now = mSensorTimeStamp
        + (System.nanoTime() - mCpuTimeStamp);
      final float sx = mSensorX;
      final float sy = mSensorY;
      particleSystem.update(sx, sy, now);
      final float xc = mXOrigin;
      final float yc = mYOrigin;
      final float xs = mMetersToPixelsX;
      final float ys = mMetersToPixelsY;
      final Bitmap bitmap = mBitmap;
      final int count = particleSystem.getParticleCount();
      for (int i = 0; i < count; i++) {
     /*
      * We transform the canvas so that the coordinate system matches
      * the sensors coordinate system with the origin in the center
      * of the screen and the unit is the meter.
      */
         final float x = xc + particleSystem.getPosX(i) * xs;
         final float y = yc - particleSystem.getPosY(i) * ys;
         canvas.drawBitmap(bitmap, x, y, null);
      }
       // and make sure to redraw asap
      invalidate();
    }
     @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {}
  }
}
```