CS378 - Mobile Computing

Sensing and Sensors

Part 2
Using Sensors

• Recall basics for using a Sensor:
  – Obtain the `SensorManager` object
  – create a `SensorEventListener` for `SensorEvents`
  – logic that responds to sensor event
  – Register the sensor listener with a `Sensor` via the `SensorManager`
Sensor Best Practices

• Unregister sensor listeners
  – when done with Sensor or activity using sensor paused (onPause method)
  – sensorManager.
    unregisterListener(sensorListener)
  – otherwise data still sent and battery resources continue to be used
Sensors Best Practices

• verify sensor available before using it
• use getSensorList method and type
• ensure list is not empty before trying to register a listener with a sensor
Sensors Best Practices

• Avoid deprecated sensors and methods
• TYPE_ORIENTATION and TYPE_TEMPERATURE are deprecated as of Ice Cream Sandwich
Sensors Best Practices

• Don't block the onSensorChanged() method
  – recall the resolution on sensors
  – 50 updates a second for onSensorChange method not uncommon
  – when registering listener update is only a hint and may be ignored
  – if necessary save event and do work in another thread or asynch task
Sensor Best Practices

• Testing on the emulator
• Android SDK doesn't provide any simulated sensors
• 3rd party sensor emulator
  • http://code.google.com/p/openintents/wiki/SensorSimulator
SensorSimulator

• Download the Sensor Simulator tool
• Start Sensor Simulator program
• Install SensorSimulator apk on the emulator
• Start app, connect simulator to emulator, start app that requires sensor data
• Must modify app so it uses Sensor Simulator library
Sensor Simulator

- Mouse in Sensor Simulator controls phone, feeds sensor data to emulator
- Can also record sensor data from device and play back on emulator
Sensor Sample - Moving Ball

- Place ball in middle of screen
- Ball has position, velocity, and acceleration
- Acceleration based on linear acceleration sensor
- Update over time, based on equations of motion, but fudged to suit application
Sensing Orientation

• Orientation of the device
• x - tangential to ground and points roughly east
• y - tangential to the ground and points towards magnetic north
• z perpendicular to the ground and points towards the sky

Orientation Sensor

• Deprecated
• Instead register listener(s) with `ACCELEROMETER` and `MAGNETIC_FIELD` sensors
• in the `onSensorChanged` method
  – pull values from `SensorEvent` for accelerometer event or magnetic field event
  – accelerometer 3 values, acceleration in x, y, z directions
  – magnetic field, ambient magnetic field in x, y, and z axis
Orientation Sensor

• with values from accelerometer and magnetic field obtain a rotation matrix
• method in SensorManager
• transforms from device coordinate system to world coordinate system
• finally, call getOrientation with rotation matrix and array to store result
Result of `getOrientation`

- method writes to array of floats
- 3 values
- [0] -> azimuth, rotation around the z axis
- [1] -> pitch, rotation around the x axis
- [2] -> roll, rotation around the y axis
- angles in radians, positive in counter clockwise direction

notice positive x and z flipped
Demo OrientationTestActivity

Azimuth (Rotation around Z Axis) 
0.648

Pitch (Rotation around X Axis) 
0.023

Roll (Rotation around Y Axis) 
-0.008
Sensor Sample - Moving Ball

- Gross Simplification
- velocity set equal to acceleration

```java
public void onSensorChanged(SensorEvent event)
{
    // set ball speed based on phone tilt
    // speed set equal to acceleration
    mBallVelocity.x = -event.values[0];
    mBallVelocity.y = event.values[1];
}
```
Sensor Sample - Moving Ball

• Alternate Implementation

```java
// try more realistic movement
float xA = -event.values[0];
float yA = event.values[1];
float aveXA = (xA + mPrevXAcc) / 2;
float aveYA = (yA + mPrevYAcc) / 2;
long currentTime = System.currentTimeMillis();
long elapsedTime = currentTime - mPrevTime;
mBallVelocity.x += aveXA * elapsedTime / 1000 / ACC_FUDGE_FACTOR;
mBallVelocity.y += aveYA * elapsedTime / 1000 / ACC_FUDGE_FACTOR;

mPrevXAcc = xA;
mPrevYAcc = yA;
mPrevTime = currentTime;
```

• position updated in separate thread which redraws the view
Sensor Sample

- Draw lines for x and y velocities

```java
//called by invalidate()
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);
    mPaint.setStrokeWidth(1);
    mPaint.setColor(0xFF00FF00);
    canvas.drawCircle(mX, mY, mR, mPaint);

    mPaint.setStrokeWidth(3);
    mPaint.setColor(0xFFFFFFFF);
    canvas.drawLine(mX, mY,
                    mX + vX * 15, mY, mPaint);
    mPaint.setColor(0xFF0000FF);
    canvas.drawLine(mX, mY,
                    mX, mY + vY * 15, mPaint);
```
Sensor Sample - TBBT

• Inspired by http://tinyurl.com/bxzaspy and http://tinyurl.com/6nhvnnv
TBBT Sound Effect App

BigBangApp

Shake the Device

BigBangApp

CRACK !!!!
Responding to Events

private class LinAccListener implements SensorEventListener {
    public void onSensorChanged(SensorEvent event) {
        if (event.sensor.getType() == Sensor.TYPE_LINEAR_ACCELERATION) {
            float x = event.values[0];
            float y = event.values[1];
            float z = event.values[2];
            float acc = (float)Math.sqrt(x * x + y * y);
            // Log.d("BBT", "+ acc);
            if (acc > 31) {
                Log.d("BBT", "+ acc);
                if (soundPlayer != null && !soundPlayer.isPlaying()) {
                    soundPlayer.start();
                    picture.setImageResource(R.drawable.crack);
                }
            }
        }
    }
}
Changing Images

• Use of an Image View
• Initial Image set in onCreate
• new image set in onSensorChange
• register listener with MediaPlayer
• on completion reset image

```java
@Override
public void onCompletion(MediaPlayer mp) {
    picture.setImageResource(R.drawable.shake);
}
```