Engineering Design w/Embedded Systems

Lecture 36—(Android Friday): Multitouch; Course Summary

Patrick Lam
University of Waterloo

April 8, 2013

Part I

Multitouch

Why multitouch?

It's fun!

Plus, it demonstrates:

- graphics;
- more event-handling; and
- using finite-state machines.

The Plan

Implement an Image Viewer app that recognizes drag and pinch-to-zoom.

Credit: Ed Burnette, “How to use Multi-touch in Android 2”.

`http://www.zdnet.com/blog/burnette/how-to-use-multi-touch-in-android-2/1747`

Excerpted from *Hello, Android 3rd edition*, Pragmatic Bookshelf.

Starting out: an image viewer app

Like in Lecture 22.

- Create an app. This time, use theme: `@android:style/Theme.NoTitleBar.Fullscreen`.
- Add an image in the `drawable-nodpi` folder.
- Add an `ImageView` and point it to your image.
- Put `scaleType=”matrix”` in the ImageView definition.

Handling Touch Events

In your `onCreate()` method, create an `OnTouchListener`.

Call `setOnTouchListener` on the `ImageView`.

The `onTouch()` method can be empty for now.

Inspecting Touch Events

Now, let’s look at the `onTouch` events.

```
// create and add an OnTouchListener with the following code:
@Override public boolean onTouch(View v, MotionEvent event) {
  dumpEvent(event);
  return true;
}

private void dumpEvent(MotionEvent event) {
  String names[] = { "DOWN", "UP", "MOVE", "CANCEL", "OUTSIDE",
                     "POINTER_DOWN", "POINTER_UP",
                     "7?", "8?", "9?" };
  int actionCode = event.getActionMasked();
  String pid = "";
  // ...
  Log.d("T", dump);
}
```

Adding Fields

We will need the following fields in the `MainActivity`:

```
ImageView imgView;
Matrix matrix = new Matrix();
PointF start = new PointF();
int dpi;

// could use an enum
static final int NONE = 0;
static final int DRAG = 1;
static final int ZOOM = 2;
int mode = NONE;
```

onCreate; onTouch skeleton

Implementation for `onCreate`:

```
imgView = (ImageView) findViewById
                  (R.id.imageView1);
imgView.setOnTouchListener(this);
```

Skeleton code in `onTouch`:

```
switch(evt.getActionMasked()) {
  ...
}
imgView.setImageMatrix(matrix);
return true;
```

Implementing Drag I

OK, now we'd better add some code to `onTouch`:

```
switch(evt.getActionMasked()) {
case MotionEvent.ACTION_DOWN:
  savedMatrix.set(matrix);
  start.set(evt.getX(), evt.getY());
  dpi = evt.getPointerId(0);
  mode = DRAG;
  break;
case MotionEvent.ACTION_UP:
case MotionEvent.ACTION_POINTER_UP:
  mode = NONE;
  break;
```

This starts the drag event but doesn't do any dragging yet.

Implementing Drag II

We also need to handle the move event:

```
case MotionEvent.ACTION_MOVE:
  if (mode == DRAG) {
    matrix.set(savedMatrix);
    matrix.postTranslate(evt.getX(dpi) - start.x,
                         evt.getY(dpi) - start.y,
  }
  break;
```

Now you can drag the picture around.

Implementing pinch/zoom: helper functions

So far we only have single-touch. Let’s do multi-touch.

First, two helper functions:

```java
private float spacing(MotionEvent event) {
  float x = event.getX(0) - event.getX(1);
  float y = event.getY(0) - event.getY(1);
  return (float) Math.sqrt(x*x+y*y);
}

private void midPoint(PointF point, MotionEvent event) {
  float x = event.getX(0) + event.getX(1);
  float y = event.getY(0) + event.getY(1);
  point.set(x/2, y/2);
}
```

(These should use `pointerIndex`es instead of hard-coding the pointers.)

Implementing pinch/zoom: handling events I

First, handle the ACTION_POINTER_DOWN, initiating a pinch:

```
case MotionEvent.ACTION_POINTER_DOWN:
  oldDist = spacing(evt);
  if (oldDist > 10f) {
    savedMatrix.set(matrix);
    midPoint(mid, evt);
    mode = ZOOM;
  }
  break;
```

(We also add savedMatrix and oldDist fields.)

Implementing pinch/zoom: handling events II

And, we need to handle the `ACTION_MOVE`:

```
case MotionEvent.ACTION_MOVE:
if (mode == DRAG) {
  // as before
} else if (mode == ZOOM) {
  float newDist = spacing(evt);
  if (newDist > 10f) {
    matrix.set(savedMatrix);
    float scale = newDist / oldDist;
    matrix.postScale(scale, scale,
                     mid.x, mid.y);
  }
}
```

Now we can zoom!

Part II

Course Summary

Four Parts

- Embedded Systems
- Android
- Practical Software Engineering Tips
- Software Engineering Concepts

About Embedded Systems

Lectures: L01, L02, L08, L17

- Definitions.
- Why embedded systems are hard.
- Sensors and actuators; ADC/DAC.
- Polling versus interrupts.
- Timers (watchdog, interval).

About Android

Lectures: L04, L05, L06, L10, L13, L16, L17, L18, L22, L25, L36.

- Event-Driven Programming
  (inner classes, callbacks, asynchronous vs synchronous)
  and inversion of control.
- XML (manifests, layouts).
- Runnables and Timers.
- Activities.
- Reading from Sensors.
- Intents (specifying a query).
- Toasts, broadcast receivers, lists.
- JUnit for Android; Robotium.
- Graphics. Inflating XML.
- Persistent storage, files.
- Multitouch.

Tools: Practical Software Engineering

Lectures: L03, L04, L07, L08, L11, L12, L16, L18, L26, L27, L30.

- Integrated Development Environments.
- Inheritance: interfaces and classes.
- Implementing Finite State Machines.
- Debugging tactics; asserts, logging, breakpoints.
- Version control.
- Software bricolage.
- Refactoring.
- Tests (JUnit, Android JUnit, Robotium).
- Reviews.

Software Engineering Concepts

Lectures: L04, L05, L09, L14, L15, L16, L18, L21, L23, L24, L28, L29, L30, L31, L32, L33, L34, L35.

- Event-driven programming.
- Assertions, tests.
- Simulation.
- Engineering design and analysis.
- Computational Decision Making.
- UML.
- Software lifecycle models (particularly XP).
  - Requirements.
  - Planning and estimation.
  - Verification (including testing, code review) & validation.
  - Software maintenance.