Touch and Multitouch

Technologies
Using touch
Multitouch operations
Gestures
Touch Technology
Capacitors

A capacitor consists of two conductive plates with insulation between them.

The insulation can be air, plastic, polymers, etc.
Capacitors

If you connect a power source, an electrical field forms in the space between the plates.

The electrical field stores energy.

The capacitance, the amount of energy stored for a given voltage, is dependent on the spacing between the plates and characteristics of the insulator.

You can put energy into a capacitor and take energy out.
RC Circuits

A resistor resists the flow of electricity

Electricity can only flow at a given rate
It will take time for the capacitor to reach its full charge
RC Circuits

\[ V_C(t) = V(1 - e^{-t/RC}) \]

If the capacitance increases, it will take longer
This is easy to measure
These times can be in the nanoseconds!

It will take a certain amount of time to reach 80% of the maximum voltage
One way to increase capacitance

Another charge holding object can have capacitance between the capacitor plate and the object. This is called *mutual capacitance*. 

![Graph showing voltage vs. distance]
How to make a touch screen out of this

Conductive Glass

Insulating Polymer

Sandwiched over the LCD

bottom grid polymer
top grid
How to make a touch screen out of this

Scan over the X,Y rows, charging and discharging the capacitor at each intersection.

Energy goes in and energy comes out, so very little loss.
Pressure?

Size of the pattern denotes the finger pressure.
Pressure

Pressure is determined by how many intersections are activated

Low Pressure  High Pressure
Android Touch

@Override
public boolean onTouchEvent(MotionEvent event) {
    return super.onTouchEvent(event);
}

Function is called whenever any touch activity occurs: down, up, move, etc.

This can be done for either or both the Activity and the View. I’m doing it in the View, since I am drawing the results.
## MotionEvent actions

MotionEvent.getActionMasked()

<table>
<thead>
<tr>
<th>Action</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>MotionEvent.ACTION_DOWN</td>
<td>First touch</td>
</tr>
<tr>
<td>MotionEvent.ACTION_UP</td>
<td>Last touch released</td>
</tr>
<tr>
<td>MotionEvent.ACTION_POINTER_DOWN</td>
<td>Touches after the first</td>
</tr>
<tr>
<td>MotionEvent.ACTION_POINTER_UP</td>
<td>Releases other than the last one</td>
</tr>
<tr>
<td>MotionEvent.ACTION_MOVE</td>
<td>Motion</td>
</tr>
<tr>
<td>MotionEvent.ACTION_OUTSIDE</td>
<td>Activity outside view bounds</td>
</tr>
<tr>
<td>MotionEvent.ACTION_CANCEL</td>
<td>Touch activity canceled</td>
</tr>
</tbody>
</table>
What happens for touches and releases

*If only one touch:*
ACTION_DOWN, then ACTION_UP

*If multitouch:*
ACTION_DOWN for first touch
ACTION_POINTER_DOWN for subsequent touches
ACTION_POINTER_UP for releases other than the last one
ACTION_UP for the last release
### Example

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Finger 1</td>
<td>ACTION_DOWN</td>
</tr>
<tr>
<td>Touch Finger 2</td>
<td>ACTION_POINTER_DOWN</td>
</tr>
<tr>
<td>Release Finger 2</td>
<td>ACTION_POINTER_UP</td>
</tr>
<tr>
<td>Release Finger 1</td>
<td>ACTION_UP</td>
</tr>
</tbody>
</table>

But...

<table>
<thead>
<tr>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Finger 1</td>
<td>ACTION_DOWN</td>
</tr>
<tr>
<td>Touch Finger 2</td>
<td>ACTION_POINTER_DOWN</td>
</tr>
<tr>
<td>Release Finger 1</td>
<td>ACTION_POINTER_UP</td>
</tr>
<tr>
<td>Release Finger 2</td>
<td>ACTION_UP</td>
</tr>
</tbody>
</table>

Single touch is easy – just look at ACTION_UP and ACTION_DOWN and pointer index 0
Indices and Pointer ID’s

Indices and Pointer ID’s are not necessarily the same!

An action is for a given pointer
Each pointer is assigned an ID
(integers starting at zero)
These are the numbers that the
Course Application shows

Messages are assigned an index
Also small integers

MotionEvent.getActionIndex()
tells us this number
**Example**

<table>
<thead>
<tr>
<th>Action</th>
<th>Index</th>
<th>Pointer ID</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Touch Finger 1</td>
<td>0</td>
<td>0</td>
<td>ACTION_DOWN</td>
</tr>
<tr>
<td>Touch Finger 2</td>
<td>1</td>
<td>1</td>
<td>ACTION_POINTER_DOWN</td>
</tr>
<tr>
<td>Release Finger 1</td>
<td>0</td>
<td>0</td>
<td>ACTION_POINTER_UP</td>
</tr>
<tr>
<td>Release Finger 2</td>
<td>0</td>
<td>1</td>
<td>ACTION_UP</td>
</tr>
</tbody>
</table>

Messages are for an index

From that you can get a pointer ID using `MotionEvent.getPointerId(index)`

The indices are kept small and sequential
Pointer ID’s stay fixed
Detecting first and second fingers
**Member variables – Housekeeping**

```java
/**
 * Text size in dp
 */
private final static int TEXT_SIZE = 250;

/**
 * Paint to use for text
 */
private Paint textPaint;

/**
 * Text size in pixels
 */
private float textSize;

/**
 * Text offsets so they center
 */
private float offset1;
private float offset2;
```
private void init(Context context) {
    textPaint = new Paint(Paint.ANTI_ALIAS_FLAG);

    // Convert dp to pixels
    DisplayMetrics metrics = getResources().getDisplayMetrics();
    textSize = TEXT_SIZE * (metrics.densityDpi/160f);
    textPaint.setTextSize(textSize);

    // Offset so we can center the text
    offset1 = -textPaint.measureText("1") / 2;
    offset2 = -textPaint.measureText("2") / 2;
}
Keeping track of current touch

/**
 * The two touch pointer id's or -1 if none
 */
private int touch1 = -1;
private int touch2 = -1;

/**
 * x,y for both of the current touches
 */
private float touch1x;
private float touch1y;
private float touch2x;
private float touch2y;
onDraw()

@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    if(touch1 >= 0) {
        canvas.drawText("1", touch1x+offset1, touch1y+textSize/2, textPaint);
    }

    if(touch2 >= 0) {
        canvas.drawText("2", touch2x+offset2, touch2y+textSize/2, textPaint);
    }
}
onTouchEvent

@Override
public boolean onTouchEvent(MotionEvent event) {
    int id = event.getPointerId(event.getActionIndex());

    switch(event.getActionMasked()) {
        case MotionEvent.ACTION_DOWN:
            touch1 = id;
            touch2 = -1;
            getPositions(event);
            return true;
        default:
            return super.onTouchEvent(event);
    }
}

If your function does not handle the touch event, do:
return super.onTouchEvent(event);
getPositions()

```
private void getPositions(MotionEvent event) {
    for(int i=0; i<event.getPointerCount(); i++) {
        int id = event.getPointerId(i);

        if(id == touch1) {
            touch1x = event.getX(i);
            touch1y = event.getY(i);
        } else if(id == touch2) {
            touch2x = event.getX(i);
            touch2y = event.getY(i);
        }
    }

    invalidate();
}
```

We call this when we move, but also need to call it when a touch initially occurs.
MotionEvent.ACTION_POINTER_DOWN

case MotionEvent.ACTION_POINTER_DOWN:
    if(touch1 >= 0 && touch2 < 0) {
        touch2 = id;
        getPositions(event);
        return true;
    }
    break;

    This case occurs for touches 3, 4, ... as well
case MotionEvent.ACTION_POINTER_UP:
    if(id == touch2) {
        touch2 = -1;
    } else if(id == touch1) {
        touch1 = touch2;
        touch2 = -1;
    }
    invalidate();
    return true;

If we release the #1 finger, the second one becomes the new #1
case MotionEvent.ACTION_UP:

case MotionEvent.ACTION_UP:
  case MotionEvent.ACTION_CANCEL:
    touch1 = -1;
    touch2 = -1;
    invalidate();
    return true;

Treat ACTION_CANCEL and ACTION_UP both as ends of all touch activity
Last but not least

```java
case MotionEvent.ACTION_MOVE:
    getPositions(event);
    return true;
```

ACTION_MOVE is somewhat different in that it is only called once for all touches that are active

```java
private void getPositions(MotionEvent event) {
    for(int i=0;  i<event.getPointerCount();  i++) {
        int id = event.getPointerId(i);

        if(id == touch1) {
            touch1x = event.getX(i);
            touch1y = event.getY(i);
        } else if(id == touch2) {
            touch2x = event.getX(i);
            touch2y = event.getY(i);
        }
    }

    invalidate();
}
```
Fancier Gestures with GestureDetector

GestureDetector more commonly installed in the Activity

Detects common gestures like flings and taps
## GestureDetector interfaces

<table>
<thead>
<tr>
<th>GestureDetector.OnGestureListener</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>onDown</td>
<td>First touch</td>
</tr>
<tr>
<td>onFling</td>
<td>Rapid swipe of the finger</td>
</tr>
<tr>
<td>onShowPress</td>
<td>After slight delay – time to ensure not some other gesture</td>
</tr>
<tr>
<td>onSingleTapUp</td>
<td>When an up is thought to be a single tap</td>
</tr>
<tr>
<td>onScroll</td>
<td>X,Y dragging movement</td>
</tr>
<tr>
<td>onLongPress</td>
<td>After longer delay</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GestureDetector.OnDoubleTapListener</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>onDoubleTap</td>
<td>Double tap</td>
</tr>
<tr>
<td>onDoubleTapEvent</td>
<td>Events during double tap</td>
</tr>
<tr>
<td>onSingleTapConfirmed</td>
<td>We know it is not a double tap</td>
</tr>
</tbody>
</table>
public class GestureActivity extends Activity implements GestureDetector.OnGestureListener, GestureDetector.OnDoubleTapListener {

private GestureDetectorCompat gestureDetector;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.gesture_activity);

    // Instantiate the gesture detector for us and we are the listener
    gestureDetector = new GestureDetectorCompat(this, this);

    // Set the gesture detector as the double tap listener.
    gestureDetector.setOnDoubleTapListener(this);
}

// Instantiate the gesture detector for us and we are the listener
gestureDetector = new GestureDetectorCompat(this, this);

// Set the gesture detector as the double tap listener.
gestureDetector.setOnDoubleTapListener(this);
@Override
public boolean onTouchEvent(MotionEvent event) {

    gestureDetector.onTouchEvent(event);

    // Be sure to call the superclass implementation
    return super.onTouchEvent(event);
}
Then implement the interface functions

```java
@Override
public boolean onDown(MotionEvent event) {
    add("onDown");
    return true;
}

@Override
public boolean onFling(MotionEvent event1,
                        MotionEvent event2, float velocityX, float velocityY) {
    add("onFling " + velocityX + ", " + velocityY);
    return true;
}

@Override
public void onLongPress(MotionEvent event) {
    add("onLongPress");
}

@Override
public boolean onScroll(MotionEvent e1, MotionEvent e2,
                        float distanceX, float distanceY) {
    return true;
}

@Override
public void onShowPress(MotionEvent event) {
    add("onShowPress");
}

@Override
public boolean onSingleTapUp(MotionEvent event) {
    add("onSingleTapUp");
    return true;
}

@Override
public boolean onDoubleTap(MotionEvent event) {
    add("onDoubleTap");
    return true;
}

@Override
public boolean onDoubleTapEvent(MotionEvent event) {
    add("onDoubleTapEvent");
    return true;
}

@Override
public boolean onSingleTapConfirmed(MotionEvent event) {
    add("onSingleTapConfirmed");
    return true;
}
```
Dragging Stuff Around

These are your choices

Place directly
Pointers tell you were something goes

Move differentially
Pointers tell you how much to move something

Move relative to a starting point
Pointers tell you how much something moved since touch began
Place Directly

**Puzzle could work this way**

**Touch location directly sets**

This is the easiest, but the least flexible. Pieces snap center to the touch point

```java
case MotionEvent.ACTION_MOVE:
    // If we are dragging, just set the
    // new location and force a redraw
    if (dragging != null) {
        dragging.setLoc(relX, relY);
        view.invalidate();
        return true;
    }
    break;
```
Differential

You move instead of “set”

Much easier to implement fancier multitouch features
Can seem more realistic

public void move(float dx, float dy) {
    x += dx;
    y += dy;
}

Step 3 Puzzle
Somewhat more flexible...

case MotionEvent.ACTION_MOVE:
    // If we are dragging, move to
    // the new location and force a redraw
    if (dragging != null) {
        dragging.move(relX - lastRelX, relY - lastRelY);
        lastRelX = relX;
        lastRelY = relY;
        view.invalidate();
        return true;
    }
    break;

Subject to cumulative rounding errors
Move relative to a starting point

Keep track of where something was when you started

Apply the moves to the starting point each time

Most precise solution, but more difficult to implement

dragging.setLoc(startX + dx, startY + dy);
We’ll concentrate here on the differential method

Gnome in Touch activity works this way

Every time we move we know where we were before and where we are now

public void touchMove(float x1, float y1, float x2, float y2) {
    touchLast.copyFrom(touchNow);
    touchNow.x1 = x1;
    touchNow.y1 = y1;
    touchNow.x2 = x2;
    touchNow.y2 = y2;
}
What are we moving?

Common example...
Bitmap (maybe transparent background)
We have a position for the top left corner and a rotation angle

You draw this this way

```javascript
canvas.save();
canvas.translate(x, y);
canvas.rotate(angle);
canvas.drawBitmap(bmp, 0, 0, null);
canvas.restore();
```
Grabbing

Grabbing is single-touch

Figure out if the touch is on the thing you are grabbing

If so, set a reference to it (grabbed)

grabbed = gnome;
Translation is easy...

```
float dx = touchNow.x1 - touchLast.x1;
float dy = touchNow.y1 - touchLast.y1;
```

**How much did we move?**

```
grabbed.move(dx, dy);
```

**Move by that much**

Just adds dx, dy to the x,y location of the object.
Multitouch rotation

As second finger moves around the first finger, the object rotates.

Can rotate the scene around touch 1 or around a point halfway between touches.

If you are also doing scaling, it does not matter.
Rotation is a bit harder than translation

We need to know the rotation angle before and after

We add the rotation, but we have to move the position...
Figuring the rotation angle

\[ \theta \]
Figuring the rotation angle

\[
\theta = \arctan\left( \frac{y_2 - y_1}{x_2 - x_1} \right)
\]

won’t work, BTW
Figuring the rotation angle

atan((y2-y1) / (x2-x1)) won’t work, BTW
Returned angle is in the range: -pi/2 through pi/2

\[
\begin{align*}
\theta &= \arctan \left( \frac{y2 - y1}{x2 - x1} \right) \\
\text{public float angle()} &\{ \\
\quad \text{float dx = x2 - x1;} \\
\quad \text{float dy = y2 - y1;} \\
\quad \text{return (float) Math.toDegrees(Math.atan2(dy, dx));}
\}
\end{align*}
\]
What is the change in the angle?

```java
float angle1 = touchLast.angle();
float angle2 = touchNow.angle();

// Add to the image rotation angle
float dAngle = angle2 - angle1;
grabbed.rotate(dAngle, x1, y1);
```

```java
public void rotate(float dAngle, float x1, float y1) {
    // Change the rotation
    angle += dAngle;
}
```
Problem: Rotating the corner

For the object to appear to rotate around the touch point, the corner point has to be rotated around the touch point by the same angle.
Rotating around a point

Move the point we are rotating to the origin:
\[ x' = x - x_c \]
\[ y' = y - y_c \]

Rotate point around the origin:
\[ x'' = x'\cos(\theta) - y'\sin(\theta) \]
\[ y'' = x'\sin(\theta) + y'\cos(\theta) \]

Translate the origin back to the point we are rotating around:
\[ x_r = x'' + x_c \]
\[ y_r = y'' + y_c \]
Rotation function

```java
public void rotate(float dAngle, float x1, float y1) {
    // Change the rotation
    angle += dAngle;

    // Compute the radians angle
    double rAngle = Math.toRadians(dAngle);
    float ca = (float) Math.cos(rAngle);
    float sa = (float) Math.sin(rAngle);
    float xp = (x - x1) * ca - (y - y1) * sa + x1;
    float yp = (x - x1) * sa + (y - y1) * ca + y1;

    x = xp;
    y = yp;
}
```
Scaling

Scaling has the same problem
Scaling is around the origin

I’m pretty sure you can figure this one out...
Scaling around a point

Scaling around point $(x_c, y_c)$ by scale factor $s$
Scale function