

# G54MDP

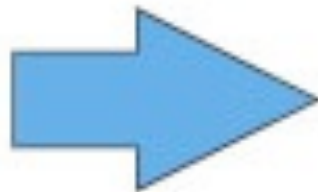
# Mobile Device Programming

Lecture 16 – Touch, Gestures

# Interfaces

- Android UI interaction metaphor
  - So far has been seen to be similar to PC
  - Different syntax, but similar concepts
    - Widgets, buttons, scrolling
    - onClick events
- Significant difference between mobiles and PC
  - What?

1	5	10	5
5	15	25	10
2	12	15	5
0	2	5	1

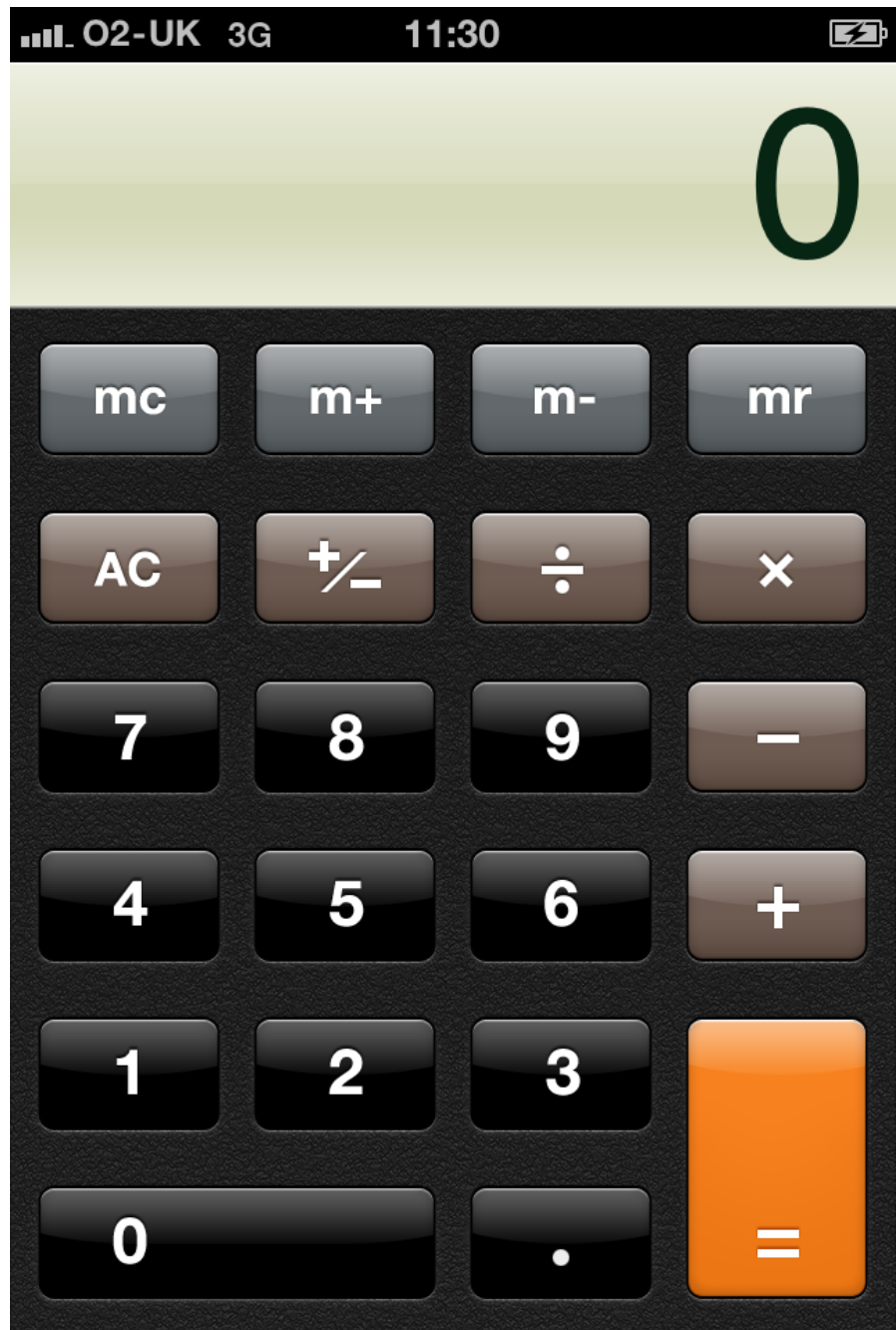


**Weighted Finger Position**

1	5	10	5
5	15	25	10
2	12	15	5
0	2	5	1

# Touch and the UI

- Touch relies on finger contact with the display
  - This has to alter the way we design our displays
- Size of the finger sets the properties of the UI, not the size of a display
  - 5” vs 1200 pixels
  - The size of ‘buttons’ must be big enough that the user can touch them
  - Ditto the spacing between them
  - If they get too small, or too close together then it will be hard for the user to accurately use them
- Size is fixed relative to display

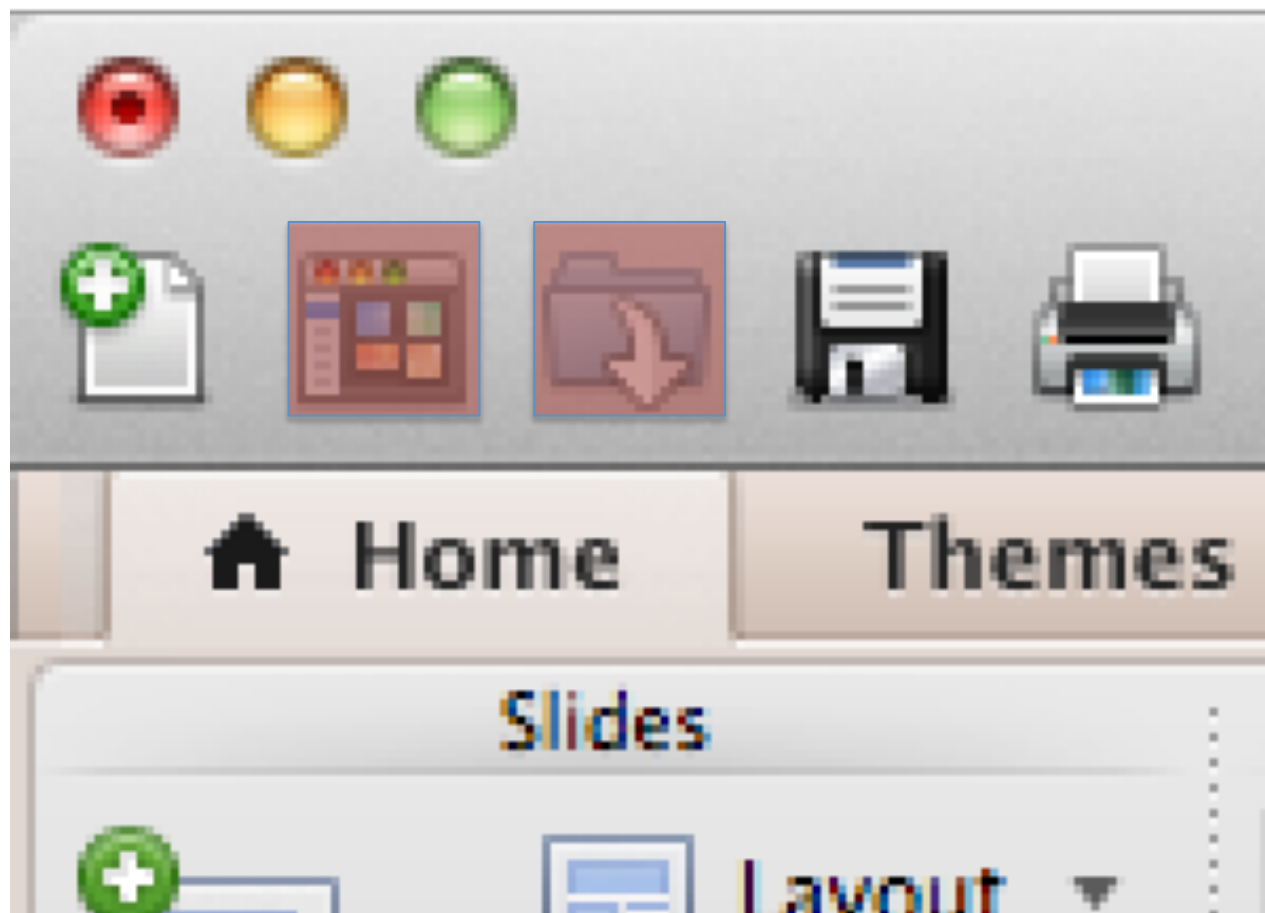


# What size?

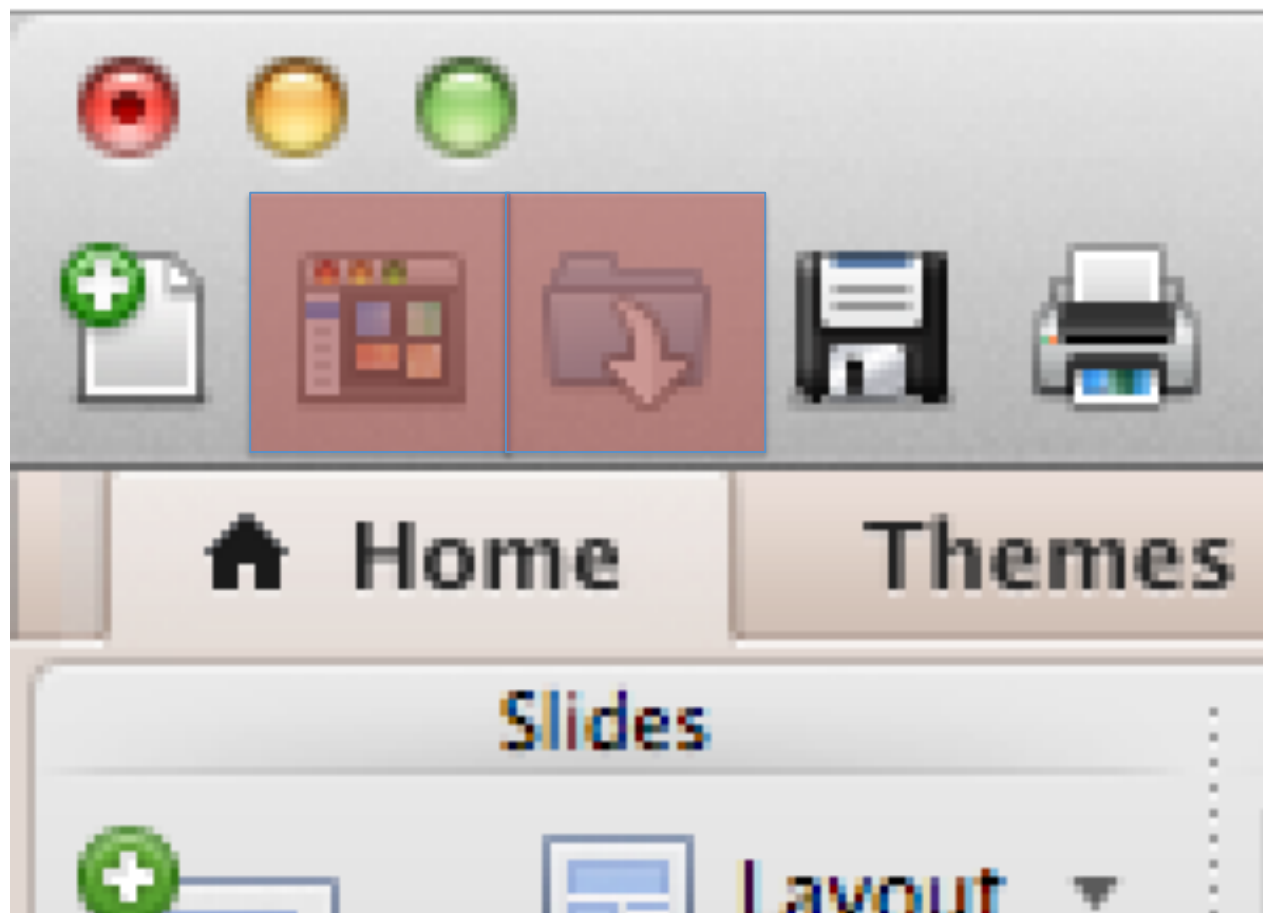
- Things are slightly more complicated on Android
- Display size, shape and resolution varies considerably from device to device
- A button that is the right size on one device would be too small/large on another
- Hence, the use of relative layouts
  - Not just about filling white-space

# Hit-box size

- Has the user touched a widget or not?
  - A touch has “hit” within a bounding box
- Think about the handles used to interact with a text frame in Word or something
- Need to be big enough that the user can accurately touch them
- Or rather the hit test area needs to be big enough
- Potentially decouple visual area from tested area







# Device size

- On mobile devices, the UI is constrained by the ratio of the device size - finger size - resolution
  - Apple 44x44 pixels
  - Microsoft min 26 pixels, ideal 34
  - Nokia 1cmx1cm, 28x28 pixels
- A UI that works on a 10" display won't necessarily "work" on a 7"
  - Steve Jobs comments about "not having to sand your fingers"
- Is a 7" device a distinct enough class of device?

Natural finger position,  
completely covers visible target



Using finger tip shows target,  
but have to reposition hand



# Fingers

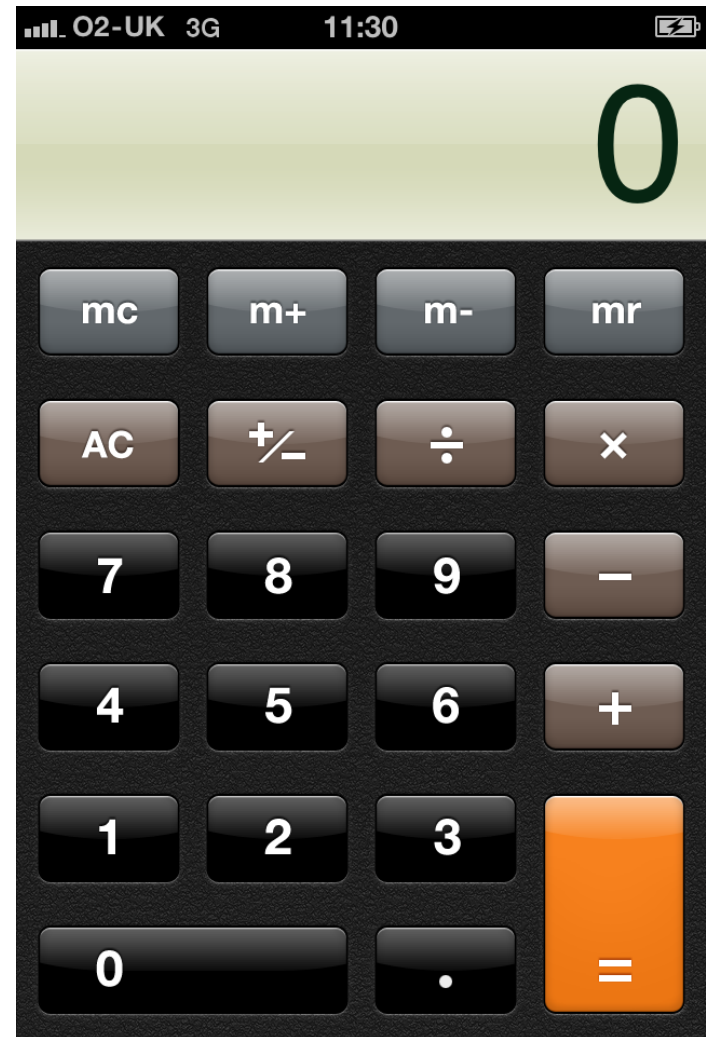
- Small touch targets
  - Touch errors
    - Finger overlaps on to neighbouring hit boxes
  - Fitts law
  - Thumbs are even bigger
- Average index finger width is 16-20mm
  - MIT study
  - 47-57 pixels on an average device
- Average thumb width is 25mm
  - 72 pixels
  - Edges of the control are visible

# Visual Feedback

- On a phone, it is quite possible for a finger to obscure the button completely
- Flashing button effect would effectively become invisible for a small button
  - Common in desktop interfaces
- Need to find alternative approaches

# iPhone calculator

- iPhone calculator takes several approaches
- Some buttons have obvious effects (e.g. digit entry)
- For operators, it leaves the button highlighted
  - Can be seen when the finger is removed



# Visual Feedback

- Other options include making the buttons bigger
  - Again, on the iPhone many buttons are the full width of the phone
  - Recall that the user is only actively engaging in one task using the phone at a time
    - Break the task into small, discrete Activities
    - Make full use of the available space
      - If not enough space, make more activities
- Need to think about how to give the user either implicit or explicit feedback that the touch was registered





# Touch Metaphors

- Finger location only known when user touches
  - Desktop UI paradigms
    - mouseovers, hover, no longer possible
  - Instead need to create new interaction mechanisms
    - e.g. touch and hold without moving
      - Analogous to hover?
- Touch (and especially multitouch) provide opportunities for new UI paradigms
  - Particularly popular on mobiles are the use of **gestures**
  - These are complex touch movements made on the device to signify an operation

# Gestures

- Tap
- Hold
- Drag
- Pinch
- Rotate
- Swipe / fling

# Programming for Touch

- Much like programming for mouse
  - onClick, onMouseDown, onMouseMove
- Handle a sequence of events
  - TouchBegin
  - TouchMoved
  - TouchEnded
- There just happens to be more than one of them
  - Need to parse a sequence of events into a gesture
  - A gesture is a series of touch events that occur over a period of **time**
    - Change in Touch

# Touch Events

- Two ways to get touch events
- Either register a new `OnTouchListener` with a view, (with `setOnTouchListener(...)`)
- Or implement `onTouchEvent()` in a custom View
- Either way, we are delivered a series of `MotionEvent`s

# MotionEvent

- This object encapsulates information about Touch events
  - Sent when a touch begins (ACTION\_DOWN)
  - When the finger moves (ACTION\_MOVE)
  - And finally when the touch ends (ACTION\_UP)
- Additional events also sent for multitouch
- Rely on the underlying OS to translate a “fat finger” into a discrete pointer

# Action Down

- A gesture starts when a finger is pressed
- A MotionEvent is generated for this ACTION\_DOWN
- Can find the action by calling `getAction()`
- This can also have the identifier of the 'pointer' so use `getActionMasked()` instead
  - If we care about multi-touch

# Action Move

- As the finger moves, a series of ACTION\_MOVE events will be sent
- Can find the new position using getX() and getY() (return floats)
  - Touch resolution is not necessarily the same as screen resolution
- Note that Android may bundle up a series of touch events
  - Ability to get 'historic' touches

# Action Up

- A gesture ends in three ways, the normal is for an ACTION\_UP event
  - This signifies that the (last) finger has been taken off the display
- If the touch event has been cancelled for any reason
  - (e.g. phone rings), then an ACTION\_CANCEL event is sent
  - ACTION\_OUTSIDE if the finger moves outside the relevant view



# Single Touch

- So a touch gesture will be formed by
  - A single ACTION\_DOWN
  - Zero or more ACTION\_MOVE
  - An ACTION\_UP to finish
- Can use the data from these to perform some interaction
  - Use position delta to move an object around the screen
    - Delta = change from original
  - Use movement velocity for a swipe / fling

# Dragging / Scrolling

- Store original location of thing to move
  - Store x,y-pair from ACTION\_DOWN
- Calculate delta from stored value and value returned from ACTION\_MOVE or ACTION\_UP
  - Change the location of thing being moved by adding delta to original location
  - Note: need to adjust as position returned is relative to View origin

# Swipe / Fling

- Can do similar for a swipe / fling
- Rather than moving the object, calculate the velocity with which it is moving
  - Speed
  - Direction
- On ACTION\_UP
  - Continue to move the object with that velocity
- Gives the user the impression of “flinging” UI elements across the screen
  - Obvious visual feedback

# Multi Touch

- Very similar to single touch
- Same sequence of events as before with a few more events thrown in
  - ACTION\_POINTER\_DOWN and ACTION\_POINTER\_UP tell us that a **new** pointer has been pressed
- Support for 256, but some Android devices only support 2

# Which finger?

- `getActionIndex()` tells us the index for the pointer caused this event for
  - `ACTION_POINTER_DOWN/ACTION_POINTER_UP`
  - However, number of pointers can change as fingers are lifted or placed
- Each pointer given an id that won't change
  - But its **index** within a bundle of movement events might
  - Need to track both the id and past locations of pointers to move things about

# Which finger?

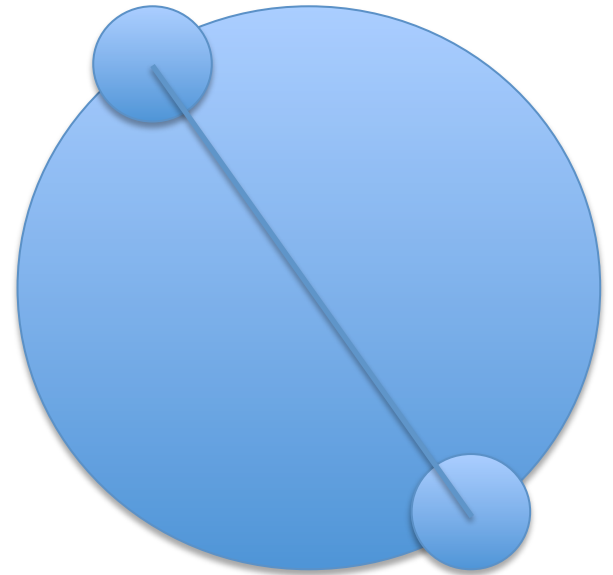
	Action	ID
#1 touch →	ACTION_DOWN	0
#2 touch →	ACTION_POINTER_DOWN	1
#3 touch →	ACTION_POINTER_DOWN	2
	ACTION_MOVE	0
#2 lift →	ACTION_POINTER_UP	1
#1 lift →	ACTION_POINTER_UP	0
#3 lift →	ACTION_UP	2

# Pointers to Gestures

- Maintain state of pointer IDs
  - Track movement of multiple fingers
- How do we convert these into gestures?
  - Pinch to zoom
  - Two-finger rotation
- Little SDK support for specific gestures
  - Implement ourselves with some simple trigonometry

# Pinch to Zoom

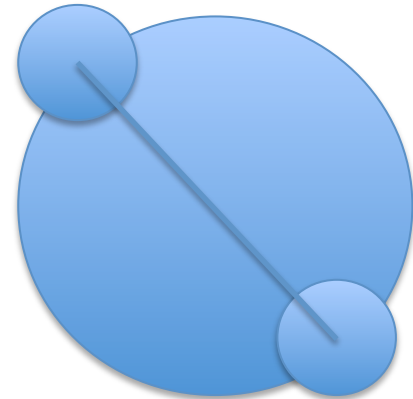
- Imagine the two points lie on the circumference of a circle
  - ACTION\_POINTER\_DOWN
- Can easily calculate the diameter of the circle
  - distance between the two points
- Use Pythagoras to calculate it





# Pinch to Zoom

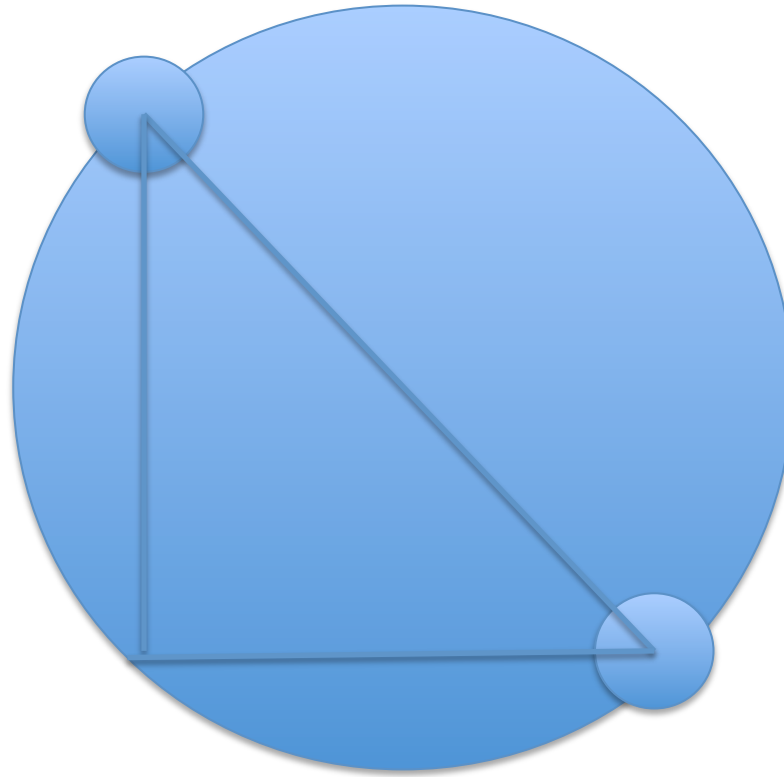
- As fingers move, diameter of the circle will change
  - store the initial diameter of the circle
- The ratio of new diameter to old diameter will give the zoom ratio
- With this and the original size of the item
  - calculate how to rescale the new item



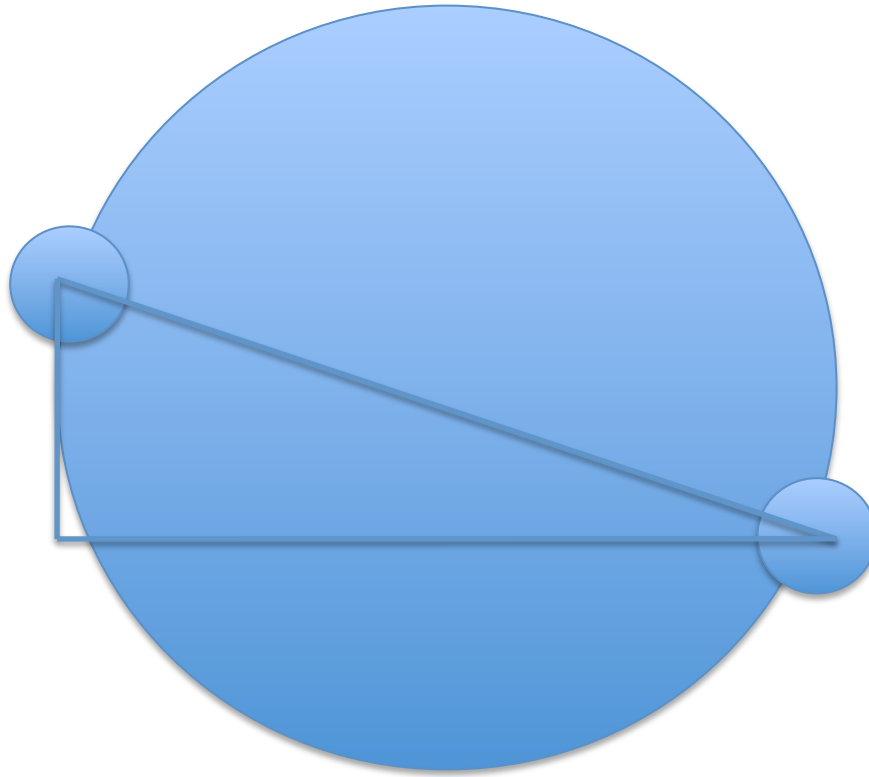
# Two-finger rotation

- Very similar approach for rotation
- The difference in position can (using basic trigonometry) give us the angle of the line bisecting the circle between the two points
- As the points move the angle will change
- Can use the difference between this and the original angle to work out the rotation
  - Magnitude may not be as important as direction
    - Rotate a photograph, document

# Two-finger rotation



# Two-finger rotation



# Android Support

- We could implement any number of complex gestures
- Android provides built-in callbacks for common gestures via `GestureDetector.SimpleOnGestureListener` class
  - `onSingleTap`
  - `onDoubleTap`
  - `onShowPress`
  - `onLongPress`
  - `onScroll`
  - `onFling`
- Can create and save custom gestures as binary resources
  - A pattern of movements
- `GestureLibrary` attempts to recognise the gesture
  - Detects attempt at complex gesture
  - Returns predications as to which gesture the input may match
  - Inspect confidence of a match
    - Gestures are an imprecise science

Let's have a look...



# References

- <http://developer.android.com/training/gestures/index.html>
- <http://developer.android.com/training/gestures/multi.html>