Gesture-controlled TV

Technology: Canesta depth camera (think Kinect)
UI: Kicker Studio, SF
How worthwhile is this course?

1 = not worthwhile, 7 = worthwhile
Lecture Pace

- Too Slow
- About Right
- Too Fast
Hours Spent per Week

- 2-4 Hours
- 4-6 Hours: 16
- 6-8 Hours: 14
- 8-10 Hours: 7
- > 10 Hours: 5
What you like

1. Discussion of UI Examples in class
2. Demonstrations and Videos
3. Android programming assignments
4. Connecting theory and practice
Areas for improvement

1. **Readings**: Some are long, tedious
2. **Not enough guidance on Android programming in class**
3. **More explanation for losing points**
Groups

1. Most groups seem to be on a productive track.
2. If yours isn’t, NOW is the time to correct course.
User Interface Components

Each component is an object with

Bounding box
Paint method for drawing itself
Drawn in the component’s coordinate system
Callbacks to process input events
Mouse clicks, typed keys

Java:
```java
public void paint(Graphics g) {
    g.fillRect(...); // interior
    g.drawString(...); // label
    g.drawRect(...); // outline
}
```

Cocoa:
```cocoa
(void)drawRect:(NSRect)rect
```
Layout: Containment Hierarchy

Window

Panel

Label
TextArea

Panel

Button

Button

Enter Text:
Events
Events

User input is modeled as “events” that must be handled by the system and applications.

Examples?

- Mouse input (and touch, pen, etc.)
  - Mouse entered, exited, moved, clicked, dragged
  - Inferred events: double-clicks, gestures
- Keyboard (key down, key up)
- Sensor inputs
- Window movement, resizing
Anatomy of an Event

Encapsulates info needed for handlers to react to input

- Event Type (mouse moved, key down, etc)
- Event Source (the input component)
- Timestamp (when did event occur)
- Modifiers (Ctrl, Shift, Alt, etc)

Event Content

- Mouse: x,y coordinates, button pressed, # clicks
- Keyboard: which key was pressed
Callbacks

Slider

- onMouseOver(Event e){...}
- onMouseDown(Event e){...}
- onMouseClick(Event e){...}
- onMouseMoved(Event e){...}
Event Dispatch

Application

Cocoa

Carbon

Window Server

I/O Kit

Mouse

Keyboard

Tablet & stylus

Application environments

Application Services

Core Services

Kernel environment

Apple, Cocoa Event-Handling Guide
Event Dispatch Loop

Event Queue
• Queue of input events

Event Loop (runs in dedicated thread)
• Remove next event from queue
• Determine event type
• Find proper component(s)
• Invoke callbacks on components
• Repeat, or wait until event arrives

Component
• Invoked callback method
• Update application state
• Request repaint, if needed

Mouse moved \((t_0, x, y)\)

Example event: Mouse moved \((t_0, x, y)\)
Event Dispatch Loop

1) Events from input devices enter here

2) Event is added to FIFO event queue

3) Main loop processes one event per iteration

Main run loop

Event source

Mach port

Window server

Apple, Cocoa Event-Handling Guide
Event Dispatch

Event Queue
- Mouse moved \((t_0,x,y)\)
- Mouse pressed \((t_1,x,y,1)\)
- Mouse dragged \((t_2,x,y,1)\)
- Key typed \((t_3, ‘F1’)\)
- …

(queues and dispatches incoming events in a dedicated thread)

/* callback for TextArea */
public void mouseMoved(e) {
    // process mouse moved event
}
Mouse/Touch vs. Keyboard Events

Mouse/touch Events are (usually) routed to the top-most (in z-order) visible component underneath the cursor using hit testing.

Keyboard events are (usually) routed to the component that has key focus.
Key Focus: Form Example

Contact Information
Please fill in all fields.

1. First name
2. Last name
3. Street address
4. City
5. Postal code
6. State
7. Business phone
8. Mobile phone
9. Home phone
10. E-mail address
Abstracting Events

Level of abstraction may vary. Consider:

**Mouse down vs. double click vs. drag**

**Touch move vs. gesture**
Event Listeners vs Event Handlers

Two ways to receive events in Android:

**Event Handlers:**
Used with *custom components* (i.e., you created a custom widget). Override member function of your View-derived class to receive events. Examples:
- `onKeyDown(int, KeyEvent)`,
- `onTouchEvent(MotionEvent)`

**Event Listeners:**
Used to be notified of events on *standard components* by outside classes (e.g., your Activity). Examples:
- `onKey(View, int, KeyEvent)`,
- `onTouch(View, MotionEvent)`
Detecting Gestures

Two different kinds of gestures:

**Continuous manipulation gestures:**
(e.g., pinch-to-zoom)

**Stroke recognition gestures**
(e.g., Handwriting recognition, Swype)

Android Gesture Search:
http://www.youtube.com/watch?v=umos1GZKbKw
Detecting Gestures

Most event architectures assume there is a single, “correct” response to a single input event.

This model is not well suited to describing multitouch interactions. Why?

Recognition, co-existence of different gesture types complicate the picture: input can match multiple possible interpretations

How to deal with uncertainty is still a research topic in HCI.
Model-View-Controller Architecture
Model-View-Controller

**OO Architecture for interactive applications**
introduced by Smalltalk developers at PARC ca. 1983
Model

Information the app is manipulating

Representation of real world objects

circuit for a CAD program
logic gates and wires connecting them
shapes in a drawing program
geometry and color
“The user's conceptual model of the system captures the semantics of objects, relationships, and behavior” (Collins)
View

Implements a visual display of the model

May have multiple views
  e.g., shape view and numerical view
Multiple Views
View

Implements a visual display of the model

May have multiple views
e.g., shape view and numerical view

Any time model changes each view must be notified so it can update
e.g., adding a new shape
Controller

Receives all input events from the user

Decides what events mean and what to do
communicates with view to determine the objects being manipulated (e.g., selection)
calls model methods to make changes on objects
model makes change and notifies views to update
Why MVC?
Why MVC?

Combining MVC into one class will not scale
model may have more than one view
each is different and needs update when model changes

Separation eases maintenance and extensibility
easy to add a new view later
model info can be extended, but old views still work
can change a view later, e.g., draw shapes in 3D
flexibility of changing input handling when using separate controllers
“pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)”
– Olsen, Chapter 5.2
“pattern of behavior in response to user events (controller issues) is independent of visual geometry (view issues)”

– Olsen, Chapter 5.2

But controller must usually contact view to interpret what user events mean (e.g., selection)
Combining View & Controller

View and controller are tightly intertwined
lots of communication between the two

Almost always occur in pairs
i.e., for each view, need a separate controller

Many architectures combine into a single class (“MV”)
Terminology

Is an android.view.View object an MVCView?
What about an Activity?
Model-ViewController in Android

**Model:**
Inherit from `java.util.Observable` class.
Provide accessors and mutators for state.
Call `setChanged()` and `notifyObservers()`

**Activity:**
Implement `java.util.Observable`:
add `update()` method
Example Application

Blue circles: 4
Cardinal squares: 2
Changing the Display

How do we redraw graphics when a shape moves?
Moving Cardinal Square

Blue circles: 4
Cardinal squares: 2
Erase w/ Background Color and Redraw

Blue circles: 4
Cardinal squares: 2
Changing the Display

**Erase and redraw**

using background color to erase fails
drawing shape in new position loses layer ordering
Damage / Redraw Method

View informs windowing system of areas that are damaged
does not redraw them right away…

Windowing system
batches updates
clips them to visible portions of window

Next time waiting for input
windowing system calls Repaint() method
passes region that needs to be updated
Damage old, Change position in model, Damage new

Blue circles: 4
Cardinal squares: 2
From the Android Reference:

**How Android Draws Views**

“When an Activity receives focus, it will be requested to draw its layout. […]”

Drawing begins with the root node of the layout. Drawing is handled by walking the tree and rendering each View that intersects the *invalid region*. The framework will not draw Views that are not in the invalid region. […]

You can force a View to draw, by calling `invalidate()`.
Threading in User Interfaces
A **thread** is a **partial virtual machine**. Each thread has its own stack (and local variables) but shares its heap with other threads in the same application.

Threads can be independently scheduled by the OS/VM.

```c
for (i=0; i<n; i++)
{
    tmp = A[i];
    A[i] = B[i];
    B[i] = tmp;
}
```

Thread1

Thread2
Threads vs. Processes

A **process** is a **complete virtual machine** with its own stack and heap.

Threads share memory – processes don’t.

Threads can communicate through shared memory, processes need other mechanisms (IPC = inter-process communication).
Pros and Cons

Why use threads?
Useful model of concurrent execution, both on single processors (time-division multiplexing) and on multi processor/multi-core systems.
Threads are relatively cheap to create, versatile because of shared memory.

Why wouldn’t one use threads?
Complicated programming model. Multithreaded programming is one of the biggest productivity killers of all time.
(locks, semaphores, monitors, mutexes, signals, spawn, fork, join,...)
“After a long and careful analysis the results are clear: 11 out of 10 people can't handle threads.”

— Todd Hoff
Why use multithreading for UIs?

Interactive programs need to respond quickly to user input. Direct manipulation assumes that objects onscreen respond to user’s touch/cursor.
Why use multithreading for UIs?

Not all code can complete quickly inside an event handler. Examples?
btnStart.setOnClickListener(new OnClickListener() {
    public void onClick(View arg0) {
        // start long computation
        Thread.sleep(60000);
        // update UI when done
        txtResult.setText("Done.");
    }
});
Event Dispatch Loop

Event Queue
- Queue of input events

Event Loop (runs in dedicated thread)
- Remove next event from queue
- Determine event type
- Find proper component(s)
- Invoke callbacks on components
- Repeat, or wait until event arrives

Component
- Invoked callback method:
  -compute...
  -compute...
- Update application state
- Request repaint, if needed

Mouse moved \((t_0,x,y)\)

Long-running operation
Stopped the event loop!
btnStart.setOnClickListener(new OnClickListener() {
    public void onClick(View v) {
        new Thread(new Runnable() {
            public void run() {
                // start long computation
                Thread.sleep(10000);
                // update UI when done
                txtResult.setText("Done.");
            }
        }).start(); // start new thread
    }
});
android.view.ViewRoot$CalledFromWrongThreadException:

Only the original thread that created a view hierarchy can touch its views.
Event Dispatch Loop

Event Queue
• Queue of input events

Event Loop (runs in dedicated thread)
• Remove next event from queue
• Determine event type
• Find proper component(s)
• Invoke callbacks on components
• Repeat, or wait until event arrives

Component
• Invoked callback method
• Update application state
• Request repaint, if needed

Mouse moved \((t_0, x, y)\)

Launch Thread
...Compute...
Update app state
Updating the UI from another thread

All common UI frameworks have a single UI thread
You are only allowed to modify the UI from the main thread.

Two fundamental rules:
Do not block the UI thread
Background threads they must not modify the UI.

Solution: When worker thread completes, request update back in the UI thread.
How to properly update the UI

Almost all GUI frameworks offer some convenient mechanism to notify the main thread from another thread.

Android has at least three such mechanisms:

1. Call `View.post(Runnable)` from worker thread
2. Subclass `AsyncTask` – creates threads behind the scenes
3. Send messages in one thread with `Handler.sendMessage()` – message is received in another thread (like IPC)
Handler.sendMessage Example

**Main thread**

- Handle event
- Handle event
- btn.OnClick()
- Handle event
- Handle event
- Handle event

- handleMessage()
- update GUI

**Helper thread**

- Start new thread

- Long computation
  - 
  - 
  - sendMessage("done")

**Message queue**
public class ThreadDemo extends Activity {
    final Handler handler = new Handler() {
        public void handleMessage(Message msg) {
            // update UI
            txtResult.setText((String)msg.obj);
        }
    };
}
public void onClick(View arg0) {
    new Thread(new Runnable() {
        public void run() {
            // long computation…
            Message msg = new Message();
            msg.obj = "Done."
            handler.sendMessage(msg);
        }
    }).start();
}
CS160 Thread Demo

Start Long Computation

Result:

Do Something Else
Working with Sound
Raw audio data – bit depth

Bit depth is size per sample
As 16-bit mono (1 channel)
Signed short: -32768 to +32767
May need to convert to float: -1.0 to 1.0
Raw audio data – sampling rate

Sampling rate: number of values (samples) per second
8000 Hz: telephone (adequate for speech)
16000 Hz: modern VoIP products
44100 Hz: audio CD quality
Basic audio record/playback

// create AudioRecord and AudioTrack
int minBufferSize = AudioRecord.getMinBufferSize(SAMPLE_RATE, AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT);
buffer = new byte[minBufferSize];
recorder = new AudioRecord(
    MediaRecorder.AudioSource.MIC, SAMPLE_RATE,
    AudioFormat.CHANNEL_IN_MONO,
    AudioFormat.ENCODING_PCM_16BIT, minBufferSize);
audioTrack = new AudioTrack(AudioManager.STREAM_MUSIC,
    SAMPLE_RATE, AudioFormat.CHANNEL_OUT_MONO,
    AudioFormat.ENCODING_PCM_16BIT, minBufferSize,
    AudioTrack.MODE_STREAM);

recorder.startRecording();
audioTrack.play();

// in another thread, while recording:
int bufferReadResult = recorder.read(buffer, 0, buffer.length);
audioTrack.write(buffer, 0, bufferReadResult);
Frequency representation

Fast Fourier Transform (FFT) converts signal to frequency domain
Spectrogram representation

FFT over many small windows of the signal
Column: one time frame of the audio
Row: frequency
Color: amplitude of frequency at time
Pitch detection

**Using TarsosDSP:**

```java
PitchProcessor mPitchProc = new PitchProcessor(PitchProcessor.PitchEstimationAlgorithm.AMDF, SAMPLE_RATE, bufSize, this);

// in audio processing thread
// after creating AudioEvent
mPitchProc.process(audioEvent);

// elsewhere, implement handlePitch
```
Low-pass filter

Original

After filter with 1000Hz stop band
Low-pass filter

Using TarsosDSP:

```java
LostPassFS mLowPassFilter = new LowPassFS(0, SAMPLE_RATE);
// in audio processing thread, after // creating an AudioEvent:
mLowPassFilter.process(audioEvent);
byte[] filteredBuffer = audioEvent.getByteBuffer();
mAudioTrack.write(filteredBuffer, 0, filteredBuffer.length);
```
Speech to text

Need a recognizer installed (e.g., Google Voice Search)

```java
SpeechRecognizer mSpeechRecognizer = SpeechRecognizer.createSpeechRecognizer(this);
Intent i =
    new Intent(RecognizerIntent.ACTION_RECOGNIZE_SPEECH)
        .putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.LANGUAGE_MODEL_FREE_FORM)
        .putExtra(RecognizerIntent.EXTRA_PROMPT, "Speak!");
    .putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE, "com.example.speechtotext");
mSpeechRecognizer.startListening(i);
```
Text to Speech

```java
TextToSpeech mTTS = 
new TextToSpeech(this, this);

mTTS.speak("I am a computer",
TextToSpeech.QUEUE_ADD, null);
```