Input part 3:
Interaction Techniques

Georgia Tech

Interaction techniques

- A method for carrying out a specific interactive task
  - Example: enter a number in a range
    - could use… (simulated) slider
    - (simulated) knob
    - type in a number (text edit box)
  - Each is a different interaction technique

Interaction techniques in libraries

- Generally ITs now come in the form of "Widgets", "controls", "components", "interactors"
- Typically in reusable libraries
  - e.g. widget sets / class libraries
- Also need custom ones

Design of interaction techniques

- Just going to say a little
- Guidelines for interaction technique design
  - Affordance
  - Feedback
  - Mechanics (incl. performance)

  - Gee, these sound sort of familiar...

Affordance

- Can you tell what it does and what to do with it by looking at it?
  - Most important for novices
    - but almost all start as novices
    - if people don't get past being novices you fail
-

(Historical sidebar on “affordances”)

- *Affordances* as a concept originally introduced by J.J. Gibson (1977), a perceptual psychologist
  - Referred to “actionable properties” between the world and a person
    - Relationship between these things, not always visible or even known
- Appropriated by Don Norman in *Psychology of Everyday Things*
  - But he basically redefined Gibson’s term to mean: does the thing make it self-obvious what we can do with it?
- Has since clarified (backtracked?) but his older definition has stuck with much of HCI
  - Should have written “perceived affordances”
  - But be careful when using the term, as affordance purists will likely take objection...

Feedback

- Can you tell what its doing?
- Can you tell the consequences of the actions?
  - e.g. Folders highlight when you drag over them indicating that if you "let go" the file will go inside the folder
    - very important to reliable operation
  - important for all users

Mechanics: "feel" and difficulty

- Fitts' law tells us about difficulty
  - predicts time to make a movement
- "Feel" is trickier
  - Can depend on physical input dev
    - physical movements, forces, etc.
  - Really gets back to the difficulty of the movement, but harder to characterize
- Important for all, but esp. experts

Fitts' law

$$\text{Time} = A + B*\log_2(\text{Dist}/\text{Size} + 0.5)$$

- Time is linearly proportional to log of "difficulty"
  - proportionality constants depend on muscle group, and device
  - Difficulty controlled by distance and required accuracy (size of target)

Fitts' law

- The *mechanical* component of true expert performance tends to be closely related to time required for movements
  - not well related to learning (or performance) of novices
  - still need to consider "cognitive load"

Fitts' law

- Actual numbers from Fitts' law generally not all that helpful
    - that level of detailed analysis is hard
- General guideline: this all boils down to a few simple properties:
    - Keep required movements (accuracy & distance) firmly in mind
        - Avoid device swapping
        - Avoid disturbing focus of attention

Mini case study #1
The original "Macintosh 7"

- Macintosh (1984) was first big success of GUIs
  - originally came with 7 interactors built into toolbox (hence used for majority)
- Most not actually original w/ Mac
  - Xerox Star (+ Smalltalk & earlier)

The Macintosh 7

- Generally very well designed (iterated with real users!)
    - very snappy performance
        - dedicated whole processor to updating them (little or no “OS”)
- Huge influence
    - These 7 still cover a lot of today’s GUIs (good and bad to that)

Button

- Shaped as rounded rectangles

  (about "modern" square corners…)

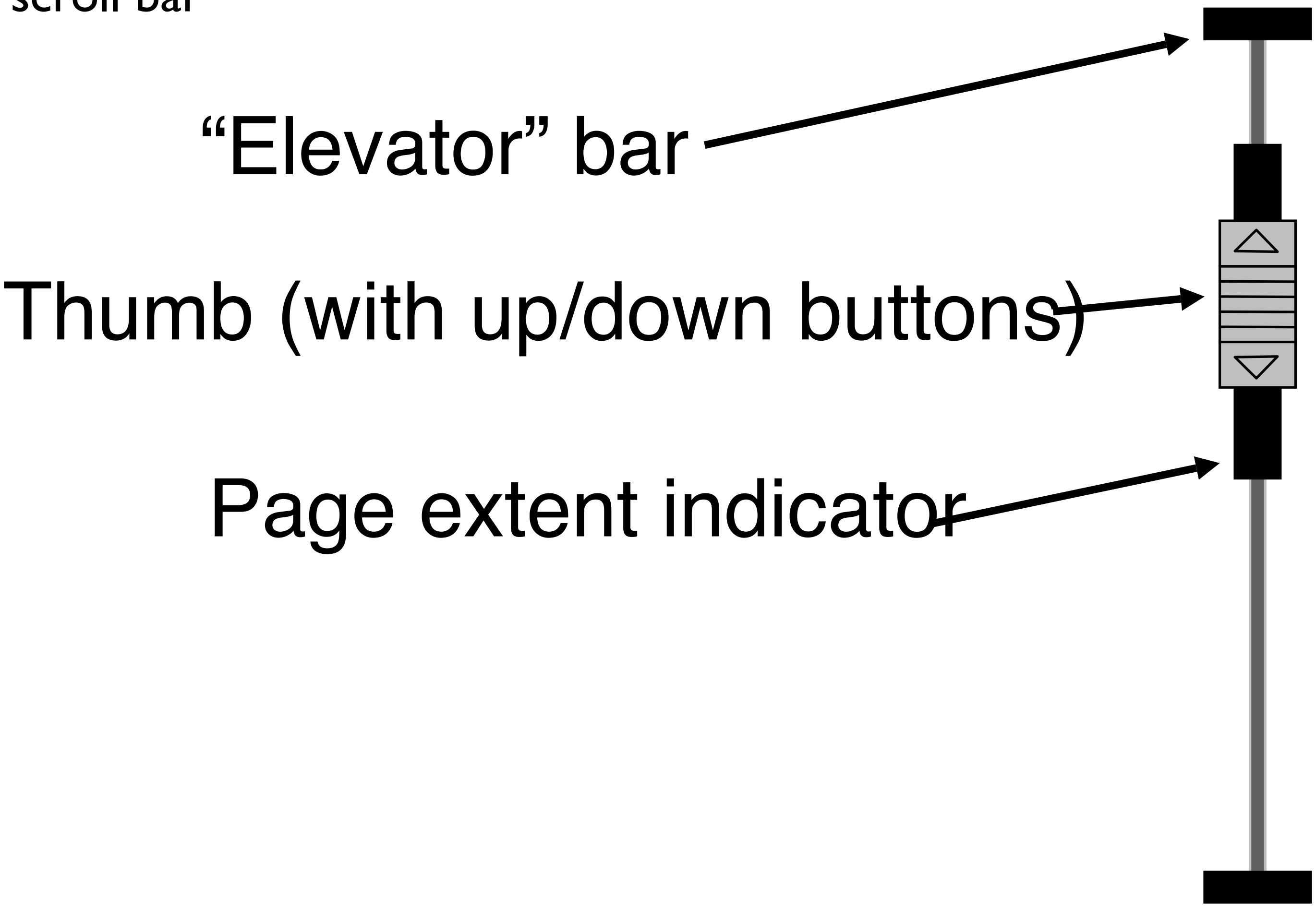
- Inverted for feedback
  - Recall Mac was pure B/W machine
  - Pseudo 3D appearance hard and hadn't been invented yet

Slider

- Used for scroll bars (but fixed size "thumb")
  - Knurling on the thumb
  - "Pogo stick" problem

Aside: a different scrollbar design

- Openlook scroll bar



“Elevator” bar

Thumb (with up/down buttons)

Page extent indicator

Pulldown menu

- This was original with Mac
- Differs slightly from Windows version you may be familiar with
  - had to hold down button to keep menu down (one press-drag-release)
  - Changed in later versions
- Items highlight as you go over
- Selected item flashes

Check boxes, radio buttons, text entry / edit fields

- Pretty much as we know them
- Single or multi-line text supported from the beginning

File pick / save

- Much more complex beast than the others
  - built from the others + some
    - e.g. no affordance, by you could type and file list would scroll to typed name

Original Mac also had others

- Window close and resize boxes
- Drag & open file icons and folders
- Not made generally available
  - not in toolbox, so not (re)usable by other programmers

Second major release of Mac added a few

- Lists
  - single & multiple selection
  - from textual lists (possibly with icons)
- Hierarchical ("pull-right") menus
- Compact ("in-place") menus
  - select one-of-N pulldown
- Window zoom box

Have seen a few more added since then

- Tabbed dialogs now widely used
- Hierarchical lists (trees)
- "Combo boxes"
  - Combination(s) of menu, list, text entry
- A few more + variations on things
- Typically don't see much more than that

Almost all GUIs supported with the above 10-12 interactor types

- Good ones that work well
  - uniformity is good for usability
- But, significant stagnation
  - "dialog box mindset"
  - opportunities lost by not customizing interaction techniques to tasks

Mini case study 2: Menus

- Menu
  - supports selection of an item from a fixed set
  - usually set determined in advance
  - typically used for "commands"
  - occasionally for setting value (e.g., picking a font)

Design alternatives for menus

- Simple, fixed location menus
  (see these on the web a lot)
  - easy to implement
  - good affordances
    - easy for novices (always same place, fully visible)
  - Focus of attention problems
  - Screen space hog

Popup menus

- Menu pops up under the cursor (sometimes via "other button")
  - close to cursor
    - not under it, why?

Popup menus

- Menu pops up under the cursor (sometimes via "other button")
  - close to cursor
    - What does Fitts' law say about this?

Popup menus

- Menu pops up under the cursor (sometimes via "other button")
  - close to cursor
    - Fitts law says: very fast
    - also focus not disturbed
  - takes no screen space (until used)
  - can be context dependent (!)
  - poor (non-existent) affordance

Getting best of both: Mac pulldown menus

- Menu bar fixed at top of screen, with pull-down submenus
  - benefits of fixed location
  - provides good affordance
  - good use of space via partial popup
  - but splits attention & requires long moves

Fitts' law effects

- Windows menus at top of windows, vs. Mac menus at top of screen
  - Interesting Fitts' law effect
    - what is it?

Fitts' law effects

- Windows menus at top of windows, vs. Mac menus at top of screen
  - Interesting Fitts' law effect
    - thin target vertically (direction of move) => high required accuracy
    - hard to pick
    - but…   (anybody see it?)

Fitts' law effects

- With menu at top of screen can overshoot by an arbitrary amount

  (Example of a "barrier" technique)

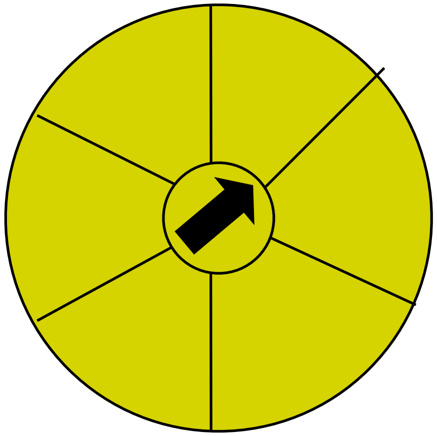
  - What does Fitts' law say about that?

Fitts' law effects

- With menu at top of screen can overshoot by an arbitrary amount
  - very large size (dominated by horizontal which is wide)
  - Original Mac had 9" screen so distance not really an issue
  - very fast selection

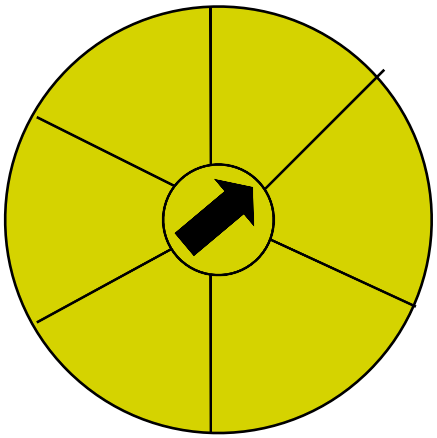
Pie menus

- A circular pop-up menu
  - no bounds on selection area
    - basically only angle counts
    - do want a "dead area" at center
  - What are Fitts' law properties?



Pie menus

- A circular pop-up menu
  - no bounds on selection area
    - basically only angle counts
    - do want a “dead area” at center
  - Fitts’ law properties:
    - minimum distance to travel
    - minimum required accuracy
    - very fast



Pie menus

- Why don't we see these much?

Pie menus

- Why don't we see these much?
  - Just not known
  - Harder to implement
    - particularly drawing labels
    - but there are variations that are easier
  - Don't scale past a few items
    - No hierarchy

Beating Fitts' law (a hobby of mine)

- Can't really beat it
  - property of being human
  - but you can "cheat"!
- One approach: avoid the problem
  - use a non-pointing device
    - shortcuts & fixed buttons
    - mouse wheel for scrolling

Beating Fitts' law

- Not everything can be a shortcut
- Other major approach: manipulate interface to reduce difficulty
  - distance (put things close)
    - but not everything can be close
    - have to make them smaller!

Beating Fitts' law

- Most ways to "cheat" involve manipulating size
  - typically can't make things bigger w/o running out of screen space (but look at that as an option)
  - but… can sometimes make things act bigger than they are

“Cheating” on target size

- Consider targets that are not just passive
  - not all movements end in “legal” or useful positions
  - map (nearby) “illegal” or non-useful onto “legal ones”
    - hit of “illegal” position treated as legal
      - e.g. positions above Mac menubar
    - effective size bigger

Snapping (or "gravity fields")

- Treat movement to an "illegal" point as if it were movement to closest "legal" (useful / likely) pt
  - Cursor or other feedback snaps to "legal" position
  - Drawn to it as if it has gravity

Snapping

- Simplest: grids
- Constrained orientations & sizes
  - 90° & 45°, square
- More sophisticated: semantic
  - only attach circuit diagram items at certain spots

Snapping

- Even more sophisticated: dynamic semantics
  - Check legality and consequences of each result at every move
    - don’t catch errors, prevent them!