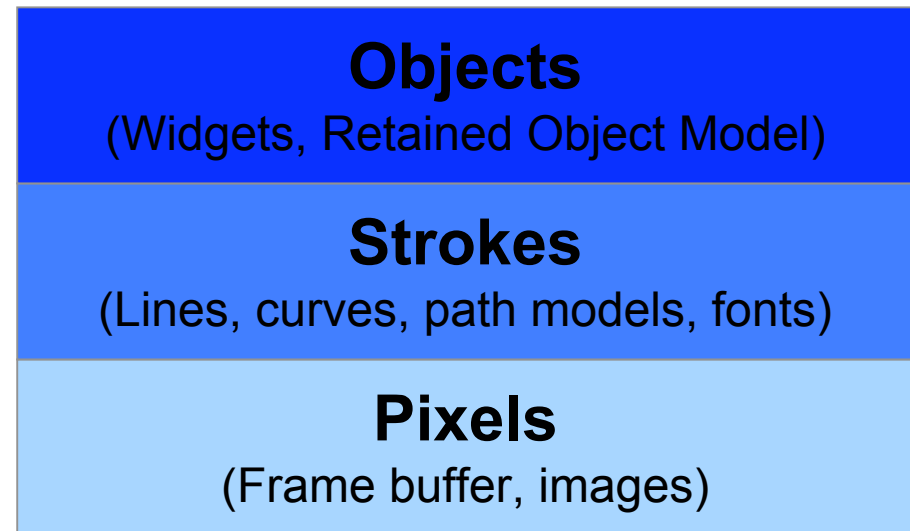


GUI Input: Devices and Input Models



Recap on GUI Output

- Hardware
- Different layers
- Damage / Redraw
- Layout Strategies



- Now switch to other large aspect of GUIs: Input

Input

- Generally, input is harder than output
 - More diversity, less uniformity
 - More affected by human properties
- Will start with hardware and then to software

Input devices

- Keyboard
 - Ubiquitous, but somewhat boring...
 - Quite mature design, average 50-60 wpm



- QWERTY key layout
 - Where did it come from?

QWERTY key layout

- From typewriter, Christopher Sholes 1868
- Urban Legends:
 - Salespeople could type “typewriter” on first row only
 - Designed to slow people down
- Originally designed to spread out likely adjacent key presses to overcome jamming problem of very early mechanical typewriters
 - Left / right / left / right



Other Keyboard Layouts

- Other layouts have been proposed
 - Dvorak is best known
 - Widely believed to be better



- Experimental and theoretical evidence casts doubt on this
 - Alternating hands of QWERTY are a win since fingers move in parallel

QWERTY keyboard layout

- Whether or not Dvorak layout is better, it did not displace QWERTY
 - Economists call this “lock-in” or “path dependence”
 - Lesson: once there is sufficient critical mass for a standard, nearly impossible to dislodge (even if apparently better)
 - Sometimes things are “good enough”
- We will see this issue again and again
 - Pie menus, other research

Chorded Keyboards

- Fast, less space, but lot more training



Twiddler

- One-handed chorded keyboard
 - After 400 minutes of practice, ten novices averaged over 26 words per minute



Visual Keyboard

- People with disabilities
 - Combine word prediction with eyegaze
 - Note that word prediction not always better



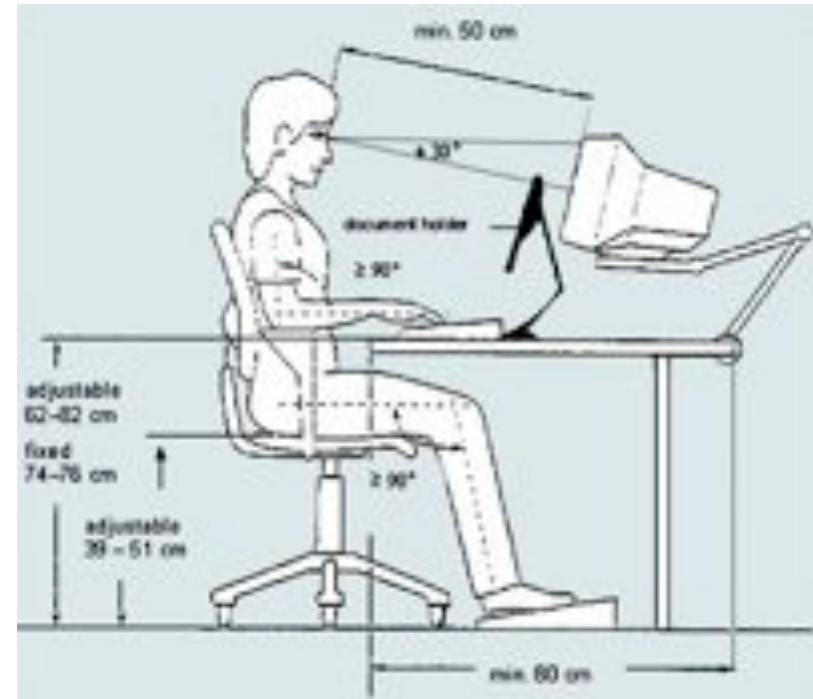
Optimus Keyboard

- Still vaporware at this point
 - Add OLEDs to keys, reconfigurable displays



Keyboards

- Repetitive Stress Injury
 - Switching keyboard to mouse and back a lot
- Take this seriously!
 - Can be a VERY big deal
 - Adjust your work environment (e.g. chair height)
 - Take breaks
 - If you have pain: stop



Buttons

- Similar to keyboard, but not for typing letters
 - Used to be common in old days
 - These days, primarily on mouse



Valuators

- Returns a single value in range
- Major implementation alternatives:
 - Potentiometer (variable resistor)
 - Similar to typical volume control
 - Shaft encoders
 - Sense incremental movements



Locators (Pointing Devices)

- Returns a location (x,y point)
 - usually screen position
- Examples
 - Mice (current defacto standard)
 - Track balls, joysticks, tablets, touch panels, etc.
- Could people use devices not coupled to screen?

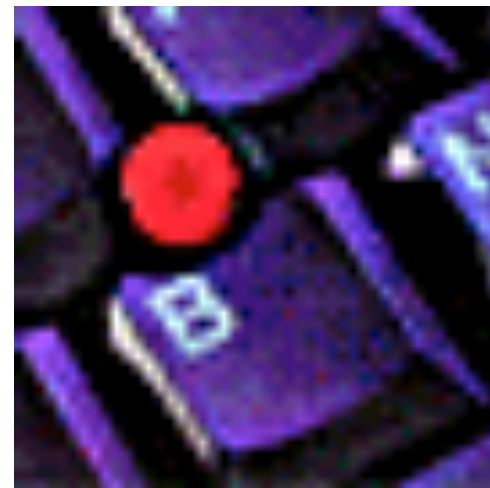


Locators

- Two major categories:
 - Absolute vs. Relative locators
- Absolute: One-to-one mapping from device movement to input
 - Ex. Tablets, touch screens
 - Easier to develop motor skills
 - But doesn't scale past fixed distances
 - bounded input range
 - Can be less accurate for same range of physical movement

Relative locators

- Relative or incremental mapping
- E.g., maps movement into rate of change of input
 - Ex. Joystick or TrackPoint
 - More accurate (for same range of movement)
 - Harder to develop motor skills
 - Not bounded (can handle infinite moves)



Recap

- Absolute: One-to-one mapping from device movement to input
 - Ex. Tables, touchscreens
 - Bounded input range
- Relative:
 - Ex. Joystick, TrackPoint
 - Unbounded input range
- Q: Mouse absolute or relative locator?
 - Ignore “acceleration” for the moment

Discuss for 2 minutes

Q: Mouse relative or absolute locator?

- Third major type: “Clutched absolute”
 - Within a range its absolute
 - Can disengage movement (pick it up) to extend beyond range
 - picking up == clutch mechanism
- Other examples:
 - Camera phone as mouse
 - Uses optical flow to estimate direction



Pointing and Selection

- New possibility: camera mouse
 - Already a product
 - Works like optical mouse
- Also tried:
 - Tilt sensors
 - Hard to get good precision

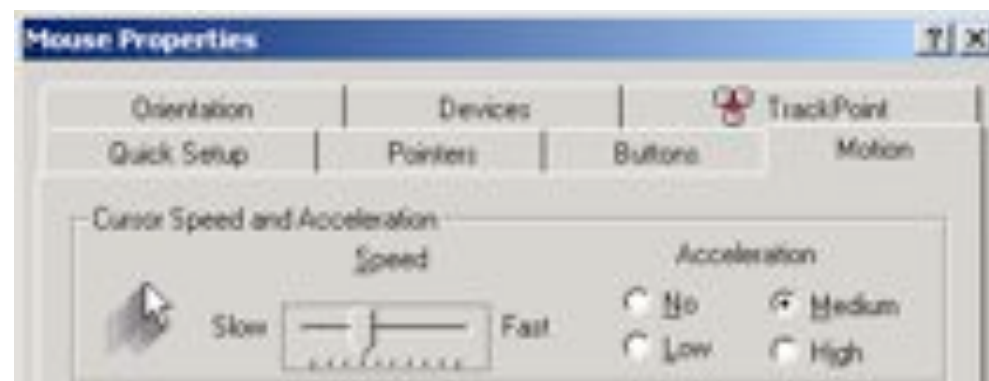


Clutched absolute locators

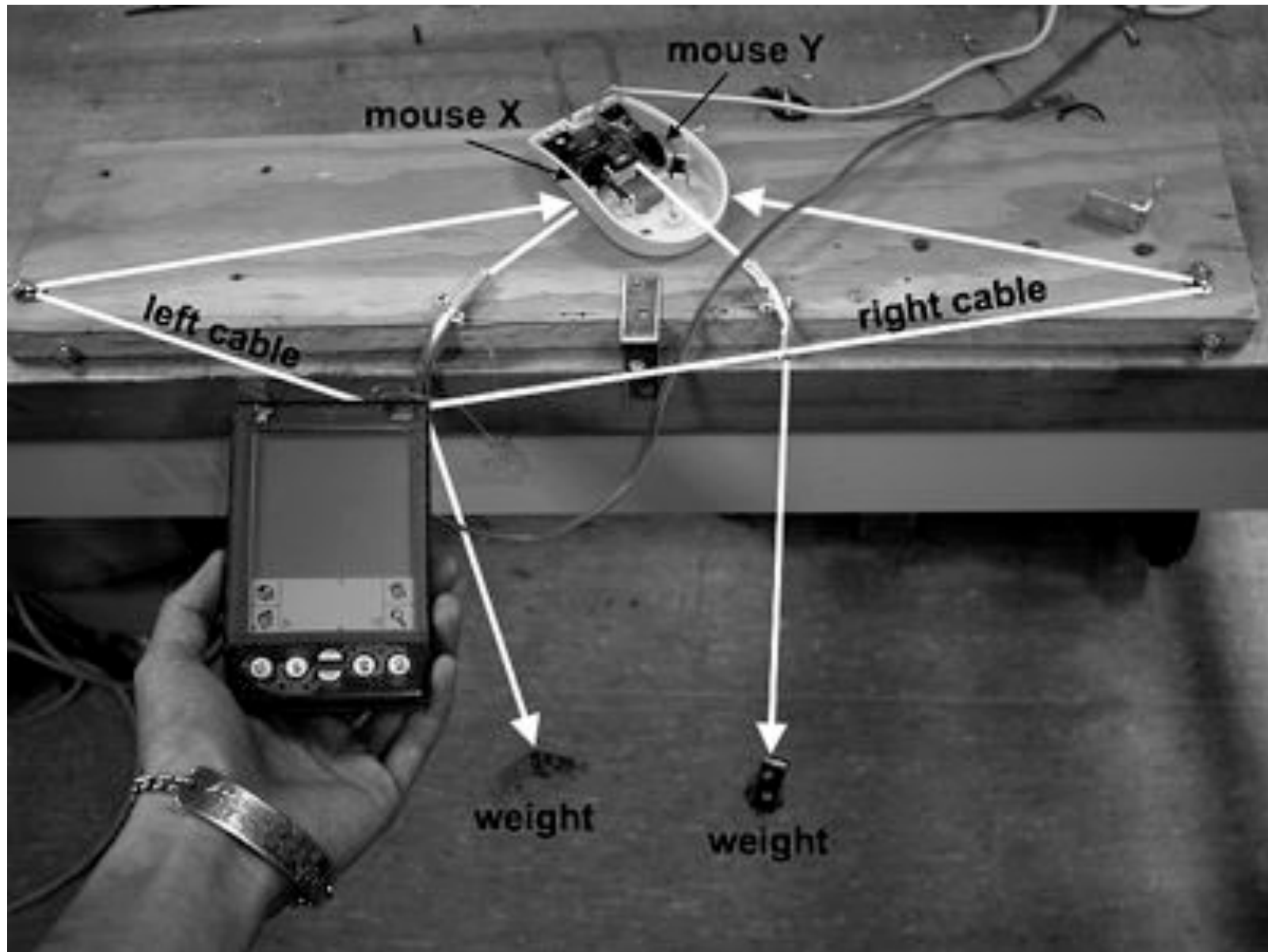
- Good compromise
 - Get one-to-one mapping when “in range” (easy to learn, fast, etc.)
 - Clutch gives some of benefits of a relative device (e.g., unbounded)
- Trackballs also fall into this category

Mouse Acceleration

- Since mouse is unbounded we can play a clever trick
- Increase speed when mouse is moving fast
 - Middle of movement
- Normal when moving slow
 - Start and end of movement
- Interesting perceptual effect:
people basically don't notice this



Peephole Display



Touch panel

- What kind of a device?

Touch panel

- Absolute device
- Possible to do input and output together in one place
 - actually point at things on the screen
- Resolution limited by size of finger (“digital input”)
 - Or requires a pen

Touch panel construction

- Membrane
 - resistive, fine wire mesh
- Optical
 - finger breaks light beam
- Surface acoustic waves
- Capacitive
 - PDA screens, SmartBoards
 - Single touch only



Drawing tablet

- Absolute or relative?

Drawing tablet

- Absolute device
- Normally used with pen / stylus
 - Allows “real drawing” (try drawing with a mouse vs. a pen)
 - Can often trace over paper images

Interesting device: Virtual Ink Mimio



- Updated acoustic tablet
 - recording whiteboard
 - ultrasonic chirps
 - 100dpi resolution over ~8ft

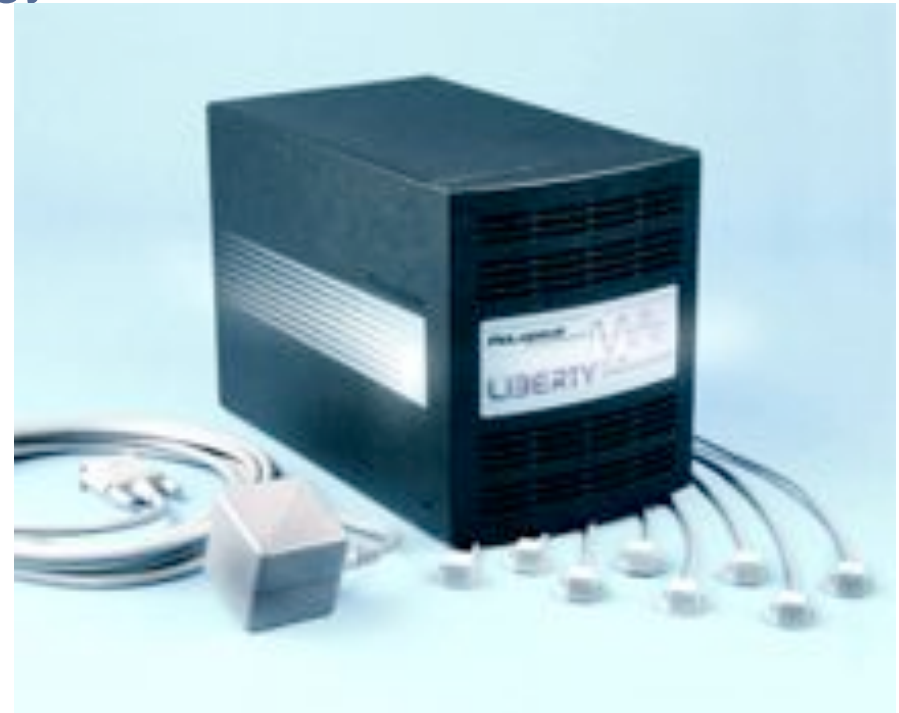


Interesting device: CrossPad



3D locators

- Can extend locators to 3 inputs
- Ex. Polhemus tracker
 - 6D device (x,y,z + pitch, roll, yaw)
 - Magnetic sensing technology



3D locators

- Can extend locators to 3 inputs
- Ex. Phantom
 - Haptic feedback



3D locators

- Can extend locators to 3 inputs
- Ex. Data Glove



Lots of other emerging possibilities

- QR Codes



Lots of other emerging possibilities

- Camera Phones for semi-literate people
- Grameen Bank
 - Microloans

(452) 5552589-101 Record ID _____	
	Loan Application Mahakalasm SHG Trust 7 
1  Date _____	3  Loan Amount _____
2  Account No. _____	4  Instalments _____

Reviewing an entered value

Editing a value by clicking on the associated barcode button. An audio prompt is also played.

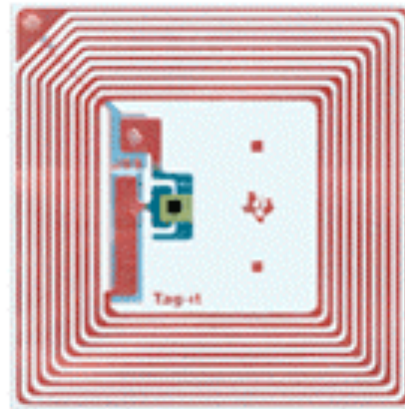


Pointing and Selection

- Barcodes



- RFIDs

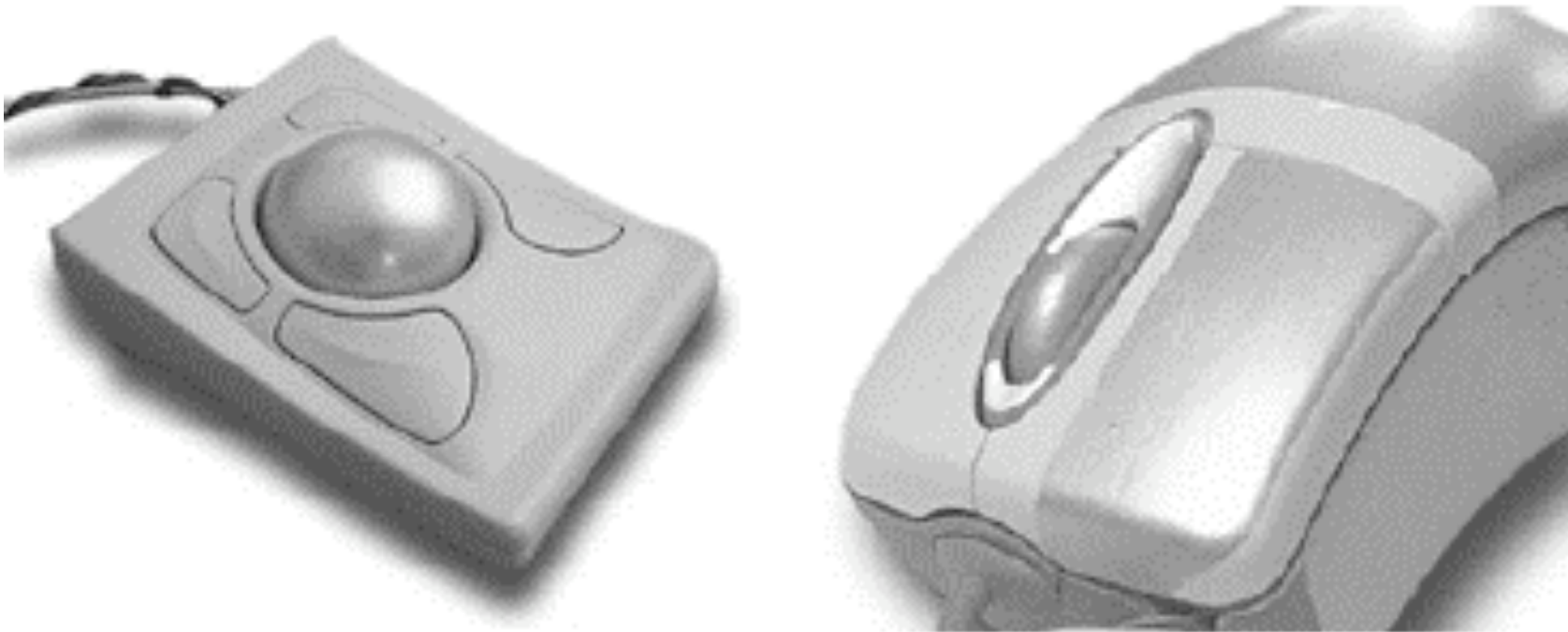


- QR Codes



Lots of other emerging possibilities

- Touch sensitivity



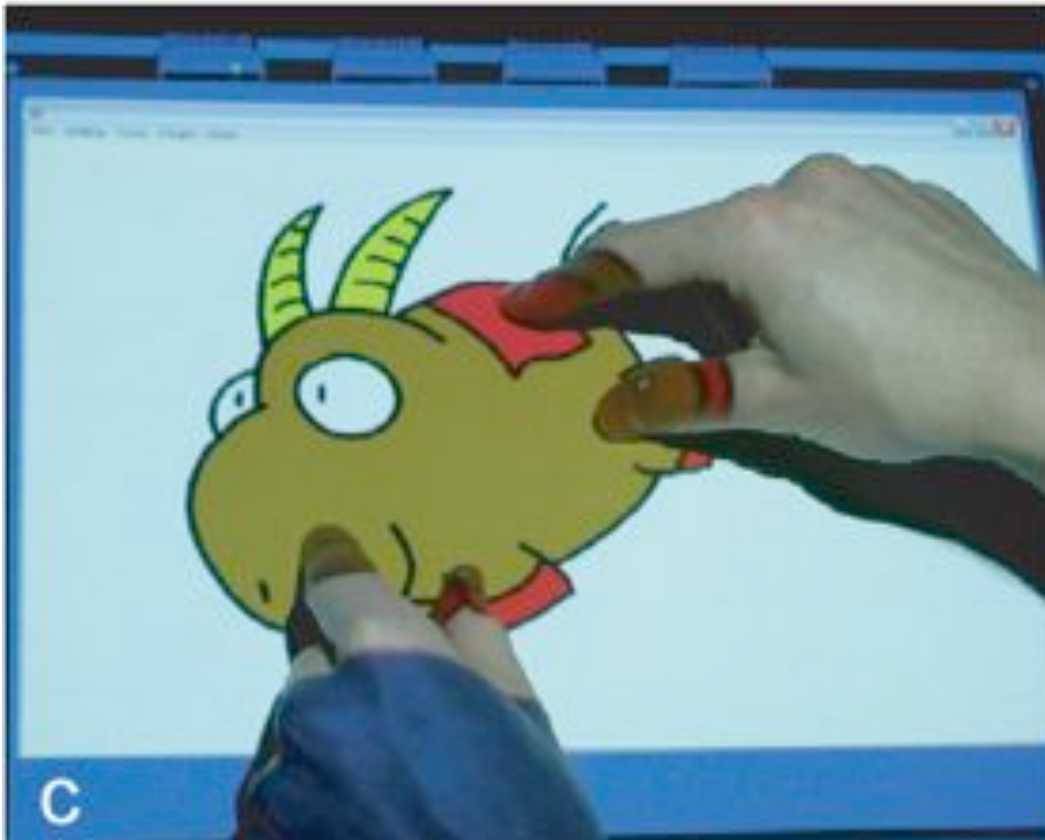
Lots of other emerging possibilities

- Speech
 - As data
 - As recognition



Lots of other emerging possibilities

- Really direct manipulation with hands
 - Uses “SmartSkin”



Lots of other emerging possibilities

- RFIDs
 - Listen Reader



Lots of other emerging possibilities

- RodDirect
 - Hack up an optical mouse, use pen as input



Lots of other emerging possibilities

- Biometrics
 - Fingerprints, weight, voice pattern, eye scan
- Smart Dust



Some Thoughts on Input Devices

- Keyboard and Mouse most prevalent
 - Path lock-in, they are good enough for most things
 - GUI research really stuck in rut from mid 80s to mid 90s
- But can we do better?
 - New situations, new domains, mobile + web tech
 - Mobile people, people with disabilities, toys
 - Cost also an important factor

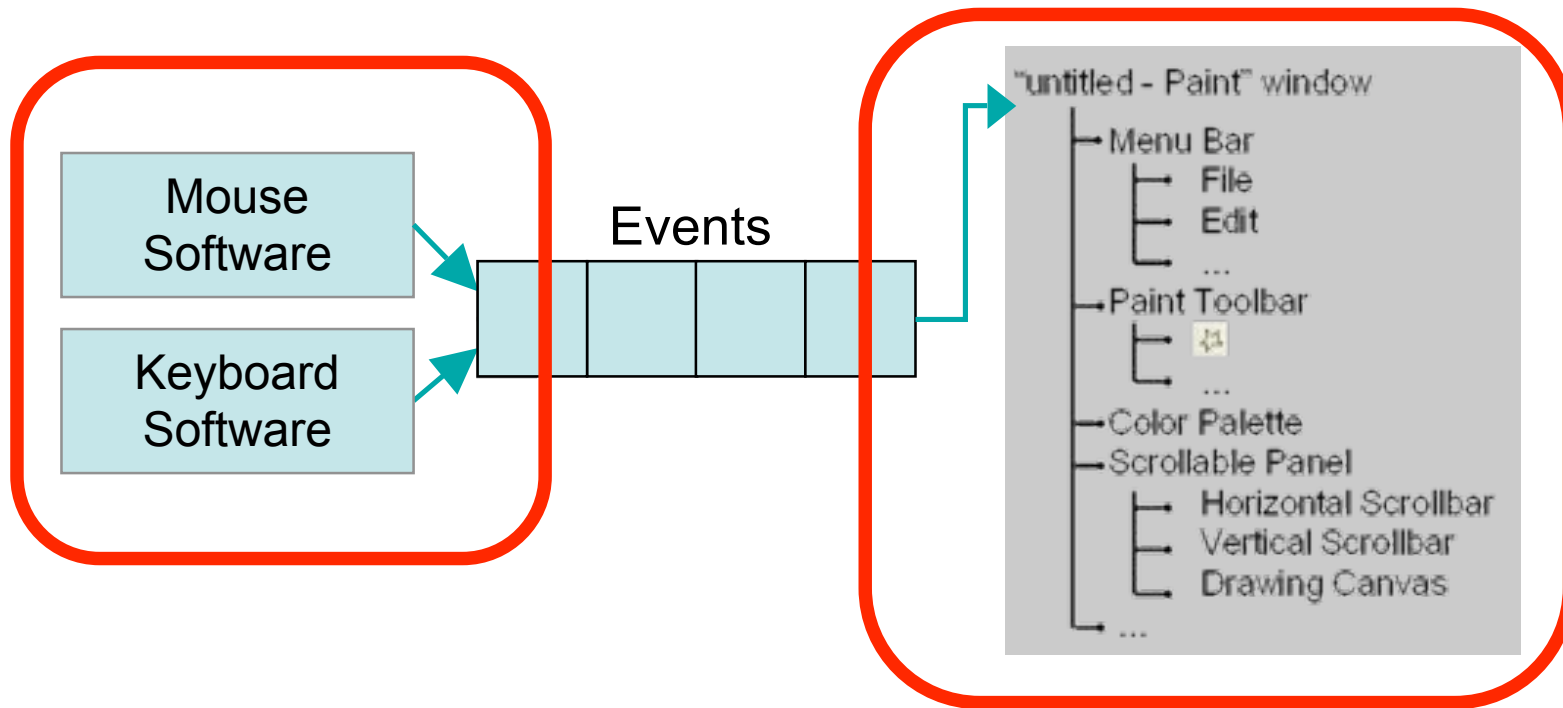


Break

- Any questions?
- HOMEWORK: Create a Facebook account, and add me as your friend.

Input is Harder than Output

- How to deal with diversity in input devices?
 - Need a higher level abstraction for input
- How to get event to right widget?



Logical Device Approach

- One approach is to use “logical devices”
 - Predefine a set of devices to support
 - Logical device characterized by its software API
 - the set of values it returns
- Old “Core Graphics” standard had 6 logical input devices
 - Valuator → returns a scalar value
 - Button → returns integer value
 - Locator → returns position on a logical view surface
 - Keyboard → returns character strings
 - Stroke → obtain input from a digitizer
 - Pick → select an object

Logical device approach

- If actual device missing, device simulated in software
 - Valuator → simulated slider
 - 3D locator → 3 knobs
- 1st step towards today's widgets
- Abstraction of logical device model good
- But... abstracts away too many details
 - some are important
 - example: mouse vs. pen on palm pilot
 - Both are locators
 - What are some differences (?)



Logical Devices not successful but..

- Still useful to think in terms of “what information is returned”
- Categorization of devices useful
 - Two broad classes emerged
 - Event devices
 - Sampled devices

Categorization of Devices

- Event Devices

- Time of input is determined by user
- Best example: button
- When activated, creates an “event record” (record of significant action)



- Sampled Devices

- Time of input is determined by the program
- Program polls for current value when it needs it
- Best example: valuator or locator
- Value is constantly updated
 - Might best think of as continuous



A Unified Model

- A way to easily program for both?
 - Rather than programming event devices one way and sampled devices another way, a way to do both?

A Unified Model: The Event model

- Model everything as events
 - Sampled devices handled as “incremental change” events
 - Each measurable change a new event with new value
- Example we’ve already seen:
 - `MouseListener` `mouseClicked()` (discrete)
 - `MouseMotionListener` `mouseMoved()` (continuous)

Simulating Sampling under Event Model

- Lots of little events is a potential problem
 - Can quickly fall behind if doing a lot of computation / redraw for every event
 - machines are fast, but can still get behind
 - Sampling provided built-in throttling
 - Would only poll data periodically
 - Whoever posts events needs to be careful here
 - Not too fast, not too slow, human perception
 - Rare you will deal with this issue however

Relevant facts

- What do we need to know about each event?
 - What
 - Where
 - When
 - Value
 - Additional Context

What Where When Value Context

- What (exactly) caused the event?
 - e.g., left mouse button went down
 - for “method based” systems this may be implicit in what handler gets called
 - Ex. `mouseMoved()` or `mousePressed()`

X-Windows defines 33 different types of events:

 **buttonPress**

 **buttonRelease**

 **keyPress**

 **keyRelease**

 **motionNotify**

 **enterNotify**


 **leaveNotify**

 **focusIn**

 **focusOut**

 **keymapNotify** (change keymap)

 **expose**

 **graphicsExpose** (source of copy not available)

 **noExpose** (source of copy is available)

 **colormapNotify**

 **propertyNotify** (some property changed)

 **visibilityNotify** (become covered)

 **resizeRequest**

 **circulateNotify** (stacking order)

 **configureNotify** (resize or move)

 **destroyNotify** (was destroyed)

 **gravityNotify** (moved due to gravity)

 **mapNotify** (became visible)

 **createNotify**

 **reparentNotify** (in diff. window)

 **unmapNotify** (invisible)

 **circulateRequest**

 **configureRequest**

 **mapRequest**

 **mappingNotify** (keyboard mapping)

 **clientMessage**

 **selectionClear** (for cut and paste)

 **selectionNotify**

 **selectionRequest**

What Where When Value Context

- Where was the primary locator (mouse) when the event happened?
 - x,y position
 - also, inside what window, what object, etc.
 - can't tell what mouse button down means without this



What Where When Value Context

- When did the event occur?
 - Stored in Event Queue until program can get to it
- Why do we need to record time?
 - If we have a queue, ordering already guaranteed(?)
 - Hint: mouse setting and keyboard setting
 - important for e.g., double-clicks and auto-repeat



What Where When Value Context

- Input value
 - e.g., ASCII value of key press
 - e.g., value of valuator
 - some inputs don't have a value
 - e.g. button press

What Where When Value Context

- Status of important buttons
 - shift, control, and other modifiers
 - possibly the mouse buttons



Events in Java (Swing & AWT)

- Subclasses of `java.util.EventObject`
 - and mostly `java.awt.AWTEvent`
 - See `java.awt.event.*` and `javax.swing.event.*`
- Each kind of event has its own class
 - A little hard to find all the parts defined in one place
 - Harder to deal with uniformly
 - But easily extensible for new event types
 - Ex. `MouseEvent`



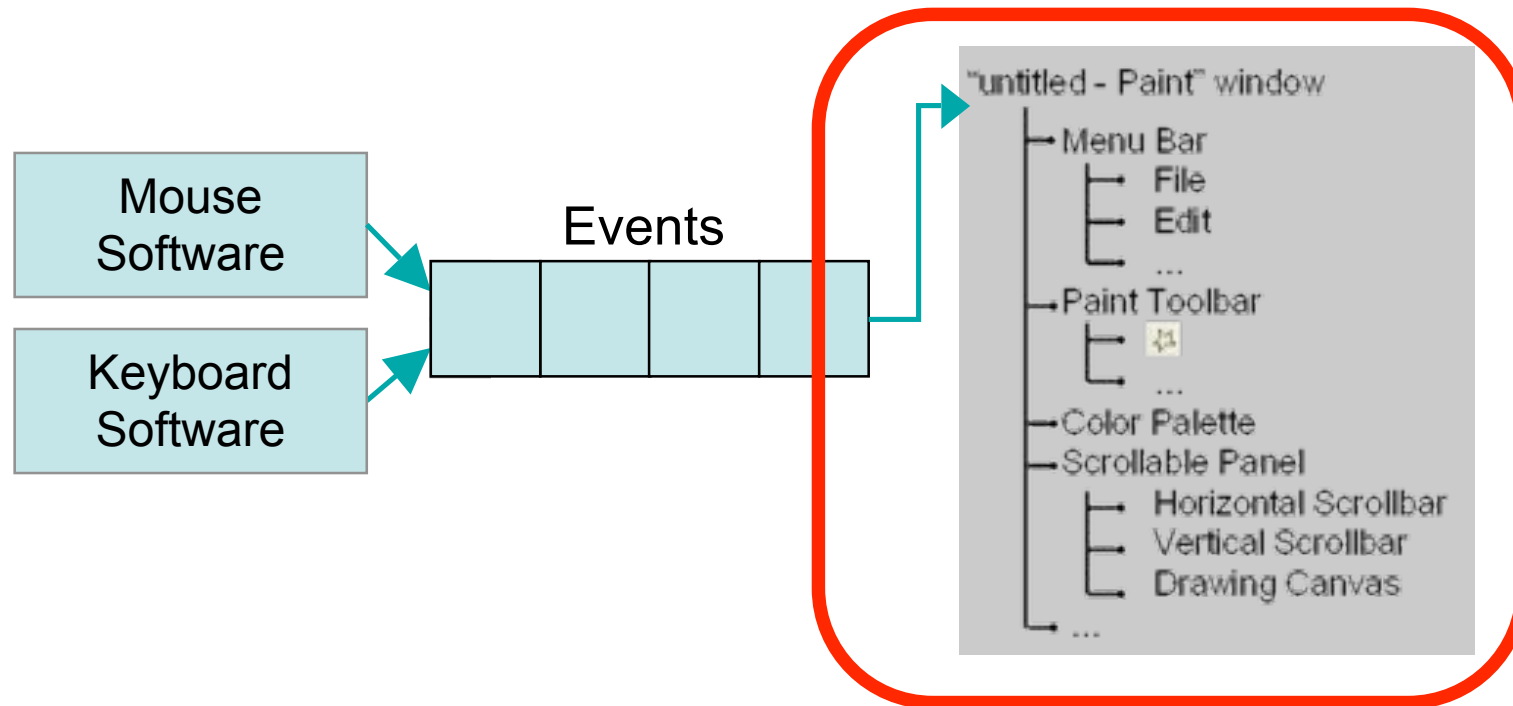
Extending the Event Model

- Events can extend past simple user inputs
 - A nice way of raising the level of abstraction
- Examples of “higher-level” events
 - window enter/exit, region enter/exit
 - system tracks mouse internally
 - See `java.awt.event.WindowListener`
 - Redraw / damage events
 - See `java.awt.event.PaintEvent`
 - Resize & window move events
 - See `java.awt.event.ComponentListener`

“Artificial” Events in Java

- Added to the event queue like user events
 - `java.awt.event.FocusEvent`
 - Window becomes or loses focus
 - `java.awt.event.WindowEvent`
 - Window being closed, iconified, etc.
- Why do it this way?
 - **Simple** programming model, highly consistent
 - Almost everything in GUI is an event
 - Extend your GUI via Listeners

Roadmap



Using Events from an Event Queue

- What object(s) gets the event?

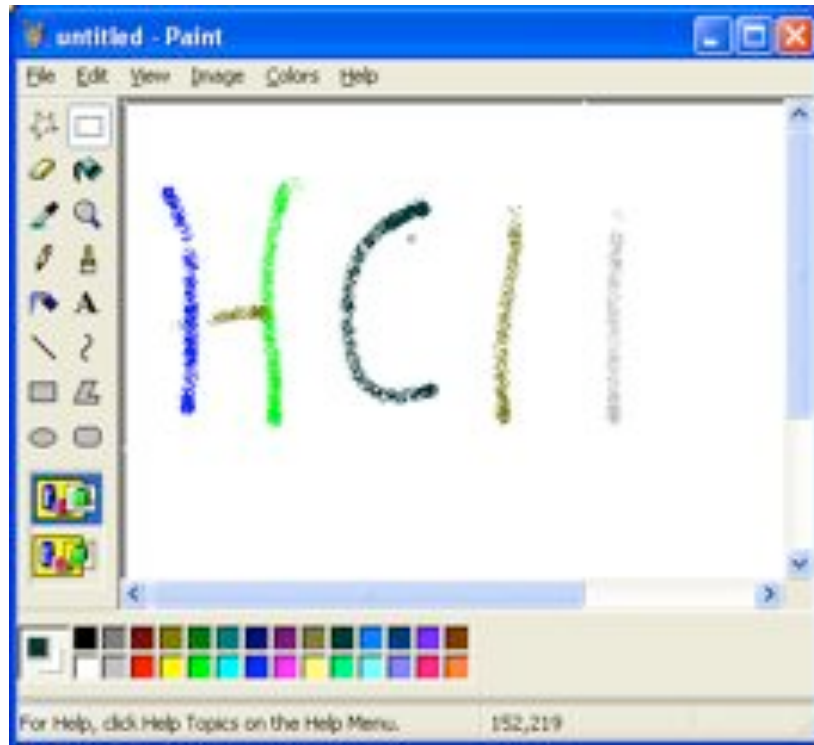


Dispatch Strategies


What object(s) gets the event?

- “Bottom first” dispatch strategy
 - lowest object in interactor tree that overlaps the position in event gets it

Bottom-First Dispatch Strategy



"untitled - Paint" window

- Menu Bar
 - File
 - Edit
 - ...
- Paint Toolbar
 - 
 - ...
- Color Palette
- Scrollable Panel
 - Horizontal Scrollbar
 - Vertical Scrollbar
 - Drawing Canvas
- ...

Dispatch Strategies

What object(s) gets the event?

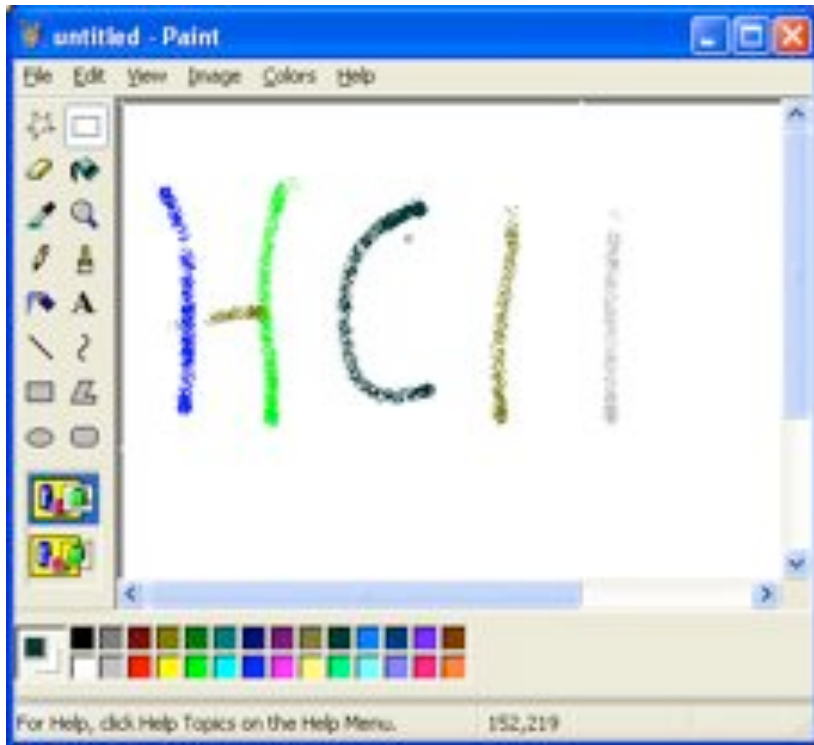
- “Bottom first” dispatch strategy
- If that widget doesn’t handle event, then:
 - Ignore the event
 - Pass the event up to its parent
- Technically, in Java, event always handled
 - Once a widget gets an event, widget handles it, and then forwards event to listeners
 - Why this approach?
 - Why not forward to listeners directly(?)
 - Java events don’t go back up to parent

Dispatch Strategies

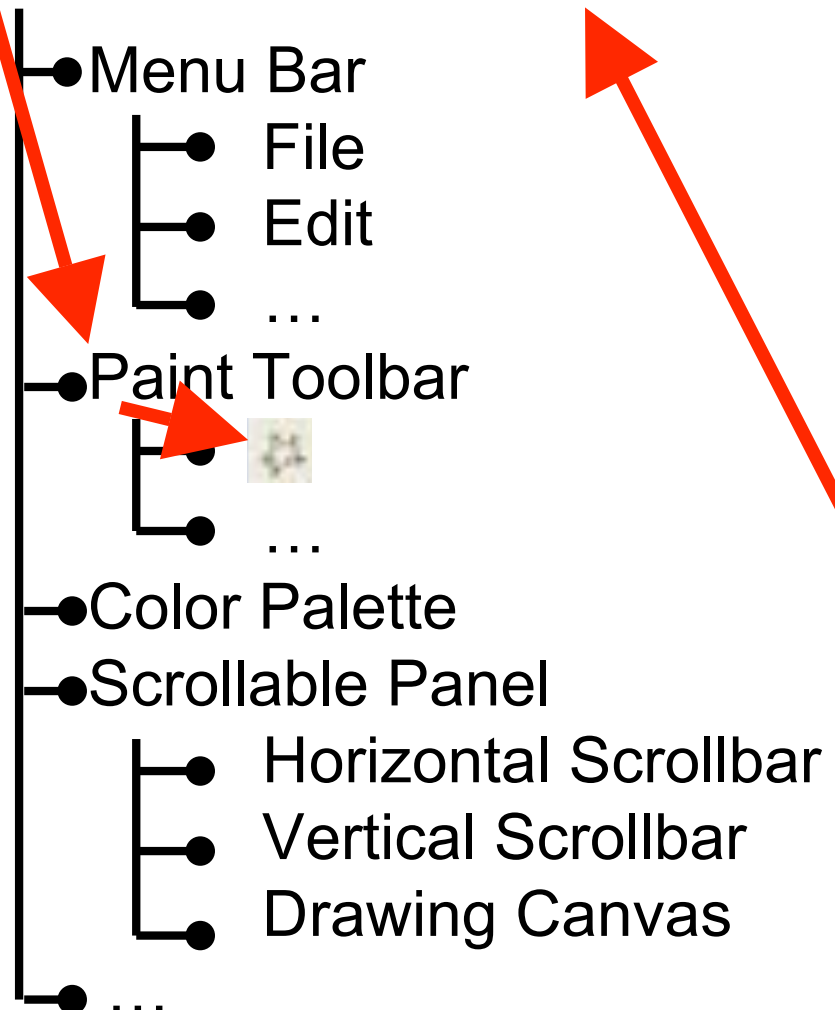
What object(s) gets the event?

- Can also do “top-down”
 - Root of tree gets it first, can act on or modify event
 - If not handled, then gives to right child
 - Root has another chance to act on it if child (and its children) doesn't handle it

Top-Down Dispatch Strategy



"untitled - Paint" window



Dispatch Strategies

What object(s) gets the event?

- Why Top-Down?
- Easy to impose high-level control
 - Ex. No scrolling anywhere
- Did this for gesture recognition
 - Parent figures out gesture, and then figures out who to dispatch to



Two Major Ways to Dispatch Events

- Positional dispatch
 - Event goes to an object based on position of the event
- Focus-based dispatch
 - Event goes to a designated object (the current focus) no matter where the mouse is pointing
- Q: Would mouse move events be done by positional or focus dispatch(?)

Discuss 2 Minutes

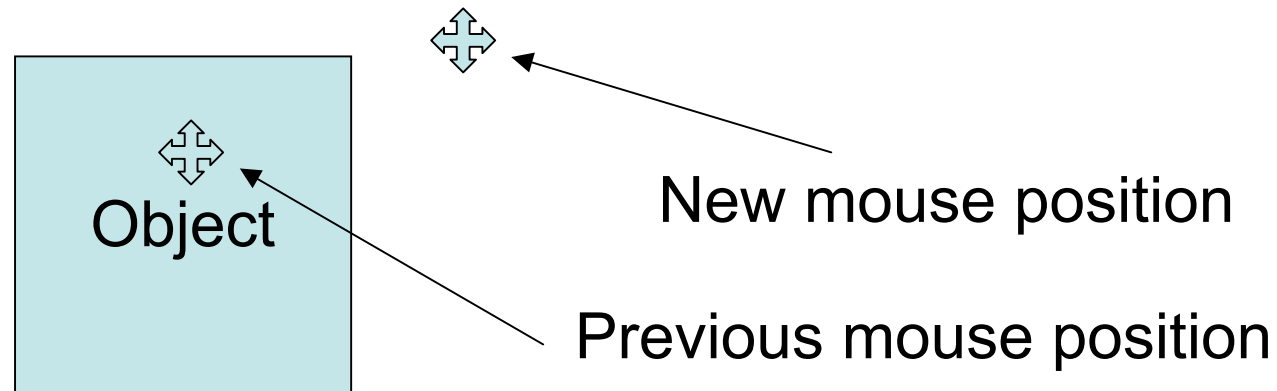
**(Consider painting vs
dragging an object)**

Question & Answer

- Q: Would mouse move events be done by focus or positional dispatch?
- A: It depends...
 - painting: use positional
 - dragging an object: need focus (why?)

Dragging Needs Focus Dispatch

- Why? What if we have a big jump?



- Cursor now outside object and won't get next event!
 - Think scrolling as well, too easy to move outside of elevator

Positional and focus based dispatch

- Will need both at different times

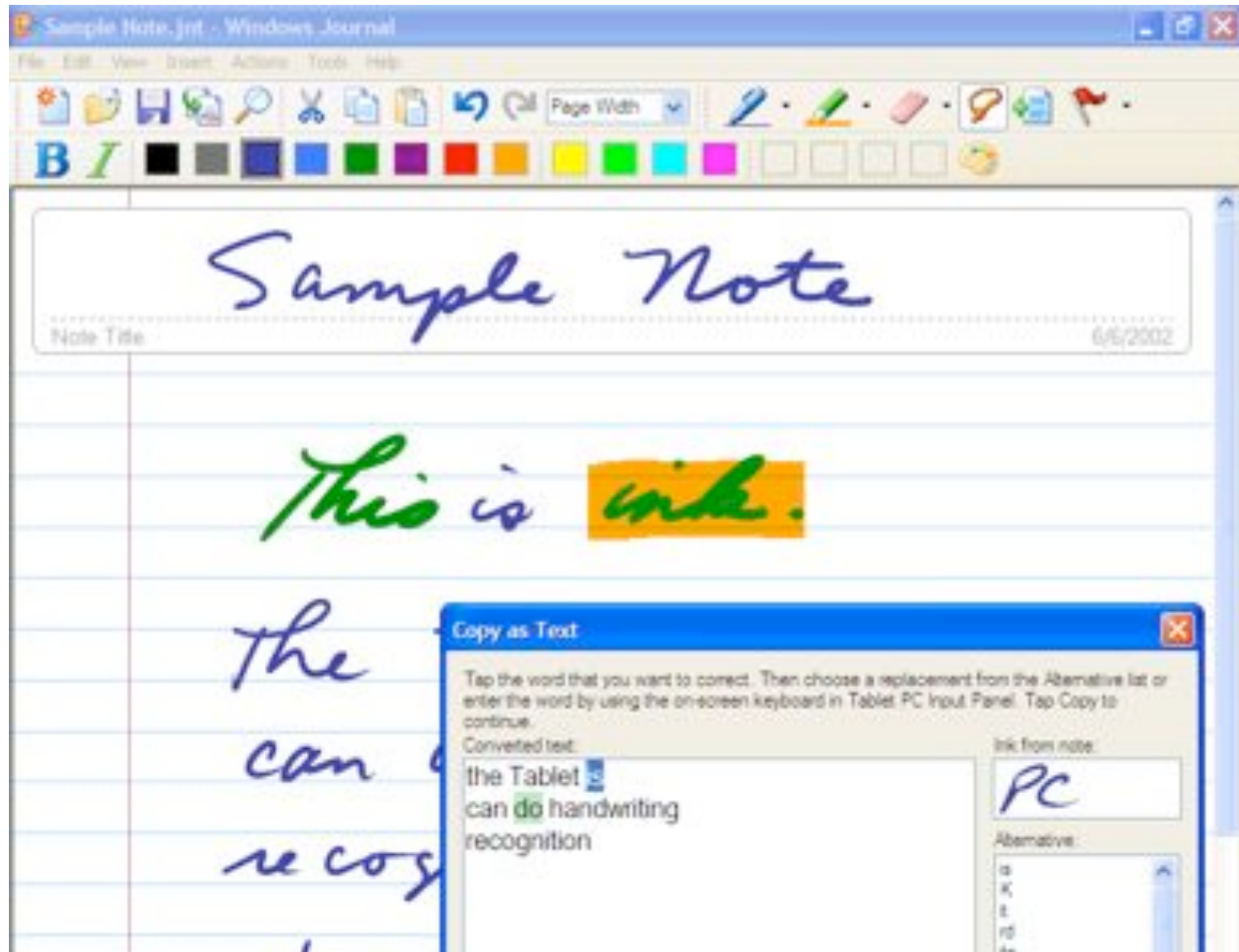
Summary

- Lots of input hardware
- Devices
 - Logical: Valuator, Locator, Button, etc
 - Event / Sampled
- Event model to unify Event and Sampled Devices
- What events look like
- Dispatch Strategies
 - Bottom-first and top-down

Implications of Queued Events

- We are really operating on events from the past
- But sampled input is from the present
 - mixing them can cause problems
 - e.g. inaccurate position at end of drag
 - Need to sample fast enough
 - Need to process events quickly, don't block the event queue
- May also be useful to coalesce continuous events
 - Multiple mouse motions joined into a single mouse motion
 - Coalescing may hurt in certain situations though...(?)

Implications of Queued Events



Positional dispatch

- If dispatching positionally, need a way to tell what object(s) are “under” a location
 - This is called “Picking”
- Probably don’t want to pick on the basis of a point (single pixel)
 - Why?

Positional dispatch

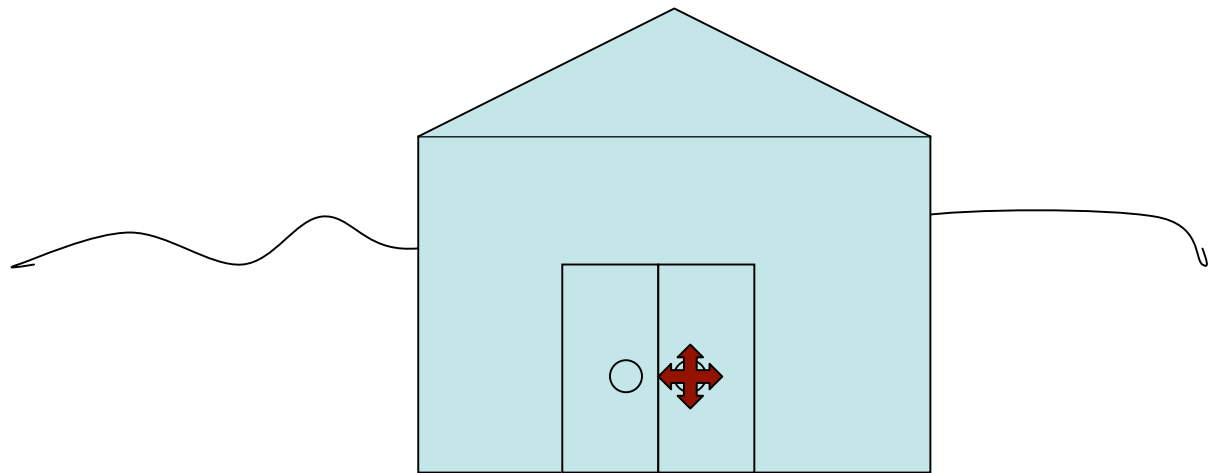
- If dispatching positionally, need a way to tell what object(s) are “under” a location
 - This is called “Picking”
- Probably don’t want to pick on the basis of a point (single pixel)
 - Why?
 - Because it requires a lot of accuracy
- Instead may want to pick anything within a small region around the cursor

Implementing Pick

- Possible to apply a clipping algorithm
 - small clip region around cursor
 - pick anything that is not completely clipped away
- Better is a recursive “pick traversal”
 - Walk down the object tree
 - Each object does a local test customized to its shape (and semantics)
 - Also tests its children recursively

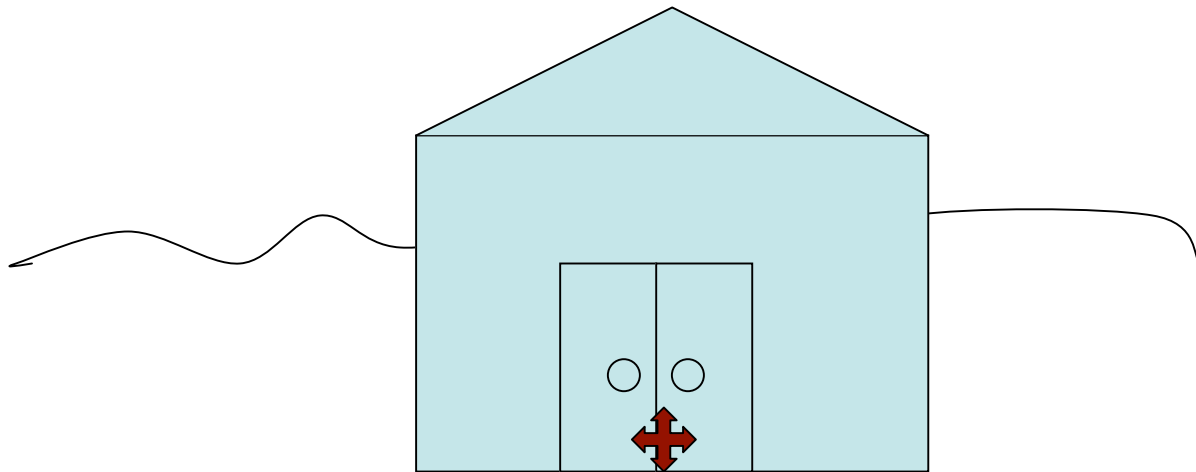
Pick ambiguity

- Classic problem, what if multiple things picked?
 - Two types
 - (1) Hierarchical ambiguity
 - are we picking the door knob, the door, the house, or the neighborhood?



Pick ambiguity

- (2) Spatial ambiguity
 - Which door are we picking?



Solutions for pick ambiguity

- No “silver bullet”, but two possible solutions
- (1) “Strong typing” (use dialog state)
 - Not all kinds of objects make sense to pick at a given time
 - Turn off “pickability” for unacceptable objects
 - Reject pick during traversal

Solutions for pick ambiguity

- (2) Get the user involved
 - direct choice
 - typically slow and tedious
 - pick one, but let the user reject it and/or easily back out of it
 - often better
 - feedback is critical
 - need a way to get at the others