Input
(Devices and Models)

Administration

- Assignment 3 (prelim) due today
- Assignment 4 set today

Where we are…

- Two largest aspects of building interactive systems: output and input
  - Have looked at basics of output
  - Now look at input

Input

- Generally, input is somewhat harder than output
  - Less uniformity, more of a moving target
  - More affected by human properties
  - Not as mature
- Will start with low level (devices) and work up to higher level

Input devices

- Keyboard
  - Ubiquitous, but somewhat boring…
  - Quite mature design
- QWERTY key layout
  - Alternatives?

QWERTY key layout

Christopher Sholes 1868

- **Originally designed to spread out likely adjacent key presses to overcome jamming problem of very early mechanical typewriters**
  - **Often quoted as “intentionally slowing down” typing, but that’s not true**

QWERTY keyboard layout

- Other layouts have been proposed
  - Dvorak is best known
  - Widely seen as better
  - Experimental and theoretical evidence casts doubt on this
    - (Is only a little better)
    - 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**
  - **Lesson: once there is sufficient critical mass for a standard it is nearly impossible to dislodge (even if there is an apparently good reason to)**

Economic phenomenon of “lock-in” or “path dependence”

Buttons

- Similar to keyboard, but not for typing letters
  - separate collection of keys with typically same form but different purpose
  - now seen as “function keys that come standard w/ keyboards
  - also show up on e.g., mouse

Ivan Sutherland, Sketchpad, 1963

Buttons

- Buttons often bound to particular commands
  - e.g., function keys
  - Improved quite a bit with labels
  - Software changeable labels would be ideal, but we don’t typically get this

Valuators (e.g. Sliders)

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

Locators (AKA pointing devices)

- Returns a location (point)
  - two values in ranges
  - usually screen position
- Examples
  - Mice (current defacto standard)
  - Track balls, joysticks, tablets, touch panels, etc.

Locators

- Two major categories:
  - Absolute vs. Relative locators

Absolute locators

- One-to-one mapping from device movement to input
  - e.g., tablet
  - Faster
  - Easier to develop motor skills
  - Doesn’t scale past fixed distances
    - bounded input range
  - less accurate (for same range of physical movement)

Relative locators

- Relative or incremental mapping
- E.g., maps movement into rate of change of input
  - e.g., joystick (or TrackPoint)

Relative locators

- –More accurate
  (for same range of movement)
- –Harder to develop motor skills
- –Not bounded
  (can handle infinite moves)

Q: is a mouse a relative or absolute locator?

(Ignore mouse acceleration for a moment)

Invented by Douglas Engelbart et al. ~1967

http://sloan.stanford.edu/MouseSite/Archive/AugmentingHumanIntellect62/Display1967.html

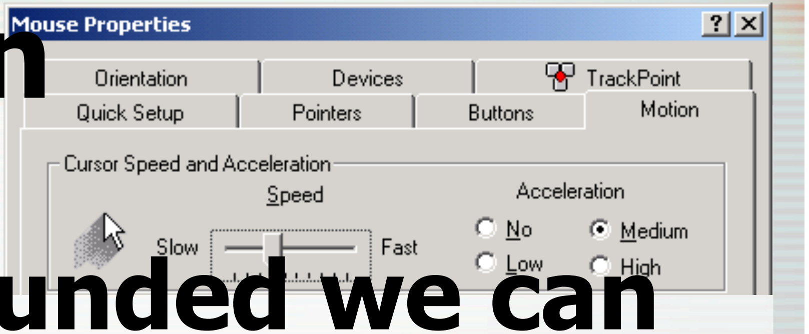
Q: is a mouse a relative or absolute locator?

- **Answer: Neither**
- **Third major type: "Clutched absolute"**
  - **Within a range it's absolute**
  - **Can disengage movement (pick it up) to extend beyond range**
    - **picking up == clutch mechanism**

Clutched absolute locators

- Very 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



Cursor Speed and Acceleration

Speed

Acceleration

○ No ● Medium

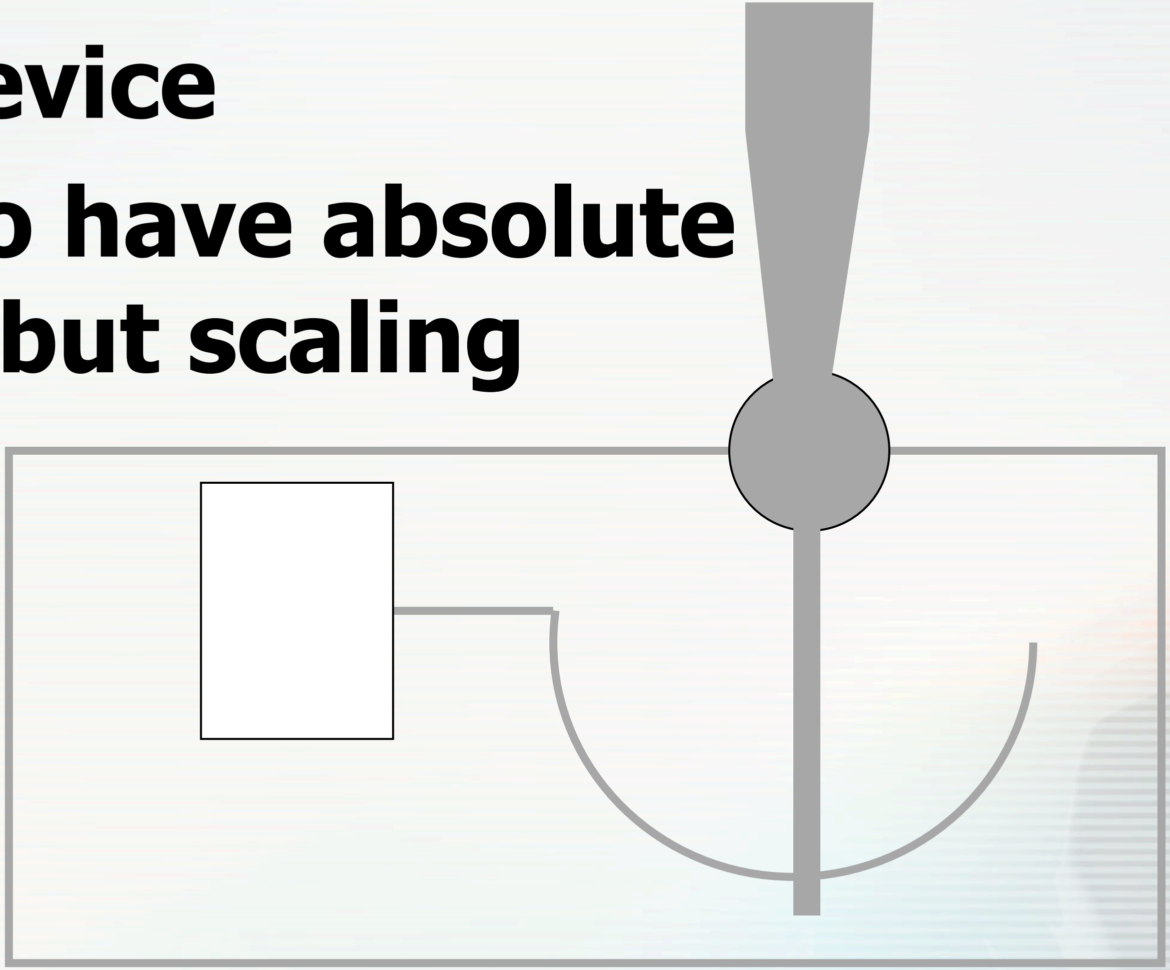
○ Low ○ High

- 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

Where we need precision

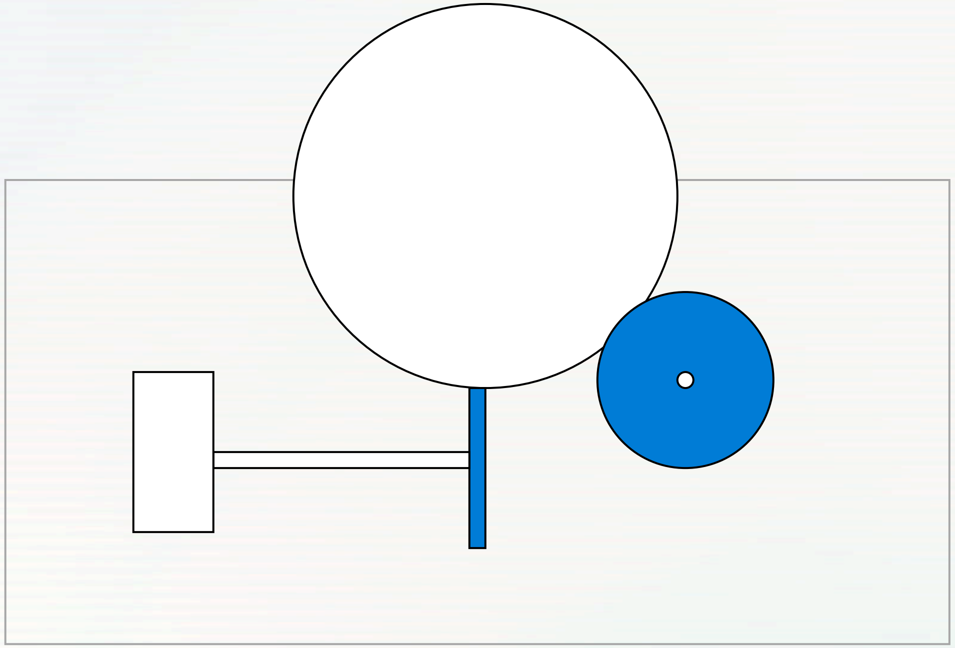
Device specifics: joysticks

- –self centering
- –relative device
- –possible to have absolute joysticks, but scaling is bad

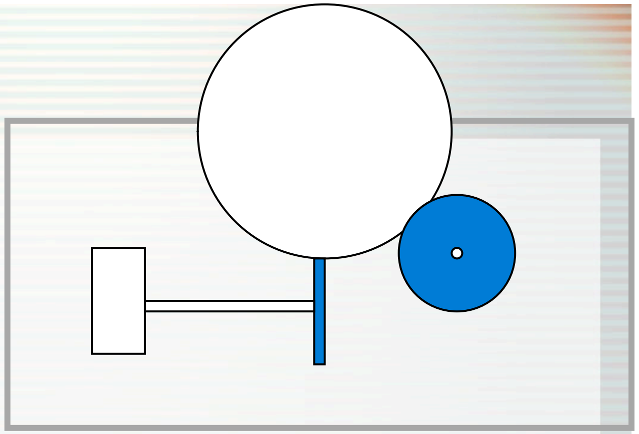


Trackballs

- (Typically large) ball which rolls over 2 wheels



Trackballs



- Clutched absolute
  - but with small movement range
- Infinite input range, etc.
- Properties vary quite a bit
  - scaling of movements
  - mass of ball

Touch panel

- **What kind of a device?**

Touch panel / Tablet

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

3D locators

- Can extend locators to 3 inputs
- Some fun older devices
  - 3D acoustic tablet
  - Wand on reels
  - Multi-axis joystick

Lots of other devices

- Still mostly KB + mouse, but increasing diversity
  - Cameras!
    - Lots of untapped potential in vision
  - Microphones
    - speech as data
    - speech recognition

Lots of other devices

- Any favorites?

Some interesting ones

- **Thumb Wheel**
- **DataGlove**
- **Motion detectors (and other sensors)**
- **Accelerometers**

- **Identification techniques**
  - **Magic apples**

Using identification as input

- If you have object identification and a network, you can appear to attach arbitrary amounts of information to an object with just identification
  - Use ID to store retrieve data in DB
    - 64 bit ID will be unique
    - 96 or 128 bits better (don’t need more)
- Make assoc in place A, carry to place B, identify, get data over network
  ⇒ data appears to have moved w/ obj

Specific identification technologies

- RFID tags
- Bar codes
- Fingerprint readers
- Sub-gram resolution scales
- Speech
- Handwriting
- Walking

Sun Microsystems Starfire

- 1994-5, Vision of 2004
- Many different interaction styles
- CHI'94 paper, video prototyping
- Book
- Apologies for stereotypes

Starfire Video

- http://www.idemployee.id.tue.nl/g.w.m.rauterberg/videos.html

Prototyping These Visions

- Styles of input/output?
- Differences/similarities with 1987 Knowledge Navigator

- **Break – 15 minutes**

Input
(Part 2: Input Models)

Dealing with diversity

- Saw lots of diversity in devices
  - actual details of devices (e.g., device drivers) is a real pain
  - how do we deal with the diversity?
- Need a model (abstraction) for input
  - like file systems abstract disks
  - higher level & device independent

Input Models are Complex

- **"One of the most complex aspects of Xlib programming is designing the event loop, which must take into account all of the possible events that can occur in a window."**

  -- Nye & O'Reilly X Toolkit Intrinsics Programming Manual, vol. 4, 1990, p. 241.

- **"The dispatching and handling of events is rather complicated."**

  -- Galaxy Reference Manual, v1.2, p. 20-5.

Logical device approach

- One approach: “logical devices”
  - A logical device is characterized by its software interface (only)
    - the set of values it returns
  - Rest of semantics (how it operates) fixed by category of device or left to the particular device

Logical device approach

- Fixed set of categories
  - old “Core Graphics” standard had 6
    - keyboard, locator, valuator, button
    - pick, stroke
- If actual device is missing, device is simulated in software
  - valuator => simulated slider
  - 3D locator => 3 knobs
- 1st step towards today’s widgets

Logical device approach

- Abstraction provided by logical device model is good
- But… abstracts away too many details (some are important)
  - example:
    mouse vs. pen on palm pilot
    - Both are locators
    - What's the big difference?

Not a success 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 the user
    - Best example: button
    - When activated, creates an "event record" (record of significant action)

Categorization of devices

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

A unified model

- Anybody see a way to do both major types of devices in one model?

A unified model: the event model

- Model everything as events
  - Sampled devices are handled with “incremental change” events
  - Each measurable change in value produces an event containing the new value
  - Program can keep track of the current value if it wants to sample

Simulating sampling under the event model of input

- Can cause problems
  - lots of little events
    - Can fall behind if doing a lot of computation/redraw for every event
      - machines are fast, blah blah blah
      - but can get behind (sampling provided built in throttling)

The event input model

- Almost all systems now use this
- An “event” is an indication that “something potentially significant” has just happened
  - in our case user action on input device
  - but, can be generalized

The event input model

- “Event records” are data structures that record relevant facts about an event
  - generally just called “events”
- Event records often passed to an “event handler” routine
  - sometimes (e.g., Flex) just encode relevant facts in parameters instead of event record

Relevant facts

- What do we need to know about each event?

Relevant facts

- What
- Where
- When
- Value
- Additional Context

What

- 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

X-Windows defines 33 types of events:

1. buttonPress
2. buttonRelease
3. keyPress
4. keyRelease
5. motionNotify
6. enterNotify
7. leaveNotify
8. focusIn
9. focusOut
10. keymapNotify (change keymap)
11. expose
12. graphicsExpose
13. noExpose
14. colormapNotify
15. propertyNotify
16. visibilityNotify (become covered)
17. resizeRequest
18. circulateNotify
19. configureNotify
20. destroyNotify
21. gravityNotify
22. mapNotify (became visible)
23. createNotify
24. reparentNotify
25. unmapNotify (invisible)
26. circulateRequest
27. configureRequest
28. mapRequest
29. mappingNotify (kbd mapping)
30. clientMessage
31. selectionClear (cut & paste)
32. selectionNotify
33. selectionRequest

Where

- Where was the primary locator (mouse) when event happened
  - x,y position
  - also, inside what window, object, etc.
  - this is specific to GUIs, but its critical
    - e.g., can’t tell what mouse button down means without this

When

- When did the event occur
  - Typically are dealing with events from the (hopefully recent) past
    - queued until program can get to them
  - In absolute time or relative to some start point
  - Hopefully at resolution of 10s of ms
    - important for e.g., double-clicks

Value

- 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

Additional context

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

Extending the event model

- **Events can extend past simple user inputs**
  - **Extra processing of raw events to get "higher level" events**
    - **window / object enter & exit**
  - **Can extend to other "things of significance"**
    - **arrival of network traffic**
    - **Low battery**
- **Generally event is a notification of the occurrence of a significant event and its convenient to use that abstraction**

Extending the event model

- Window systems typically introduce a number of events
  - window enter/exit   region enter/exit
    - system tracks mouse internally so code acts only at significant points
  - Redraw / damage events
  - Resize & window move events

Synchronization and events

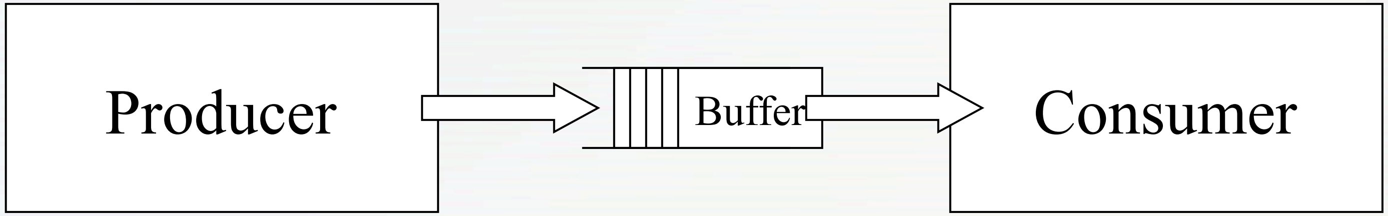
- The user and the system inherently operate in parallel
  - asynchronously
- This is a producer consumer problem
  - user produces events
  - system consumes them

Synchronization and events

- Need to deal with asynchrony
  - both parties need to operate when they can
  - but can’t apply concurrency control techniques to people
- How do we handle this?

Synchronization and events

- Use a queue (buffer) between



  - As long as buffer doesn't overflow, producer does not need to block
  - Consumer operates on events when it can

Implications of queued events

- We are really operating on events from the past
  - hopefully the recent past
- But sampled input is from the present
  - mixing them can cause problems
  - e.g. inaccurate position at end of drag

Using events from an event queue

- Two big questions:
  - What object(s) gets the event?
  - What does it do with it?
    - Interpret it based on what the event is, what the object is, and what state the object is in

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

Question

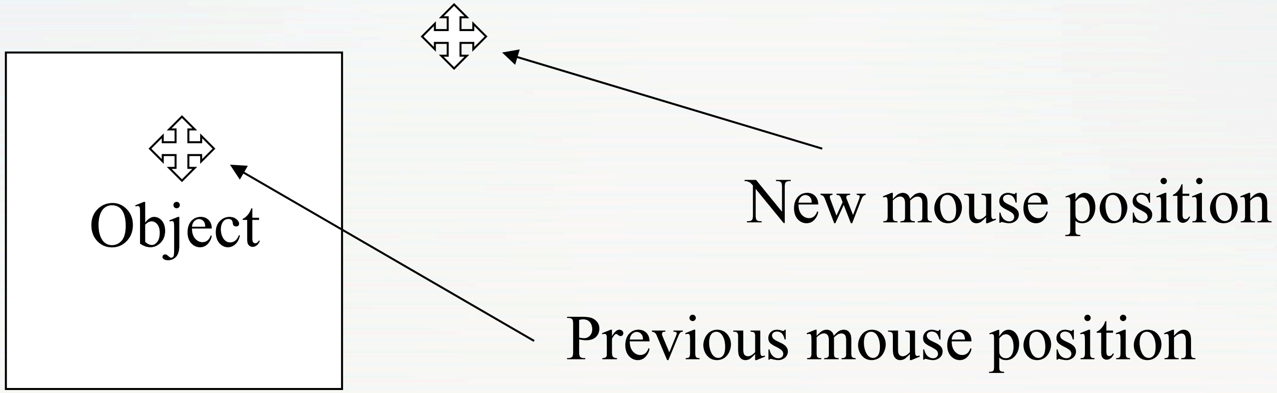
- Would mouse move events be done by focus or positional dispatch?

Question & answer

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

Dragging an object needs focus dispatch

- Why? What if we have a big jump?



- Cursor now outside the object and it doesn’t get the next event!

Positional and focus based dispatch

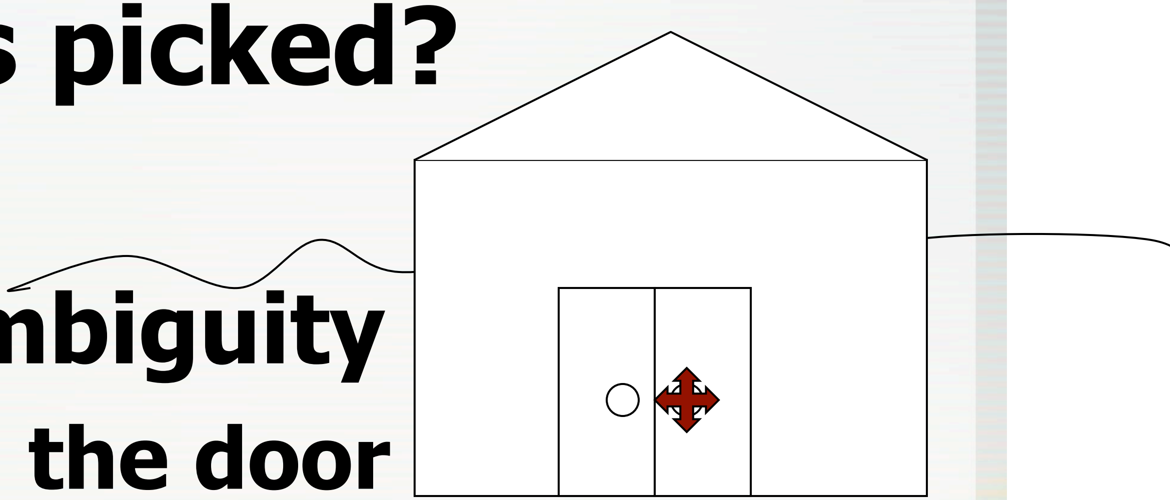
- Will need both
- Will need a flexible way to decide which one is right
  - sometimes we need one, sometimes another

Positional dispatch

- If we are dispatching positionally, need a way to tell what object(s) are “under” a location
- “Picking”

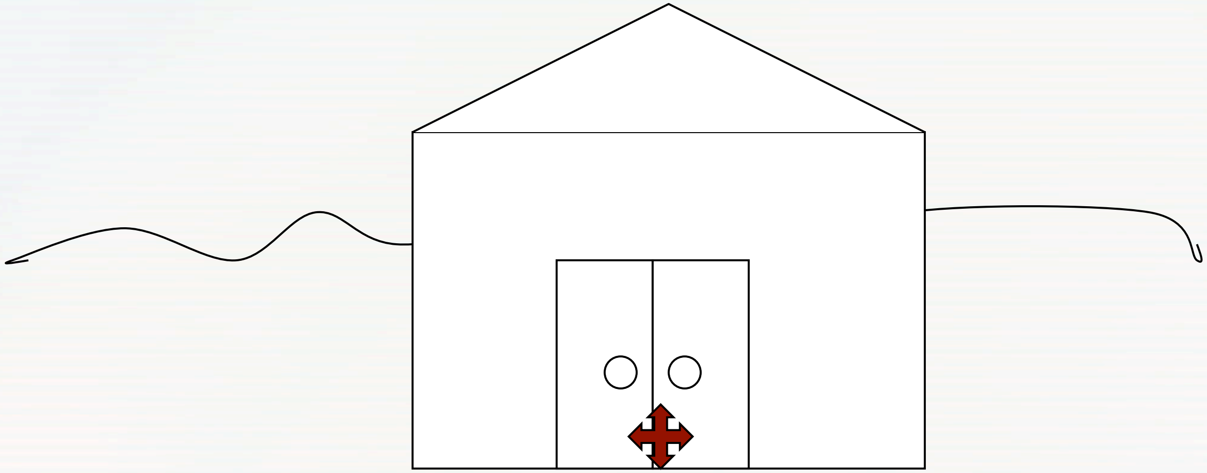
Pick ambiguity

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



Pick ambiguity

- Spatial ambiguity
  - Which door are we picking?



Solutions for pick ambiguity

- No “silver bullet”, but two possible solutions
  - “Strong typing” (use dialog state)
    - Turn off “pickability” for unacceptable objects
      - » reject pick during traversal

Solutions for pick ambiguity

- 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

Input Summary

- Lots of variety in input devices
- Event model is good abstraction
- Issues
  - How to support user asynchrony
  - Who is each event dispatched to
  - What is done with the event

Questions?