Touch Interaction and Touch Gestures

Georgia Tech

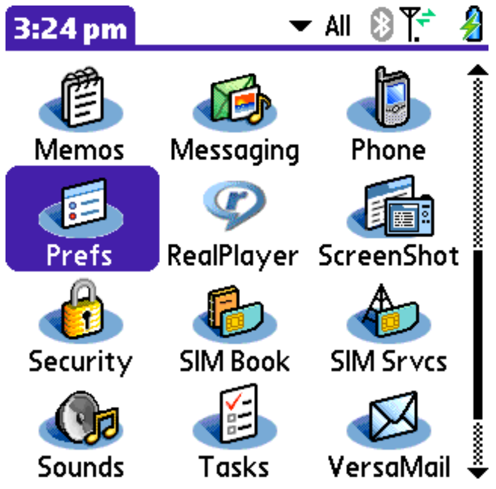
Types of Touch

- All have very different interaction properties:
  - Single touch
  - Multitouch: multiple fingers on the same hand
  - Multihand: multiple fingers on different hands
  - Multiperson: multiple hands from multiple people! (Collaborative multitouch)

Single-Touch Interaction

Single finger touch gestures

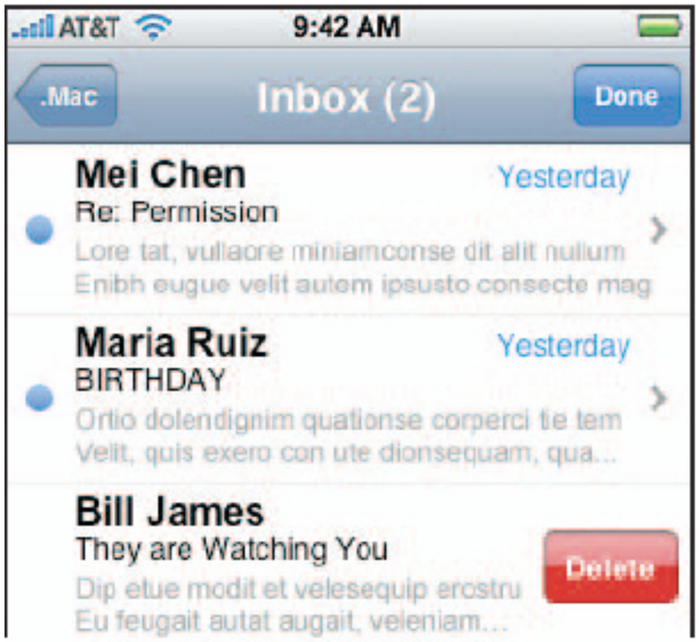
- Typically inputs used for *command* input, not *content* input
- Most common: press/tap for selection
  - Not really much of a "gesture" at all
- Slightly more complex:
  - Double-tap to select
  - Double tap, hold, and drag to move windows on OS X
  - Tap, hold and drag to select text on the iPad
- Note: some of these don't require a screen, just a touchable surface



Other examples

- One-finger:
  - Special interactions on lists, etc.
  - Example: swipe over mail message to delete
  - Specialized feedback for confirmation
  - Still no good affordances though.

- Non-finger gestures?
  - Surface--use edge of hand for special controls
  - Technically "single touch," although most hardware that can support this is probably multitouch capable


AT&T
9:42 AM
.Mac
Inbox (2)
Done
Mei Chen
Yesterday
Re: Permission
Lore tat, vullaore miniamconse dit alit nullum
Enibh eugue velit autem ipsusto consecte mag
Maria Ruiz
Yesterday
BIRTHDAY
Ortio dolendignim quationse corperci tie tem
Velit, quis exero con ute dionsequam, qua...
Bill James
They are Watching You
Dip etue modit et velesequip erostru
Eu feugait autat augait, veleniam...
Delete

Multi-Touch Interaction

Multitouch Gestures

- Multitouch: responsiveness to multiple points of input, not just a single point.
  - Extra hardware required!
  - E.g., Many single-touch systems will simply average multiple points of input.
- Allows a much richer and expressive vocabulary of gestures
  - Multiple fingers on the same hand
  - Multiple fingers of different hands
  - Multiple fingers by different people (when using table-scale or wall-scale devices, typically)
- For this section, we're going to mainly be talking about multiple fingers on the same hand.

Example multitouch gestures

- Two-finger:
  - Scale: pinch, expand two fingers
  - Rotate: two points lets you do intuitive rotation
  - Scroll window on OS X
- Three-finger:
  - Three-finger swipe: advance forward, backward (in web browser, photo browser, etc.)
- Four-finger:
  - Task management--swipe left and right to bring up task manager, up and down to hide/ show all windows on OS X
  - Swipe up to bring up multitasking controls on iPad; swipe left/right to change apps
- Five-finger
  - Five-finger pinch to return to homescreen on iPad
- Note: some of these may be used without a touchscreen (e.g., directly on a multitouch trackpad)

Pros and cons of many of these?

- Advantages of multitouch
  - Expressiveness: In many cases, *very* natural interaction that's a close map to what we do in the "real world"
    - E.g., two fingered rotation
  - Parallelism: Allows for more *degrees of freedom: higher dimensionality input, but in a very natural way.*
    - Two-fingered rotation: specifies amount of rotation, pivot point, all in one simple gesture; can combine with scaling very naturally.
- Chief disadvantage of multitouch:
  - Poor/nonexistent affordances in many cases
    - How do you know what you *can* do?
    - Depends on education (reading a manual, or contextual help, or suggestions) for people to have access to these.
- Lots of interesting work to be done in defining interaction techniques in multitouch--**better affordances, feedback, specific techniques for accomplishing specific tasks** (we'll see some when we talk multi-hand)

Two-Handed Interaction and Magic Lenses/Toolglasses

Spot the difference between these examples and GUIs

- A student turns a page of a book while taking notes
- A driver changes gears while steering a car
- A recording engineer fades out the drums while bringing up the strings

- [Examples ref. Buxton]

Quick Motivation

- The desktop paradigm does not demand much (physically) of its user.
- Then again, it doesn't take advantage of the physical abilities of the user either.
- Many tasks are handled more easily with multiple hands.

Two-handed (Bi-manual) Interaction

- Potential advantages:
  - Expressiveness: do things in a more natural way, use hands the way we use them in the real world
    - E.g., one finger in a book to hold its place, while thumbing through other pages
  - Parallelism: multiple actions at the same time. **Need to be coordinated, though, to prevent cognitive burden!**

- E.g., there's a reason we don't use two mice at the same time!
- Also need to understand relative roles of dominant/non-dominant hands

Two-handed (Bi-manual) Interaction

- Some examples:
- Simplest case today: two hands on a keyboard...
  - **Independent** actions from both hands (hitting keys)
  - Only coordinated in time; but each hand interacts with distinct keys
  - Also: controlling sliders on a mixing board, playing the violin

Two-handed (Bi-manual) Interaction

- In the "real world," though, most often hands are working **cooperatively**--working together to accomplish a task. Two forms:
  - **Symmetric**. Inputs perform similar but independent actions to accomplish the same task.
    - Examples: positioning line endpoints or rectangle bounds on a screen, stretching a rubber band.
  - **Asymmetric**. Inputs play complementary but disparate roles; one inputs role must be performed in order for the other input to perform its role (also called *compound* motion).
    - Examples: opening a jar (the hand grasping the lid can't perform its role of rotation unless the non-dominant hand holds the jar in place). Also: drawing/drafting, lab work, surgeons, dentists, etc...

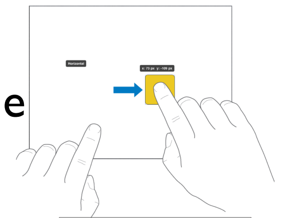
Kinematic Chain Theory

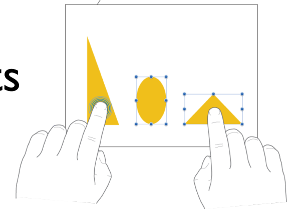
- Most of this discussion is out of scope for the class, but KCT describes how dominant and non-dominant hands work together in asymmetric cooperative action

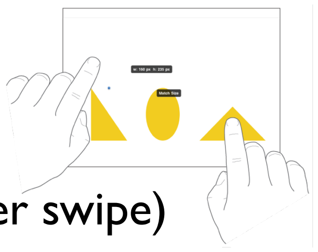
- Non-dominant hand provides the frame of reference (e.g., moving the drawing paper to the dominant hand)
- Dominant hand acts at a finer spatial-temporal scale (smaller, quicker motions) than the non-dominant hand (larger, coarse-grained motions)
- Non-dominant hand initiates the action, dominant hand terminates it

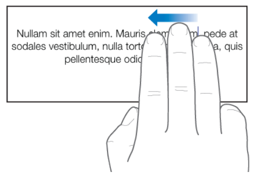
Some iPad Examples (from Keynote)

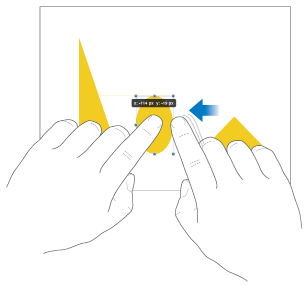
- “Normal” multitouch systems can support multi-hand input (if they’re large enough, and stably positioned of course)
- Constrained Drag: touch and hold one finger anywhere on screen while an object with the other hand; constrains movement to a perfectly straight line
- Multi-select: tap and hold one object to select it, then tap other objects with another finger
- Match sizes: hold one object while you resize another one; snaps to size of held object
- Move text insertion point by word (two finger swipe) or line (three finger swipe)
- Nudge: move an object by single pixel increments by holding it with one finger and then swiping with another finger (nudge by higher numbers of pixels by using more fingers)











Quick Quiz

- What was the first use of two-handed input with a computer?

Quick Quiz

- What was the first use of two-handed input with a computer?

- Douglas Englebart in 1968
  - Point with mouse
  - Operate chord keyboard

Next Quiz

- Why has the PC so committed to having a single pointing device?

Next Quiz

- Why has the PC so committed to having a single pointing device?

- Lots of historical baggage
  - Technical: Early systems couldn't keep up with multiple continuous devices
  - Experimental: Fitts Law has only two parameters, target distance and size; performance studies typically focus on just a single hand

Lots of Recent Interest

- N. Matsushita, Y. Ayatsuka, J. Rekimoto. Dual touch: a two-handed interface for pen-based PDAs. UIST 2000, pp. 211-212.
  - Coordinated pen-and-thumb interaction without any additional technology on contact closure PDA (e.g., Palm or PocketPC device).
- A GUI Paradigm Using Tablets, Two Hands and Transparency. G Fitzmaurice, T. Baudel, G. Kurtenbach, B. Buxton. Alias/Wavefront, Toronto. CHI 97
- K. Hinckley, M. Czerwinski and M. Sinclair. Interaction and modeling techniques for desktop two-handed input. UIST '98 pp. 49-58.
- T. Grossman, G. Kurtenbach, G. Fitzmaurice, A. Khan, B. Buxton. Creating principle 3D curves using digital tape drawing. CHI 2002
- S. Chatty. Extending a graphical toolkit for two-handed interaction. UIST '94, pp. 195-204.
- MID: Multiple Input Devices
  - http://www.cs.umd.edu/hcil/mid/

Toolglasses and Magic Lenses

- GUI interaction technique meant to capture a common metaphor for two-handed interaction
  - Basic idea:
    - One hand moves the lens
    - The other operates the cursor/pointer
  - “See through” interfaces
  - The lens can affect what is “below” it:
    - Can change drawing parameters
    - Change change input that happens “through” the lens
- For the purpose of this lecture, I’m combining both of these under the term “magic lens”

Quick Examples

- Magnification (and arbitrary transforms)
- Render in wireframe/outline
- Object editing
  - E.g., click-through buttons: position color palette over object, click through the palette to assign the color to the object

- Important concept: lenses can be *composed* together
  - E.g., stick an outline lens and a color palette lens together to change the color of an object's outline
- Second important concept: lenses don't just have to operate on the final rendered output of the objects below them
  - Can take advantage of application data structures to change presentation and semantics

Reading:

Eric A. Bier, Maureen C. Stone, Ken Pier, William Buxton and Tony D. DeRose, "Toolglass and magic lenses: the see-through interface", *Proceedings of the 20th Annual Conference on Computer Graphics*, 1993, Pages 73-80.

http://www.acm.org/pubs/articles/proceedings/graph/166117/p73-bier/p73-bier.pdf

Note...

- These techniques are patented by Xerox
- Don't know scope of patent, but its likely you would need to license to use them commercially ... if the patents haven't expired

Advantages of lenses

- In context interaction
  - Little or no shift in focus of attention
    - tool is at/near action point
  - Alternate views in context and on demand
    - can compare in context
    - useful for "detail + context" visualization techniques

Detail + context visualization

- Broad category of information visualization techniques
  - Present more detail in area of interest
    - More than you could typically afford to show everywhere
    - Details may be very targeted
  - Present in context of larger visualization

Advantages of lenses

- Two handed interaction
  - Structured well for 2 handed input
    - non-dominant hand does coarse positioning (of the lens)
      - examples also use scroll wheel with non-dominant hand
        - scaling: again a coarse task
    - dominant hand does fine work

Advantages of lenses

- Spatial modes
  - Alternative to more traditional modes
  - Use "where you click through" to establish meaning
  - Typically has a clear affordance for the meaning
    - lens provides a "place to put" this affordance (and other things)

Examples

- Lots of possible uses, quite a few given in paper and video
- Property palettes
  - Click through interaction
  - Again: no context shift + spatial mode

Examples

- Clipboards
  - Visible
    - invisibility of typical clipboard is a problem
  - Lots of interesting variations
    - multiple clipboards
    - “rubbings”
  - Can do variations, because we have a place to represent them & can do multiple specialized lenses

Examples

- Previewing lenses
    - Very useful for what-if
    - Can place controls for parameters on lens
- Selection tools
    - Can filter out details and/or modify picture to make selection a lot easier

Examples

- Grids
    - Note that grids are aligned with respect to the object space not the lens

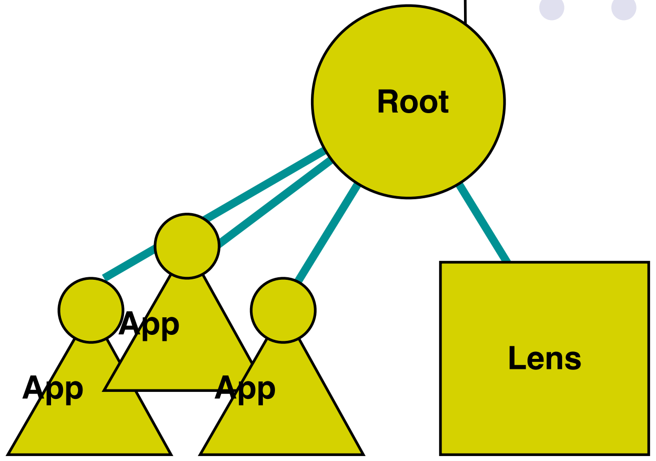
Examples

- Debugging lenses
  - Show hidden internal structure in a GUI
  - Not just surface features
- "Debugging Lenses: A New Class of Transparent Tools for User Interface Debugging," Hudson, Rodenstein, Smith. UIST'97

Implementation of lenses

- Done in a shared memory system
  - All "applications" are in one address space
  - Can take advantage of application-internal data structures
    - Different than OS-provided magnifying glass, for example
  - Like one giant interactor tree
  - Also assumes a common command language that all applications respond to

Implementation of lenses



- Lens is an additional object "over the top"
  - Drawn last
  - Can leave output from below and add to it (draw over top)
  - Can completely overwrite output from below
    - can do things like "draw behind"

Implementation of lenses

- Input side
  - Changed way they did input
    - originally used simple top-down dispatch mechanisms
    - now lens gets events first
      - can modify (e.g., x,y) or consume
    - possibly modified events then go back to root for “normal dispatch

Implementation of lenses

- Input side
  - Special mechanism to avoid sending events back to lens
  - Also has mechanism for attaching "commands" to events
    - assumes unified command lang
    - command executed when event delivered

Implementation of lenses

- Output side
- Damage management
  - Lenses need to be notified of all damage
    - Lens may need to modify area due to manipulation of output (e.g. mag)

Implementation of lenses

- Output side
- Redraw
  - Several different types of lenses
    - Ambush
    - Model-in / model-out
    - Reparameterize and clip

Types of lens drawing

- Ambush
    - catch the low level drawing calls
        - typically a wrapper around the equivalent of the Graphics object
    - and modify them
        - e.g. turn all colors to “red”
    - Works transparently across all apps
    - But somewhat limited

Types of lens drawing

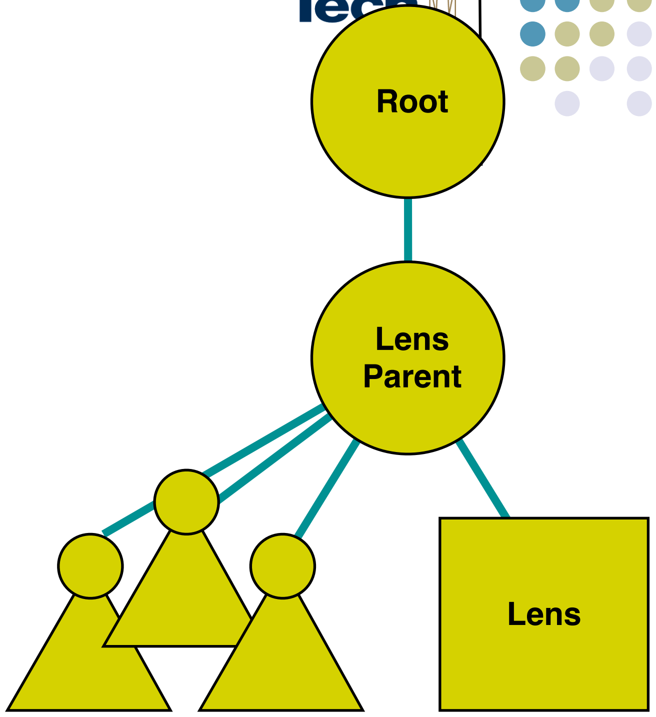
- Reparameterize & clip
  - similar to ambush

  - modify global parameters to drawing
  - redraw, but clipped to lens
  - best example: scaling

Types of lens drawing

- Model-in / model-out
  - create new objects and transform them
    - transforms of transforms for composition
  - very powerful, but…
    - cross application is an issue
    - incremental update is as issue

Lenses in subArctic



- Implemented with special "lens parent" & lens interactors
- Input
  - Don't need to modify input dispatch
  - Lens may need to change results of picking (only positional is affected)
    - in collusion with lens parent

Lenses in subArctic

- Damage management
  - Lens parent forwards all damage to all lenses
  - Lenses typically change any damage that overlaps them into damage of whole lens area

Lenses in subArctic

- Replace vs. draw-over just a matter of clearing before drawing lens or not
- Two kinds of output support
  - Ambush
    - Via wrappers on drawable
    - Extra features in drawable make ambush more powerful
  - Traversal based (similar to MIMO)

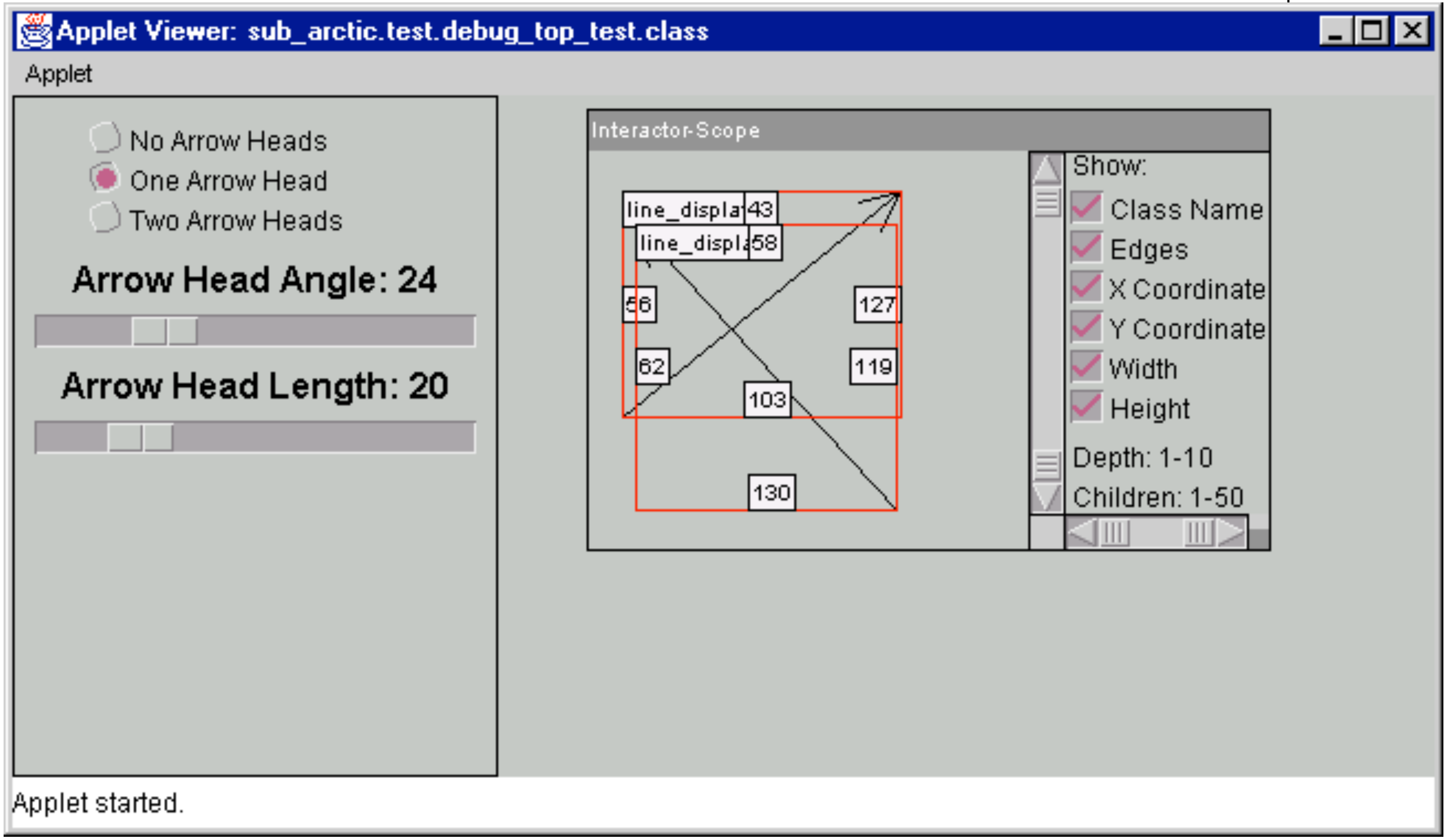
Ambush features in drawable

- boolean start_interactor_draw()
- end_interactor_draw()
  - called at start/end of interactor draw
  - allows tracking of what is being drawn
  - drawing skipped if returns false
- allows MIMO effects in ambush
  - isolated drawing
  - predicate selected drawing

Lenses in subArctic

- Also support for doing specialized traversal
    - walk down tree and produce specialized output
    - can do typical MIMO effects

Example: Debugging Lens



Applet Viewer: sub_arctic.test.debug_top_test.class

Applet

Arrow Head Angle: 24

Arrow Head Length: 20

Show:

- [✓] Class Name
- [✓] Edges
- [✓] X Coordinate
- [✓] Y Coordinate
- [✓] Width
- [✓] Height

Depth: 1-10

Children: 1-50

Applet started.

Lenses in Swing

- Two things to do:
  - #1: Make sure that your lens is drawn over other components
    - Easiest way: add a special component as the "Glass Pane" of a JFrame
    - GlassPane is hidden by default; when visible, it's like a sheet of glass over the other parts of your frame.
    - Generally, set a custom component as the glass pane with a paintComponent() method to cause things to be drawn
      - myFrame.setGlassPane(myNewLensPane)
      - myNewLensPane.setVisible(true)
  - #2 Create your lens class itself
    - Extend JCompnoent
    - Implement whatever listeners you want to get events for
    - Implement paintComponent so that when you draw yourself, you actually draw components under you (however you want to draw them) -- note that the lens itself likely won't have children

Swing GlassPane

- Hidden, by default
- Like a sheet of glass over all other parts of the JFrame; transparent unless you set it to be a component that has an implementation of paintComponent()
  - Don't actually have to *do* anything in paintComponent unless you want the pane itself to be visible
- Useful when you want to catch events or paint over an area that already contains components
  - E.g., deactivate mouse events by installing a class pane that intercepts the events

GlassPane Resources

- Tutorial on how to use the various panes in a JFrame:
  - http://java.sun.com/docs/books/tutorial/uiswing/components/rootpane.html
- Example of using glass pane:
  - http://blog.elevenworks.com/?p=6
- Another example of using glass panes for graphical overlay:
  - http://weblogs.java.net/blog/joshy/archive/2003/09/swing_hack_3_ov.html

Making a Lens

- Basically, a specialized component that’s a child of the glass pane
  - Output:
    - The lens should draw itself (title bar, gizmo to make it go away, its borders)
    - Also draw the components in the frame that are under it, although perhaps not in their original form
  - Input:
    - Redispatch events to components in the content pane
    - May need to tweak their coordinates/details (transform to the new component’s coordinate system, for example)
      - See SwingUtilities.convertMouseEvent(), SwingUtilities.convertPoint(), etc.

Lens Resources

- Swing Hacks, hack #56: Create a Magnifying Glass Component
- Blog entry on magic lenses in Swing:
  - http://weblogs.java.net/blog/joshy/archive/2003/11/swing_hack_5_a.html
- Lens details from an earlier version of this class:
  - http://www3.cc.gatech.edu/classes/AY2001/cs4470_fall/a4.html
- Passing events through to underlying components
- Tweaking component drawing
  - SwingUtilities.paintComponent
  - Lets you call a component's paint method on an arbitrary graphics object (e.g., one of your own choosing; can disable/reimplement certain functions, look at the call stack, etc., in drawing)
- Drawing the lens itself
  - Consider using JInternalFrame as the base class for your Lens, as you'll get some basic window decorations.

Collaborative Multitouch (Very Briefly...)

Collaborative multitouch

- Most useful for large surfaces (tables, walls) instead of phones
- Examples:
  - Microsoft Surface
  - Mitsubishi DiamondTouch table
  - Nottingham Dynamo
- Special issues:
  - Orientation (for table-top displays)
  - Can you tell which finger belongs to whom?

Opportunities for expressiveness

- Use edge of hand to bring up "secret" dialog box (Wu & Balakrishnan, 2003)
- Augment with additional sensing (e.g., face recognition) to determine user identity