User Interaction

CS 465 lecture 20

User Interaction

- Input devices
- GUIs and GUI design
- Event-driven programming
- 3D interaction

Input devices

- Discrete events
  - Keyboard
  - Function keys
  - Mouse buttons
  - Game controller buttons
    - Including multi-way controllers (pseudo-joysticks)
- Valuators: generate continuous values
  - Rotary knobs (relative or absolute)
    - Recentering or free
  - Joysticks (two valuators in one)

Input devices

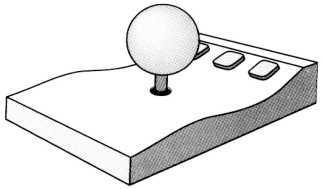
- Locators: give a continuous 2D position
  - Mechanical mouse (trackball is the same)
    - Two axes with optical encoders
      - Integrate rate of pulses on each axis
      - Result = position
  - Optical mouse
    - Image sensor looking out the bottom
      - Shift and correlate to estimate motion per frame
      - Integrate motion to get position
  - Mouse velocity scaling

Input devices

- Locators, cont.
  - Pen tablet
    - Directly senses absolute stylus position
    - Often used directly over a display
      - PDA
      - Tablet PC
  - Absolute vs. relative
  - Direct vs. indirect

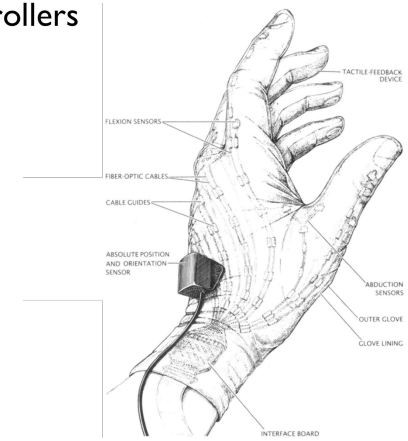
Input devices

- Multidimensional controllers
  - More exotic devices
  - Spaceball
  - Data glove
  - 3D tracker
    - Magnetic
    - Acoustic
    - Optical



Input devices

- Multidimensional controllers
  - More exotic devices
  - Spaceball
  - Data glove
  - 3D tracker
    - Magnetic
    - Acoustic
    - Optical



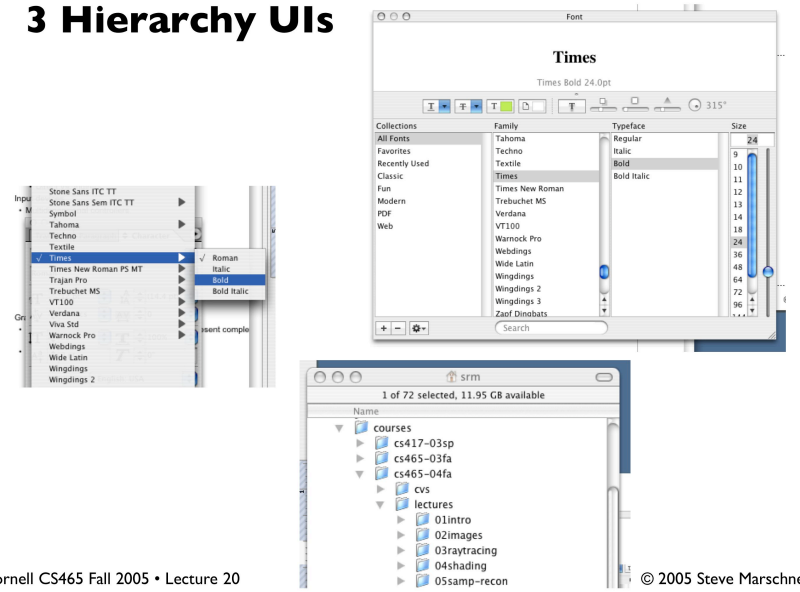
Graphical User Interfaces (GUIs)

- Using visual display coupled with pointing to present complex choices to the user
- The dominant mode of user interface today
- Many flavors exist, but all present similar widgets
  - Icons (objects)
  - Buttons (actions)
  - Menus (collections of choices/actions)
  - Lists
  - Trees

Basic interaction tasks

- Positioning
- Selection
  - From large/continuous set (part of image)
    - Rectangle, lasso
  - From unorganized discrete set (icons on desktop)
    - Click and modifier-click, or drag area
  - From linearly organized set
    - Selection from list box
  - From hierarchically organized set
    - Drop-down menus, trees, columnar lists, …

3 Hierarchy UIs



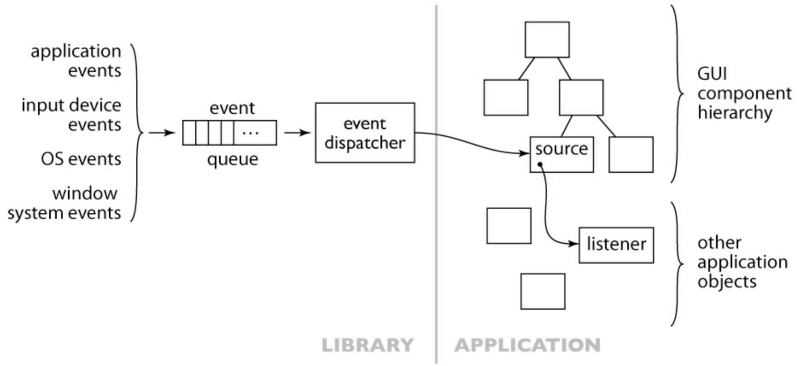
Event driven programming

- Menu-based text
- Full-screen text editor
- Interactive program with pointer
- Callback-based interactive program
- GUI program
- Event-driven GUI program
  - Event routing
  - Non-input events

An event-driven system: Java/Swing

- The window system is in control (it's the outer loop)
- All application code executes, directly or indirectly, in response to events
  1. *Event listeners* are registered on components
  2. Event generated (asynchronously, by input device, system, or application)
  3. Event goes on event queue
  4. Event is dequeued in the main event loop and dispatched
     a) Identify event source
     b) Call event-handling method on source
     c) Event-handling method calls all interested listeners

Event handling machinery



Event handling issues

- Listener/callback style programming encodes the event routing in references (listener lists, etc.), which are dynamic, rather than in code, which is error-prone and static
- For responsiveness, must avoid long operations in event handlers
  - Long computations
  - Network and file operations
  - Solution: events for asynchronous I/O
  - Solution: fragmented computation and idle events
  - Solution: multiple threads

Interaction in graphics

- Graphics interaction (e.g. drawing) is less constrained
- Normally based on mouse events
  - Input events:
    - Mouse down, mouse up
    - Mouse moved
    - Mouse clicked (down-up w/o motion)
    - Mouse dragged (motion while button down)
    - All come with info about modifier keys
  - Resize, redraw events
  - Timer events
- Several common interaction tasks exist

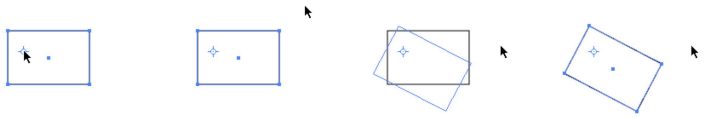
Creating 2D objects

- Rubberbanding
  - Mouse-down: first point; mouse-up: second point
  - In between, give feedback
  - Rubberbanding is a “spring loaded” mode

Manipulating 2D objects

- Transformations
  - Handle-based interface: easy to learn but limited

  - Center-and-drag interface: more flexible but less direct



Implementing interaction

- How to do this by handling events?
- Mode implies the need to keep state

```
mouseDown(x, y) {
   startPt.set(x, y);
   endPt.set(x, y);
   rubberband = true;
}
```

```
mouseMove(x, y) {
   if (rubberband) {
      endPt.set(x, y);
      repaint();
   }
}
```

```
mouseUp(x, y) {
   if (rubberband) {
      createLine(startPt, endPts);
      rubberband = false;
      repaint();
   }
}
```

  - (drawing code needs to draw the tentative line)

Implementing interaction

- State machines: an abstraction for modes

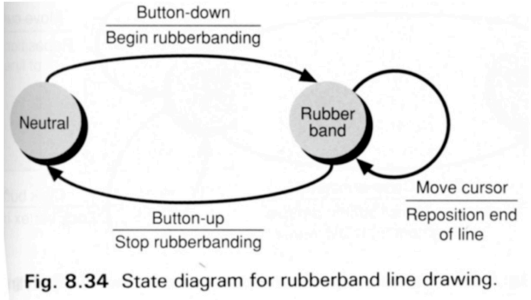
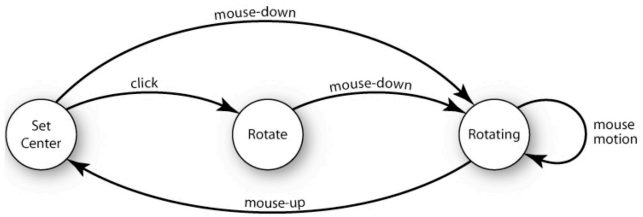


Fig. 8.34 State diagram for rubberband line drawing.

  - Thinking this way can help with coding complex interactions

More complex state machine

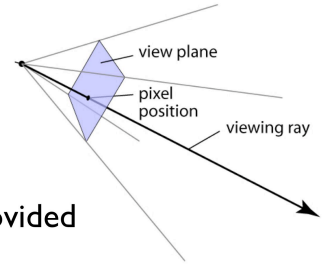
- For center-based rotation in Adobe Illustrator



  - (demo)

Interaction with 3D scenes

- Basically 2D interaction with feedback given via projected view
- Main challenge: mapping 2D mouse events to 3D operations
- The camera is very important
  - Determines what can be seen
  - Determines mapping between 2D and 3D
  - Reverse projection is ill-defined!
- Often multiple cameras are provided

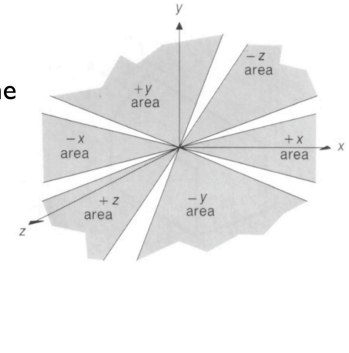


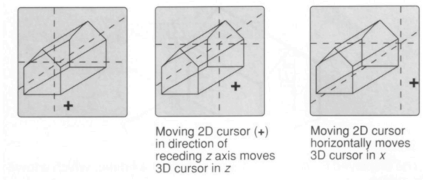
3D interaction: points

- Principle: point should go where you click
- Creating a point
  - Click specifies a ray, but where is the point?
  - Common solution: intersect with known plane (e.g. a ground plane)
- Moving a point
  - Mouse motion specifies two rays
  - One possibility: intersect with plane as above
  - Second possibility: move parallel to camera plane
    - Scale motion by z distance to ensure following mouse

3D interaction: translation

- Principle: click and drag moves object as expected
- Again, many 3D directions project to the user's 2D motion vector
  - Move parallel to view plane
  - Move parallel to coordinate plane
  - Move parallel to coordinate axis





3D interaction: translation

- Manipulator: more refined version of same idea
  - present 3D widgets to select axis
  - example: manipulator in project

3D interaction: scale

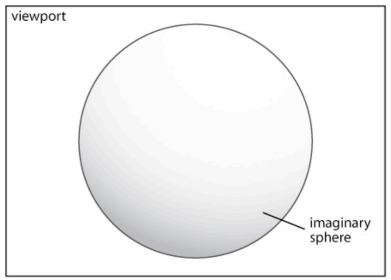
- Uniform scale: a 1D input; just use one mouse coord
- Nonuniform scale: 2 inputs for 3 axes
  - Project to closest axis (like translation example)
  - Map mouse *x* and *y* to closest axes
- Example of manipulator from project

3D interaction: rotation

- Very common operation
- 2D input for 3 rotations
  - Provide sliders for 3 axes
  - Up/down rotates about horiz.; left/right rotates abot vert.
  - Manipulator to choose axis
  - Virtual trackball

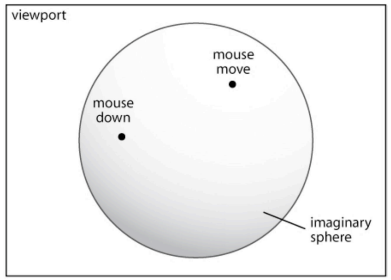
Virtual trackball rotation

- A very popular interface for general rotations
- Concept: you're turning a sphere by grabbing a point and moving it
  - The sphere makes the smallest rotation that follows the mouse
  - Rotation axis is perpendicular to the radii to the two points
  - Rotation angle is determined by angle between radii



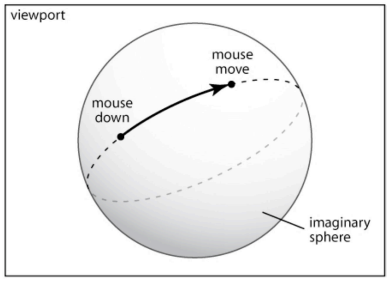
Virtual trackball rotation

- A very popular interface for general rotations
- Concept: you're turning a sphere by grabbing a point and moving it
  - The sphere makes the smallest rotation that follows the mouse
  - Rotation axis is perpendicular to the radii to the two points
  - Rotation angle is determined by angle between radii



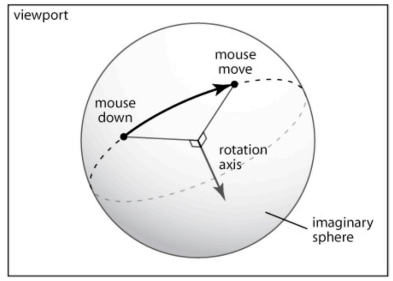
Virtual trackball rotation

- A very popular interface for general rotations
- Concept: you're turning a sphere by grabbing a point and moving it
  - The sphere makes the smallest rotation that follows the mouse
  - Rotation axis is perpendicular to the radii to the two points
  - Rotation angle is determined by angle between radii



Virtual trackball rotation

- A very popular interface for general rotations
- Concept: you're turning a sphere by grabbing a point and moving it
  - The sphere makes the smallest rotation that follows the mouse
  - Rotation axis is perpendicular to the radii to the two points
  - Rotation angle is determined by angle between radii



Selection of 3D objects (picking)

- How to determine what object a click falls on?
- Same as visible surface determination problem
  - One solution: use ray tracing (but requires separate infrastructure)
  - Pipeline solution: object ID buffer
    - Draw in hidden buffer with color per object
    - Read back color to determine object belonging to a pixel
  - Pipeline solution: draw scene with tiny viewport
    - Observe what primitives are actually drawn
    - OpenGL provides a callback mechanism for this

Manipulating cameras in 3D

- Quite similar to transforming objects
  - Except you apply the inverse transformation to the camera
- Nomenclature comes from cinematography
  - Camera translation: "dolly" for *w*; "track" for *u* (and also *v* by abuse of terminology)
  - Camera rotation about viewpoint: "pan" for about *v*; "tilt" for about *u* (common abuse = pan for both); "roll" for about *w* (probably from aviation)
  - Camera rotation about target point: "orbit"
  - Changing angle of view: "zoom"

