CS 160: Lecture 10

Professor John Canny
Spring 2003
March 3
Administrivia

- In-class midterm on Weds
  * Closed book (no calcs or laptops)
  * Material up to last Weds
- Anonymize online contextual inquiries!!
Outline

Output
* basic 2-D computer graphics
* color models

Input
* event overview
* windowing systems
* window events
* event dispatching

Widget communication & layout
2-D Computer Graphics

- Models for images
  * strokes, pixels, regions
- Coordinate systems
  * device, physical
- Canvas
- Drawing
  * paths, shapes, text
- Clipping
Stroke Model

Describe image as strokes (w/ color/thickness)
  + Line ((10, 4), (17,4), thick 2, red)
  + Circle (( 19, 13), radius 3, thick 3, white)

Maps to early vector displays & plotters
Most UI toolkits have stroked objects
  * arcs, ellipses, rounded rectangles, etc.
Problems with Stroke Model?

How would you represent with strokes?
Solution?
Pixel Model

- Break-up complex images into discrete “pixels” & store color for each

- Resolution
  * spatial: number of rows by columns
    + e.g., 1280 x 1024 is a good monitor display
    + quality laser printer: 6000 x 4800 (600 dpi)
  * image depth (i.e., number of bits per pixel)
    + several styles... 8-bit, 24-bit, 32-bit
Image Depth

- Bit map - 1 bit/pixel (on/off)
  * B&W screens or print-outs
Image Depth (cont.)

- **Gray scale - 2-8 bits/pixel**
  * group pixels (some on, some off)

- **Full color - 24 bits/pixel**
  * 8 bits per primary color (Red, Green, Blue)
Image Depth (cont.)

- **Full color - 32 bits/pixel**
  * Usually just 24-bit color (used for efficiency)
  * Extra 8-bits are optional - can be used for “alpha” (transparency)

- **Color mapped - 8 bits/pixel**
  * store index @ pixel - map into table w/ 24 bits
  * cuts space & computation
  * problem????
Aliasing

- Smooth objects (e.g., lines) appear jagged since resolution is too low
- Antialiasing - fill-in some jagged places w/ gray scale or primary colors
Anti-Aliasing

- Pixels colored in proportion to relative amount of line that crosses them.
- Equivalently, draw the line in B/W at finer resolution and then color each pixel in proportion to number of colored sub-pixels.
Cleartype

The pixel matrix for a laptop or LCD screen.
Use sub-pixel color pixels as though they were gray pixels (can cause color anomalies).
Region Model

- Use the stroke model to outline region
- Fill the region with
  - colors & blendings (i.e., patterns)
- Advantages??
  - allows representation of filled shapes w/
    - little memory
    - independence from display resolution
- Text represented this way & converted to bitmaps inside of the printer
Outline Fonts

Used by both Postscript & TrueType
Coordinate Systems

Device coordinates
- coordinates of the display device
- origin usually at upper left

Window coordinates
- toolkit presents window as an abstraction
- virtual display with frame around it (title, etc.)
- program’s coordinates placed in window as if frame doesn’t exist
- like device coords, always expressed in pixels
- mouse events may be in device coords - check
Coordinate Systems (cont.)

- Physical coordinates
  - pixel-based coords don’t deal well w/ devices of different resolutions (e.g., monitor vs. printer)
  - specify coordinates in physical units (e.g., inches, centimeters, or printer points)

- Model coordinates
  - coordinate system relative to drawn objects
  - need to convert from model to physical/window coordinates & back
Canvas

- Abstraction for the drawing surface
  * most toolkits support one
- Defines methods used for drawing
- Each instance has a height, width, & defines its physical units
- Use the same method interface for
  * windows
  * image in memory
  * printed output
- Called Graphical Device Interface (GDI) by MS
Drawing

Could specify with:

* void Canvas::Rectangle (x1, y1, x2, y2, lineWidth, lineColor, fillColor)

Lots of parameters!

* shapes have properties in common
  + geometry, line/border width, line/fill color, pattern

Use current settings of canvas

* Usually there is a “graphicscontext” or similar abstraction that defines all the parameters needed for drawing.
Text Font Selection

Font family

* Garamond, Arial, Modern, Times Roman, Courier
* defines the general shape of the characters
  + some are mono-spaced (“i” gets same space as “G”)
  + serif (e.g., Times) vs. sans serif (e.g., Arial)
    ~ serifs have “feet” at baseline -> easier to track eye
      but look bad on low-resolution displays.

Style

* normal, bold, italic, bold italic

size in points (1 point = 1/72 inch)
Text (cont.)

- Usually simple to draw
  + Canvas Cnv;
  + Cnv.SetFont ("Times", Bold, 10);
  + Cnv.Text (10, 20, "This is the text");

- Outline vs. Bitmapped fonts
  * need pixels to draw on screen so may store as BM
  + problems: takes lots of space font in several sizes
  * instead store as a closed shape (e.g., outline only)
  * easy to scale to any size and convert to bitmap
Clipping

- Limit drawing to particular area of screen. Why?
  * for performance

- Commonly clip to rectilinear regions. Why?
  * can be done very quickly
Color Models

- 256 levels for each primary adequate
  * -> 24 bits / pixel

- RGB model
  * specify color by red, green, & blue components

- HSV model - hue, saturation, & value
  * hue is primary wavelength (i.e., basic color)
  * saturation is a measure of how pure light is
    + high is pure, low means it is mixed w/ white/gray
  * value is intensity (dark vs. light)
Color Models (cont.)

- **HSV** is easier for people to use
  * there is a direct conversion to RGB

- **CMY model**
  * in terms of mixtures of pigments
  * pigment gets color from light it absorbs and does not reflect
  * mix *Cyan, Magenta, Yellow*
    + subtractive primaries
  * used by printers and artists
Images sometimes have a 4th channel called “alpha“(α) to encode transparency (e.g. png)

\[ C = \alpha \times C_f + (1-\alpha) \times C_r \quad - \text{each color channel} \]
Break
Sequential Programs

- Program takes control, prompts for input
- Examples include
  * command-line prompts (DOS, UNIX)
  * LISP interpreter
- The user waits on the program
  * program tells user it’s ready for more input
  * user enters more input
Sequential Programs (cont.)

- **Architecture**
  - Program reads in a line of text
  - Program parses the text
  - Program evaluates the result
  - Maybe some output
  - Loop back to beginning

- But how do you model the many actions a user can take?
  - for example, a word processor?
  - need to do printing, editing, inserting, whenever user wants to
Sequential Programs (cont.)

- Usually end up with lots of *modes*
  * lots of state variables
- Other examples of modes
  * paint programs (line, bucket-fill, rectangle, etc)
  * universal remotes with TV / VCR mode
  * *vi* edit mode and command mode
- Problems with modes?
Sequential Programs (cont.)

- Problems with modes?
  - gets confusing if too many modes
  - can be easy to make errors
  - need feedback as to what mode you are in
  - how to switch between modes?

- We’ll need a more advanced model to simplify windows programming
Event-Driven Programming

Instead of the user waiting on program, have the program wait on the user.

All communication from user to computer is done via “events”.

An event is something “interesting” that happens in the system:

* mouse button goes down
* item is being dragged
* keyboard button was hit
Event Example

close box

title bar

close box

folder

scroll bar

size control
Major Issues

- How to decompose the UI into interactive objects?
- How to distribute input to the interactive objects
- How to partition between application & system software?
- Models for programming interactive objects
- Models for communications between objects
Windowing Systems

- Partitioning to prevent chaos
- Infrastructure to support common services

Two major aspects
- software services to applications
  - create and organize windows
  - implement interaction in those windows
- window manager
  - UI allowing user to control size & placement of windows
Interactor Tree

- Decompose interactive objects into a tree
  * interactive objects also known as "widgets"
  * based on screen geometry of objects
  * nested rectangles

- Used for dispatching events
  * events are dispatched (sent) to code in widget
  * the code then handles the event

- Variety of methods for dispatching events
  * return to this later
Interactor Tree

Display Screen

- “F:\cs160\Public” window
  - Inner Window
    - title bar
    - horizontal scroll bar
    - contents area
      - “CDJukebox” folder
      - “Home Ent...” folder
      - ... size control
    - “Web Newspaper” window
      - ...
Interactor Tree

Display Screen

?-Outer Win [black]

93.54
Interactor Tree

Display Screen
- Outer Win \([black]\)
- Inner Win \([green]\)

Result Win \([tan]\)
- Result String

Keypad \([Teal]\)
- \(=\) button
- - button
- + button
- 0 button
Interactor Tree (Java)

Display Screen

- Frame [black]
- Panel [green]

Text Entry [tan]
- Result String

Keypad Panel [Teal]
- Button(“=”)
- Button(“-”)
- Button(“+”)
- Button(“0”)

93.54

7 8 9
4 5 6
1 2 3
0 + -
= ENT
Windows

- Top level windows known as *root windows*
  - provide UI abstraction for multiple apps
  - windowing system arbitrates interactive resources

- Each root window belongs to an app.
  - all descendant windows belong to same app
  - violated by ActiveX

- **Windows vs. widgets/controls**
  - X, NeXTStep, MS Windows
    - everything is window
Summary

- Concepts:
  - 2D vector graphics
  - Raster graphics - color, antialiasing
  - Event-driven programming
  - Interactors