

CS 349

Graphic Abstractions

Byron Weber Becker
Winter 2012



CS 349 Winter 2012 1

Announcements: Jan 11

- Section 002 is moving to EV3 1408 effective MONDAY.
- ExamQuestions: post to a second instance of Piazza.
- HCI+Security talk: Thu Jan 19, 4:00pm DC1302
 - Security and privacy systems with usability problems are far more likely to result in overall security or privacy failures regardless of the systems' technical soundness. I argue that an integrated design approach combining human-computer interaction (HCI) and security is vital to success, and without it security is inevitably weakened. In this talk, I discuss some of our recent research, including work on knowledge-based user authentication, usable privacy and security of mobile devices, and improving users' mental models of security and privacy.



CS 349 Winter 2012 2

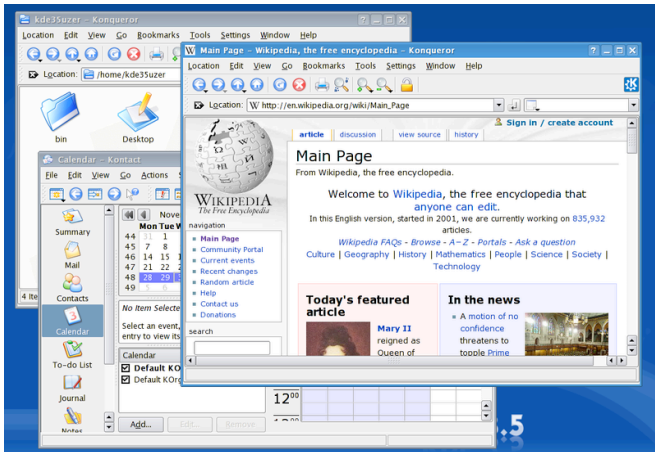
Lecture Overview

- Architecture
 - Base Window System
 - Window Managers
- Windows
 - Canvas
 - Abstraction: screen, printer, etc.
 - Clipping
 - Interactor trees
 - Coordinate systems
- Pixelated Displays
 - Human Perception
 - Models



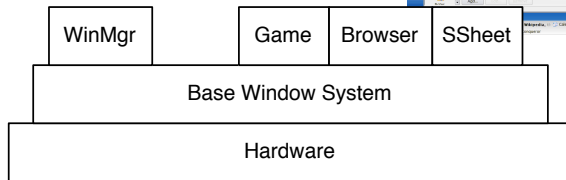
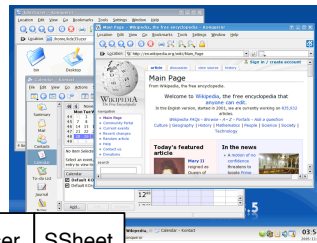
CS 349 Winter 2012 3

Architecture



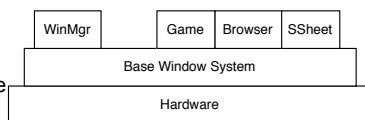
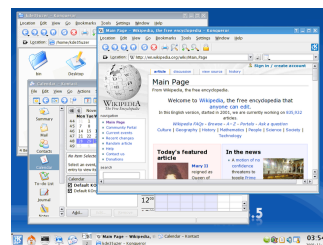
Screenshot from <http://en.wikipedia.org/>

Base Window System



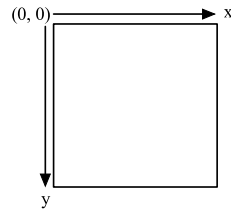
Base Window System

- Lowest level abstraction for windowing system
- Provides routines for creating, destroying, managing windows
- Routes input to correct window
- Ensures only one application changing frame buffer (video memory) at a time
 - Is one reason why you see only single-threaded / non-thread-safe GUI architectures



Base Window System

- Creates canvas abstraction for applications
 - Applications shielded from details of frame buffer, visibility of window, other application windows
- Each window has its own coordinate system
 - BWS transforms between coordinate systems
 - Each window does not need to worry where it is on screen, always assumes its top-left is (0,0)
- Provides basic graphics routines for drawing

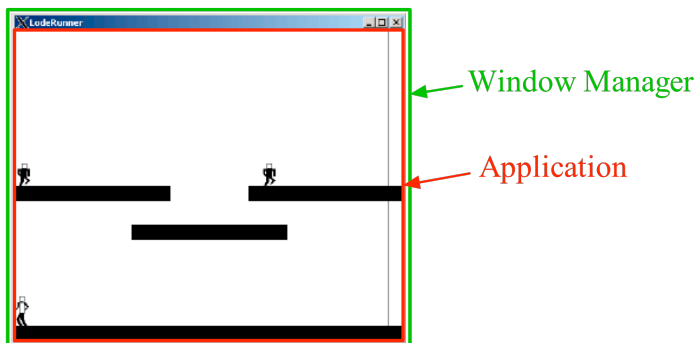


Window Manager

- Window Manager provides conceptually different functionality
 - Layered on top of Base Window System
 - Provides interactive components for windows (menus, close box, resize capabilities)
 - Creates the “look and feel” of each window

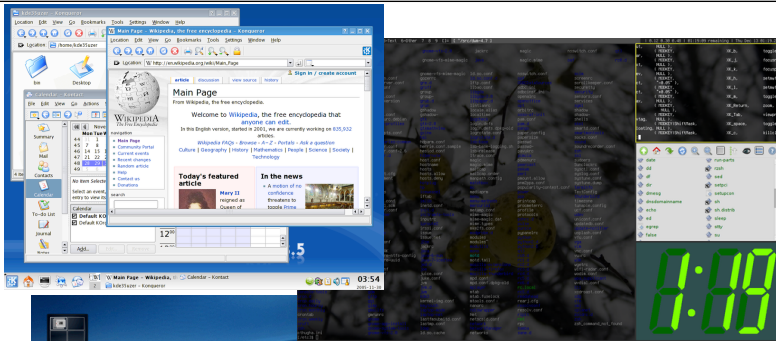
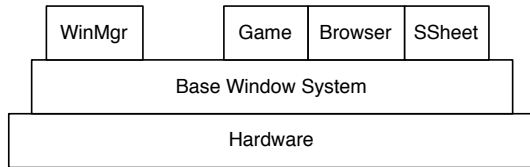
Window Manager

- Frame vs. content area (actual canvas)



BWS vs. Window Managers

- X separates Base Window System from Window Manager
 - Enables many alternative “look and feels” for windowing system (e.g., KDE, GNOME, fwm...)
 - One of the keys to its lasting power: Can continue to grow by changing the Window Manager layer
- Each a separate process



<http://en.wikipedia.org/wiki/File:Dwm-screenshot.png>



Types of window managers:

- Stacking
- Tiling
- Compositing

<http://en.wikipedia.org/wiki/KWin>

BWS vs. Window Managers

- Macintosh, Windows bundle Base Window System and Window Manager together
 - *Very difficult* for 3rd party to achieve alternative look and feel
- Trade-offs in approaches?
 - Look and feel...
 - Window management possibilities...
 - Input possibilities...



Windows and Widgets

- Window
 - The high-level unit managed by window manager
 - Has canvas
 - May contain sub-windows/widgets
- Widgets
 - Individual elements within window user sees, interacts with
 - Label, slider, text component, etc.
 - AKA “controls” (Microsoft) and “components” (Java)
- Widgets are either:
 - Actual windows themselves (e.g., X, MS Windows)
 - Objects implemented by the GUI toolkit (Java)



Canvas

- Every system provides a *Canvas*-like abstraction
- The method by which one draws in the window
 - Canvas represents window’s content area
- Canvas more than a “surface”
 - A “surface” *and* a set of routines for manipulating that surface
 - DrawLine(), DrawRectangle(), DrawString()...
- Graphics context (state)
 - State representing parameters for future drawing operations
 - Foreground colour, line width, font...
 - Clip



Canvas

- Division of entities not standardized
- Examples:
 - X: Display + XLib routines + GC
 - Java: Component + Graphics/Graphics2D object
 - Mac OS X’s Cocoa: NSView + NSGraphicsContext
 - Windows: Device Context + Graphics Device Interface (GDI)
- Java rolls a lot into Graphics/Graphics2D object
 - Graphics context (foreground color, font, clip...)
 - Drawing routines
 - *Only* way to manipulate canvas’s graphics



Graphics Abstraction

- Abstraction of drawing routines useful to create *device independence*
 - In theory, same drawing routines, regardless of where output rendered (e.g., CRT, LED display, printer)
 - Don't need to know hardware capabilities, just draw
- But devices *do* matter, in some cases
 - Output to screen vs. printer

Graphics Abstraction Issues

- Rendering to printer rather than display
 - Papers have “hard” edges, so content can't go on endlessly
 - Pagination issues with printing
 - Resolution a big deal with printing
 - Huge differences in DPI (dots per inch) between display devices, printing devices
 - 72 DPI vs. 300 DPI
 - So same drawing routines for screen and printer desirable, but not exactly possible
- Color models also an issue
 - RGB vs. CMYK

The Clip

- Need to ensure each window/canvas only draws in its own area
- Need to optimize drawing routines
- *Clip* provides this functionality

The Clip

- A (potentially) arbitrary region that defines where drawing operations will / will not have effect
- Part of the GC (graphics context) in X
- Part of Graphics/Graphics2D in Java
- Manipulable by programmer, but can never draw outside of your own window

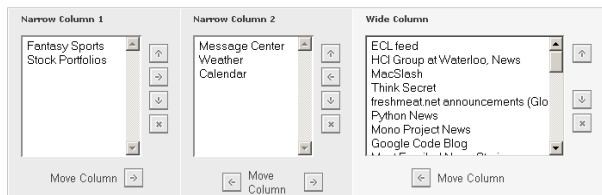


```
XRectangle clip_rect;
clip_rect.x = 0;
clip_rect.y = 20;
clip_rect.width = 30;
clip_rect.height = 40;

while (1)           // event loop
{ XEvent event;
  XNextEvent( display, &event );
  switch( event.type ) {
  case KeyPress:
    clip_rect.x += 10;
    XSetClipRectangles(display, gc, 0, 0, &clip_rect, 1, Unsorted);
    repaint(display, window, gc);
    break;
    ...
  }}}

void repaint(Display* display, Window window, GC gc)
{ XClearWindow( display, window );
  XDrawString(display, window, gc, 30, 50, "String test", strlen("String test"));
  XDrawLine(display, window, gc, 30, 50, 200, 50);
  XFlush(display);
}
```

Interactor Tree



- Components contained within components
 - *Containment hierarchy or interactor tree*

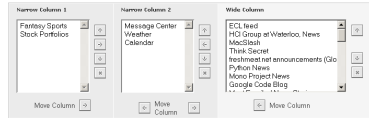


Interactor Tree

- Window

- First Panel

- Narrow Column 1 Label
 - List Box
 - Up arrow
 - Right arrow
 - ...



- Second Panel

- Narrow Column 2 Label
 - List Box
 - ...

- Third Panel...



Interactor Tree

- Interactor tree represents a hierarchy of containers
 - Components contained within components
- Containment hierarchy helps decide where to target events
- Hierarchy also used for *drawing* components
 - Child components “draw” within parent components
 - Child’s painting is clipped to parent component’s bounds; parent likely restricts child even further



Coordinate Systems

- Component’s location and bounds are represented in coordinate system of parent component
- But...
- ...Drawing within component is assumed to be relative to component’s top-left corner
 - Top-left corner is always (0, 0)
- Has important consequences for delivering event information to components, drawing into components...



Drawing in Windows/Components

- Need elegant way to set up drawing routines so they are always relative to top-left corner of component
- In X, MS Windows, where a window == a component, base window system automatically sets up coordinate system so drawing routines are relative to component
- For lightweight architectures (Java's Swing), coordinate system needs to be explicitly set

Pixelated Displays: Terminology

- Pixel
 - Image element on a display, a location
- Bitmap
 - Black and white image
 - Rectangular grid of 1's and 0's
- Pixmap
 - Same as bitmap but with colour
 - Each pixel contains one of
 - 24+ bits
 - an offset into a colour table

Performance Issues

- Display technology is fairly expensive
 - Consider 100Hz refresh rate on high quality 1024 x 768 display:
 - Frame rate: 10 milliseconds
 - Line rate: 768 lines every 10 ms or 13 microseconds
 - Pixel rate: 1024 pixels every 13 microseconds = ~10 nanoseconds
- Because display is so expensive, only use the resolution that's necessary. No redundant pixels.

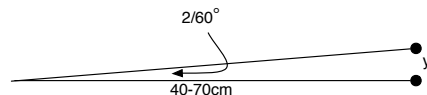
Announcements: Jan 13

- Section 002 is NOT moving to EV3 1408.
- ExamQuestions: post to a second instance of Piazza.
- HCI+Security talk: Thu Jan 19, 4:00pm DC1302
 - Security and privacy systems with usability problems are far more likely to result in overall security or privacy failures regardless of the systems' technical soundness. I argue that an integrated design approach combining human-computer interaction (HCI) and security is vital to success, and without it security is inevitably weakened. In this talk, I discuss some of our recent research, including work on knowledge-based user authentication, usable privacy and security of mobile devices, and improving users' mental models of security and privacy.



Spatial limits of vision

- Spatial acuity
 - 2 minutes of arc seems co-located
 - 1 minute of arc = 1/60 degrees
- Monitor viewing distance?
 - Canadian Centre for Occupational Health and Safety says 40-70cm.



- Pixel size?
 - Common monitor values are 0.25mm or less

$$\tan(2 / 60) = \frac{y}{400\text{mm}}$$
$$0.00058 * 400\text{mm} = y$$
$$y = 0.23\text{mm}$$

$$\tan(2 / 60) = \frac{y}{700\text{mm}}$$
$$0.00058 * 700\text{mm} = y$$
$$y = 0.41\text{mm}$$



iPhone's 'Retina' Display




- Just a marketing name Apple came up with
 - 2" x 3" display with 640x960 resolution
 - About 326dpi
 - Apple claims that the limit of spatial acuity at 25 cm (unreasonably close!) is 326 dpi
 - The preceding formula indicates that 0.15mm seems co-located.
- How big is an iPhone pixel?

$$\frac{1\text{in}}{326\text{dots}} * \frac{25.4\text{mm}}{1\text{in}} = .078\text{mm}$$



Drawing to a pixelated display

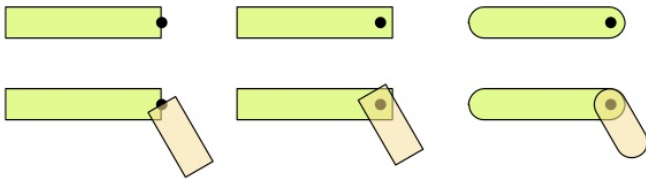
- Three different models for creating images:

Pixel		SetPixel(x, y, color) DrawImage(x, y, w, h, img)
Stroke		DrawLine(x1, y1, x2, y2) DrawRect(x, y, w, h)
Region		DrawLine(x1, y1, x2, y2) DrawRect(x, y, w, h)

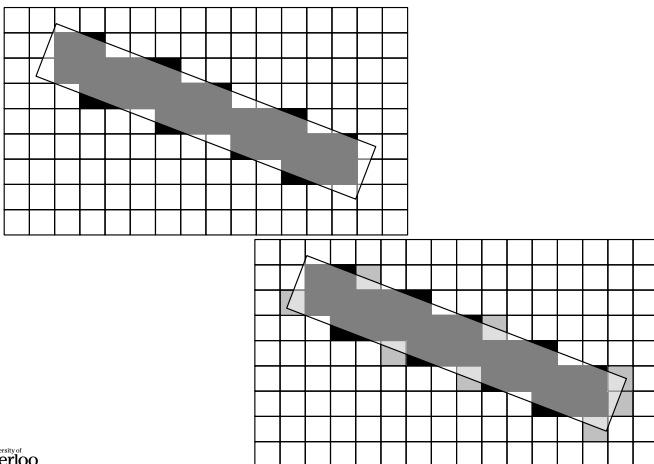


Drawing

- Issues with regions: how do lines join?



Drawing: Aliasing



Recap

- Architecture
 - Base Window System
 - Window Managers
- Windows
 - Canvas
 - Abstraction: screen, printer, etc.
 - Clipping
 - Interactor trees
 - Coordinate systems
- Pixelated Displays
 - Human Perception
 - Models