Multiple Windows

X Design Criteria 1

- *The X Window System* by Robert W. Scheifler and Gim Gettys in *ACM Transactions on Graphics*, Vol. 5, No. 2, April 1986, defines the following design goals for X. See the full paper on the course web site.

1. "implementable on a variety of displays"
2. "applications must be device independent"
3. "system must be network transparent"
4. "must support multiple applications displaying concurrently"
5. "support many different application and management interfaces"

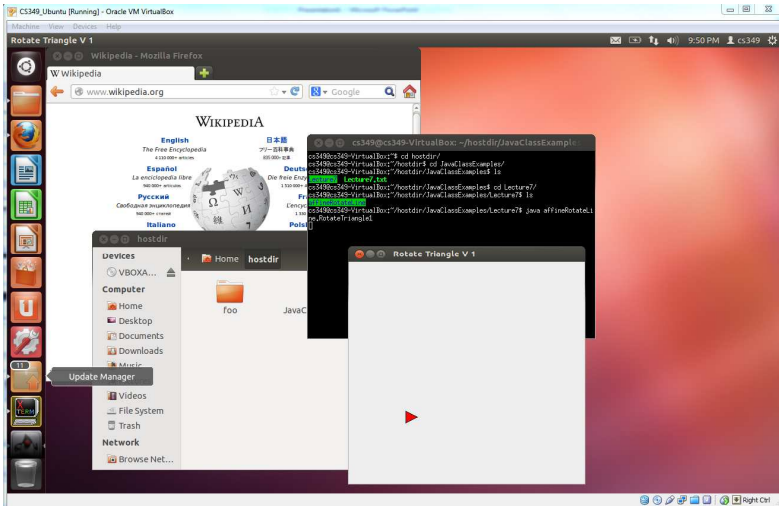
X Design Criteria 2

6. “must support overlapping windows, including output to partially obscured windows”
7. “support a hierarchy of resizable windows, and an application should be able to use many windows at once.”
8. “provide high-performance, high-quality support for text, 2-D synthetic graphics, and imaging”
9. “system should be extensible”

- How to support multiple windows?

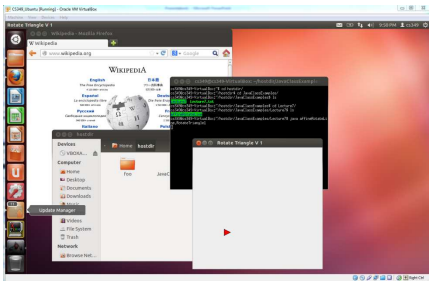
3

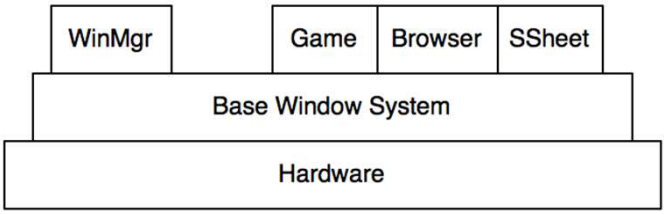
Multiple Windows



4

Base Window System

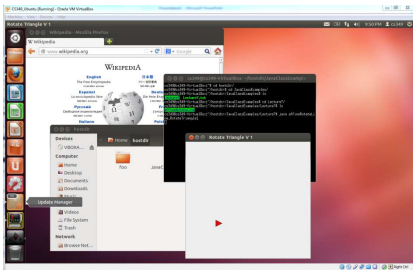


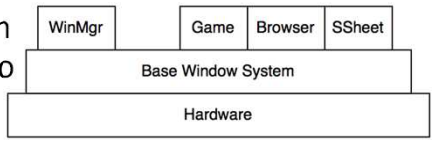


5

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

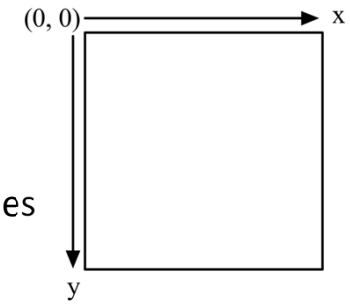




6

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



7

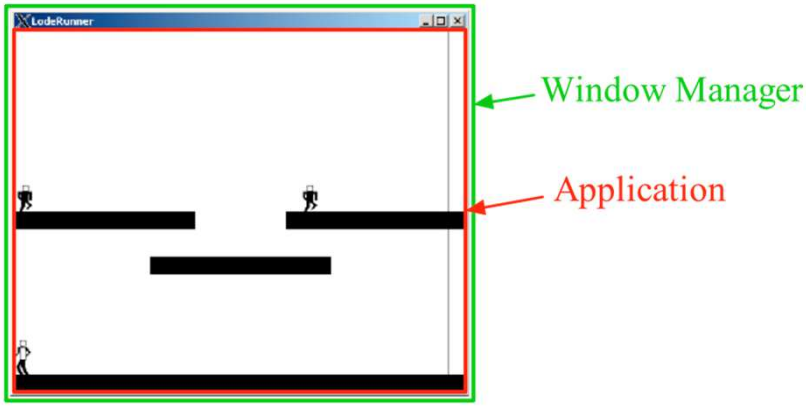
Window Manager

- 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

8

Window Manager

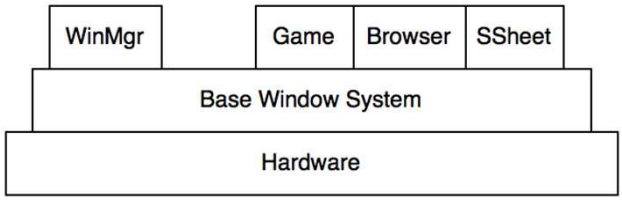
- Frame vs. content area (actual canvas)



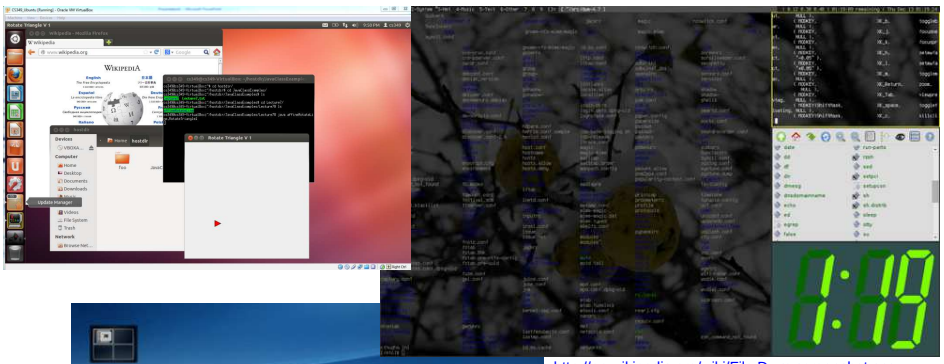
9

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, fvwm…)
  - One of the keys to its lasting power: Can continue to grow by changing the Window Manager layer
- Each a separate process



10



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

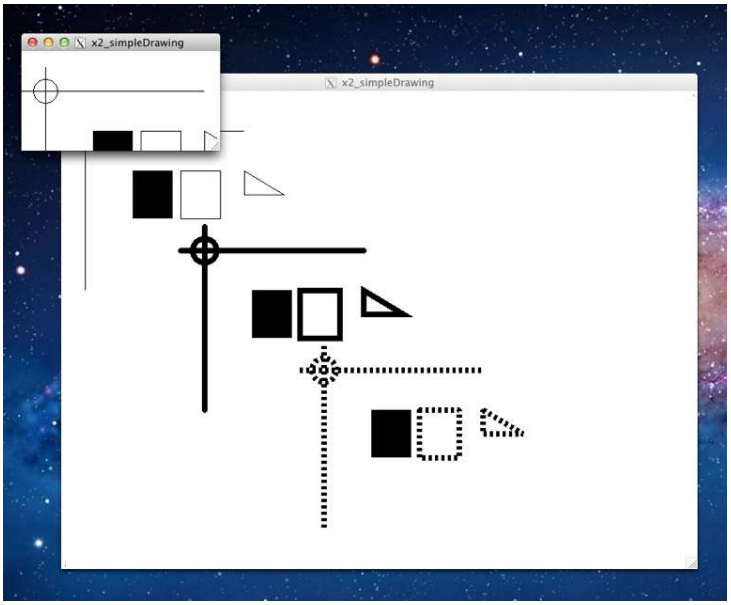
- Macintosh, Windows bundle Base Window System and Window Manager together
  - Very difficult for 3rd party to achieve alternative look and feel
- Trade-offs in approaches?
  - Look and feel…
  - Window management possibilities…
  - Input possibilities…

12

Multiple Windows in X

- Modified x3_events.cpp to produce x7_multiwindow.cpp:
  - initX: some one-time initialization moved to the main program. Added parameters for the root window, and location/size.
  - repaint: replaced with repaintWindow1 and repaintWindow2, each with slightly different code
  - main: declared xInfo1 and xInfo2. Called initX twice to initialize them.
  - modifications to the event loop (forthcoming)
- Result on next slide

13



14

Event Loop

```
XEvent event;

while( true ) {
        XNextEvent( display, &event );
        switch( event.type ) {
                case ButtonPress:
                        if (event.xany.window == xInfo1.window)
                                cout << "Got button press in window 1!\n";
                        else if (event.xany.window == xInfo2.window)
                                cout << "Got button press in window 2!\n";
                        break;
                case KeyPress:
                        if (event.xany.window == xInfo1.window)
                                cout << "Got key press in window 1!\n";
                        else if (event.xany.window == xInfo2.window)
                                cout << "Got key press in window 2!\n";
                        break;
                case Expose:
                        if (event.xany.window == xInfo1.window)
                                repaintWindow1(xInfo1);
                        else if (event.xany.window == xInfo2.window)
                                repaintWindow2(xInfo2);
                        break;
        }
}
```

15

Nested Windows

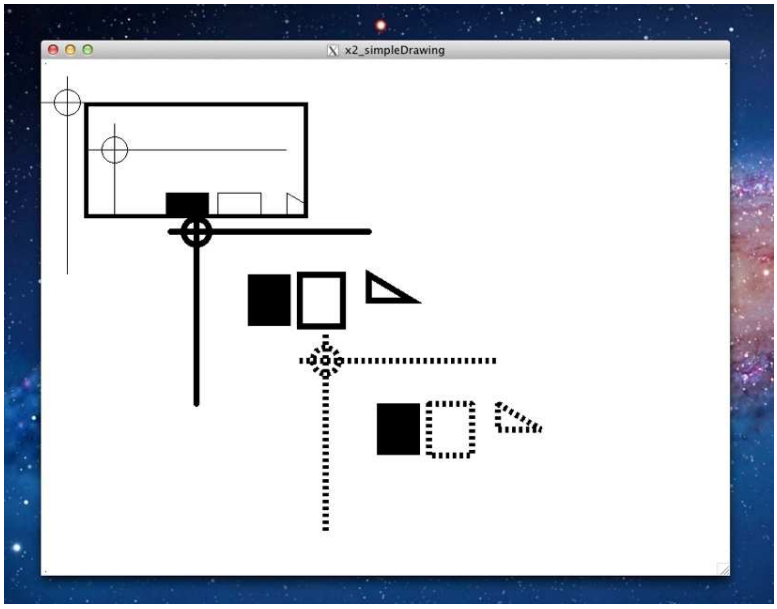
```
xInfo1.display = display;
xInfo1.screen = screen;
initX(argc, argv, xInfo1, DefaultRootWindow( display ),
100, 100, 800, 600);

xInfo2.display = display;
xInfo2.screen = screen;
initX(argc, argv, xInfo2, DefaultRootWindow( display ),
50, 50, 300, 200);
```

```
xInfo1.display = display;
xInfo1.screen = screen;
initX(argc, argv, xInfo1, DefaultRootWindow( display ),
100, 100, 800, 600);

xInfo2.display = display;
xInfo2.screen = screen;
initX(argc, argv, xInfo2, xInfo1.window,  // Change “root” window
50, 50, 300, 200);
```

16



17

Widgets

- Nested windows are the beginnings of widgets...
  - "Widgets" is a generic name for parts of an interface that have their own behavior: buttons, progress bars, sliders, drop-down menus, spinners, file dialog boxes, ...
  - Can have their own appearance
  - Receive and interpret their own events
  - Put into libraries (toolkits) for reuse
  - ...

18