

# Basic Single-Window X

X: The Basics

1

## Overview

- Basic X Architecture
- Drawing
- Events and the Event Loop

2

## Windowing Systems

- 1960's: Doug Englebart used a mouse-driven cursor with multiple (non-overlapping?) windows.
- 1973: Xerox PARC developed the Alto -- bit-mapped graphics, desktop metaphor, GUI. Heavily influenced PERQ, Apple Lisa/Mac, Sun workstations. Followed by Xerox Star. Alto stacked windows; Star mostly tiled.
- 1984: Apple Macintosh released. First commercially successful multi-window GUI.
- 1984: Work on X windowing system begins.
- 1985: Microsoft releases Windows 1.0; doesn't really take off until 1990 release of Windows 3.0.

3

## Valuable X References

- CS349 web site, see Resources page
  - *The X Window System*
    - background, design goals, basic architecture
    - PDF available online
  - The Xlib manual: <http://tronche.com/gui/x/xlib>
    - link and a PDF available on CS349 site
  - *Basic Graphics Programming With The Xlib Library* tutorial
  - Sample code

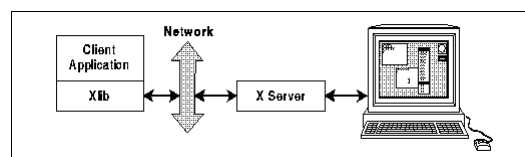
4

## Notes on this lecture

- All examples were implemented and tested on VM
  - Should be reasonably generic
  - You may need to tweak some things if you want to program on your own computer
  - ... But remember that assignments must run on VM
- Class examples use standard C programming language
  - No objects, no STL
  - You can use c or c++
- TAs will be looking at your sourcecode ...

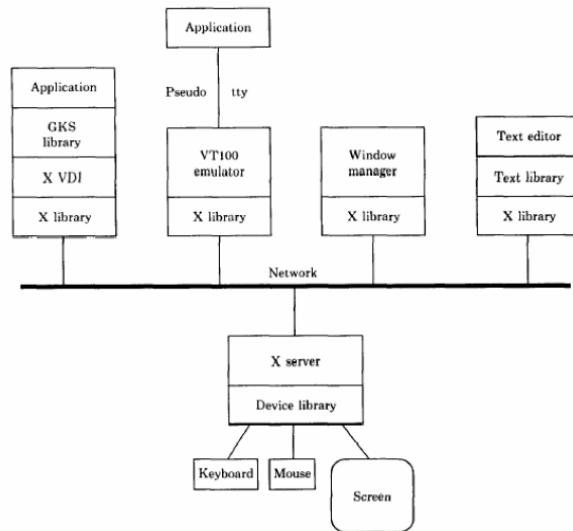
## XWindows System (1)

- Client-Server model of an application and UI
  - The client is the application that is running on a computer
  - The server is the display that is being drawn



## XWindows System (2)

- Goal was flexibility
  - Many clients (perhaps on multiple machines)
  - One display
- Generalization of model-view-controller architecture
  - Model = client application
  - View/Controller = terminal device running Xserver



## XWindows and XLib

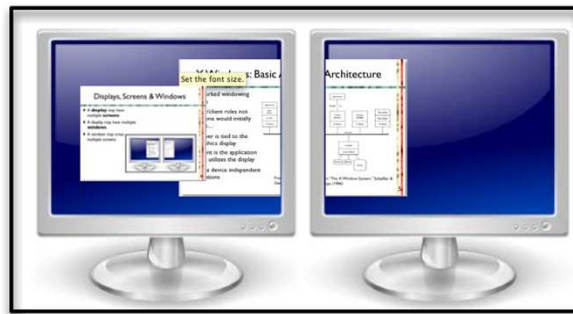
- To avoid implementing message passing every time a new program written, created XLib
  - Every client who “speaks” XLib could interact with any XServer
  - So what?
    - VERY NOVEL in the ‘80s.
    - Character terminals and proprietary drawing routines
- BUT
  - No common look and feel

## Programming XWindows

- Concepts
  - Display
  - Screen
  - Window
  - Graphics Context
  - Events

## Displays, Screens & Windows

- A **display** may have multiple **screens**
- A display may have multiple **windows**
- A window may cross multiple screens
- For now, we will work with a single window.



10

## Structure of a Basic GUI Program

1. Perform initialization routines.
2. Connect to the X server.
3. Perform X-related initialization.
4. While not finished:
  1. Receive the next event from the X server.
  2. handle the event, possibly sending various drawing requests to the X server.
  3. If the event was a quit message, exit the loop.
  4. Do any client-initiated work
5. Close down the connection to the X server.
6. Perform cleanup operations.

## A basic program

```

#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>

Display* display;

int main(){
display = XOpenDisplay(":0");
if (display == NULL) {
    printf("Cannot connect");
    exit (-1);
}
else{
    printf("Success!");
    /* do program stuff here */
    XCloseDisplay(display);
}
}

```

- Line 1:
  - Xlib header file
- Line 4:
  - A variable to hold the display
- main function
  - Try to open the display on this computer
    - Indicated by ":0"
  - If display is NULL
    - Print error message
  - Else
    - Connected to display
    - Do rest of program stuff here
  - Close display
- **g++ -o ex1 ex1.cpp -L/usr/X11R6/lib -lX11**

## Displaying a Window

```

#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
Display* display;
Window window;
int main( int argc, char *argv[] ){
    display = XOpenDisplay("");
    if (display == NULL) exit (-1);
    int screennum = DefaultScreen(display);
    long background = WhitePixel(display, screennum);
    long foreground = BlackPixel(display, screennum);
    window = XCreateSimpleWindow(display, DefaultRootWindow(display),
                                10, 10, 800, 600, 2, foreground, background);
    XSetStandardProperties(display, window, "x1_openWindow", "OW",
                          None, argv, argc, None);

    XMapRaised(display, window);
    XFlush(display);
    sleep(15);
    XCloseDisplay(display);
}

```

- Open display
- Get default screen
- Get background
- Get foreground
- Create window
- Map onto screen
- Flush XServer buffer
- Pause for 15 seconds
- Move all code to windowinit() function

## Code Review

- Review x1\_openWindow.cpp on resources page
- Same Functions/Procedures:
  - XOpenDisplay
  - DefaultScreen
  - XWhitePixel, XBlackPixel
  - XCreateSimpleWindow
  - XSetStandardProperties
  - XMapRaised
  - XFlush
- Difference is cleaner coding practice, but longer code as well.
- `g++ -o x1_openWindow x1_openWindow.cpp \`  
`-L/usr/X11R6/lib -lX11 -lstdc++`

## Drawing



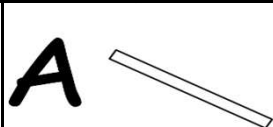
- Draw on a *canvas*
- Most basic primitive is `writePixel(x, y, colour)`
- But need higher level routines; need a model



15

## Drawing to a pixelated display

- Three different models for creating images:

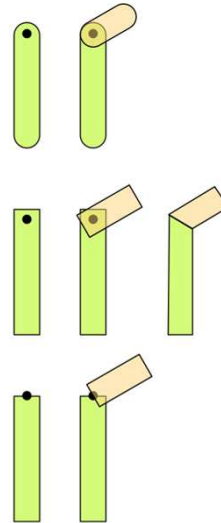
Pixel		<code>SetPixel(x, y, color)</code> <code>DrawImage(x, y, w, h, img)</code>
Stroke		<code>DrawLine(x1, y1, x2, y2)</code> <code>DrawRect(x, y, w, h)</code>
Region		<code>DrawLine(x1, y1, x2, y2)</code> <code>DrawRect(x, y, w, h)</code>

16



## Drawing

- Issues with lines:
  - Where are the end points?
  - How should the ends overlap?
  - What shape do the endpoints have? Is the line dashed or solid?
- How to communicate all the options?
- Observation: most choices are the same for multiple calls to drawLine.



17

## Graphics Context

- Gather all this information into a single structure that's passed to the drawing routines
  - X: GC structure
  - Java: Graphics object
- All graphics environments choose some variety of this approach.
- X: Graphics Context is stored on server
  - Use multiple contexts to reduce network traffic
  - But limited memory on server
  - Global to the application: need a policy

18

```

typedef struct {
int function;           // how the source and destination are combined
unsigned long plane_mask; // plane mask
unsigned long foreground; // foreground pixel
unsigned long background; // background pixel
...
int line_width;        // line width (in pixels)
int line_style;        // LineSolid, LineDoubleDash, LineOnOffDash
int cap_style;         // CapButt, CapRound, CapProjecting
int join_style;        // JoinMiter, JoinRound, JoinBevel
int fill_style;        // FillSolid, FillTiled, FillStippled,
FillOpaqueStippled
int fill_rule;         // EvenOddRule, WindingRule
int arc_mode;          // ArcChord, ArcPieSlice
...
Font font;             // default font
...
} XGCValues;

```

19

## Create and Use Graphics Context

```

GC gc = XCreateGC(display, window, 0, 0);
XSetForeground(display, gc, BlackPixel(display, screen));
XSetBackground(display, gc, WhitePixel(display, screen));
XSetFillStyle(display, gc, FillSolid);
XSetLineAttributes(display, gc, 1, LineSolid, CapButt, JoinRound);

...

XDrawLine(display, window, gc, x, y-30, x, y+200);
XFillRectangle(display, window, gc, x+60, y+50, 50, 60);

```

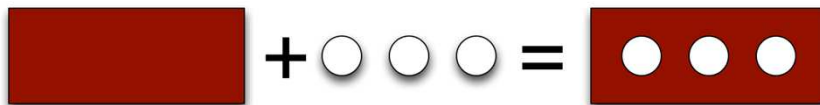
## Code Review

- `x2_simpleDrawing.cpp`
  - `initX` initializes three graphics contexts
  - `main` changed to call several procedures to draw
  - `drawPointsInCorners`
    - get window attributes (eg width and height)
    - use of `XDrawPoint`
  - `drawStuff`
    - parameters say which GC and where to draw
    - use of `XDrawLine`, `XDrawArc`, `XDrawRectangle`, `XFillRectangle`

21

## Painter's Algorithm

- The basic graphics primitives are... primitive.
- To draw more complex shapes...
  - Draw back-to-front, layering the image
  - Called "Painter's Algorithm"



22

## Display List

```
/* * An abstract class representing displayable things. */  
class Displayable{  
    public:    virtual void paint(XInfo &xinfo) = 0;  
};
```

23

```
/* * Display some text where the user clicked the mouse. */  
class Text : public Displayable{  
    public:  
    virtual void paint(XInfo &xinfo)  
    {  
        XDrawImageString( xinfo.display, xinfo.window, xinfo.gc,  
            this->x, this->y, this->s.c_str(), this->s.length() );  
    }  
    // constructor  
    Text(int x, int y, string s):x(x), y(y), s(s) {}  
  
    private:  
    int x;  
    int y;  
    string s;  
};
```

24

```
list<Displayable *> dList;    // list of Displayables

/* * Function to repaint a display list */
void repaint( list<Displayable *> dList, XInfo &xinfo) {
    list<Displayable *>::const_iterator begin = dList.begin();
    list<Displayable *>::const_iterator end = dList.end();
    XClearWindow( xinfo.display, xinfo.window );
    while( begin != end ) {
        Displayable *d = *begin;
        d->paint(xinfo);
        begin++;
    }
    XFlush( xinfo.display );
}
```

25

## Painting Advice

- Keep it simple
  - Clear the window and redraw everything
  - Get fancier (eg clipping, double buffering) only if you really need to for performance reasons
- Repaint when necessary -- but no oftener.
- Flush the buffer often enough -- but no oftener.
  - Unless you're debugging!

26

## Events Defined

- Event: noun: a thing that happens, especially one of importance. Example: the media focused on events in Egypt
- Event: a structure used to notify an application of an event's occurrence
- Examples:
  - Keyboard (key press, key release)
  - Pointer Events (button press, button release, motion)
  - Window crossing (mouse enters, leaves)
  - Input focus (gained, lost)
  - Window events (exposure, destroy, minimize)
  - Timer events

27

## Events: Why do we need them?

- Users have lots of options in a modern interface
- Need a uniform, well-structured, way to handle them
- Need to be able to handle any event, including those that aren't appropriate given the current state of the app
  - eg: clicking on a button that is currently disabled

28

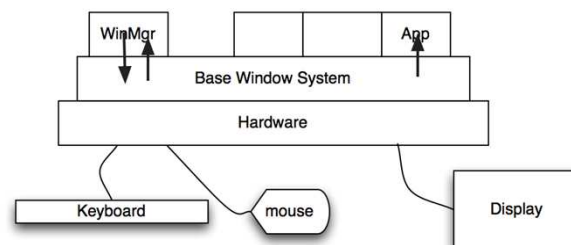
## Role of the X Server

- Collect event information
- Put relevant information in a known structure
- Order the events by time
- Decide to which application/window the event should be dispatched
- Deliver the event.

29

## Collecting Events

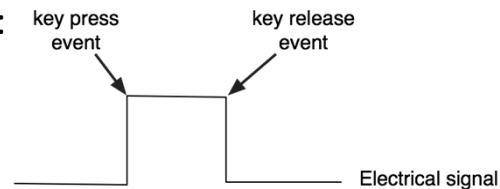
- Some events come from the user via the underlying hardware; some from the window manager.



30

## Collecting Events

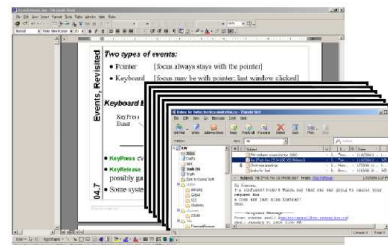
- Key events:
  - Key press: put char on screen
  - Key release: usually ignored except to tell if key is being held for auto-repeat purposes
  - Scan codes
- Mouse button events:
  - press and release are differentiated
- Mouse motion events:
  - mouse moved
  - mouse dragged



31

## Collecting Events

- Damage Events
  - May be a long sequence of damage events
  - Responding to them all can bog the system down
  - X includes a field indicating a minimum number that are yet to come. Ignore damage event unless that field is 0.
  - Or perhaps until a timeout



32



## Collecting Events

- Don't always need all of the events. (Why?)
- XSelectInput(display, window, ButtonPressMask | KeyPressMask | ExposureMask )
- Defined masks: NoEventMask, KeyPressMask, KeyReleaseMask, ButtonPressMask, ButtonReleaseMask, EnterWindowMask, LeaveWindowMask, PointerMotionMask, PointerMotionHintMask, Button1MotionMask, Button2MotionMask, ..., ButtonMotionMask, KeymapStateMask, ExposureMask, VisibilityChangeMask, ...

33

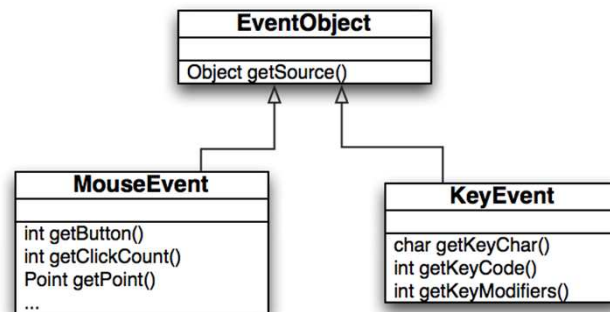
## Known structure : X

- X uses a C union
  - typedef union {
  - int type;
  - XKeyEvent xkey;
  - XButtonEvent xbutton;
  - XMotionEvent xmotion;
  - // etc.
  - }
- Each structure contains at least the following
- typedef struct {
  - int type;
  - unsigned long serial; // sequential #
  - Bool send\_end; // from SendEvent request?
  - Display\* display;
  - Window window;
  - } X\_\_Event

34

## Known Structure: Java

- Java uses an inheritance hierarchy
- Each subclass contains additional information, as required (not shown)



35

## Order by Time

- Use an Event Queue to maintain a list in order.
- In X, applications get the next event with
  - XNextEvent(Display\* display, XEvent\* evt)
    - Gets and removes the next event in the queue.
    - If empty, it blocks until another event arrives.
  - XPending(Display\* display)
    - How many events are pending in the event queue?
    - Never blocks.

36

## Responding to Events

- Take the first event off the event queue
- Handle it
- Repeat

```
XEvent event;
while (true)
{ XNextEvent(xinfo.display, &event);
  switch (event.type)
  { case Expose:
    if (event.xexpose.count == 0)...
    break;

    case ButtonPress:
    // handle event
    break;

    case ...:
    }
  repaint(...);
}
```

37

## Code Review

- Review x3\_events.cpp
  - XSelectInput
  - eventLoop
  - handleKeyPress

38

## Animation

- Goals:
  - Move things around on the screen
  - Repaint 30-60 times per second
  - Make sure events are handled on a timely basis
  - Don't use more CPU than necessary

39

```

XEvent event;
unsigned long lastRepaint = 0;
while( true ) {
    if (XPending(xinfo.display) > 0) {
        XNextEvent( xinfo.display, &event );
        switch( event.type ) {
            case MotionNotify:
                handleMotion(xinfo, event);
                break;
            ...
        }
    }
    unsigned long end = now();
    if (end - lastRepaint > 1000000/FPS) {
        handleAnimation(xinfo);
        repaint(xinfo);
        lastRepaint = now();
    }
    if (XPending(xinfo.display) == 0) {
        usleep(1000000/FPS - (end - lastRepaint));
    }
}

```

40

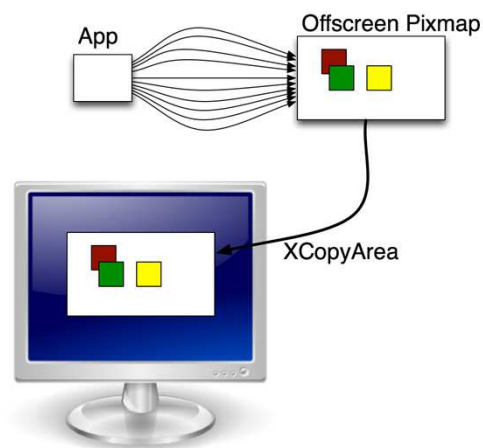
## Demo Code

- x4\_animate.cpp
  - Event loop conforms to previous slide
  - New events used

41

## Double Buffering

- Flickering: when an intermediate image is on the display
  - e.g.: Clear, then redraw strategies
- Solution:
  - Create an offscreen image buffer
  - Draw to the buffer
  - Copy the buffer to the screen as quickly as possible



42

## Double Buffering Logistics

- Creating the off-screen buffer:  

```
int depth = DefaultDepth(display, DefaultScreen(display));
pixmap = XCreatePixmap(display, window,
                       width, height, depth);
```
- Drawing on the buffer:  

```
XFillRectangle(display, pixmap, gc, x, y, width, height);
```
- Copying from buffer to window:  

```
XCopyArea(display, pixmap, window, gc,
          0, 0, width, height, // region of pixmap to copy
          0, 0);             // top left corner of destination
```
- Freeing an unused off-screen buffer:  

```
XFreePixmap(display, pixmap);
```

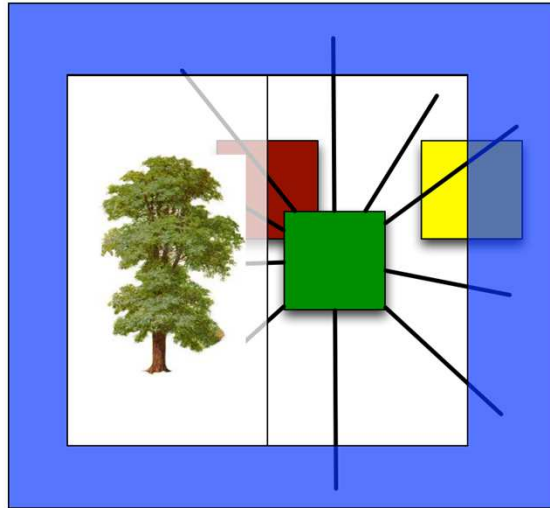
43

## Double Buffering Issues

- Window size changes
  - ResizeRedirectMask doesn't do what you think
- Memory; location
- Smaller-than-window uses

44

## Clipping



45

## Code Demo

- `x6_clip.cpp`
  - `XSetClipRectangle`

46

## Summary

- Basic X architecture (client, server, network)
- Windows: opening, disposing
- Drawing
  - Models (pixel, stroke, region)
  - graphics contexts
  - Painter's Algorithm; Display lists
- Events (structure, selecting, event loop, etc)
- Animation
- Double Buffering
- Clipping