

# CS 349

## X Windows

Byron Weber Becker  
Winter 2012



CS 349 Winter 2012 1

## 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.



CS 349 Winter 2012 2

## X Design Criteria 1

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"



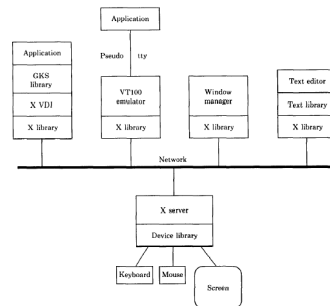
CS 349 Winter 2012 3

## 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”

## X Windows: Quick Background

- Networked windowing system
- Server/client roles not what one would initially expect...
  - Server is tied to the graphics display
  - Client is the application that utilizes the display
- Enables *device independent* applications



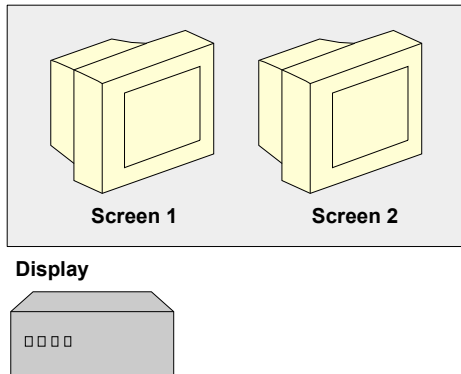
From "The X Window System," Scheifler & Gettys (1986)

## Invaluable X Refs

- The X Window System (PDF available online)
- The Xlib manual: <http://tronche.com/gui/x/xlib>
- Sample code linked in A01

# Basic X Programming Elements

- Display
- Screen
- Window
- Graphics Context
- Other X objects



# Basic X Programming Elements

- Display: The structure used for interacting with the server
  - Manages all the screens available on the system (X allows 1+ screens)
  - Used to obtain screens, send messages back and forth between client/server
- Screen: Represents a physical device (display)
- Window: A window within the screen
- GC (Graphics Context): Used to set drawing options (e.g., drawing color)

# Programming X

- Basic functions for creating/displaying a window:
  - Display \*XOpenDisplay(char \*display\_name)
  - int XDefaultScreen(Display \*d)
  - Window XCreateSimpleWindow(Display \*d, Window parent, ...)
  - void XSetWMProperties(Display \*d, Window w, ...)
  - void XMapRaised(Display \*d, Window w)
  - void XDraw...(Display \*d, ...)
  - void XFlush(Display \*d)

# Getting User Input & Drawing

- Contrast batch/conversational interfaces with GUI's
  - How do we need to structure our code for batch/conversational interfaces?
  - How do we need to change this structure for a GUI?



# Events + Event Loops

- Since user can initiate action at anytime (keyboard, mouse input, button press) need to be flexible and check whether any action has taken place
- *Events* encapsulate actions and changes to system
- Look for new events using *event loops*
- Basic event loop algorithm in pseudocode:

```
while (true)
  if (event_available())
    e = get_event(...);
    handle_event(e);
```



# Events in X

- Need to tell X what types of events we want to receive
  - Mouse, keyboard, exposure (repaint needed), etc.
- Use `XSelectInput(...)`:

```
XSelectInput(display, window,
             ButtonPressMask
             | KeyPressMask
             | ExposureMask );
```
- Will notify us when mouse button is pressed, key is pressed, or the window has been “exposed” (needs updating)



# Events in X

- XEvent is event type (a union in C)
- Getting an event
  - XNextEvent()
    - *Blocks* until a new event is received
  - XPending()
    - Returns number of events in event queue (non-blocking)
- Based on event type, process accordingly
  - Repaint window
  - Interpret mouse, keyboard input
  - ...

# Painting in X

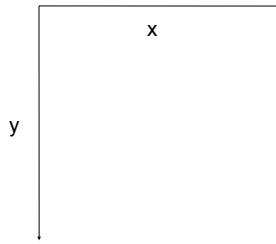
- Will be notified whenever window has been “damaged” via Expose event
  - Need to repaint our window according to current state of application
  - Then flush the display

# Painting in X

- To fill a rectangle: XFillRectangle(...)
- To draw a rectangle: XDrawRectangle(...)
- Other shapes:
  - XDrawPoint(...)
  - XDrawLine(...)
  - XDrawArc(...)
  - XDrawLines(...)

# Painting

- Coordinate system in X:
  - y increasing in value down
  - x increasing in value right



# Painting in X

- Text:
  - `XDrawImageString()`
- Remember to flush display to see changes!
  - `XFlush ()`

# Painting in X

- How can we specify the colour or other attributes of what we paint?
- Can specify this information via a Graphics Context (GC)
  - `XCreateGC()`
  - `XFreeGC()`

## More Sophisticated Drawing

- How could we draw a “brick” with holes in it?

## Painter’s Algorithm

- To draw “holes” or shapes with portions “cut out” of them, can draw background of object, then draw other shapes on top of it
- To draw a brick
  - Create graphics context with foreground color
  - Create graphics context with background color
  - Fill rectangle using foreground color GC
  - Fill set of arcs using background color GC (the holes in the brick)

## Putting it all together...

- Sample code on web site (see A01 Resources):
  - hello.cpp (mouse events)
  - xdemo\_str.c (key events, clipping)
  - xdemo\_2.c (expose events)
- Up next... hello.cpp

```

/* Header files for X functions */
#include <X11/Xlib.h>
#include <X11/Xutil.h>

using namespace std;
const int Border = 5;
const int BufferSize = 10;

/* Information to draw on the window. */
struct XInfo
{
    Display *display;
    Window window;
    GC gc;
};
... /* skip lots of stuff */

/* Execution starts here... */
int main ( int argc, char *argv[] )
{
    XInfo xinfo;

    initX(argc, argv, xinfo);
    eventloop(xinfo);
}

```

22

```

void initX(int argc, char *argv[], XInfo &xinfo)
{
    XSizeHints hints;
    unsigned long background, foreground;
    int screen;

    xinfo.display = XOpenDisplay( "" );
    if ( !xinfo.display )
        error( "Can't open display." );

    screen = DefaultScreen( xinfo.display );
    background = WhitePixel( xinfo.display, screen );
    foreground = BlackPixel( xinfo.display, screen );

    hints.x = 100;
    hints.y = 100;
    hints.width = 400;
    hints.height = 300;
    hints.flags = PPosition | PSize;
    xinfo.window = XCreateSimpleWindow( xinfo.display,
        DefaultRootWindow( xinfo.display ),
        hints.x, hints.y, hints.width, hints.height,
        Border, foreground, background );
    XSetStandardProperties( xinfo.display, xinfo.window, "Hello1", "Hello2", None,
        argv, argc, &hints );
}

```

## Initialize X:

- Open the display
- Get the default screen
- Create a window
- Set window properties
  - Set up the Graphics Context
  - Say what kind of events we're interested in
  - Show the window to the user

23

```

xinfo.gc = XCreateGC( xinfo.display, xinfo.window, 0, 0 );
XSetBackground( xinfo.display, xinfo.gc, background );
XSetForeground( xinfo.display, xinfo.gc, foreground );

XSelectInput( xinfo.display, xinfo.window,
    ButtonPressMask | KeyPressMask | ExposureMask );

XMapRaised( xinfo.display, xinfo.window );
}

```

## Initialize X:

- Open the display
- Get the default screen
- Create a window
- Set window properties
  - Set up the Graphics Context
  - Say what kind of events we're interested in
  - Show the window to the user

24

```

void eventloop(XInfo &xinfo)
{ XEvent event;   KeySym key;   char text[BufferSize];
  list<Displayable *> dList;    // list of Displayables

  while( true )
  { XNextEvent( xinfo.display, &event );
    switch( event.type ) {

      case Expose:
        if ( event.xexpose.count == 0 )
          repaint( dList, xinfo);
        break;

      case ButtonPress:
        dList.push_front(new Text(event.xbutton.x, event.xbutton.y, "Urrp!"));
        repaint( dList, xinfo );
        break;

      case KeyPress:
        int i = XLookupString( (XKeyEvent *)&event, text, BufferSize, &key, 0 );
        if ( i == 1 && text[0] == 'q' )
          error( "Terminated normally." );
        break;
    }
  }
}

```

Get events from the user and decide how to handle them.

25

```

class Displayable
{ public:
  virtual void paint(XInfo &xinfo) = 0;
  virtual ~Displayable() {}
};

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) {}
  virtual ~Text() {}

private:
  int x;
  int y;
  string s;
};

```

Data structures to store information used to repaint the display.

26

```

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++;
  }

  /* flush our instructions to the display */
  XFlush( xinfo.display );
}

```

Repaint the entire display.

27

# Compiling X...

To compile X, use gcc and link in Xlib library:

```
gcc -o xdemo xdemo.c -L/usr/X11R6/lib -lX11
```

Outputs a file named xdemo

1. Compiles xdemo.c
2. Adds /usr/X11R6/lib as a search path for linking libraries
  - This command should work in a typical X11 installation
3. Links in library X11
4. You may also need to use the -I switch to indicate where your header files are located
5. On some machines, add -lstdc++



# The Makefile

```
.phony: all
all:
    gcc -o xdemo xdemo.c -L/usr/X11R6/lib \
    -lX11
```



# Hints

- XMapRaised can be split into two calls, allowing you to prepare what the user sees before the window is displayed.
- Drawing:
  - Keep it simple unless you can't: simply clear the screen and redraw everything.
  - Use the painter's algorithm.
  - In 2D, don't bother to check if the object is going to be visible. The GPU may be able to push the pixels faster than you can push the commands.
  - Don't repaint more frequently than you need to -- flickers. But do it often enough -- at least every response to the user.
  - XFlush: call if things aren't showing up that should, but don't call more often than needed.

