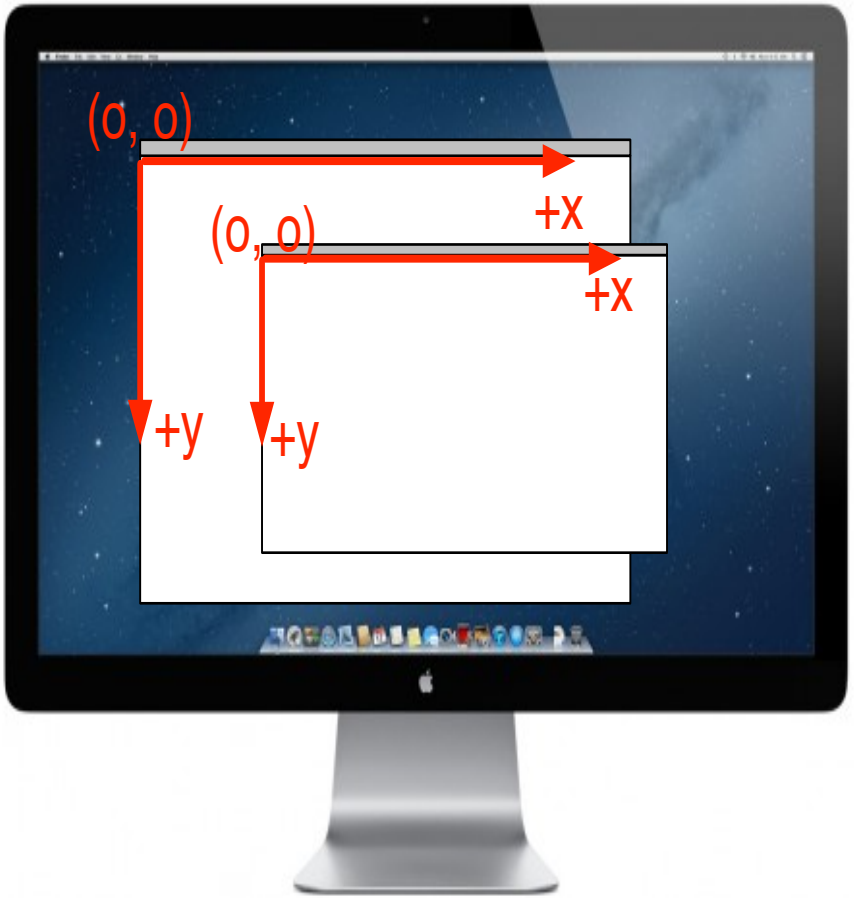
Basic GUI Drawing

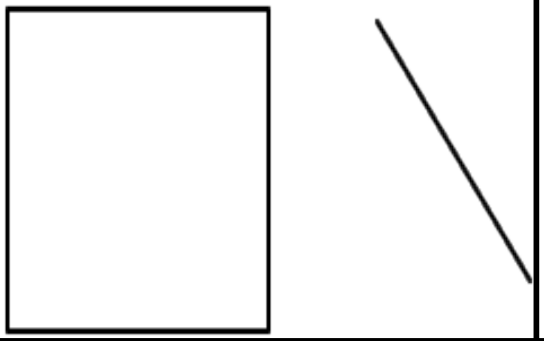
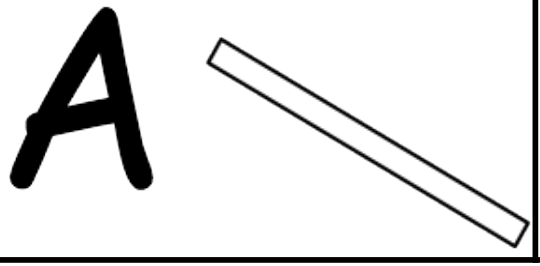
Drawing models, Graphics context, Display lists, Painter's Algorithm

Local Coordinates

- Any modern OS manages multiple windows
- Base Window System (BWS) manages:
  - where the window is located, whether it is covered by another window, etc...
  - enables drawing using *local* coordinate system for window

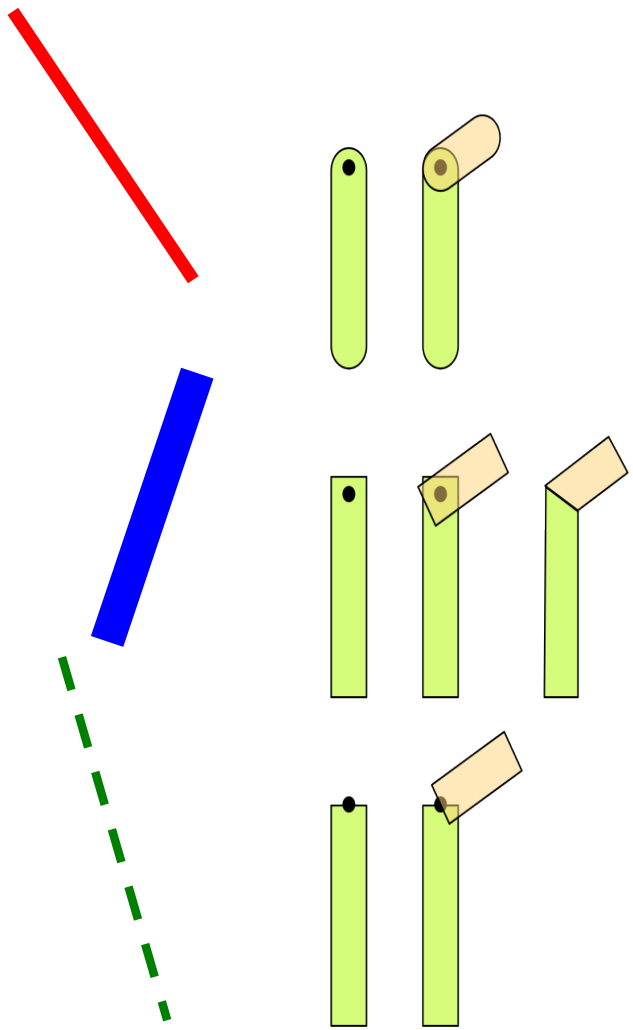


Three different *conceptual* drawing models:

Pixel		SetPixel(x, y, colour) DrawImage(x, y, w, h, img)
Stroke		DrawLine(x1, y1, x2, y2, colour) DrawRect(x, y, w, h, colour)
Region		DrawLine(x1, y1, x2, y2, color, thick) DrawRect(x, y, w, h, colour, thick, fill) DrawText("A", x, y, colour)

Drawing Options

- Lots of options for drawing
  - e.g. drawLine(x1,y1,x2,y2)
    - what colour?
    - how thick?
    - dashed or solid?
    - where are the end points?
    - should the ends overlap?
- Observation: most choices are the same for multiple calls to drawLine()
- How to communicate all the options?
  - lots of parameters?
  - functions that are more specific?
  - some other way?



Graphics Context (GC)

- Solution: Gather all drawing options into a single structure and pass it to the drawing routines
  - In X, this is the Graphics Context (GC) structure
- All graphics environments use variation on this approach
  - Java/C#: Graphics Object
  - OpenGL: Attribute State
- In X, the graphics context is stored on X server
  - Can switch between multiple saved contexts to reduce network traffic (but limited memory on X server)
  - There is a default (global) graphics context shared by all applications.
- With modern applications, we don't separate client application and X server UI routines, but this assumption of repeated attributes (and GC) still applies

Drawing in X Windows (1)

Step 1: setup display and open a window

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>

int main() {
    // setup local display
    Display * display = XOpenDisplay( "" );
    int screen = DefaultScreen( display );

    // setup window
    int win_x=0, win_y=0, win_width=500, win_height=500, border=5;
    Window window = XCreateSimpleWindow(
        display,                            // display where window appears
        DefaultRootWindow( display ),   // window's parent in window tree
        win_x, win_y, win_width, win_height, border,    // parameters
        BlackPixel(display, screen),    // window border colours
        WhitePixel(display, screen)     // window background colour
    );
    XMapRaised( display, window );
    XFlush(display);
    sleep(1);           // hack to let server get set up before drawing
```

Drawing in X Windows (2)

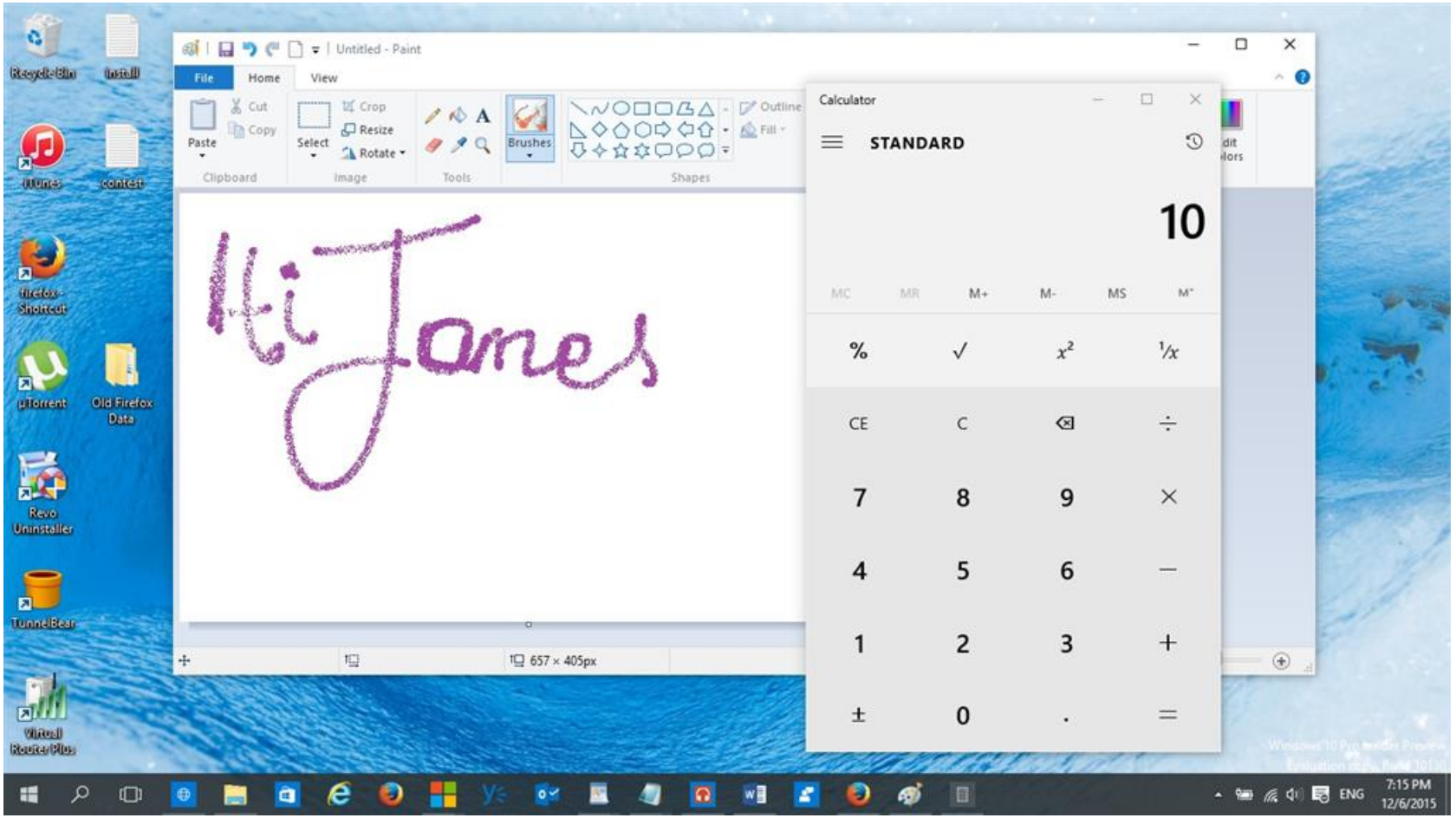
Step 2: setup GC and use to draw shapes

```
    // setup GC
    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);

    // draw in window using GC
    int x1=0, x2=100, y1=0, y2=100, width=75, height=75;
    XDrawLine(display, window, gc, x1, y1, x2, y2);
    XDrawRectangle(display, window, gc, x1, y1, width, height);
    XFillRectangle(display, window, gc, x1, y1, width, height);
    XFlush(display);

    // close when complete
    XCloseDisplay(display);
}
```

See *drawing.cpp* for a structured example of this.

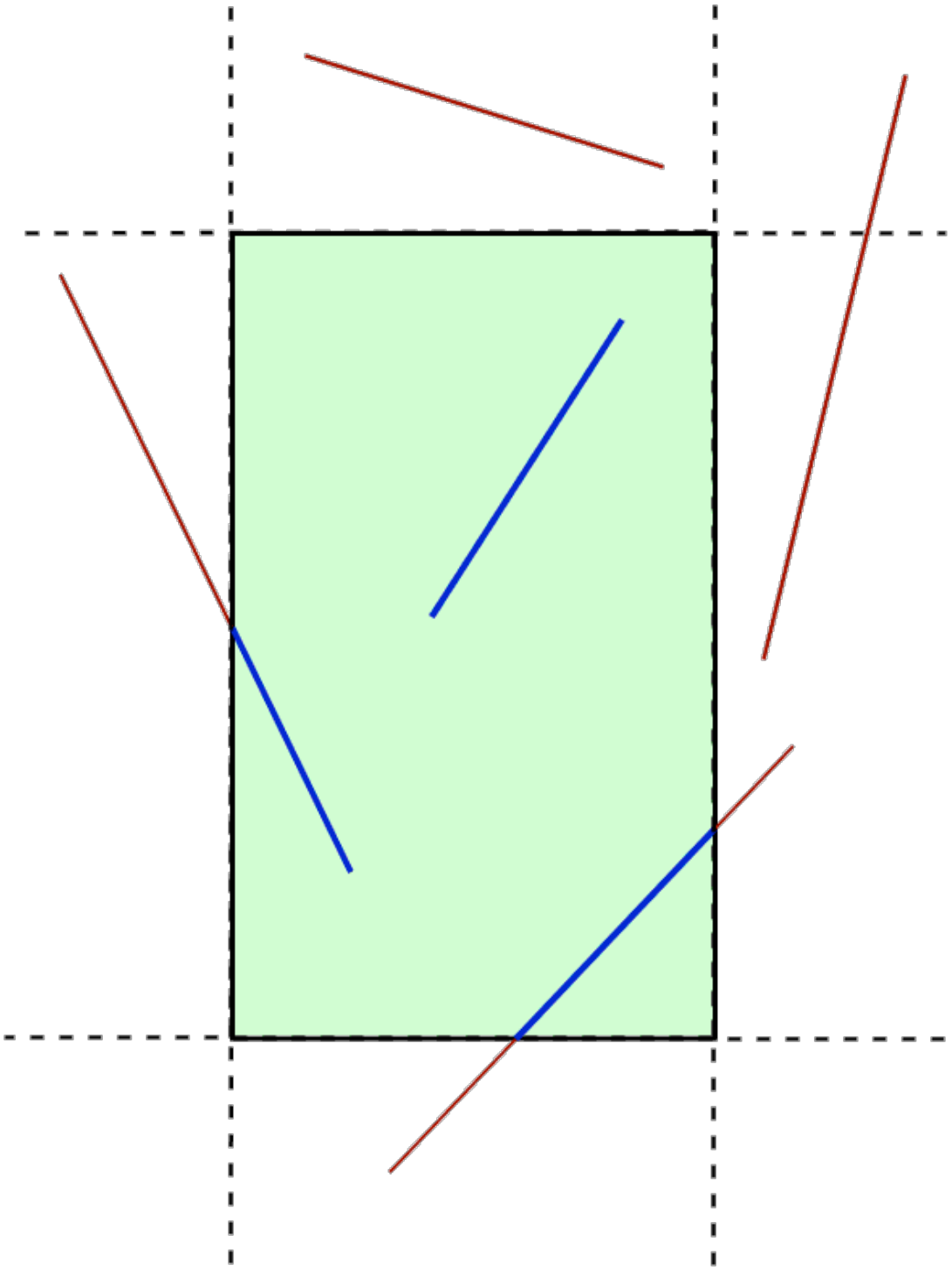
Simplifying Drawing With Clipping

- What are some other problems that might arise when trying to draw on a computer display?



Clipping and the Painter's Algorithm

Clipping: More on this later …

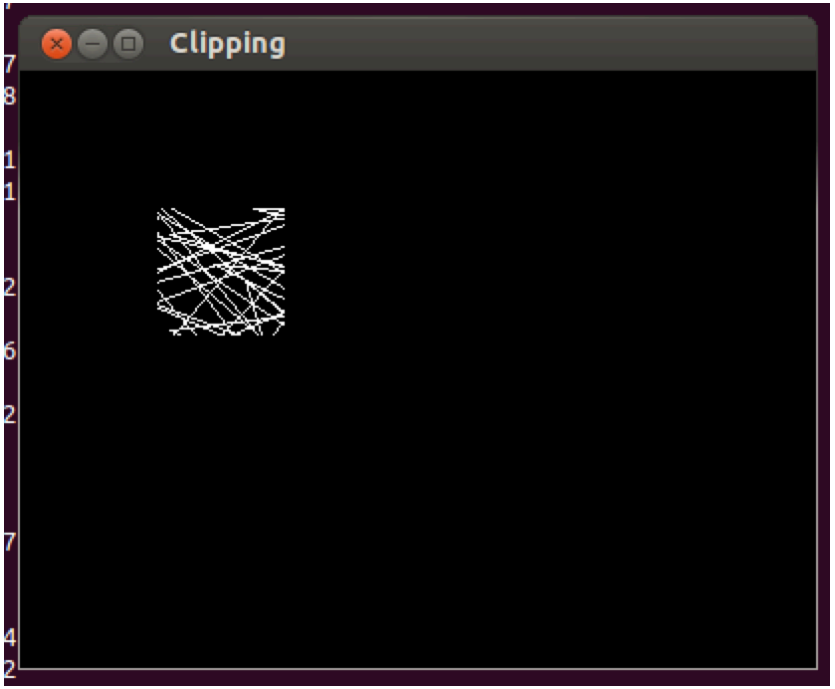


Clipping in X Windows

- XSetClipMask
- XSetClipRectangles

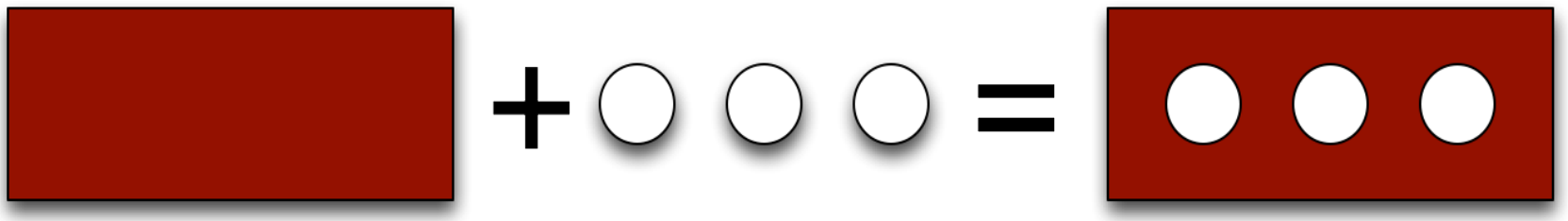
```
if (!is_clipping)
    XSetClipMask(display, gc, None);
else
   XSetClipRectangles(
      display, gc, 0, 0, &clip_rect, 1,Unsorted);
```

See *clipping.cpp* for a working demo of this code.



Painter's Algorithm

- The basic graphics primitives are simple and only support drawing basic shapes.
- To draw more complex shapes (composite):
  - Draw back-to-front, layering the image
  - Called "Painter's Algorithm"
- Also allows "stacking" shapes to simulate depth



Painters Algorithm Analogy

Fast and with music: http://youtu.be/ghHxTjXAnM4

Using the Painters Algorithm

- How to implement the painter's algorithm:
  - Describe shapes that you wish to paint on-screen as "displayable" objects
    - Each object needs to be capable of displaying itself on the screen
    - Implement as base class with a "paint" method
    - Define derived classes for different kinds of displayables: text, game sprites, etc.
  - Keep an ordered list of "displayables"
    - Order the list back-to-front.
  - To repaint
    - Clear the screen (window).
    - Repaint everything in the display list, in back-to-front order.

Displayable Base Class

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

Displayable Text

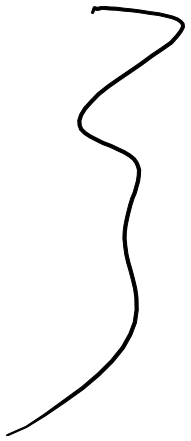
```
/* Display some text  */
class Text : public Displayable {
    public:
        virtual void paint(XInfo &xinfo) {
            XDrawImageString( xinfo.display,
                xinfo.window, xinfo.gc, this->x,
                this->y, this->s.c_str(),
                this->s.length() );
        }

        // constructor
        Text(int x, int y, string s):x(x), y(y), s(s)  {}

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

Displayable Polyline

```
/* Display a polyline */
class Polyline : public Displayable {
    public:
        virtual void paint(XInfo& xinfo) {
            XDrawLines(xinfo.display, xinfo.window,
            xinfo.gc, &points[0],
            points.size(), CoordModeOrigin );
        }

        //
        Polyline(int x, int y) {  add_point(x,y); }

        void add_point(int x, int y) {
            XPoint p; // XPoint is a built in struct
            p.x = x;   p.y = y;
            points.push_back(p);
        }

    private:
        vector <XPoint> points;      // XPoint is a built in struct
};
```

Displaying the Display List

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

dList.push_front(new Paddle(…));
dList.push_front(new Ball(…));
dList.push_front(new Background(…));

/* 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 );
}
```



Display List Examples

- These examples all make use of the display list:
  - *doublebuffer.cpp*
  - *animation.cpp*
  - *eventloop.cpp*
- We'll examine these in more depth as we discuss events (next!).

Summary

- Models (pixel, stroke, region)
- Graphics contexts
- Drawing primitives
- Painter's algorithm
- Display lists