A Brief intro to X11 Programming

This is meant to be a simple overview of X11 programming. It will contain the basics necessary to create a single window X program. All of the information of this will pertain simply to programming with the X11 libraries which come standard with X11R[5/6]. It goes without saying, but this page is under constuction at all times.

Topics:

Overview of X11 Programming

X is a fun-filled graphical system for most flavors of UNIX. There are many toolkits for programming X, but learning to use the X Library calls is a tried and true method which yields highly portable code. This document is meant to be a short introduction to some of the things I have learned about it myself. It will probably always remain "under construction". Strap yourself in and hold on! This train's bound for glory, this train's coming through and one of these days these trains are going to walk all over you!

Header Files for X

Here are the header files required to do much of anything in X. The order **is** important:

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

These files can typically be found in /usr/include/X11. I highly recommend looking though the header files themselves, but don't get caught up on trying to figure out everything at once! The best way to learn them is to continually consult them as you continue to work with X.

Important X Variables

X has a number of important variables for handling windows:

```
Display *dis;
int screen;
Window win;
GC gc;
```

The display points to the X Server. The screen refers to which screen of the display to use. Setting up the connection from the X Client to the X Server typically involves a line like: setenv DISPLAY my.machine.where.ever:0. The my.machine.where.ever is tied in with the Display* and the screen is connected with the :0 part of the variable. The Window controls the actual window itself (duh! ;^). And the GC is the graphics context.

The graphics context has a lot to do with how things are displayed/drawn in the window. Different masks can be set, etc. and you can get pretty funky with it.

Typically calls to get input and output to the window use one or more of the above variables. See the appropriate sections for details.

Here is a simple routine to create a simple X Windows. It assumes that the above variable have been declared globally:

```
void init_x() {
        /* get the colors black and white (see section for details) */
        unsigned long black,white;

        /* use the information from the environment variable DISPLAY
           to create the X connection:
        */
        dis=XOpenDisplay((char *)0);
        screen=DefaultScreen(dis);
        black=BlackPixel(dis,screen),   /* get color black */
        white=WhitePixel(dis, screen);  /* get color white */

        /* once the display is initialized, create the window.
           This window will be have be 200 pixels across and 300 down.
           It will have the foreground white and background black
        */
        win=XCreateSimpleWindow(dis,DefaultRootWindow(dis),0,0,
                200, 300, 5, white, black);

        /* here is where some properties of the window can be set.
           The third and fourth items indicate the name which appears
           at the top of the window and the name of the minimized window
           respectively.
        */
        XSetStandardProperties(dis,win,"My Window","HI!",None,NULL,0,NULL);

        /* this routine determines which types of input are allowed in
           the input.  see the appropriate section for details...
        */
        XSelectInput(dis, win, ExposureMask|ButtonPressMask|KeyPressMask);

        /* create the Graphics Context */
        gc=XCreateGC(dis, win, 0,0);

        /* here is another routine to set the foreground and background
           colors _currently_ in use in the window.
        */
        XSetBackground(dis,gc,white);
        XSetForeground(dis,gc,black);

        /* clear the window and bring it on top of the other windows */
        XClearWindow(dis, win);
        XMapRaised(dis, win);
};
```

This section is really only about 10 lines of code if you take out all of the goofy comments, and it can be reused with only minor modifications over and over. Some may object to the "cut-and-paste" method of reusing code, but it works... This snippet of code will pop up an X Window on your default server. Most of the calls have a lot of functionality, but this is enough to get you started.

Another important task is closing the window correctly:

```
void close_x() {
/* it is good programming practice to return system resources to the
   system...
*/
        XFreeGC(dis, gc);
        XDestroyWindow(dis,win);
```

```
        XCloseDisplay(dis);
        exit(1);
}
```

For more information about the routines above, see the following sections (where applicable) and/or try the man pages. They are a great resource.

Handling Input

Input in X is "event driven". This is a big fancy-pants word given to programs which wait for stuff to happpen before doing stuff. Here is a very simple event driven program:

```
main () {
        char tmp[512];
        while (1) {
                scanf("%s",tmp);
                printf("%s",tmp);
        }
}
```

As you can see, "event driven" programming is nothing new or special. The way that events are handled in X is, however, a bit on the fun side. Basicly you have an XEvent. This can be keypress, mouse button press/release, mouse movement, etc. Anything which can be used as input to an X Window.

How can one variable handle all these different types of events you ask? The answer is that it is a big, complex structure! Don't be afraid! Even though it is big and scary, you don't have to learn every aspect of it to use it.

The key concepts to using the XEvent structure are:

1. setting up the input masks
2. creating an instance of the XEvent
3. checking for events
4. handling events.

Here is code snippet which gives an example of using XEvent's. It assumes that a call has been made to init_x() (see previous section) where the call:

```
XSelectInput(dis, win, ExposureMask|ButtonPressMask|KeyPressMask)
```

was made. This call sets up the masks. Now only Expose, ButtonPress and KeyPress events are regarded as valid. The Expose event type refers to when a portion of the window which had been obscured by another window becomes visisble again. This is usually handled by some sort of user routine to repaint the window. Note: the redraws must be handle **by the user**! The below example assumes a routine redraw() has been written. Also notice how the KeyPress event is handled. Pressing a 'q' will exit the program.

```
XEvent event;           /* the XEvent declaration !!! */
KeySym key;             /* a dealie-bob to handle KeyPress Events */
char text[255];         /* a char buffer for KeyPress Events */

/* look for events forever... */
while(1) {
        /* get the next event and stuff it into our event variable.
           Note:  only events we set the mask for are detected!
        */
        XNextEvent(dis, &event);

        if (event.type==Expose && event.xexpose.count==0) {
        /* the window was exposed redraw it! */
                redraw();
        }
        if (event.type==KeyPress&&
            XLookupString(&event.xkey,text,255,&key,0)==1) {
        /* use the XLookupString routine to convert the invent
```

```
                   KeyPress data into regular text.  Weird but necessary...
                */
                        if (text[0]=='q') {
                                close_x();
                        }
                        printf("You pressed the %c key!\n",text[0]);
                }
                if (event.type==ButtonPress) {
                /* tell where the mouse Button was Pressed */
                        printf("You pressed a button at (%i,%i)\n",
                                event.xbutton.x,event.xbutton.y);
                }
        }
};
```

The printf statements will, of course, print to terminal you ran the X program from.

There are a lot of Event types and masks. Read the man pages on: XSelectInput and XNextEvent for more information. Also see /usr/include/X11 and search for NoEventMask. This will give you a complete list of all the different sorts of events there are. It is worth noting that XNextEvent will block until another event comes through. So if no events occur, your program just sits there. XCheckWindowEvent is similar to XNextEvent (though it takes different arguments), but it returns a true (1) or a false (0) about whether or not an event occured. If an event did occur it fills an XEvent structure with info about the event. In either case, the program execution continues unabated.

Drawing in the Window

Now that you have a nice spiffy windows up, and it can accept input, the next step is to respond with output! Here are some basics to get you started:

- XDrawLine(dis,win,gc, x1,y1, x2,y2);
- XDrawArc(dis,win,gc,x,y, width, height, arc_start, arc_stop);
- XDrawRectangle(dis, win, gc, x, y, width, height)
- XFillArc(dis,win, gc, x, y, width, height, arc_start, arc_stop);
- XFillRectangle(dis,win,gc, x, y, width, height);

Note: I using the notation that the variables:

```
Display *dis;
Window win;
GC gc;
```

have been declared and initialized (ala init_x()). All of these calls are pretty straight forward, but a look at the man pages will give you more useful information. By appending an 's' to the end of each of the above you can get routines to draw more than one of the same (e.g.: XDrawArcs(...)). Something to keep in mind, is that anything that is drawn uses the current foreground and background colors. See Working with Colors for more info!

Placing Text in the Window

Placing text in a window is very similar to drawing in a window. The routine to remember is:

```
XDrawString(dis,win,gc,x,y, string, strlen(string));
```

Pretty simple, huh? There are other string routines, which you can learn about by consulting the man page for XDrawString.

Working with Colors

Colors can get pretty complicated. In the interest of simplicity, I will assume that you are working with a display which can handle the standard 256 colors (e.g.: SVGA). With X it is possible to sense the type of display and adjust the color routines appropriately, but why worry about that? If you are writing some serious application and you are concerned about interoperability (EGA, CGA, TrueColor, Real Color, etc) then read the man pages like a big boy/girl.

Ahem, now on with the discussion. For the purpose of this introduction I am going to stick with two basic ideas: shared and private colormaps. Colormaps is another word for palettes. I am only going to talk about 256 color colormaps.

Getting the colors you want from a shared colormap is really pretty simple. The easiest way is to consult the rgb.txt file (usually found as /usr/X11/lib/X11/rgb.txt). It list all the string names associated with all the colors of the rainbow! Here is how you use them:

```
void get_colors() {
        XColor tmp;

        XParseColor(dis, DefaultColormap(dis,screen), "chartreuse", &tmp);
        XAllocColor(dis,DefaultColormap(dis,screen),&tmp);
        chartreuse=tmp.pixel;
};
```

Once again, this routine assumes you have called the init_x() routine. It also assumes chartreuse is a globally declared unsigned long int to hold the pixel value which refers to the closest match to chartreuse in the current palette. As you can probably imagine, you don't always get exactly what you ask for. This is why many application use a private colormap (or have it as an option).

Creating a private colormap is a bit more involved, but if you want the colors to be perfect in your window, it is the easiest way.

```
void create_colormap() {
        int i;
        Colormap cmap;
        XColor tmp[255];

        for(i=0;i<255;i++) {
                tmp[i].pixel=i;
                tmp[i].flags=DoRed|DoGreen|DoBlue;
                tmp[i].red=i*256;
                tmp[i].blue=i*256;
                tmp[i].green=i*256;
        }
        cmap=XCreateColormap(dis,RootWindow(dis,screen),
                DefaultVisual(dis,screen),AllocAll);
        XStoreColors(dis, cmap, tmp,255);
        XSetWindowColormap(dis,win,cmap);
}
```

This initializes an array of 256 XColors into which we store the RGB values, set the pixel and flag values. Then the colormap is created (note: the funky call to DefaultVisual, is a clue about how to handle different display types ;^). Next the colors are loaded into the colormap and finally the colormap is set for our window. If you set up a private colormap, you may also want to add a line like: XFreeColormap(dis,cmap); to your close_x() routine.

Now that your know how to get the pixel values for the color you want (either with a share colormap or private), here is what you can do with this information:

```
        XSetForeground(dis,gc,chartreuse);
        XDrawArc(dis,win,gc,x,1, 17,17,0,23040);
```

You basicly use the pixel value to set the foreground (or background) color and then draw in the color of your choice! What could be simpler?

Compiling your Code

Now for the fun part. Compiling your code! Let's say you have written an X program called hi.c. In it you placed the #include lines with care in hope that a window soon would appear! What you do to compile it depends on how your system is setup. Try each of the following until you find which one(s) will work for you. I have listed them in order of commonality:

1. gcc hi.c -o hi -lX11
2. gcc hi.c -o hi -I /usr/include/X11 -L /usr/X11/lib -lX11
3. gcc hi.c -o hi -I /sowhere/else -L /who/knows/where -lX11

OK, I admit the third one is wacky, but it is meant to indicate that the -lX11 flag by itself was not sufficient and that the file X.h is not in /usr/include/X11 and that libX11.so was not in /usr/X11/lib. In this case you will need to find them, install them, have them installed, or give up like a lamer! Try using whereis and find to locate them (maybe locate too...).

Before running your program don't forget to set your DISPLAY environment variable! In csh: setenv DISPLAY my.ip.number.here:0. In others try: DISPLAY=my.ip.number.here:0; export DISPLAY.

Conclusions

This document is a very basic introduction. This is its first draft (2/9/96). I hope to add more information (XImage, .xbm files, cursors, etc) when I have more time later. As I continue my own process of learning X, I hope to expand these pages further.

Here is a very simple sample program for your to try out.

Origional Author of this page: bhammond@blaze.cba.uga.edu
