Notes on creating graphical output using X-11 graphics

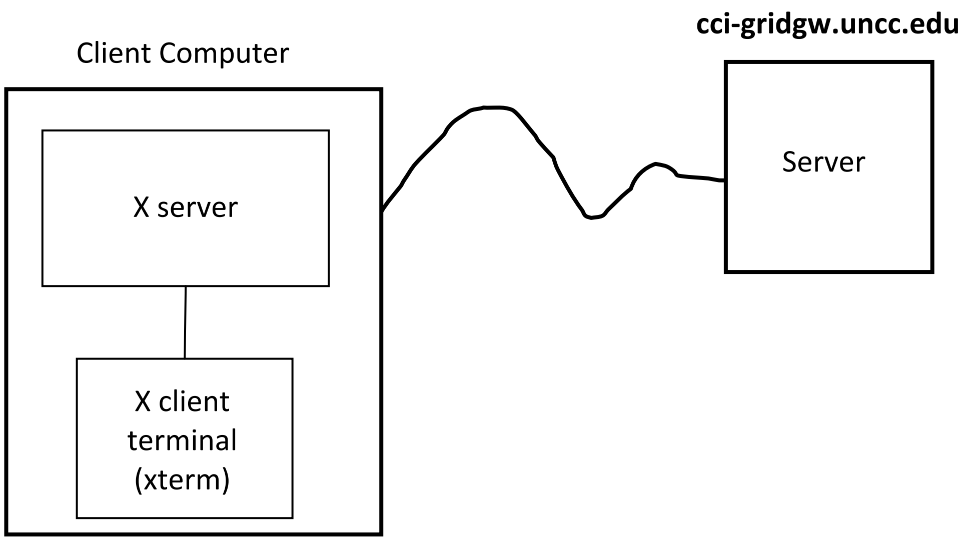
Ubuntu Virtual Machine

B. Wilkinson, Sept 4, 2014

Assignments and projects may require graphical output, which is especially relevant for problems such as the heat distribution problem to show the heat contours or the *N*-body problem to show movement of bodies. When executing programs on a remote server such as **cci-gridgw.uncc.edu**, the graphical output has to be forwarded to the client computer for display. In these notes, we will explain how to create and forward basic X11 graphics.[1] Note these notes only apply to using the provided Ubuntu virtual machine (or Ubuntu directly installed) and an interactive connection a server (not batch through a job scheduler.)

X-11 graphics

X-11 refers to version 11 of the X Window System first developed in the 1980's. It is chosen because it is part of the Linux distribution, it is relatively easy to write simple graphics, and easy to forward to a client. It is said to be "almost universally" used on Unix-like systems [Wikipedia X Window System]. The ability to forward the graphics is critical in our application. X is a client-server model that relies on an X-server running on the client:



An X-server will be available and usually already running if you are using a Linux distribution on the client.[2] In other cases, an X server would needs to be installed, such as Xming on a Windows system.

First test your X11 environment by running the X11 clock program in the background with the command:

```
xclock &
```

[1] Note: Rather than use the basic X 11 libraries directly as described here, one could use other graphics libraries that use X 11 forwarding such as Cairo if the libraries are installed.

[2] startx command will initialize an X session if needed.

X-11 drawing code

Before calling any X 11 routine to draw a figure, you have to first do a rather long sequence of code to set up the X window environment, which is given below and also in the file **sample.c** in the virtual machine directory **~ParallelProg/X11/**:

```
#include <stdio.h>
#include <stdlib.h>

#include <X11/Xlib.h>         // X11 library headers
#include <X11/Xutil.h>
#include <X11/Xos.h>

#define X_RESN 800            // x resolution
#define Y_RESN 800            // y resolution


int main (int argc, char **argv ) {

/* --------------------------- X11 graphics setup ------------------------------ */

       Window    win;                 // initialization for a window
       unsigned int   width, height, // window size */
                 win_x,win_y,         // window position
                 border_width,        // border width in pixels
                 display_width, display_height,      // size of screen
                 screen;                             // which screen

       char           *window_name = "My graphics program", *display_name = NULL;
       GC             gc;
       unsigned long  valuemask = 0;
       XGCValues      values;
       Display   *display;
       Pixmap    bitmap;
       XPoint    points[800];
       FILE      *fp, *fopen ();
       char      str[100];

       XSetWindowAttributes attr[1];

       if (  (display = XOpenDisplay (display_name)) == NULL ) {   // connect to Xserver
          fprintf (stderr, "Cannot connect to X server %s\n",XDisplayName (display_name) );
          exit (-1);
       }

       screen = DefaultScreen (display);                   // get screen size, not used here
       display_width = DisplayWidth (display, screen);
       display_height = DisplayHeight (display, screen);

       width = X_RESN;                                            // set window size
       height = Y_RESN;
       win_x = 0;     win_y = 0;                                  // set window position

       border_width = 4;                                          // create opaque window
       win = XCreateSimpleWindow (display, RootWindow (display, screen),
                             win_x, win_y, width, height, border_width,
                             BlackPixel (display, screen), WhitePixel (display, screen));

       XStoreName(display, win, window_name);

       gc = XCreateGC (display, win, valuemask, &values);         // create graphics context

       XSetBackground (display, gc, WhitePixel (display, screen));
       XSetForeground (display, gc, BlackPixel (display, screen));
       XSetLineAttributes (display, gc, 1, LineSolid, CapRound, JoinRound);

       XMapWindow (display, win);
       XSync(display, 0);

/* ---------- End of X11 graphics setup, continue with application code, sample given here ------- */

       usleep(100000);                              // some delay appears necessary
```

```
        XClearWindow(display, win);                  // clear window for next drawing
        XSetForeground(display,gc,(long) 0xDC143C);  // color of foreground (applies to object to be drawn)

        //XDrawPoint (display, win, gc, 400, 400);   // draw point at location x, y in window

        XFillArc (display,win,gc,400,400,50,50,0,23040);     // draw circle of size 50x50 at location 400,400

        XFlush(display);                                     // necessary to write to display

        usleep(10000000);                                    // provide a delay beween each drawing

        return 0;
}
```

Useful X11 routines

Once the code to set up the X window environment is in place, you get down to the business of drawing an image, using routines such as

```
XClearWindow(display, win);                                  // clear window for next drawing
XSetForeground(display,gc,(long) color);                     // color of foreground (object to be drawn)
XDrawPoint (display, win, gc, x, y);                         // draw point at location x, y in window
XFillArc (display,win,gc,x,y,width,height,angle1,angle2);    // draw arc/circle
XFlush (display);                                            // necessary to write to display
```

The long integer color is a 24-bit number that specifies the color, as give in the Wikipedia entry for X-11 color names. For example, 0xDC143C would give Crimson. (Note: the number can be given as a hexadecimal number.) To create a circle with **XFillArc()**, the start and end angles would be 0 and 23040 (degrees x 64). You drawing routines can be repeated in a loop to display movement. Include **usleep()** or **sleep()** to get the appropriate speed for the motion.

Compiling C code with X-11 graphics

You will need to compile with the X11 libraries in addition to any other libraries such the Math libraries, e.g. to compile **sample.c**: [3]

```
        cc -o sample sample.c -lm –lX11
```

On some systems (Macs e.g.), you may need to provide the full path to the X11 libraries:

```
        cc -o sample sample.c -lm -L/usr/X11R6/lib -lX11
```

Make file

A make file is most convenient especially if the compilation command is getting long. For example, a file called **makefile** with the contents:

Target name, used when invoking make with make **<target>**

```
Hello: hello.c
        cc -o hello hello.c -lm

Sample: sample.c
        cc -o sample sample.c –lm –lX11
```

Dependencies – here check source file has been updated. Will not recompile if not necessary.

Command line to execute

[3] Order of libraries on command line is important. Libraries must follow the source file. Symbols are resolved from left to right.

will compile either a regular C program, **hello.c**, or an X11 program **sample.c** with the commands:

```
make Hello
make Sample
```

Note the commands in the make file, (cc … in the example) MUST begin with a tab character.

Using a remote server

To execute X11 programs on a remote server and see the graphical output on your client computer, one has to forward the graphics. To forward X-11 graphics from a terminal, include the –X option:

```
ssh cci-gridgw.uncc.edu –X –l <username>
```

You will also need to specify your username on the remote server with the **–l** option if it different to the local computer.

Test X11 forwarding

Test the connection and forwarding by running **xclock** in the background:

```
xclock &
```

Servers without an external Internet connection

On the UNCC cluster, internal nodes such as **cci-grid05** are not accessible directly and one needs to first ssh into **cci-gridgw.uncc.edu** remembering to forward X11 graphics (-X option) and then ssh from **cci-gridgw.uncc.edu** to the internal node, again remembering to forward X11 graphics, e.g. from **cci-gridgw.uncc.edu** to **cci-grid05**:

```
ssh ccigrid05 –X
```

Test the connection and forwarding by running xclock in the background. The clock graphics should forward back through the two servers and to your client machine. (You would get two clocks if you also forwarded one from the first server.)

Servers that are not accessible from off-campus. A similar procedure is necessary to reach servers such as **coit-grid06.uncc.edu** from off-campus.

Useful references

Wikipedia X Window System http://en.wikipedia.org/wiki/X_Window_System
Wikipedia entry: X-11 color names   http://en.wikipedia.org/wiki/X11_color_names
XLib Manual http://tronche.com/gui/x/xlib/
X11 graphics routines http://tronche.com/gui/x/xlib/graphics/

Some key X11 routines: (See manual for more details on functions)

XOpenDisplay()	Open a connection to the X server that controls a display **(XCloseDisplay()** closes a display or disconnect from the X server)
XCreateSimpleWindow()	Creates a window
XCreateGC()	Creates a graphics context and returns a GC
XSetBackground()	Specifies background for display/GC
XSetForeground()	Specifies foreground for display/GC
XMapwindow()	Maps window to display
XSync()	Flushes output buffer and then waits until all requests have been received and processed by X server.
XClearWindow()	Clears entire area in specified window
XFlush()	Flushes the output buffer
XSetLineAttributes()	Sets line width and line styles for specified GC