A Compact Windowing System for the Curl Environment
by
Edward Alan Otte

Submitted to the
Department of Electrical Engineering and Computer Science

July 16, 1998

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer [Electrical] Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Curl is a programming language for creating web content. It is capable of running under a Linux operating system using Xwindows for a graphics interface. There are applications of Curl which do not need such a large and complex graphics system as Xwindows. The compact windowing system eliminates much of the unnecessary functionality of Xwindows while implementing the necessary components. The system makes use of the video hardware using svgalib for Linux. The system also make use of the Freetype TrueType font library to use TrueType fonts for text rendering. This eliminates one incompatibility between the Linux and MS Windows versions of Curl.

Thesis Supervisor: Steve Ward
Title: Professor of Electrical Engineering and Computer Science
# TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION ........................................................................... 4  
CHAPTER 2 OVERVIEW ................................................................................. 6  
CHAPTER 3 IMPLEMENTATION ....................................................................... 8  
  GENERAL ........................................................................................................ 8  
  WINDOWS, PIXMAPS, AND XIMAGES ............................................................... 10  
     Initialization .................................................................................................... 10  
     Structure ........................................................................................................ 10  
     Window Management ....................................................................................... 13  
     Copying Areas ................................................................................................. 16  
  GRAPHICS ....................................................................................................... 17  
     GC .................................................................................................................. 17  
     Color ............................................................................................................. 18  
     Drawing ......................................................................................................... 19  
  EVENTS, THE KEYBOARD, AND THE POINTER ........................................... 26  
     Events ........................................................................................................... 26  
     The Keyboard .................................................................................................. 28  
     The Mouse ..................................................................................................... 29  
  FONT HANDLING ............................................................................................. 31  
CHAPTER 4 CONCLUSION ............................................................................ 34  
  FUTURE WORK ................................................................................................. 34  
BIBLIOGRAPHY ............................................................................................. 35  
APPENDICES .................................................................................................. 36  
  APPENDIX A XWINDOW FUNCTIONS AND STRUCTURES FROM XWINDOWS.C ......................................................... 36  
  APPENDIX B MY.H .......................................................................................... 38  
  APPENDIX C WINSTUFF.C ............................................................................... 47  
  APPENDIX D GC.C .......................................................................................... 62  
  APPENDIX E COLOR.C .................................................................................... 64  
  APPENDIX F GRAPHICS.C .............................................................................. 66  
  APPENDIX G EVENT.C ................................................................................... 86  
  APPENDIX H FONT.C ..................................................................................... 104  

# TABLE OF FIGURES

Figure 1 Representation of winnode structures and relationships ............................................. 13  
Figure 2 Examples of rect_intersect computation ................................................................. 15  
Figure 3 Reference picture for equation 1 ........................................................................... 20  
Figure 4 Illustration of divisions in triangle and polygon functions ..................................... 21  
Figure 5 Reference figure for equations 2 and 3 .................................................................. 22  
Figure 6 Illustration of miter joined lines ............................................................................ 23  
Figure 7 Actual polygons filled by lines function ................................................................ 24  
Figure 8 Division of area by fill_arc .................................................................................. 25
Chapter 1 Introduction

Curl\textsuperscript{1} is new programming language for creating web content. The range of documents Curl can produce runs from simple text to complex interactive programs, for Curl is essentially a powerful object-oriented programming language specialized to produce web documents. A central feature of Curl is that it was designed to be a "gentle slope system." This means that it is simple to make the transition from unsophisticated text documents to complex interactive pages via a series of small incremental steps. In other words, there are no discontinuities or large steps in the learning curve of the language. Also, since Curl is a full programming language, it alone is sufficient to produce complex web documents, making unnecessary the need to use many languages such as HTML, Java, etc, in one document. Using only Curl also makes sharing information between modules a simple affair.

One potential use of the Curl environment is in the MASC\textsuperscript{2} (Modular Appliance Super Computer) Information Appliances System. These MASC computers are small, modular computer cards being developed by the Information Appliances Project at LCS. Their purpose is to create a seamless, computerized network of appliances sharing data and control. The gentle slope paradigm of Curl makes it interesting for educational use because the capabilities grow nicely with the user's knowledge. This allows all levels of users to be accommodated by the system with it proving neither too challenging nor too limited for any of them. The MASC system provides a method for delivering the Curl environment into the home easily.

A likely candidate operating system for the MASC cards is Linux. Curl currently has versions which run under Linux, SunOS, and Microsoft Windows 95 or NT. The graphical interface on the Linux platform is Xwindows. Linux is a free implementation of a Unix system,

\textsuperscript{1} See http://www.cag.lcs.mit.edu/curl/
\textsuperscript{2} See http://www.sds.lcs.mit.edu/InfoAppl/start.html
and can be modified to operate the MASC cards. However, it and its Xwindows system are complex, and a more simple specialized windowing system for Curl would be more manageable in a MASC-Curl implementation. The entirety of the Linux Xwindows system is simply not necessary, and a smaller windowing system will suffice while conserving resources of the cards.

The purpose of this project is to implement a compact window system to enable Curl to run under Linux without using Xwindows. Many components of Xwindows are not necessary, such as provisions for it to run with many applications or over a network. Xwindows is structured such that an application calls procedures in xlib, the library that is linked to the application. Xlib acts as a client to another program, the X server, and communicates with the server using the X protocol. The overhead of this protocol and client-server is eliminated in the compact system by having the entire graphics system linked to the Curl application.
Chapter 2 Overview

The new system uses svgalib to interface the low level hardware such as the video card, the keyboard, and mouse. Svgalib is a low level Linux graphics library which provides some drawing primitives in the C programming language. While most of Curl is implemented in Curl itself, the graphics interface is in C. The interface to the display used in the Linux version of Curl is xlib for Xfree86. Specifically, there are about sixty host functions defined in Curl and implemented in C. The new system implements these host functions by using svgalib functions. The svgalib library supports only those video driver chipsets for which a driver has been written. These drivers give the library the ability to set the chip specific registers in order to set up the video chips for the specific svga modes necessary.

In particular, eight of these host functions deal with font handling. These are implemented by incorporating the FreeType\(^3\) library functions into the window system. The FreeType library handles TrueType fonts, which are not handled by the Linux Xwindows. Making a Linux windowing system which supports TrueType fonts solves an incompatibility problem between the Linux and Windows implementations of Curl. FreeType allows text to be rendered in high quality, and once rendered the text can be displayed using svgalib.

The host functions are implemented in a file called xwindows.c. This file is the only link between Curl and Xfree86. There were several possibilities for how to go about implementing these host functions without using Xwindows. They could be redone from scratch, or they could be left alone and those functions from xlib could be implemented. The latter course was chosen for several reasons. Firstly, it would allow as little modification as possible to the Curl system code. Secondly, the xlib functions were better specified and documented, and therefore they were more easily constructed to do exactly as expected. Lastly,

\(^3\) See http://www.freetype.org
by implementing the xlib functions the window system may prove useful in the future for some other application which uses a similar subset of Xwindow functionality. Many Xwindow applications may be able to be used with this system by simply adding a few if any additional xlib function re-implementations.

The implementation is divided into six different modules, each of which handles a different subset of tasks. These modules are found in the files graphics.c, font.c, event.c, winstuff.c, gc.c, and color.c. Most of the actual drawing functions are implemented in graphics.c. The font handling, rendering, and drawing functions can be found in font.c. Xwindows is an event driven system. This means that information from the window system such as keyboard or mouse input or events such as a mouse entering a window or a window needing to be redrawn are delivered to the application in an event queue which is queried whenever necessary. The code handling this event system as well as the keyboard and mouse can be found in event.c. The window management functions are contained in winstuff.c. This includes creating, destroying, and manipulating the windows as well as copying areas to and from them. Xwindow objects Windows, Pixmaps, and Images are all handled here, as they are very similar in their workings. The functions dealing with creating and changing the Xwindow object graphics contexts are in gc.c. Finally, the functions dealing with color allocation, parsing, and handling are implemented in color.c. These functions all include a common header file, my.h, which contains constant definitions and function prototypes. This is the interface which is to be included in the final xwindows.c which will provide the link between Curl and the new system.
Chapter 3 Implementation

General

The Curl code calls about seventy different functions from xlib and uses about ten structures defined by it. Most of the functions are implemented by the new system, while some are eliminated from xwindows.c or remain implemented only as stubs that simply return doing nothing. The latter functions either are not needed or their functionality was eliminated to simplify the system. The following box shows these functions.

**ELIMINATED:**

- XStringListToTextProperty
- XTextProperty
- XErrorEvent
- XGetErrorDatabaseText
- XGetErrorText
- XSetErrorHandler

**STUBS:**

- XSync
- XFlush

The eliminated functions deal with one of two things. One group handles error codes generated and returned by Xwindows, and the other deals with the window manager. Errors are not generated and sent as events in the new system, so these functions are not necessary. A window manager is a separate application which controls the positioning of windows on a display, among other related tasks. Since there is no other application running and communicating with the window system, there is no window manager, per se. The few necessary window management tasks are incorporated into the system itself.
In Xwindows, the XCreateFontCursor function returns a Cursor object which can be associated with a window, and that cursor is displayed when the mouse pointer is inside that window. This functionality is not yet implemented in the new system. Currently, a simple arrow pointer is displayed in all windows at all times. The Cursor variable type is currently assigned to an integer using a \textit{typedef} statement in \textit{my.h}. The XFlush and XSync functions force the server to complete all tasks it has been given and send all events that may have been sent. Since the new system is not a client server model, the situations where these are necessary never occur, so they simply return immediately.

Xwindows uses several concepts to abstract the hardware devices and capabilities that are used. These include identifiers such as Display, Screen, Visual, Colormap, and Depth. These variable types are used in many of the xlib functions as arguments or return values. The Display and Screen types identify the specific physical monitor and associated data which should be affected by the function call. This is necessary in Xwindows because a single xserver has the capability of controlling many such physical devices across a network. However, the new system is limited to one monitor on one machine, so these values are no longer relevant. The function prototypes have remained the same as the Xwindows versions to eliminate the need to make many changes to the code in \textit{xwindows.c}, but these arguments are dummy ones and hence are not used. (Except for one exception, XLoadQueryFont, which will be discussed later.) The variable types are assigned to integers or pointers to integers using a \textit{typedef} statement in \textit{my.h}.

Xwindows also is capable of using many colormaps, depths (bits used per pixel to represent color), and visual type. The visual type is the way pixels are translated to produce a color. The new system uses only one of each of these, so a variable type indicating which is to be used is not necessary. The Display also has a name of type string, and an integer connection number which are unnecessary. The functions which return these values are defined as macros
to return zero or one, since their values are irrelevant. The macro which returns the default depth returns the `svgalib` macro `BITS PER PIXEL`. The macro which returns the display name returns an empty string.

**MACROS:**

- `XdisplayName`
- `XdefaultScreen`
- `XdefaultDepth`
- `XdefaultColormap`
- `ConnectionNumber`

### Windows, Pixmaps, and XImages

**Initialization**

Normally, in Xwindows, the connection to the xserver is established before any other window action by calling `XOpenDisplay`. The necessary initialization steps for the new system are found in this function. It creates and initializes the root window and other structures. It calls functions to initialize the font engine and event structures. It also sets up the video mode and allocates and sets the GraphicsContext for the physical screen.

**Structure**

Xwindows uses three types of objects to hold graphics. These three types are Windows, Pixmaps, and XImages. Windows represent the actual windows the user sees on the screen, and thus contain the visible graphics. Pixmaps are similar to Windows, except they don’t represent anything other than a container for graphics. Windows and Pixmaps are both of type `Drawable`, which means they can be passed as an argument to a drawing function and be drawn into. The differences between the two are that Pixmaps do not have attributes such as location on the screen, border color and width, parent, etc. Also, in Xwindows, the graphics data in Window objects is not guaranteed to remain in the memory of the server, or cached. If the window is not
visible on the screen (unmapped or obscured by other windows), the graphics data may be
thrown away. If it is later needed and not available, and Expose event will be sent to the
application, indicating that the Window should be redrawn. Pixmaps, however, always remain
available in the servers memory until explicitly destroyed.

In the client server Xwindows setup, there are distinct memory areas for the server and
the client application. In fact, the server may be running on an entirely different machine. The
data for Windows and Pixmaps reside in the server’s memory, and thus can not be accessed
directly by the application, but only through xlib function calls which communicate with the
server. However, when an XImage object is created, its data resides in the application’s
memory area, and thus can be accessed directly for fast manipulation. In the new system, which
is not a client server system, there is only one memory area. Thus, both Drawables and
XImages are in the application memory area. Thus, the chief distinction between Pixmaps and
XImages does not occur. Now the only difference is that Pixmaps, being Drawables, have
fields to supports the attributes of Windows, which are also Drawables.

Another distinction is that XImages are usually handled and passed around using
pointers to their structures. Drawables, however, are in Xwindows simply identifiers for server
held objects. In the new system they are actually defined as pointers to a Drawable_Struct
structure. They should still be treated as identifiers and only be modified by calling one of the
provided functions in order to keep the system’s internal state up to date and correct.

In the new system, unlike Xwindows, Window contents are always cached when
unmapped or obscured by other windows. The only times that a Window needs to be redrawn
due to lost graphics data are when is it first created and mapped or when it is resized.

The XImage structure has three fields, integer width and height and a pointer to an
unsigned char to hold the graphics data. A Drawable_Struct structure contains the same three
fields as an XImage but with several more. One field is an integer flag indicating if the
Drawable is a Window or a Pixmap. If its value is 0, the object is a Pixmap and the remaining fields are unused. If the flag’s value is 1, the object is a Window. For a Window, there are other relevant fields. There are two pair of integer coordinates, one indicating the Window’s position on the screen relative to its parent Window, and one indicating its absolute position relative to the upper left corner of the screen. Coordinates increase moving down and to the right. The structure also has a flag to indicate if the Window has been mapped. If it is mapped, it is displayed on the screen if not covered by other windows. It has fields containing the width and color of the border that should surround the window. It also contains pointers to structures to hold additional attributes, XWindowAttributes and XSetWindowAttributes. Finally, it has a pointer to a winnode structure which gives information about the window’s parent and children windows and where it is in the stacking order relative to its siblings.

Now a point to avoid possible confusion. Both Xwindows and vgagl, a graphics library used with svgalib, have objects called graphic contexts. These are distinct and serve distinct purposes. An Xwindows graphics context holds information needed to completely specify drawing requests, such as which color or font to use, what the line width should be, which logical operator should be applied to the operation, or what the clipping area should be. A vgagl graphics context contains information such as the size of the current area to be drawn into, be it the screen or an area of memory. With vgagl, virtual contexts can be defined and set to send svgalib’s primitive drawing operations into system memory areas instead of directly to the screen. This method is used to draw into Pixmaps, Windows, and XImages. The vgagl graphics contexts will be referred to as GraphicContexts and the Xwindows graphics contexts will be referred to as GC’s to avoid confusion.
Every Window has associated with it a winnode structure. This structure contains a pointer back to the Window, and pointers to winnodes for the Window's parent, its list of children, its last child, and its next and previous sibling in the stacking order. If any of these do not exist, the values are set to NULL. The system keeps an updated window structure by modifying the winnode structures when necessary. By starting at the winnode of the root window, complete stacking order can be determined for both which Window should be displayed on top of others and which visible Window the pointer is currently contained in.

There is a function called stacknext which takes a single Window argument and returns the Window next in the stacking order, or NULL if the argument is the last Window. It determines this by returning the bottom most child if the window has any children, or returning the next
sibling if it exists, or finally stepping up the order parent by parent and checking if any of the Window’s ancestors have siblings that must be dealt with, continuing until the root is reached.

To update the actual displayed graphics on the screen, there is a function called refresh which also takes a single Window as an argument. To update the entire display, refresh is called with the root window as an argument. Many times, only the stack order from a certain Window to the top or the order needs to be refreshed, such as when a Window’s contents are updated. Then refresh is called with that Window. The refresh function displays the entire Window it is called with if it and all of its ancestors are mapped. An exception to this is if part of the Window lies outside of its parent or one of its ancestors. Then this portion is not displayed. This action is called clip by parent. The function then recursively calls itself with the next Window in the stack order using stacknext. Thus, when a Window is updated, the entirety of the non-clipped part of the Window along with all Windows above it in the stacking order is redisplayed.

To determine when a Window needs to be clipped by an ancestor, two helper functions are used, p_in_rect and rect_intersect. The p_in_rect function takes two integer coordinates and an XRectangle and returns 1 if the point is contained in the rectangle and 0 if it is outside. The XRectangle structure simply abstracts a rectangle, with fields of width, height, and coordinates of the top left corner. The rect_intersect function uses p_in_rect to find the intersection of two rectangles. It determine which corners of the two rectangles are contained in the other rectangle and uses this information to compute the intersection. For instance, if only the bottom right corner of one rectangle and the top left of the other lie inside the opposite rectangle, then the intersection ranges from the top left of the one to the top right of the other. Or, if only the two left corners of one rectangle lie inside the other, the intersection lies from these two corners to the left edge of the other rectangle.
When a Window is mapped or unmapped, the appropriate flag is set or cleared and the screen is refreshed from the Window or the Root, respectively. Also, when a Window is mapped, an expose event is sent to the Window so it will be redrawn. When a Window is raised or lowered, its and possibly its parent’s winnode structures are updated, and the screen is refreshed from the Window. When a Window is moved on the screen, first its absolute and parent relative locations are updated. Then a call is made to the \texttt{move\_kids} function, which recursively updates the absolute positions of the Window’s children and subsequent descendants, since all of their parents absolute positions have changed. Then the screen is then refreshed from the root. When a Window is resized, the graphics area is reallocated, since more or less space is needed. Then expose and configure notify events are sent to the Window, and the screen is refreshed from the root.

When a Window is created, the winnode structure must also be created and updated, along with any other relevant parent or sibling winnode structures. The graphics area must also be allocated. This is done by calling \texttt{malloc} with an argument of the window width times the window height times \texttt{BYTESPERPIXEL}, a macro defined by \texttt{svgalib}. Areas for Pixmaps and XImages are allocated similarly. When a Drawable or XImage needs to be drawn into, a virtual
GraphicsContext is defined and setup by calling the svgalib vgagl function `gl_setcontextvirtual`. Then subsequent drawing actions occur into the memory area argument given to `gl_setcontextvirtual` until the current context is changed to something else, either another virtual context or the physical screen.

**Copying Areas**

When Windows are displayed on the screen, the vgagl function `glutbox` is used. This takes the graphics information in the specified system memory area and copies it into the appropriate section of video memory. To display to the screen, it is called when the physical screen is the current GraphicsContext. To copy an area from one Drawable to another, or to get an XImage from a Drawable, the function `gl_getbox` is used in a similar fashion. The area to be copied is put into a temporary memory that is allocate to be of the appropriate size, and then this area is copied into the destination. The temporary area is then freed.

In the XPutImage function, the image is not always intended to copied straight into the destination Drawable. Sometimes, as indicated by the GC also passed as an argument, the XImage should be or’ed or and’ed by pixel onto the destination. Thus, the result of putting a pixel value should not be the source value, but the source value combined with the current destination pixel value by a logical binary operator. The function accomplishes this by allocating two temporary buffers and filling one with the source rectangle and one with the destination rectangle. Then the two buffers are stepped through byte by byte and each pair of bytes is combined using the appropriate operator as specified in the GC, the result being placed in one of the two buffers. Then this resultant buffer is copied to the destination rectangle and the buffers are freed.
Window Functions:

<table>
<thead>
<tr>
<th>Function</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCreateWindow</td>
<td>XPutImage</td>
</tr>
<tr>
<td>XDestroyWindow</td>
<td>XPutPixel</td>
</tr>
<tr>
<td>XLowerWindow</td>
<td>XGetImage</td>
</tr>
<tr>
<td>XRaiseWindow</td>
<td>XCreateImage</td>
</tr>
<tr>
<td>XMapWindow</td>
<td>XCreatePixmap</td>
</tr>
<tr>
<td>XUnmapWindow</td>
<td>XDestroyImage</td>
</tr>
<tr>
<td>XMoveWindow</td>
<td>XFreePixmap</td>
</tr>
<tr>
<td>XResizeWindow</td>
<td>XGetWindowAttributes</td>
</tr>
<tr>
<td>XMoveResizeWindow</td>
<td>XGetWindowAttributes</td>
</tr>
<tr>
<td>XCopyArea</td>
<td>XSetWindowAttributes</td>
</tr>
</tbody>
</table>

**Graphics**

**GC**

The Xwindows graphics context, or GC, is used to pass supplementary information necessary to complete drawing requests. The GC itself is simply a structure with fields for holding the information. There are functions to create and destroy GC’s and to change or lookup values stored in them. Not all the capabilities of GC’s are implemented in the new system, because they are limited to the parts that Curl makes use of in xwindows.c. The used values are foreground, function, line width, clipping region, and font. Other values, such as line style and arc mode, are only semi-implemented. In these cases, the values are always the constant of the default, so instead of checking every time, the relevant drawing functions simply always use the default behavior. Other styles are not implemented, for they are not necessary.

The foreground gives the color which the drawing should be done in. Color handling will be discussed in the next section. The function tells how the destination pixels should be combined with the source pixels. The default is GXcopy, which simply overwrites the
destination pixels. GXand, GXor, and GXxor are also used. These are discussed in the
previous section on copying areas, and will come up again in the section on drawing. The line
width specifies how thick in pixels lines and outlines of circles should be. The font is a pointer
to a XFontStruct that should be used for drawing text. If the clip flag of the GC is set, then all
drawing is limited to the rectangle given in the GC. Nothing outside this rectangle will be
drawn. This is done using gl_setclippingrectangle.

<table>
<thead>
<tr>
<th>GC Functions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XCreateGC XSetFunction</td>
</tr>
<tr>
<td>XChangeGC XSetArcmode</td>
</tr>
<tr>
<td>XGetGCValues XSetForeground</td>
</tr>
<tr>
<td>XFreeGC XSetPlaneMask</td>
</tr>
<tr>
<td>XSetClipRectangles XSetFont</td>
</tr>
</tbody>
</table>

Color

Color handling in the new system is somewhat less complicated than in Xwindows.
Firstly, there is only a single colormap and a constant depth, whereas in Xwindows these can
both vary from window to window. An XColor structure is still used. The main components of
this structure are the pixel value and the red, green, and blue color components. The main
function are XAllocColor and XParseColor. XAllocColor takes an XColor with the rgb
components set, and determines and sets the pixel value of the XColor. It does this by calling
gl_rgbcolor to get the pixel value. The actually allocated rgb values are likely to not be exactly
what was given due to limited numbers of simultaneous colors, so XAllocColor also returns the
actually used rgb values. It does this by temporarily writing one pixel of the pixel value to the
screen, and then retrieving the rgb values using gl_getpixelrgb. It then replaces the original
displaced pixel value. This is a somewhat odd way of doing this, but the most straightforward
way provided by vgagl.
XParseColor takes a string and looks it up in a text file, rgb.txt, and returns the rgb translation of the string if found. The only not completely straightforward about this function is that the string comparison must be case insensitive, so the characters of the string are compared one at a time, checking to see if they are the same letter, regardless of capitalization, at each step.

<table>
<thead>
<tr>
<th>Color Handling Functions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XAllocColor</td>
</tr>
<tr>
<td>XParseColor</td>
</tr>
<tr>
<td>XQueryColors</td>
</tr>
</tbody>
</table>

Drawing

All of the drawing functions have a few things in common. First, a drawing function saves the current GraphicsContext so it can be restored after drawing into a virtual context. Then the virtual context is set, using the width, height, and data area given by the Drawable argument. The GC is then checked, and if the clip flag is set, clipping is enabled. The actual drawing is then done, and clipping is disabled and the original context is restored.

For the most part, the drawing is done using several functions to draw abstract shapes which are implemented using vgagl drawing primitives. The vgagl functions which are used extensively are `gl_hline`, `gl_line`, and `gl_setpixel`. These functions draw horizontal lines, any lines, and single pixels. The drawing functions are `circle`, `triangle`, `polygon`, `line`, `xorline`, `lines`, `fill_quad`, and `fill_arc`. Many of these functions use others of the group to build more and more complex drawing capabilities.

Many of the functions deal with non-integer values at some point in their computations in order to be more correct. When actually drawing, however, integers must be used to describe the location on the screen. In the cases where non-integer values must be passed to a function
requiring integers, the macro \textit{Rnd} is used, which rounds the value to the nearest integer. This results in more accurately specified and drawn shapes.

The \textit{triangle} function is the basis for several of the others. This function takes three points and a pixel value and draws a solid triangle with vertices at the points. It does this by drawing a series of horizontal lines. Each triangle is actually broken up into two smaller right triangles, one on top of the other, and drawn in two sections. The function determines which point is between the other two along the vertical axis, and splits the triangle there. Then one triangle is drawn by drawing successive horizontal lines, starting from the \( y \) coordinate of the highest point continuing down to the \( y \) coordinate of the middle point. The same is done from the middle point to the bottom point. For each \( y \) coordinate, the two \( x \) coordinates of the horizontal line are computed. This is done by using the equation of a line and solving for the \( x \) value at a given \( y \) value, as in equation 1.

\[
\frac{(x_1, y_1)}{(x_a, y)} \frac{(x_b, y)}{(x_2, y_2)}
\]

\( x_3, y_3 \)

\textbf{Figure 3 Reference picture for equation 1}
\[ xa = x_1 + \frac{(y - y_1)(x_3 - x_1)}{(y_3 - y_1)} \]
\[ xb = x_1 + \frac{(y - y_1)(x_2 - x_1)}{(y_2 - y_1)} \]

Equation 1

The polygon function makes use of successive calls to triangle. It divides each n-gon into n-2 triangles and fills each of them using triangle. Specifically, it takes as argument an array of XPoints and the number of points to process. If the number is three, it simply calls triangle with the points. Otherwise, it calls triangle with the first point and last two points and then recursively calls itself with the same array of points, but with the number of points being one less. This method is illustrated in the following figure.

![Figure 4 Illustration of divisions in triangle and polygon functions](image)

The actions of triangle are actually slightly more complicated. There is an additional argument, xorflag. If this is passed with a value of one, then the function xor should be applied to the drawing instead of a straight copy. In this case, triangle calls no gl_hline, but xorhline instead. The function xorhline combines the horizontal line using exclusive or with the destination pixel values. It does this in a similar fashion to XCopyArea by allocating a temporary buffer with
height one, putting the destination values in it, and xor'ing their values to the pixel values passes to it before putting the box back into the destination.

The line function also makes use of the polygon function. If the line width passed to line is zero or one, then the line is drawn with gl_drawline. These lines are called thin lines. If a thin line is to be drawn, but the GC indicates that the function should be GXxor, then a temporary buffer is allocated which the line is drawn into, and this buffer is copied as in XCopyArea to the destination. If the line is thick, that is with a width greater than one, then a call to polygon is used to draw a quadrilateral. The coordinates are given by the following equations. If, in the case of thick lines and a function of GXxor, an xorflag of 1 is passed to polygon.

![Figure 5 Reference figure for equations 2 and 3](image)

- **point 1**: \((x_1 - (w/2) \sin(\Theta), y_1 + (w/2) \cos(\Theta))\)
- **point 2**: \((x_1 + (w/2) \sin(\Theta), y_1 - (w/2) \cos(\Theta))\)
- **point 3**: \((x_2 + (w/2) \sin(\Theta), y_2 - (w/2) \cos(\Theta))\)
- **point 4**: \((x_2 - (w/2) \sin(\Theta), y_2 + (w/2) \cos(\Theta))\)

**Equation 2**
The function \textit{lines} uses \textit{polygon} also, but in a slightly different way. The \textit{lines} function differs from repeatedly calling \textit{line} because of the way the lines are joined. When drawn using \textit{lines}, they are joined using a miter style. That is, the outside edges of the lines are extended to where they meet and this area is also filled, forming a pointed shape. This is illustrated in the following figure.

\begin{figure}
\centering
\includegraphics[width=\textwidth]{figure6.png}
\caption{Illustration of miter joined lines}
\end{figure}

The \textit{lines} function uses the \textit{miter} function to compute the point of intersection of the two outside borders of the two lines. Then a call to \textit{polygon} for each line is made, using the miter point and opposite corner as the two vertices of that side of the quadrilateral, as shown below.
The XDrawLine function simply calls \textit{line}, while the XDrawLines function calls \textit{lines}. XFillPolygon calls \textit{polygon}, while XFillRectangle calls the simpler \textit{gl\_fillbox} is used. XDrawRectangle draws four lines using \textit{line}. The only two remaining drawing functions are XDrawArc and XFillArc.

These two functions make use of the other helper functions, \textit{circle}, \textit{fill\_quad}, and \textit{fill\_arc}. The \textit{circle} function draws part of an arc, starting from one angle and continuing through a specified angle. The function is not limited to drawing circles, however, it can also draw arcs of an ellipse. The \textit{circle} function draws the arc point by point, calculating the set of points by stepping through angles. At each step using trigonometric functions to calculate the offset from the center of the circle. The step size of the angle varies with the size of the bounding rectangle, with larger rectangles demanding smaller step sizes. This allows small circles to be computed quickly while not leaving gaps in the arcs inscribed in larger rectangles.
The XDrawArc function uses *circle* to draw. If the line width specified by the GC is greater than one, then more than one call to *circle* is made. The drawing produced by each call to *circle* is aligned with the others by passing it a shifted and shrunken bounded rectangle.

The XFILLArc function makes use of *fill_arc*. The *fill_arc* function is similar to *circle* except it fills the arc and does not just draw the contour. First, *fill_arc* determines which full quadrants the arc to be filled spans and fills each of these with *fill_quad*. The *fill_quad* function works like *triangle*, but it uses the y axis for the start point and computes the end point as in the *circle* function. All arc filling function do the computations for the first quadrant only, and then transform those resultant coordinates into the other quadrants if necessary. Then *fill_arc* has one or two partial quadrants to fill. It does this by breaking up the area again, into a triangle and the rest. This is illustrated in the following figure. The triangle is filled by a call to *triangle*, and the remaining part is filled as in *fill_quad*.

![Figure 8 Division of area by fill_arc](image-url)
Events, the Keyboard, and the Pointer

Events

Xwindows is an event driver system. This means that events arrive and applications deal with them or wait to deal with them by entering event loops. Events occur when the user gives input with the keyboard or mouse, or when the window system needs to inform the application of something, for instance the window system tells the application when windows need to be redrawn. The new system delivers events to the Curl application by using the same interface and functions as Xwindows.

The application selects which types of events it is interested in for each Window by calling the function XSelectInput. In this implementation, the XSelectInput function actually just returns right away, for all the Windows used by Curl select the same event types, and these types are the only ones dealt with by the system. Thus all events handled are sent to all Windows, and no check is done to determine if it was selected, for it always is. The functions maintain a list of events that is added to as necessary, and events are removed in response to calls of XNextEvent. This function returns the next event, or blocks and waits for one to arrive if there are no events present in the queue. XPeekEvent is similar, but it returns the event without removing it from the queue. Actually, neither of these return the event object, but they copy the relevant information into a structure passes to them as an argument.

The event queue is implemented as a pointer to an eventnode, which simply contains an XEvent structure and a pointer to the next eventnode, or NULL if it is the last. There is also a count kept of the number events in the queue. XEventsQueued simply returns this number when called. Much of the complexity of Xwindows is removed here, because in that system events can arrive asynchronously and are copied by xlib into a local queue, so events may have arrived but not yet been copied. Thus, the functions XFlush and XSync, as discussed in a previous section, are not necessary.
The XEvent structure is actually a union of many specific types of event structures. When an event arrives (or needs to be added to the queue), the program branches according to the type member of the XEvent and then uses the appropriate structure. The event types that are selected by Curl windows, and thus the typed that are handled by the system, are the following: MotionNotify, ButtonPress, ButtonRelease, EnterNotify, LeaveNotify, KeyPress, KeyRelease, Expose, and ConfigureNotify. MotionNotify events occur when the mouse is moved in a window. ButtonPress and Release events are generated when the mouse buttons are pressed or released. Likewise for KeyPress and Release events regarding the keyboard. Enter and LeaveNotify events occur when the pointer enters or leaves a Window. Expose events occur when a Window needs to be redrawn.

ConfigureNotify events occur when a window changes size. ConfigureNotify events only occur when XResizeWindow or XMoveResizeWindow are called. These calls also generate Expose events. In the new system, when Expose events are sent, the entire window is specified as the area needing to be refreshed, since never is only part of it not available. The only other time Expose events are sent is when a Window is mapped with XMapWindow. Each of these events is added to the event queue by calling a *send_* function, such as *send_expose*, *send_button*, or *send_motion*, with appropriate argument values. Note, these do not always actually send events, and may send more than one in some cases. For instance, if the mouse is in the root window, there is no application to handle input events and none is sent. Also, a mouse click on a window border does no generate an event to a window, but simply raises the window. This is an example of the previously discussed incorporation of window manager functionality into the system. Normally in Xwindows, the click on the border would be sent as an event to the window manager program, which may then raise the window. Part of this chain of events is bypassed. More than one event can be sent by a button press, as will be discussed in a following section on the mouse.
These functions to send input events are called by either the keyboard or mouse handler functions. These in turn are called by the svgalib function *vga_waitevent*. The specific handlers are set up to be called by *vga_waitevent* in the event initialization function called by XOpenDisplay.

Most event structures have fields called state and time. The state holds the state of pointer buttons and modifier keys such as control and shift at the time of the event. The keyboard and event handlers keep structures up to date so that a call to *update_state* can update the state variable, whose value can be written into the event structure. In Xwindows the time field contains the time from server startup, expressed in milliseconds. Currently in the new system, the time field is simply filled with a counter which is incremented each time an event is added to the queue. This at least gives a strict ordering for the events.

The Keyboard

Whenever the keyboard sends information, *vga_waitevent* receives this information and interprets it. It then calls the keyboard handler, *my_keyh*. The arguments to the handler are a scancode and a flag indicating weather the key was pressed or released. If a key is held down, many presses will be sent before the release. The scancode corresponds to a physical key on the keyboard. The handler is simple, it updates an array which keeps track of all keys, updates the state, and sends an event to the Window the mouse pointer is currently in. The only other action it undertakes is to keep track of the state of the CapsLock button. It treats this as still pressed when updating the state even after the first release. Only after a second release is it truly no longer active.

The additional translation of a key event occurs later. After an application receives a key event, it usually calls XLookupString. This function looks at the scancode and current state
of the modifiers and translates the event into a keysym and, if applicable, a string. The keysyms are constants defined in `keysymdef.h` and correspond more meaningfully to the event. This translation is done with a long switch statement and conditionals to account for the modifiers. The application can then branch according to the keysym and take the appropriate action.

The Mouse

Just as keyboard input is handled by a handler function, mouse input is handled by the mouse handler, `my_mouse_h`. This function makes use of two other functions, `m_in_win` and `cur_win`. The `m_in_win` function takes a point and a Window and returns 1 if the point is in the window, 2 if the point is on the border, and 0 if it is completely outside the Window. The function makes use of the `p_in_rect` function discussed earlier. The `cur_win` function steps up the stacking order from the root, checking if the mouse's current position is in each Window. It then returns the highest Window the pointer is in, and this gives the pointer the mouse is in on the screen.

The mouse handler takes as arguments the current mouse button state, and the amount of movement in the x and y directions. The handler first checks if there was movement, and if so, redraws the cursor. Whenever the cursor is drawn, the destination square for the cursor is saved in a buffer. Then before the cursor is drawn, the old square is restored by drawing over the cursor at the old position, then a box is retrieved from the new position, then the cursor itself is drawn. This eliminates trails of the mouse cursor. The handler also checks to see if the mouse would move off the screen, and if so only changes the position enough for it to remain on the screen.

Then the handler calls `m_in_win` to check if the mouse is still in the same window. If not, a LeaveNotify is sent to the previous window and an EnterNotify to the new window. If
there was mouse motion, a MotionNotify is sent to the current window as well. There is one last Xwindow function dealing with events that has yet to be mentioned. When an application receives a MotionNotify event, it calls XQueryPointer to get the current state and position of the pointer. This function simply returns the relevant information from the variables which hold the state with no further processing. The handler then checks if the state of the buttons has changed, and if so, calls send_button with the relevant information.

<table>
<thead>
<tr>
<th>Event Functions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XNextEvent XSelectInput</td>
</tr>
<tr>
<td>XPeekEvent XQueryPointer</td>
</tr>
<tr>
<td>XEventsQueued XLookupString</td>
</tr>
</tbody>
</table>

The send_button function does by far the most processing of any of the send_ functions. This occurs because of passive button grabbing and clicking on borders. If the mouse is clicked on the border of a window, no event is actually sent, but the window is simply raised to the top of the stacking order relative to its siblings. Button grabbing is the other complication. When a button is clicked in a window, the window passively grabs the pointer, so that the corresponding button release is sent to the same window. If, however, one button is pressed in one window, then the pointer is moved to another window and another button is pressed, then the new window grabs the pointer on this button, and a release of the original button is sent to the original window even though the button is still pressed. Also, the original button is counted as pressed in the new window, so it receives a button press event. If the button was grabbed by a window border, the corresponding release generates no event. The send_button function actually just does these computations, and sends events when necessary by calling really_send_button.
Font handling

One large incompatibility between the Linux and MS Windows versions of Curl is font handling. The Windows version uses TrueType fonts, while the Linux version uses only Linux fonts. TrueType fonts are scalable to any size, and each character is rendered from a description stored in the file. Linux fonts are stored as bitmaps of a specific size, so a different font file needs to be used for each pointsize. The font handling Curl host functions is the only section of *xwindows.c* which needed to be modified significantly. Instead of passing a different filename for every pointsize to be loaded to the Xwindow function used to load fonts, the name of the font file and a pointsize is passed. If a bold, italic, or both bold and italic font is desired, then a different font file is needed. The file is loaded and the characters are rendered at the desired size.

The function used to load fonts is `XLoadQueryFont`. Its arguments are the Display and the name of the font file to open. This was sufficient for Linux fonts, but now more information is needed. Since the Display argument is not needed, and was simply kept to keep the function prototype the same, this argument is changed to allow the passing of the desired pointsize to `XLoadQueryFont`. Conveniently, the Display was assigned as a pointer to an integer using a `typedef` statement in *my.h*. This is therefore used to pass the pointsize to the function, still preserving the prototype of the function.

As stated previously, the FreeType library is used to render TrueType fonts. When `XLoadQueryFont` is called, it uses FreeType functions to open the font file and get the necessary information. `XLoadQueryFont` and the function `setfontsize` render each character into a bitmap of minimum size and cache them in a list. This list is pointed to by an entry in the `XFontStruct` returned by `XLoadQueryFont`. There are several options for caching. The rendered bitmaps could be cached, the outlines of each character could be cached, or no caching could be done and each character could be rendered each time it is to be drawn. Outlines are
comparable to an intermediate step between caching the complete bitmaps and no caching at all. The system caches the entire bitmaps to allow fast drawing of text once the font is loaded and rendered. This, however, results in a noticeable delay the first time a font of a particular size is loaded. If XLoadQueryFont is called again with the same arguments and the XFontStruct has yet to be freed, this is detected and the previously rendered structure is returned, saving this delay and extra memory use whenever possible.

XLoadQueryFont also does several other related tasks. The character metrics of each character are also cached. These include the width, height, advance width (space which should be given before the next character begins), and left bearing and ascent. The bearings are needed to translate the minimally sized bitmap so the characters will be lined up when being drawn.

The instance resolutions of the screen are also set before rendering occurs. For a svga display, the resolutions are usually 96 units, but if the resolution is smaller than usual this number may be too large and the characters will appear larger on the screen than intended. The appropriate unicode charmap from the file to use is also determined to make sure the mapping between the intended characters draw match those actually looked up.

<table>
<thead>
<tr>
<th>Font functions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLoadQueryFont</td>
</tr>
<tr>
<td>XTextWidth</td>
</tr>
<tr>
<td>XDrawString</td>
</tr>
</tbody>
</table>

The two remaining Xwindows font functions are XDrawText and XTextWidth. XTextWidth takes a string of characters and returns the width in pixels of the string if it were drawing in the specified font. It simply sums up the advance widths cached with each character and returns the number. This function is used by the application for text placement. XDrawText actually draws the text into a Drawable. It does this by starting at the specified origin, looking at each bit of the character bitmap, and setting a pixel of the destination to the specified value if each bit is
set. The origin is first shifted by the left bearing of the character and the ascent. When the
drawing of the character is done, the origin is shifted back, and then the x coordinate is
incremented by the advance width of the character.
Chapter 4 Conclusion

Future Work

There are many things to be done to continue the development of the compact windowing system for Curl. For a simple example, the ability to select and change the cursor’s appearance depending on the window the pointer is in and the window’s state would be a simple addition. A more complex change would be to keep better track of relative window positions, in order to eliminate the need to copy the entire window’s contents to the screen when updated or refreshed. The amount of overhead of this tactic and therefore size of the gain is unclear, and it would add significantly to the size and complexity of the code. The current system is acceptable because there are generally few windows active in the Curl application, moreover, there is only one application to deal with. This further limits the number of windows.

Another matter is the driver used by svgalib to access the video card. Most common cards are supported, but the list is still not as extensive or up to date as those supported by Xwindows. The svgalib programs will run in vga mode on any vga capable video card, but in order to get the higher resolution and number of colors specific information on the cards registers must be available to set the chips up for the proper svgs mode operation. An intended destination for this system is in the MASC system, which will require a new driver for the video interface in any case. It should be just as easy to adapt the driver to work with svgalib as any other system which could be chosen, so this should not pose a problem. The system was designed and tested on several video cards with Cirrus Logic chipsets.
Bibliography


“The FreeType Engine Core Library Reference,” http://www.freetype.org/docs/apiref.txt
Appendices

Appendix A  Xwindow functions and Structures from xwindows.c

XEvent
XNextEvent
XPeekEvent
XEventsQueued
XSelectInput
XQueryPointer
XFlush
XSync
XLookupString

XStringListToTextProperty
XTextProperty
XErrorEvent
XGetErrorDatabaseText
XGetErrorText
XSetErrorHandler
XWMHints
XClassHint
XSizeHints
XSetWMNormalHints
XSetWMProperties

XCreateFontCursor

XCreateWindow
XDestroyWindow
XLowerWindow
XMapWindow
XUnmapWindow
XMoveResizeWindow
XMoveWindow
XResizeWindow
XRaiseWindow
XQueryTree

XOpenDisplay
XDisplayName

XDefaultScreen
XDefaultVisual
XDefaultDepth
XDefaultColormap

XBell

XFree

XColor
XAllocColor
XParseColor
XQueryColors
XGCValues
XCreateGC
XChangeGC
XGetGCValues
XFreeGC
XSetFont
XSetFunction
XSetArcMode
XSetForeground
XSetPlaneMask
XSetClipRectangles
XPoint
XRectangle
XDrawArc
XDrawLine
XDrawLines
XDrawRectangle
XFillArc
XFillPolygon
XFillRectangle
XCOPYArea
XImage
XPutImage
XPutPixel
XGetImage
XCreateImage
XCreatePixmap
XDestroyImage
XFreePixmap

XWindowAttributes
XSetFont
XGetWindowAttributes
XChangeWindowAttributes
AllPlanes
BlackPixel
ConnectionNumber

XFontStruct
XTextWidth
XSetFont
XLoadQueryFont
XDrawString
Appendix B my.h

#include <stdlib.h>
#include <stdio.h>
#include <vga.h>
#include <vgagl.h>
#include <math.h>
#include <freetype.h>
#include <vgamouse.h>
#include <vgakeyboard.h>
#include "keysymdef.h"

#define
#define
#define
#define
#define
#define
#define
#define
#define
#define
#define
#define

typedef
typedef
typedef
typedef
typedef
typedef

typedef int Display;
typedef int Colormap;
typedef int Bool;
typedef int Visual;
typedef int Screen;
typedef int Cursor;

#define AllPlanes -OL

int BlackPixel(Display *d, int s);
Display *XOpenDisplay(char * display_name);
void XBell(Display *display, int percent);
static int dummy=0;

/* Window attribute value mask bits */
#define CWBackPixmap (1L<<0)
#define CWBackPixel (1L<<1)
#define CWBorderPixmap (1L<<2)
#define CWBorderPixel (1L<<3)
#define CWBitGravity (1L<<4)
#define CWWinGravity (1L<<5)
#define CWBackingStore (1L<<6)
#define CWBackingPlanes (1L<<7)
#define CWBackingPixel (1L<<8)
#define CWOverrideRedirect (1L<<9)
#define CWSaveUnder (1L<<10)
#define CWEventMask (1L<<11)
#define CWDontPropagate (1L<<12)
#define CWColormap (1L<<13)
#define CWCursor (1L<<14)

/* Values */

#if 0
typedef struct {
  Pixmap background_pixmap; /* background, None, or ParentRelative */
  unsigned long background_pixel;/* background pixel */
  Pixmap border_pixmap; /* border of the window or CopyFromParent*/
  unsigned long border_pixel; /* border pixel value */
  int bit_gravity; /* one of bit gravity values */
  int win_gravity; /* one of the window gravity values */
  int backing_store; /* NotUseful, WhenMapped, Always */
  unsigned long backing_planes; /* planes to be preserved if possible*/
};
unsigned long backing_pixel; /* value to use in restoring planes */
Bool save_under; /* should bits under be saved? (popups) */
long event_mask; /* set of events that should be saved */
long do_not_propagate_mask; /* set of events that should not propagate */
Bool override_redirect; /* boolean value for override_redirect */
Colormap colormap; /* color map to be associated with window */
Cursor cursor; /* cursor to be displayed (or None) */
} XSetWindowAttributes;
#endif

typedef struct {
  long event_mask; /* NOT SURE set of events that should be saved */
  long do_not_propagate_mask; /* NOT SURE set of events that should not propagate */
  Cursor cursor; /* cursor to be displayed (or None) */
} XSetWindowAttributes;
#if 0
typedef struct {
  int x, y; /* location of window */
  int width, height; /* width and height of window */
  int border_width; /* border width of window */
  int depth; /* depth of window */
  Visual *visual; /* the associated visual structure */
  /* was Window, not void */
  void *root; /* root of screen containing window */
  int class; /* InputOutput, InputOnly */
  int bit_gravity; /* one of the bit gravity values */
  int win_gravity; /* one of the window gravity values */
  int backing_store; /* NotUseful, WhenMapped, Always */
  unsigned long backing_planes; /* planes to be preserved if possible */
  unsigned long backing_pixel; /* value to use when restoring planes */
  Bool save_under; /* boolean, should bits under be saved? */
  bool map_installed; /* boolean, is color map currently installed */
  int map_state; /* IsUnmapped, IsUnviewable, IsViewable */
  long all_event_masks; /* set of events all people have interest in */
  long your_event_mask; /* my event mask */
  long do_not_propagate_mask; /* set of events that should not propagate */
  Bool override_redirect; /* boolean value for override_redirect */
  Screen *screen; /* back pointer to correct screen */
} XWindowAttributes;
#endif

typedef struct {
  int x, y; /* location of window */
  int width, height; /* width and height of window */
  int border_width; /* border width of window */
} XWindowAttributes;

typedef struct {
  int winflag;
  int mapped;
  unsigned char * pmap;
  unsigned int w;
  unsigned int h;
  unsigned int x;
  unsigned int y;
  unsigned int xr;
  unsigned int yr;
  int bw;
  int bc;

  struct winnode * node;
  XWindowAttributes * wa;
  XSetWindowAttributes * swa;
}
typedef Drawable_Struct * Drawable;

typedef struct {
    unsigned char *pmap;
    unsigned int w;
    unsigned int h;
} XImage;

typedef Drawable Window;
typedef Drawable Pixmap;

Window RootWindow(Display *d, int s);

void refresh(Window w);
Window stacknext(Window w);
Window stackprev(Window w);

typedef struct {
    Window win;
    struct winnode * parent;
    struct winnode * children;
    struct winnode * last_child;
    int num_children;
    struct winnode * sibprev;
    struct winnode * sibnext;
} winnode;

#define Rnd(x) ((int)(x) + ((int) (2*(x))) % 2)

typedef struct {
    short x, y;
} XPoint;

typedef struct {
    short x, y;
    unsigned short width, height;
} XRectangle;

#define DoRed 1
#define DoGreen 2
#define DoBlue 4

/* GC attribute value mask bits */
#define GCFunction (1L<<0)
#define GCPPlaneMask (1L<<1)
#define GCForeground (1L<<2)
#define GCBackground (1L<<3)
#define GCLineWidth (1L<<4)
#define GCLineStyle (1L<<5)
#define GCCapStyle (1L<<6)
#define GCJoinStyle (1L<<7)
#define GCFillStyle (1L<<8)
#define GCFillRule (1L<<9)
#define GCTile (1L<<10)
#define GCStipple (1L<<11)
#define GCTileStipXOrigin (1L<<12)
#define GCTileStipYOrigin (1L<<13)
#define GCFont (1L<<14)
typedef struct {
    int function;  /* logical operation */
    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, LineOnOffDash, LineDoubleDash */
    int cap_style; /* CapNotLast, CapButt, CapRound, CapProjecting */
    int join_style; /* JoinMiter, JoinRound, JoinBevel */
    int fill_rule; /* FillSolid, FillTiled, FillStippled, FillOpaqueStippled */
    int fill_style; /* EvenOddRule, WindingRule */
    int arc_mode; /* ArcChord, ArcPieSlice */
   Pixmap tile;  /* tile pixmap for tiling operations */
    Pixmap stipple; /* stipple 1 plane pixmap for stippling */
    int ts_x_origin; /* offset for tile or stipple operations */
    int ts_y_origin; /* default text font for text operations */
    void * font; /* default text font for text operations */
    int subwindow_mode; /* ClipByChildren, IncludeInferiors */
    Bool graphics_exposures; /* boolean, should exposures be generated */
    int clip_x_origin; /* origin for clipping */
    int clip_y_origin; /* change to rec from pixmap */
    XRectangle clip_mask; /* bitmap clipping; other calls for rects */
    int clip_flag; /* add clip flag */
    int dash_offset; /* patterned/dashed line information */
    char dashes;
} XGCValues;

#define ArcPieSlice 1;
typedef XGCValues *GC;

void XDrawArc(Display *display, Drawable d, GC gc,
              int x, int y, unsigned int width, unsigned int height,
              int angle1, int angle2);
void XDrawLine(Display *display, Drawable d, GC gc,
               int x1, int y1, int x2, int y2);
void XDrawLines(Display *display, Drawable d, GC gc,
                XPoint *points, int npoints, int mode);
void XDrawRectangle(Display *display, Drawable d, GC gc,
                    int x, int y, unsigned int width, unsigned int height);
void XFillArc(Display *display, Drawable d, GC gc,
             int x, int y, unsigned int width, unsigned int height,
             unsigned int line_width, unsigned int line_style,
             unsigned int cap_style, unsigned int join_style,
             unsigned int fill_style, unsigned int fill_rule,
             XRectangle clip_mask, int clip_flag, int dash_offset, char dashes);
int x, int y, unsigned int width, unsigned int height,
   int angle1, int angle2);
void XFillPolygon(Display *display, Drawable d, GC gc,
   XPoint *points, int npoints, int shape, int mode);
void XFillRectangle(Display *display, Drawable d, GC gc,
   int x, int y, unsigned int width, unsigned int height);
void XCopyArea(Display *display, Drawable src, Drawable dest, GC gc,
   int src_x, int src_y, unsigned int width,
   unsigned int height, int dest_x, int dest_y);

int XAllocColor(Display *display, Colormap colormap, XColor *c);
void XQueryColors(Display *display, Colormap colormap,
   XColor *defs_in_out, int ncolors);
int XParseColor(Display *display, Colormap colormap, char *spec,
   XColor *exact_def_return);
GC XCreateGC(Display *display, Drawable d,
   unsigned long valuedmask, XGCValues *values);
void XChangeGC(Display *display, GC gc, unsigned long valuedmask,
   XGCValues *values);
   /* was Status return val */
int XGetGCValues(Display *display, GC gc, unsigned long valuedmask,
   XGCValues *values_return);
void XFreeGC(Display *display, GC gc);
void XSetFunction(Display *display, GC gc, int function);
void XSetPlaneMask(Display *display, GC gc, unsigned long plane_mask);
void XSetForeground(Display *display, GC gc, unsigned long foreground);
void XSetArcMode(Display *display, GC gc, int arc_mode);
void XSetClipRectangles(Display *display, GC gc,
   int clip_x_origin, int clip_y_origin,
   XRectangle rectangles[], int n, int ordering);

/**************************dis is where i is ***********/
#define ZPixmap 0
XImage *XCreateImage(Display *display, Visual *visual, unsigned int depth,
   int format, int offset, char *data,
   unsigned int width, unsigned int height,
   int bitmap_pad, int bytes_per_line);
void XPutPixel(XImage *ximage, int x, int y, unsigned long pixel);
void XPutImage(Display *display, Drawable d, GC gc, XImage *image,
   int src_x, int src_y, int dest_x, int dest_y,
   unsigned int width, unsigned int height);
XImage *XGetImage(Display *display, Drawable d,
   int x, int y, unsigned int width, unsigned int height,
   unsigned long plane_mask, int format);
void XDestroyImage(XImage *ximage);
Pixmap XCreatePixmap(Display *display, Drawable d,
   unsigned int width, unsigned int height,
   unsigned int depth);
void XFreePixmap(Display *display,Pixmap pixmap);
Window XCreateWindow(Display *display, Window parent,
   int x, int y, unsigned int width, unsigned int height,
   unsigned int border_width, int depth,
unsigned int class, Visual* visual, unsigned long valuemask,
XSetWindowAttributes *attributes);

void XDestroyWindow(Display *display, Window w);
void XRaiseWindow(Display *display, Window w);
void XLowerWindow(Display *display, Window w);
void XMapWindow(Display *display, Window w);
void XUnmapWindow(Display *display, Window w);
void XMoveWindow(Display *display, Window w, int x, int y);
void XResizeWindow(Display *display, Window w,
    unsigned int width, unsigned int height);
void XMoveResizeWindow(Display *display, Window w, int x, int y,
    unsigned int width, unsigned int height);

int XQueryTree(Display *display, Window w,
    Window * root_return, Window * parent_return,
    Window ** children_return, unsigned int * nchildren_return);

int XGetWindowAttributes(Display *display, Window w,
    XWindowAttributes *window_attributes_return);

void XChangeWindowAttributes(Display *display, Window w,
    unsigned long valuemask,
    XSetWindowAttributes *attributes);

void XFree(void * p);

typedef struct {
    short lbearing; /* origin to left edge of raster */
    short rbearing; /* origin to right edge of raster */
    short width; /* advance to next char's origin */
    short ascent; /* baseline to top edge of raster */
    short descent; /* baseline to bottom edge of raster */
    TT_Raster_Map rm;
} XCharStruct;

typedef int XFontProp;

typedef struct {
    void * fid; /* Font id for this font */
    XCharStruct min_bounds; /* minimum bounds over all existing char */
    XCharStruct max_bounds; /* maximum bounds over all existing char */
    int ascent; /* logical extent above baseline for spacing */
    int descent; /* logical decent below baseline for spacing */
    TT_Face face;
    TT_Instance instance;
    TT_CharMap char_map;
    TT_Face_Properties properties;
    TT_Instance_Metrics imetrics;
    int num;
    void * fn;
    int rendered;
    XCharStruct *per_char; /* first_char to last_char information */
} XFontStruct;
typedef XFontStruct * Font;
void fontinit();
void fontexit();
void setfontsize(XFontStruct* fs, int ps);
void XSetFont(Display *display,GC gc,Font font);
int XTextWidth(XFontStruct *font_struct,char * string,int count);
XFontStruct *XLoadQueryFont(Display *display,char * name);
void XDrawString(Display *display,Drawable d,GC gc,int x,int y,
char * string,int length);

#endif

/****
*/

typedef int Time;

typedef struct
{
  int type;    /* ButtonPress or ButtonRelease */
  Window window;  /* `event' window it is reported relative to */
  Time time;    /* milliseconds */
  int x, y;     /* pointer x, y coordinates in event window*/
  unsigned int state; /* key or button mask */
  unsigned int button; /* detail */
} XButtonEvent;

typedef XButtonEvent XButtonPressedEvent;
typedef XButtonEvent XButtonReleasedEvent;

typedef struct
{
  int type;    /* KeyPress or KeyRelease */
  Window window;  /* `event' window it is reported relative to */
  Time time;    /* milliseconds */
  int x, y;     /* pointer x, y coordinates in event window*/
  unsigned int state; /* key or button mask */
  unsigned int keycode; /* detail */
  char* st;
} XKeyEvent;

typedef XKeyEvent XKeyPressedEvent;
typedef XKeyEvent XKeyReleasedEvent;

typedef struct
{
  int type;    /* MotionNotify */
  Window window;  /* `event' window reported relative to */
  Time time;    /* milliseconds */
  int x, y;     /* pointer x, y coordinates in event window*/
  unsigned int state; /* key or button mask */
  char is_hint; /* detail */
} XMotionEvent;

typedef XMotionEvent XPointerMovedEvent;

typedef struct
{
  int type;    /* Expose */
  Window window;
  int x, y;
  int width, height;
} XExposeEvent;

typedef struct
{
  int type;
  Window window;
  int width, height;
} XQueryPointerEvent;
typedef struct {
    int type;
    Window window; /* event window reported relative to */
    Time time; /* milliseconds */
    int x, y; /* pointer x, y coordinates in event window */
    unsigned int state; /* key or button mask */
} XCrossingEvent;

typedef union _XEvent {
    int type; /* must not be changed */
    XKeyEvent xkey;
    XButtonEvent xbutton;
    XMotionEvent xmotion;
    XCrossingEvent xcrossing;
    XExposeEvent xexpose;
    XConfigureEvent xconfigure;
    long pad[24];
} XEvent;

typedef int KeySym;

#define Expose 1
#define ConfigureNotify 2
#define ButtonPress 3
#define ButtonRelease 4
#define KeyPress 5
#define KeyRelease 6
#define EnterNotify 7
#define LeaveNotify 8
#define MotionNotify 9

int XLookupString(XKeyEvent *event_struct, char * buffer_return,
                  int bytes_buffer, KeySym * keysym_return,
                  void *dummy);

void XNextEvent(Display *display, XEvent *event_return);

void XPeekEvent(Display *display, XEvent *event_return);

void XFlush(Display *display);

void XSync(Display *display, Bool discard);

int XEventsQueued(Display *display, int mode);

Bool XQueryPointer(Display *display, Window w,
                   Window * root_return, Window * child_return,
                   int *root_x_return, int *root_y_return,
                   int *win_x_return, int * win_y_return,
                   unsigned int *mask_return);

void event_init();
void event_exit();
void draw_cursor();

/* #include <X11/cursorfont.h> */
Cursor XCreateFontCursor(Display *display, unsigned int shape);

#define MBUT1 MOUSE_LEFTBUTTON /* 4 */
#define MBUT2 MOUSE_RIGHTBUTTON /* 1 */
#define MBUT3 MOUSE_MIDDLEBUTTON /* 2 */
#define SHIFT 8
#define CTRL 16
#define ALT 12
#define CAPS 64
#define Button1   MBUT1
#define Button2   MBUT2
#define Button3   MBUT3

#define Button1Mask    MBUT1
#define Button2Mask    MBUT2
#define Button3Mask    MBUT3
#define ShiftMask      SHIFT
#define LockMask       CAPS
#define ControlMask    CTRL
#define Mod1Mask       ALT

/* not used except in xwindows.c */
#define Mod2Mask        0
#define Mod3Mask        0
#define Mod4Mask        0
#define Mod5Mask        0
#define Button4Mask     0
#define Button5Mask     0
#define Button4         98
#define Button5         99
Appendix C winstuff.c

#include "my.h"

int VGAMODE;
GraphicsContext *backscreen;
GraphicsContext *physicalscreen;

Window myroot=NULL;
winnode rootnode;

Cursor XCreateFontCursor(Display *display,unsigned int shape)
{
    return (1);
}

GraphicsContext *ps()
{
    return (physicalscreen);
}

Window RootWindow(Display *d, int s)
{
    return (myroot);
}

int BlackPixel(Display *d, int s)
{
    return (gl_rgbcolor(0,0,0));
}

Display *XOpenDisplay(char * display_name)
{
    if (myroot != NULL) printf("\nTrying to open display twice!\n");
    vga_init();
    VGAMODE = vga_getdefaultmode();
    if (VGAMODE == -1)
        VGAMODE = G800x600x64K;
    if (!vga_hasmode(VGAMODE))
        printf("Mode not available.\n");
    exit(-1);
    vga_setmousesupport(1);
    vga_setmode(VGAMODE);
    gl_setcontextvga(VGAMODE); /* Physical screen context. */
    physicalscreen = gl_allocatecontext();
    gl_getcontext(physicalscreen);
    if (COLORS == 256)
        gl_setrgbpalette();
    glclearscreen(0);
    myroot = (Window) malloc(sizeof(Drawable_Struct));
    myroot->winflag=l;
    myroot->mapped=0;
    myroot->node= (struct winnode *) &rootnode;
    myroot->x=0;
myroot->y=0;
myroot->w=WIDTH;
myroot->h=HEIGHT;

rootnode.win=myroot;
rootnode.parent=NULL;
rootnode.children=NULL;
rootnode.last_child=NULL;
rootnode.num_children=0;
rootnode.sibprev=NULL;
rootnode.sibnext=NULL;

fontinit();
event_init();
return (&dummy);

void XBell(Display *display, int percent)
{
    putchar ('\a');
    fflush(stdout);
}

void exitclean()
{
    fontexit();
event_exit();
    gl_clearscreen(0);
    vga_setmode(TEXT);
}

int p_in_rect(int x, int y, XRectangle *r)
{
    if ( (x<r->x) || (y<r->y) ) return(0);
    if ( ((x-(r->x)) <= (r->width)) && ((y-(r->y)) <= (r->height)) )
        return(1);
    else return(0);
}

void rect_intersect( XRectangle *r1, XRectangle *r2, XRectangle *out)
{
    int a1,a2,a3,a4;
    int b1,b2,b3,b4;

    a1=p_in_rect(r1->x, r1->y, r2);
    a3=p_in_rect(r1->width + r1->x, r1->height + r1->y, r2);
    if(a1 && a3)
    {
        out->x=r1->x;
        out->y=r1->y;
        out->width=r1->width;
        out->height=r1->height;
        return;
    }

    b1=p_in_rect(r2->x, r2->y, r1);
    b3=p_in_rect(r2->width + r2->x, r2->height + r2->y, r1);
    if(b1 && b3)
    {
        out->x=r2->x;
out->y=r2->y;
out->width=r2->width;
out->height=r2->height;
return;
}

if(a1 && b3)
{
    out->x=r1->x;
    out->y=r1->y;
    out->width=r2->width - r1->width - r1->x;
    out->height=r2->height - r1->y;
    return;
}

if(rl && b3)
{
    out->x=r1->x;
    out->y=r1->y;
    out->width=r2->width - r1->width - r1->height - r1->y;
    return;
}

a2=p_in_rect(rl->width + rl->x, rl->y, r2);
a4=p_in_rect(r1->x, r1->height + r1->y, r2);
b2=p_in_rect(r2->width + r2->x, r2->y, rl);
b4=p_in_rect(r2->x, r2->height + r2->x, r2->y, rl);

if(a4 && b2)
{
    out->x=r1->x;
    out->y=r1->y;
    out->width=r2->width - r1->width - r2->x;
    out->height=r1->height - r2->y;
    return;
}

if(a2 && b4)
{
    out->x=r2->x;
    out->y=r2->y;
    out->width=r1->width + r1->x - r2->x;
    out->height=r2->height - r1->y;
    return;
}

if(a1 && a4)
{
    out->x=r1->x;
    out->y=r1->y;
    out->width=r2->width + r2->x - r1->x;
    out->height=r1->height;
    return;
}

if(a2 && a3)
{
    out->x=r2->x;
    out->y=r2->y;
    out->width=r1->width + r1->x - r2->x;
    out->height=r1->height;
    return;
}

if(b1 && b4)
{
    out->x=r2->x;
    out->y=r2->y;
}
out->width=rl->x + rl->width - r2->x;
out->height=r2->height;
return;
}

if(b2 && b3)
{
  out->x=rl->x;
  out->y=r2->y;
  out->width=r2->x + r2->width - rl->x;
  out->height=r2->height;
  return;
}

if(a1 && a2)
{
  out->x=rl->x;
  out->y=r1->y;
  out->width=r1->width;
  out->height=r2->y + r2->height - rl->y;
  return;
}

if(a3 && a4)
{
  out->x=rl->x;
  out->y=r2->y;
  out->width=r1->width;
  out->height=rl->y + rl->height - r2->y;
  return;
}

if(b1 && b2)
{
  out->x=r2->x;
  out->y=r2->y;
  out->width=r2->width;
  out->height=rl->y + rl->height - r2->y;
  return;
}

if(b3 && b4)
{
  out->x=r2->x;
  out->y=r1->y;
  out->width=r2->width;
  out->height=r2->y + r2->height - rl->y;
  return;
}

out->x=0;
out->y=0;
out->width=0;
out->height=0;
return;
}

void refresh(Window w)
{
  Window pwin;
  winnode * wn;
  winnode * p;
  XRectangle r1,r2,r3;
  int draw;

  if (w==NULL) return;

  draw=1;

wn = (winnode *) w->node;

if ( w == myroot ) {
    gl_clearscreen(0);
    refresh(stacknext(w));
} else if (w->mapped)
{
    p= (winnode *) wn->parent;
    r.l.x=0;
    r.l.y=0;
    r.l.width=WIDTH;
    r.l.height=HEIGHT;

    while ( p != &rootnode )
    {
        pwin = (Window) p->win;
        if ( pwin->mapped == 0)
        {
            draw=0;
            break;
        }
        r2.x=pwin->x;
        r2.y=pwin->y;
        r2.width=pwin->w;
        r2.height=pwin->h;
        p= (winnode *) p->parent;
        rect_intersect(&r1,&r2,&r1);
    }

    if (draw)
    {
        gl_setclippingwindow(r1.x,r1.y,r1.x+r1.width,r1.y+r1.height);
        gl_putbox(w->x,w->y, w->w, w->h, w->pmap);
        if (w->bw > 1)
        {
            gl_fillbox(w->x - w->bw,w->y - w->bw, w->w + 2*w->bw, w->bw, w->bc);
            gl_fillbox(w->x - w->bw,w->y + w->w, w->w + 2*w->bw, w->bw, w->bc);
            gl_fillbox(w->x + w->w,w->y, w->w + 2*w->bw, w->bw, w->bc);
            gl_fillbox(w->x + w->w,w->y + w->w, w->w + 2*w->bw, w->bw, w->bc);
        }
        else if (w->bw==1)
        {
            gl_hline(w->x -1,w->y-1, w->x + w->w,w->bc);
            gl_hline(w->x -1,w->y + w->h, w->x + w->w,w->bc);
            gl_line(w->x -1,w->y+1, w->x+1,w->y,w->bc);
            gl_line(w->x -1,w->y+1, w->x + w->w,w->h,w->bc);
            gl_line(w->x + w->w,w->y+1, w->x + w->w,w->y,w->h,w->bc);
        }

        gl_disableclipping();
    }

    refresh(stacknext(w));
} else
{
    while((winnode *) wn->sibnext == NULL)
    {
        if (wn==&rootnode) return;
        wn = (winnode *) wn->parent;
    }
    refresh((winnode *) (wn->sibnext)->win );
}
Window stacknext(Window w)
{
  winnode * wn;

  wn= (winnode *) w->node;

  if (wn->num_children>0)
    return( ((winnode *) (wn->children))->win );

  while( (winnode *) wn->sibnext == NULL)
  {
    if (wn==&rootnode) return( (Window) NULL);
    wn = (winnode *) wn->parent;
  }

  return( ((winnode *) (wn->sibnext))->win );
}

void XCopyArea(Display *display,Drawable src,Drawable dest,GC gc,
               int src_x,int src_y,unsigned int width,
               unsigned int height,int dest_x,int dest_y)
{
  int x0,y0;
  GraphicsContext *savedcontext;
  unsigned char *bm;

  if(gc->clip_flag){
    x0=gc->clip_x_origin;
    y0=gc->clip_y_origin;
    gl_setclippingwindow((x0+gc->clip_mask.x),y0+gc->clip_mask.y,
                         x0+gc->clip_mask.x+gc->clip_mask.width,
                         y0+gc->clip_mask.y+gc->clip_mask.height);
  }

  savedcontext = gl_allocatecontext();
  gl_getcontext(savedcontext);

  gl_setcontextvirtual(src->w, src->h, BYTESPERPIXEL,
                       BITSPERPIXEL, src->pmap);
  bm = malloc(width * height * BYTESPERPIXEL);
  gl_getbox(src_x,src_y,width,height,bm);
  gl_setcontextvirtual(dest->w, dest->h, BYTESPERPIXEL,
                       BITSPERPIXEL, dest->pmap);
  glputbox(dest_x,dest_y, width, height, bm);
  free(bm);

  if(dest->winflag) refresh(dest);

  if(gc->clip_flag) gl_disableclipping();
}

XImage *XCreateImage(Display *display,Visual * visual,unsigned int depth,
int format, int offset, char *data, unsigned int width, unsigned int height, int bitmap_pad, int bytes_per_line)
{
    XImage * i;
    i = malloc(sizeof(XImage));
    i->w = width;
    i->h = height;
    i->pmap = (unsigned char *) data;
    return(i);
}

void XPutPixel(XImage *i, int x, int y, unsigned long pixel)
{
    GraphicsContext *savedcontext;
    savedcontext = gl_allocatecontext();
    gl_getcontext(savedcontext);
    gl_setcontextvirtual(i->w, i->h, BYTESPERPIXEL, BITSPERPIXEL, i->pmap);
    gl_setpixel(x, y, pixel);
    gl_setcontext(savedcontext);
}

void XPutImage(Display *display, Drawable dest, GC gc, XImage *src, int src_x, int src_y, int dest_x, int dest_y, unsigned int width, unsigned int height)
{
    int x0, y0;
    GraphicsContext *savedcontext;
    unsigned char *bm;
    int k;
    XRectangle r1, r2;
    unsigned char *temp;
    unsigned char *temp2;

    savedcontext = gl_allocatecontext();
    gl_getcontext(savedcontext);

    if (gc->function == GXcopy)
    {
        if (gc->clip_flag)
        {
            x0 = gc->clip_x_origin;
            y0 = gc->clip_y_origin;
            gl_setclippingwindow(x0 + gc->clip_mask.x, y0 + gc->clip_mask.y,
                                 x0 + gc->clip_mask.x + gc->clip_mask.width,
                                 y0 + gc->clip_mask.y + gc->clip_mask.height);
        }
        gl_setcontextvirtual(src->w, src->h, BYTESPERPIXEL,
                              BITSPERPIXEL, src->pmap);
        bm = malloc(width * height * BYTESPERPIXEL);
        gl_getbox(src_x, src_y, width, height, bm);
        gl_setcontextvirtual(dest->w, dest->h, BYTESPERPIXEL,
                              BITSPERPIXEL, dest->pmap);
        gl_putbox(dest_x, dest_y, width, height, bm);
        free(bm);
    }
}
} else {
    if ((gc->function != GXor) && (gc->function != GXand)) {
        printf("Unexpected RasterOp in XPutImage, exiting...\n");
        exit(-1);
    }
    rl.x = dest_x;
    rl.y = dest_y;
    rl.width = width;
    rl.height = height;
    if (gc->clip_flag) {
        r2.x = gc->clip_x_origin;
        r2.y = gc->clip_y_origin;
        r2.width = gc->clip_mask.width;
        r2.height = gc->clip_mask.height;
        rect_intersect(&rl, &r2, &rl);
    }
    temp = malloc(rl.width * rl.height * BYTESPERPIXEL);
    temp2 = malloc(rl.width * rl.height * BYTESPERPIXEL);
    gl_setcontextvirtual(src->w, src->h, BYTESPERPIXEL, BITSPERPIXEL, src->pmap);
    gl_getbox(src_x + (rl.x - dest_x), src_y + (rl.y - dest_y),
              rl.width, rl.height, temp2);
    gl_setcontextvirtual(dest->w, dest->h, BYTESPERPIXEL, BITSPERPIXEL, dest->pmap);
    gl_getbox(rl.x, rl.y, rl.width, rl.height, temp);
    if (gc->function == GXand) {
        for (k = 0; k < rl.width * rl.height * BYTESPERPIXEL; k++)
            *(temp + k) = *(temp + k) & *(temp2 + k);
    } else {
        for (k = 0; k < rl.width * rl.height * BYTESPERPIXEL; k++)
            *(temp + k) = *(temp + k) | *(temp2 + k);
    }
    gl_putbox(rl.x, rl.y, rl.width, rl.height, temp);
    free(temp);
    free(temp2);
}
/* here can also do physical screen */
gl_setcontext(savedcontext);
if (dest->winflag)
    refresh(dest);
if (gc->clip_flag) gl_disableclipping();

XImage *XGetImage(Display *display, Drawable d,
                  int x, int y, unsigned int width, unsigned int height,
                  unsigned long plane_mask, int format) {
    XImage *i;
    GraphicsContext *savedcontext;
    savedcontext = gl_allocatecontext();
    gl_getcontext(savedcontext);
i = malloc(sizeof(XImage));

i->w = width;
i->h = height;
i->pmap = (unsigned char *) malloc(width * height * BYTESPERPIXEL);

gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
BITSPERPIXEL, d->pmap);

gl_getbox(x, y, width, height, i->pmap);

/* here can also do physical screen */
gl_setcontext(savedcontext);

return(i);
}

void XDestroyImage(XImage *ximage)
{
free(ximage->pmap);
free(ximage);
}

Pixmap XCreatePixmap(Display display, unsigned int x,
unsigned int y, int width, unsigned int height, int depth)
{
Pixmap p;
GraphicsContext *savedcontext;

savedcontext = gl_allocatecontext();
gl_getcontext(savedcontext);

p = malloc(sizeof(Drawable_Struct));
p->winflag = 0;
p->w = width;
p->h = height;
p->pmap = malloc(width * height * BYTESPERPIXEL);

gl_setcontextvirtual(p->w, p->h, BYTESPERPIXEL,
BITSPERPIXEL, p->pmap);

gl_clearscreen(0);

/* here can also do physical screen */
gl_setcontext(savedcontext);

return(p);
}

void XFreePixmap(Display *display, Pixmap pixmap)
{
free(pixmap->pmap);
free(pixmap);
}

Window XCreateWindow(Display *display, Window parent,
int x, int y, unsigned int width, unsigned int height,
unsigned int border_width, int depth,
unsigned int class, Visual* visual, unsigned long valuemask,
XSetWindowAttributes *attributes)
{
Window w;
winnode * wn;
GraphicsContext *savedcontext;

w = malloc(sizeof(Drawable_Struct));
wn = malloc(sizeof(winnode));
w->winflag = 1;
w->mapped = 0;
w->w = width;
w->h = height;
w->x = parent->x + x;
w->y = parent->y + y;
w->xr = x;
w->yr = y;
w->pmap = malloc(width * height * BYTESPERPIXEL);

savedcontext = gl_allocatecontext();
gl_getcontext(savedcontext);
gl_setcontextvirtual(w->w, w->h, BYTESPERPIXEL, BITSPERPIXEL, w->pmap);
gl_clearscreen(0);
gl_setcontext(savedcontext);

w->bw = border_width;
w->bc = gl_rgbcolor(28, 28, 112); /* midnight blue */
w->swa = malloc(sizeof(XSetWindowAttributes));
w->wa = malloc(sizeof(XWindowAttributes));

w->node = (struct winnode *) wn;

wn->win = w;
wn->parent = parent->node;
wn->sibnext = NULL;
wn->children = NULL;
wn->last_child = NULL;
wn->num_children = 0;
wn->sibprev = ((winnode *) parent->node)->last_child;

if ( ((winnode *) parent->node)->last_child) != NULL )
    (winnode *) parent->node)->last_child->sibnext = (struct winnode *) wn;

if ( (winnode *) parent->node)->num_children == 0 )
    { 
        (winnode *) parent->node)->children = (struct winnode *) wn; 
    }

((winnode *) parent->node)->last_child = (struct winnode *) wn;
((winnode *) parent->node)->num_children += 1;

return(w);

void destroywin(Window w)
{
    int i;
    winnode * wn;
    winnode * ps;
    winnode * ns;
    winnode * p;

    wn = (winnode *) w->node;
    ps = (winnode *) wn->sibprev;
    ns = (winnode *) wn->sibnext;
    p = (winnode *) wn->parent;

    wn = w->node;
    wn->sibprev = ps;
    wn->sibnext = ns;
    wn->parent = p;

w->mapped=0;
free(w->pmap);
free(w->swa);
free(w->wa);
for(i=0;i < (wn->num_children);i++)
    {
        destroywin( ((winnode *) wn->children)->win );
    }
if( (winnode *) p->children == wn)
    p->children = (struct winnode *) ns;
if( (winnode *) p->last_child == wn)
    p->last_child = (struct winnode *) ps;
p->num_children--;
if (ns!=NULL)
    ns->sibprev= (struct winnode *) ps;
if (ps!=NULL)
    ps->sibnext= (struct winnode *) ns;
free(wn);
free(w);
void XDestroyWindow(Display *display, Window w)
{
    destroywin(w);
    if ( rootnode.num_children == 0 )
        exitclean();
    else refresh(myroot);
}
void XRaiseWindow(Display *display, Window w)
{
    winnode * wn;
    winnode * ps;
    winnode * ns;
    winnode * p;
    wn = (winnode *) w->node;
    ps = (winnode *) wn->sibprev;
    ns = (winnode *) wn->sibnext;
    p = (winnode *) wn->parent;
    if( (winnode *) p->last_child == wn) return;
    if( (winnode *) p->children == wn)
        p->children = (struct winnode *) ns;
    if (ns!=NULL)
        ns->sibprev= (struct winnode *) ps;
    if (ps!=NULL)
        ps->sibnext= (struct winnode *) ns;

    wn->sibnext=NULL;
    wn->sibprev= p->last_child;
    ( (winnode *) p->last_child)->sibnext= (struct winnode *) wn;
    p->last_child = (struct winnode *) wn;
    refresh(w);
}
void XLowerWindow(Display *display, Window w)
{
    winnode * wn;
    winnode * ps;
    winnode * ns;
    winnode * p;

    wn = (winnode *) w->node;
    ps = (winnode *) wn->sibprev;
    ns = (winnode *) wn->sibnext;
    p = (winnode *) wn->parent;

    if( (winnode *) p->children == wn) return;

    wn->sibprev=NULL;
    wn->sibnext= p->children;

    ( (winnode *) p->children)->sibprev= (struct winnode *) wn;

    p->children = (struct winnode *) wn;

    if( (winnode *) p->last_child == wn)
        p->last_child = (struct winnode *) ps;

    if (ns!=NULL)
        ns->sibprev= (struct winnode *) ps;
    if (ps!=NULL)
        ps->sibnext= (struct winnode *) ns;

    refresh(w);
}

void send_expose(Window win);

void XMapWindow(Display *display, Window w)
{
    w->mapped=1;
    send_expose(w);
    refresh(w);
}

void XUnmapWindow(Display *display, Window w)
{
    w->mapped=0;
    refresh(myroot);
}

void move_kids(Window w)
{
    winnode *wn;
    winnode *cn;
    Window child;

    wn= (winnode *) w->node;
    cn= (winnode *) wn->children;

    while (cn != NULL)
    {
        child=cn->win;
        child->x = child->xr + w->x;
        child->y = child->yr + w->y;
        move_kids(child);
        cn= (winnode *) cn->sibnext;
    }
return;
}

void XMoveWindow(Display *display, Window w, int x, int y)
{
    Window p;
    w->xr=x;
    w->yr=y;
    p= ( winnode *) ((winnode *) w->node )->parent)->win;
    w->x=p->x + x;
    w->y=p->y + y;
    move_kids(w);
    refresh(myroot);
}

void send_config(Window win);

void XResizeWindow(Display *display, Window w,
                   unsigned int width, unsigned int height)
{
    GraphicsContext *savedcontext;
    unsigned char * pm;
    unsigned char * tmp;

    pm=malloc(width*height*BYTESPERPIXEL);
    savedcontext = gl_allocatecontext();
    gl_getcontext(savedcontext);
    gl_setcontextvirtual(width, height, BYTESPERPIXEL,
                          BITSPERPIXEL, pm);
    gl_clearscreen(0);
    gl_setcontext(savedcontext);
    tmp=w->pmap;
    w->pmap=pm;
    free(tmp);
    send_config(w);
    send_expose(w);
    refresh(myroot);
}

void XMovResWindow(Display *display, Window w, int x, int y,
                    unsigned int width, unsigned int height)
{
    Window p;
    w->xr=x;
    w->yr=y;
    p= ( winnode *) ((winnode *) w->node )->parent)->win;
    w->x=p->x + x;
    w->y=p->y + y;
    move_kids(w);
    XResizeWindow(display, w, width, height);
int XQueryTree(Display *display, Window w,
              Window * root_return, Window * parent_return,
              Window ** children_return, unsigned int * nchildren_return)
{
    winnode * wn;
    Window * ch;
    int i;
    winnode * cn;
    wn = (winnode *) w->node;
    *root_return=myroot;
    *parent_return= ((winnode *) wn->parent)->win;
    *nchildren_return=wn->num_children;
    if (wn->num_children == 0)
    {
        *children_return=NULL;
        return(1);
    }
    ch=malloc( wn->num_children * sizeof(Window) );
    cn= (winnode *) wn->children;
    for (i=0;i<wn->num_children;i++)
    {
        ch[i]=cn->win;
        cn= (winnode *) cn->sibnext;
    }
    *children_return=ch;
    return(1);
}

int XGetWindowAttributes(Display *display, Window w,
                          XWindowAttributes *window_attributes_return)
{
    window_attributes_return->x = w->x;
    window_attributes_return->y = w->y;
    window_attributes_return->width = w->w;
    window_attributes_return->height = w->h;
    window_attributes_return->border_width = w->bw;
    return(1);
}

void XChangeWindowAttributes(Display *display, Window w,
                              unsigned long valuemask,
                              XSetWindowAttributes *attributes)
{
    /* only this is used in xwindows.c */
    if (CWCursor & valuemask)
    {
        (w->swa)->cursor = attributes->cursor;
    }
}

void XFree(void * p)
{
    if (p != NULL)
        free(p);
}
Appendix D gc.c

#include "my.h"

GC XCreateGC(Display *display, Drawable d,
        unsigned long valuemask, XGCValues * values)
{
    GC g;
    g = malloc(sizeof(XGCValues));
    g->line_width = 0;
    g->arc_mode = ArcPieSlice;
    g->function = GXcopy;
    g->foreground = 0;
    g->clip_x_origin = 0;
    g->clip_y_origin = 0;
    g->clip_flag = 0;
    g->plane_mask = -0;
    /* g->clip_mask = */

    // Font font; ?? */
    return (g);
}

void XChangeGC(Display *display, GC gc,
        unsigned long valuemask, XGCValues * values)
{
    if (valuemask & GCFunction) gc->function = values->function;
    if (valuemask & GCForeground) gc->foreground = values->foreground;
    if (valuemask & GCLineWidth) gc->line_width = values->line_width;
}

/* return value was Status */
int XGetGCValues(Display *display, GC gc,
        unsigned long valuemask, XGCValues * values_return)
{
    if (valuemask & GCFunction) values_return->function = gc->function;
    if (valuemask & GCForeground) values_return->foreground = gc->foreground;
    if (valuemask & GCLineWidth) values_return->line_width = gc->line_width;
}

void XFreeGC(Display *display, GC gc)
{
    free(gc);
}

void XSetFunction(Display *display, GC gc, int function)
{
    gc->function = function;
}

void XSetPlaneMask(Display *display, GC gc, unsigned long plane_mask)
{
    /* always -0 */
    gc->plane_mask = plane_mask;
void XSetFont(Display *display, GC gc, Font font)
{
    gc->font=font;
}

void XSetForeground(Display *display, GC gc, unsigned long foreground)
{
    gc->foreground=foreground;
}

void XSetArcMode(Display *display, GC gc, int arc_mode)
{
    gc->arc_mode=arc_mode;
}

void XSetClipRectangles(Display *display, GC gc, int clip_x_origin, int clip_y_origin, XRectangle rectangles[], int n, int ordering)
{
    /* n always is 1 or 0 */
    gc->clip_x_origin=clip_x_origin;
    gc->clip_y_origin=clip_y_origin;

    if (n==0)
    {
        gc->clip_flag=0;
    }
    else
    {
        gc->clip_flag=1;
        gc->clip_mask.x=rectangles[0].x;
        gc->clip_mask.y=rectangles[0].y;
        gc->clip_mask.width=rectangles[0].width;
        gc->clip_mask.height=rectangles[0].height;
    }
}
Appendix E color.c

#include "my.h"

void QC(XColor *c, int flag)
{
    int p;
    int r, g, b;
    if(flag)
    c->flags=(DoRed|DoGreen|DoBlue);

    p=gl_getpixel(100,100);
    gl_setpixel(100,100,c->pixel);
    glgetpixelrgb(100,100,&r,&g,&b);
    gl_setpixel(100,100,p);
    c->red=r;
    c->green=g;
    c->blue=b;
}

int XAllocColor(Display *display, Colormap colormap, XColor *c)
{
    int p;
    c->pixel=gl_rgbcolor(c->red,c->green,c->blue);
    QC(c,0);
    return(1);
}

void XQueryColors(Display *display, Colormap colormap, XColor *defs_in_out, int ncolors)
{
    int i;
    for (i=0; i<ncolors; i++)
        QC(&defsinout[i],1);
}

int colcomp(char* s1, char* s2, int n)
{
    int i;
    char cl, c2;
    int l1, l2;
    if (strncmp(s1, s2, n)==0) return(0);
    l1=strlen(s1);
    l2=strlen(s2);
    if ( abs(l1-l2)>1 ) return(-1);
    if ( abs(l1-l2)==1 ) n--;
    for (i=0; i<n; i++)
    {
        cl=s1[i];
        c2=s2[i];
        if ( (int)cl>64 && (int)cl<91 ) cl= (char) ((int)cl+32);
        if ( (int)c2>64 && (int)c2<91 ) c2= (char) ((int)c2+32);
        if ( (int) cl != (int) c2 ) return(-1);
    }
    return(0);
int XParseColor(Display *display, Colormap colormap,char *spec,
    XColor *exact_def_return)
{
    FILE *cols;
    int r,g,b;
    char buf[30];
    int l1,l2,l;
    int flag=0;
    int sc;
    l1=strlen(spec);
    cols=fopen("rgb.txt","r");
    r=-1;
    fscanf(cols,"%d %d %d ",&r, &g, &b);
    while (r != -1)
    {
        fgets(buf,30,cols);
        l2=strlen(buf);
        l2=l2-1;
        if (l1>l2) l=l1; else l=l2;
        sc=colcomp(buf,spec,l);
        if (sc==0)
        {
            flag=1;
            break;
        }
        r=-1;
        fscanf(cols,"%d %d %d ",&r, &g, &b);
    }
    fclose(cols);
    if (flag)
    {
        exact_def_return->red=r;
        exact_def_return->green=g;
        exact_def_return->blue=b;
        return(1);
    }
    else return(0);
}
Appendix F graphics.c

#include "my.h"

void triangle(int x0,int y0,int x1,int y1,int x2,int y2,int c, int xorflag);
void circle(float x0,float y0, float w, float h, int c, int al, int a2);
void polygon(int num, XPoint* points, int c,int xorflag);
void line(int xl, int yl, int x2, int y2, int w, int c);
void xorline(int xl, int yl, int x2, int y2, int w, int c, unsigned char *pm,int wi, int h);
void lines(int num, XPoint* p, int w,int col);
void fill_quad(float x0,float y0, float w, float h, int quad, int c);
void fill_arc(float x0,float y0, float w, float h, int c, int al, int a2);

void rect_intersect( XRectangle *r1, XRectangle *r2, XRectangle *out);

void XDrawArc(Display *display,Drawable d,GC gc, int x,int y,unsigned int width,unsigned int height, int angle1,int angle2)
{
  int x0,y0;
  int i;
  GraphicsContext *savedcontext;

  if (width==0 || height==0)
    return;

  savedcontext = gl_allocatecontext();
  gl_getcontext(savedcontext);
  gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
       BITS_PER.Pixel, d->pmap);

  if(gc->clip_flag){
    x0=gc->clip_x_origin;
    y0=gc->clip_y_origin;
    gl_setclippingwindow(x0+gc->clip_mask.x,y0+gc->clip_mask.y,
      x0+gc->clip_mask.x+gc->clip_mask.width,
      y0+gc->clip_mask.y+gc->clip_mask.height);
  }

  if (gc->line_width==0)
    circle(x,y,width,height,gc->foreground, angle1, angle2);
  for(i=0;i<gc->line_width;i++)
    circle(x+i,y+i,width-2*i,height-2*i,gc->foreground, angle1, angle2);

  gl_setcontext(savedcontext);
  if(d->winflag)
    refresh(d);
  if(gc->clip_flag) gl_disableclipping();
}

void XDrawLine(Display *display,Drawable d,GC gc,
           int xl,int yl,int x2,int y2, int w,int col)
{
  int x0,y0;
  GraphicsContext *savedcontext;

  savedcontext = gl_allocatecontext();
  gl_getcontext(savedcontext);
  gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
       BITS_PER.Pixel, d->pmap);

  gl_setcontext(savedcontext);
  if(d->winflag)
    refresh(d);
  if(gc->clip_flag) gl_disableclipping();
}
if(gc->clip_flag)
{
    x0=gc->clip_x_origin;
    y0=gc->clip_y_origin;
    gl_setclippingwindow(x0+gc->clip_mask.x,y0+gc->clip_mask.y,
                         x0+gc->clip_mask.x+gc->clip_mask.width,
                         y0+gc->clip_mask.y+gc->clip_mask.height);
}

if ( gc->function== GXcopy )
{
    line(x1,y1,x2,y2,gc->line_width,gc->foreground);
}
else
{
    if (gc->function != GXxor) {
        printf("Unexpected RasterOp in XDrawLine, exiting...
" );
        exit(-1);
    }
    xorline(x1,y1,x2,y2,gc->line_width,gc->foreground,d->pmap, d->w, d->h);
}

gl_setcontext(savedcontext);
if(d->winflag)
    refresh(d);
if(gc->clip_flag) gl_disableclipping();

void XDrawLines(Display *display,Drawable d,GC gc,
                 XPoint *points, int npoints, int mode)
{
    int x0,y0;
    GraphicsContext *savedcontext;

    savedcontext = gl_allocatecontext();
    gl_getcontext(savedcontext);
    gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
                         BITSPERPIXEL, d->pmap);

    if(gc->clip_flag){
        x0=gc->clip_x_origin;
        y0=gc->clip_y_origin;
        gl_setclippingwindow(x0+gc->clip_mask.x,y0+gc->clip_mask.y,
                             x0+gc->clip_mask.x+gc->clip_mask.width,
                             y0+gc->clip_mask.y+gc->clip_mask.height);
    }

    lines(npoints, points,gc->line_width,gc->foreground);
    gl_setcontext(savedcontext);
    if(d->winflag)
        refresh(d);
    if(gc->clip_flag) gl_disableclipping();
}

void XDrawRectangle(Display *display,Drawable d,GC gc,
                     int x,int y,unsigned int width,unsigned int height)
{
    int w,c;
XPoint p[5];
int x0,y0;
GraphicsContext *savedcontext;

savedcontext = gl_allocatecontext();
gl_getcontext(savedcontext);

gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
BITSPERPIXEL, d->pmap);

if(gc->clip_flag){
  x0=gc->clip_x_origin;
  y0=gc->clip_y_origin;
  gl_setclippingwindow(x0+gc->clip_mask.x, y0+gc->clip_mask.y, x0+gc->clip_mask.x+gc->clip_mask.width, y0+gc->clip_mask.y+gc->clip_mask.height);
}

c=gc->foreground;
w=gc->line_width;

p[0].x=x;
p[0].y=y;
p[1].x=x+w;
p[1].y=y;
p[2].x=x+w;
p[2].y=y+h;
p[3].x=x;
p[3].y=y+h;
p[4].x=x;
p[4].y=y;

if (w==0 || w==1)
  {
    p[0].x=x;
p[0].y=y;
p[1].x=x+w;
p[1].y=y;
p[2].x=x+w;
p[2].y=y+h;
p[3].x=x;
p[3].y=y+h;
p[4].x=x;
p[4].y=y;
    lines(5,p,w,c);
  }
else
  {
    line(Rnd(x-w/2),y,Rnd(x+w+w/2),y,w,c);
    line(Rnd(x-w/2),y+h,Rnd(x+w+w/2),y+h,w,c);
    line(x,y,x+y+h,w,c);
    line(x+w,y,x+w+y+h,w,c);
  }

gl_setcontext(savedcontext);
if(d->winflag)
  refresh(d);

if(gc->clip_flag) gl_disableclipping();
}

void XFillArc(Display *display, Drawable d, GC gc,
  int x, int y, unsigned int width, unsigned int height,
  int angle1, int angle2)
  {
  int x0,y0;
  GraphicsContext *savedcontext;

if (width==0 || height==0)
    return;

savedcontext = gl_allocatecontext();
gl_getcontext(savedcontext);

gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
                      BITSPIXEL, d->pmap);

if (gc->clip_flag){
    x0=gc->clip_x_origin;
    y0=gc->clip_y_origin;
    gl_setclippingwindow(x0+gc->clip_mask.x,y0+gc->clip_mask.y,
                          x0+gc->clip_mask.x+gc->clip_mask.width,
                          y0+gc->clip_mask.y+gc->clip_mask.height);
}
else {
    gl_setclippingwindow(x,y,x+width,y+height);
}

fill_arc(x,y,width,height,gc->foreground, angle1, angle2);
gl_setcontext(savedcontext);
if(d->winflag)
    refresh(d);

if(gc->clip_flag) gl_disableclipping();

void XFillPolygon(Display *display,Drawable d,GC gc,
                   XPoint *points,int npoints,int shape,int mode)
{
    int x0,y0;
    GraphicsContext *savedcontext;

    savedcontext = gl_allocatecontext();
gl_getcontext(savedcontext);

    gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
                         BITSPIXEL, d->pmap);

    if(gc->clip_flag){
        x0=gc->clip_x_origin;
        y0=gc->clip_y_origin;
        gl_setclippingwindow(x0+gc->clip_mask.x,y0+gc->clip_mask.y,
                             x0+gc->clip_mask.x+gc->clip_mask.width,
                             y0+gc->clip_mask.y+gc->clip_mask.height);
    }

    polygon(npoints, points,gc->foreground,0);

gl_setcontext(savedcontext);
    if(d->winflag)
        refresh(d);
    if(gc->clip_flag) gl_disableclipping();
}

void XFillRectangle(Display *display,Drawable d,GC gc,
                     int x,int y,unsigned int width,unsigned int height)
{
    int x0,y0,k;
    GraphicsContext *savedcontext;

    XRectangle rl,r2;
    unsigned char * temp;
    unsigned char * temp2;
if (width==0 || height==0)
    return;

savedcontext = gl_allocatecontext();
gl_getcontext(savedcontext);

if (gc->function == GXcopy)

if (width==1)
    {gl_line(x,y,x,y, height, gc->foreground);
        return;
    }

if (height==1)
    {g_hline(x,y,x+width, gc->foreground);
        return;
    }

if(gc->clip_flag){
    x0=gc->clip_x_origin;
y0=gc->clip_y_origin;
gl_setclippingwindow(x0+gc->clip_mask.x, y0+gc->clip_mask.y,
x0+gc->clip_mask.x+gc->clip_mask.width,
y0+gc->clip_mask.y+gc->clip_mask.height);
}

else
    {
    if (gc->function != GXxor)
        {
            printf("Unexpected RasterOp in FillRect, exiting...\n");
            exit(-1);
        }

    rl.x=x;
    rl.y=y;
    rl.width=width;
    rl.height=height;

    if(gc->clip_flag)
        {
            r2.x=gc->clip_x_origin;
r2.y=gc->clip_y_origin;
r2.width=gc->clip_mask.width;
r2.height=gc->clip_mask.height;
            rect_intersect(&rl, &r2, &rl);
        }

    temp = malloc(rl.width * rl.height * BYTESPERPIXEL);
temp2 = malloc(rl.width * rl.height * BYTESPERPIXEL);

    gl_getbox(rl.x, rl.y, rl.width, rl.height, temp);
    gl_setcontextvirtual(rl.width, rl.height, BYTESPERPIXEL, BITSPERPIXEL, temp2);
    gl_clearscreen(gc->foreground);

    gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL, BITSPERPIXEL, d->pmap);
for(k=0;k<rl.width * rl.height * BYTESPERPIXEL;k++)
    *(temp + k) = *(temp + k) ^ *(temp2 + k);

    gl_putbox(rl.x, rl.y, rl.width, rl.height, temp);
    free(temp);
    free(temp2);
}

    gl_setcontext(savedcontext);
    if(d->winflag)
        refresh(d);
    if(gc->clip_flag) gl_disableclipping();


void circle(float x0, float y0, float w, float h, int c, int al, int a2)
{
    float t, a, b;
    x=x0+w/2;
    y=y0+h/2;
    a=w/2;
    b=h/2;
    a2=-a2;
    if (a2>0)
        for (t= al*2*PI/(64*360);
            t<=(al+a2)*2*PI/(64*360);
            t += (PI/(2*w+2*h)))
            gl_setpixel(Rnd(a*cos(t)+x), Rnd(b*sin(t)+y), c);
    else
        for (t= al*2*PI/(64*360);
            t>=(al+a2)*2*PI/(64*360);
            t -= (PI/(2*w+2*h)))
            gl_setpixel(Rnd(a*cos(t)+x), Rnd(b*sin(t)+y), c);
}

void fill_quad(float x0, float y0, float w, float h, int quad, int c)
{
    float x, y, a, b, j;
    int i, t;
    x=x0+w/2;
    y=y0+h/2;
    a=w*w/4;
    b=h*h/4;
    for (i=(int) y0+l;i<y0+h/2;i++)
        if (j>=(a-a*(i-y)*(i-y)/b))
            switch (quad) {
                case 1:
                    gl_hline(Rnd(x),i,Rnd(x+j),c); break;
                case 2:
                    gl_hline(Rnd(x-j),i,Rnd(x),c); break;
                case 3:
                    gl_hline(Rnd(x-j),2*y0+h-i,Rnd(x),c); break;
                case 4:
                    break;
            }
if ( (int)h%2 == 0 )
switch(quad) {
    case 1:
        {gl_hline(Rnd(x),i,Rnd(x+w/2),c); break;}
    case 2:
        {gl_hline(Rnd(x-w/2),i,Rnd(x),c); break;}
    case 3:
        {gl_hline(Rnd(x-w/2),2*y0+h-i,Rnd(x),c); break;}
    case 4:
        {gl_hline(Rnd(x),2*y0+h-i,Rnd(x+w/2),c); break;}
}

int quad(float a)
{
    while (a<0)
        a += 360*64;
    while (a>=360*64)
        a -= 360*64;
    if (a<=90*64) return(1);
    else if (a<=180*64) return(2);
    else if (a<=270*64) return(3);
    else return(4);
}

void farc(float x0,float y0, float w, float h, int c,
           float a1, float a2, int q)
{
    float x,y,a,b,j;
    int i,t;
    float t1, t2;
    float as,bs;
    float x1,x2;
    float y1,y2;
    float temp,r;
    float th1,th2;
    x=x0+w/2;
    y=y0+h/2;
    a=w/2;
    as=a*a;
    bs=b*b;
    switch (q){
    case 1:
        t1=a1;
        t2=a2;
        break;
    case 2:
        t2=a1;
        t1=a2;
        t1=180*64-t1;
        t2=180*64-t2;
        break;
    case 3:
        t1=a1;
        t2=a2;
        t1=(180*64);
        t2=(180*64);
        break;
    case 4:
        t1=a1;
        t2=a2;
        t1=(180*64);
        t2=(180*64);
        break;
    }
case 4:
    t2=a1;
    t1=a2;
    t1=(360*64-t1);
    t2=(360*64-t2);
    break;
}
while(tl<0) tl+=360*64;
while(tl>360*64) tl-=360*64;
while(t2<tl) t2+=360*64;

    tl=tl/360/64*2*PI;
    t2=t2/360/64*2*PI;

    th1=fabs(atan(a/b*tan(tl)));
    th2=fabs(atan(a/b*tan(t2)));

    for (i=Rnd(y-b*sin(th2));i<(l+Rnd(y-b*sin(thl)));i++)
    {
        y1=i;
        x1=x+sqrt( (as-as*(i-y)*(i-y)/bs) );
        x2=x-(i-y)/tan(t2);
        switch(q)
        {
            case 1:
                break;
            case 2:
                { x1=2*x-x1;
                x2=2*x-x2;
                break; }
            case 3:
                { x1=2*x-x1;
                x2=2*x-x2;
                y1=2*y-y1;
                break; }
            case 4:
                { y1=2*y-y1;
                break; }
        }
        if (x1<x2) {
            gl_hline(Rnd(x1),y1,Rnd(x2),c);
        } else {
            gl_hline(Rnd(x2),y1,Rnd(x1),c);
        }
    }

if (i<y) {
    x1=x+a*cos(th1);
    y1=y-b*sin(th1);
    x2=x+(y-y1)/tan(th2);
}
switch(q)
{
    case 1:
    {
        break;
    }
    case 2:
    {
        x1=2*x-x1;
        x2=2*x-x2;
        break;
    }
    case 3:
    {
        x1=2*x-x1;
        x2=2*x-x2;
        y1=2*y-y1;
        break;
    }
    case 4:
    {
        y1=2*y-y1;
        break;
    }
}

triangle(Rnd(x),Rnd(y),Rnd(x1),Rnd(y1),Rnd(x2),Rnd(y1),c,0);

void fill_arc(float x0,float y0, float w, float h, int c, int al, int a2)
{
    float t1, t2, tt;
    int q1, q2;
    if (a2>=360*64 || a2<-360*64)
    {
        fill_quad(x0,y0,w,h,1,c);
        fill_quad(x0,y0,w,h,2,c);
        glhline(Rnd(x0),Rnd(y0+h/2),Rnd(x0+w),c);
        fill_quad(x0,y0,w,h,3,c);
        fill_quad(x0,y0,w,h,4,c);
        return;
    }
    t1=al;
    t2=al+a2;
    if (a2<0){
        tt=t1;
        t1=t2;
        t2=tt;
    }
    while(t1<0) t1+=360*64;
    while(t1>360*64) t1-=360*64;
    while(t2<tt) t2+=360*64;
    q1=quad(t1);
    q2=quad(t2);
    if (q1!=q2)
    switch (q1)
    {
        case 1: {
            farc(x0,y0,w,h,c,t1,90*64,1);
            switch (q2)
            {
                case 2: {
                    farc(x0,y0,w,h,c,90*64,t2,2);
                }
            }
        }
    }
}
break;
case 3: {
farc(xO,yO,w,h,c,180*64,t2,3);
fill_quad(xO,yO,w,h,2,c);
break;
}
case 4: {
farc(xO,yO,w,h,c,270*64,t2,4);
fill_quad(xO,yO,w,h,3,c);
gl_hline(Rnd(x0),Rnd(y0+h/2),Rnd(x0+w/2),c);
fill_quad(xO,yO,w,h,1,c);
break;
}
break;
}
case 2: {
farc(xO,y0,w,h,c,t1,180*64,2);
switch (q2) {
  case 3: {
farc(xO,y0,w,h,c,180*64,t2,3);
break;
  }
case 4: {
farc(xO,y0,w,h,c,270*64,t2,4);
fill_quad(xO,y0,w,h,3,c);
break;
  }
case 1: {
farc(x0,y0,w,h,c,0,t2,1);
fill_quad(xO,yO,w,h,4,c);
fill_quad(xO,yO,w,h,3,c);
break;
  }
case 3: {
farc(xO,y0,w,h,c,tl,270*64,3);
switch (q2) {
  case 4: {
farc(xO,y0,w,h,c,0,t2,1);
break;
  }
case 2: {
farc(xO,yO,w,h,c,90*64,t2,2);
fill_quad(xO,y0,w,h,4,c);
gl_hline(Rnd(x0+w/2),Rnd(y0+h/2),Rnd(xO+w),c);
fill_quad(xO,yO,w,h,1,c);
break;
  }
}
break;
}
case 3: {
farc(xO,y0,w,h,c,tl,180*64,2);
switch (q2) {
  case 4: {
farc(xO,y0,w,h,c,180*64,t2,3);
break;
  }
case 4: {
farc(xO,y0,w,h,c,270*64,t2,4);
fill_quad(xO,y0,w,h,3,c);
break;
  }
}
break;
}
case 3: {
farc(xO,y0,w,h,c,t1,270*64,3);
switch (q2) {
  case 4: {
farc(xO,y0,w,h,c,270*64,t2,4);
break;
  }
case 1: {
farc(x0,y0,w,h,c,0,t2,1);
fill_quad(xO,yO,w,h,4,c);
fill_quad(xO,yO,w,h,3,c);
break;
  }
case 2: {
farc(x0,y0,w,h,c,90*64,t2,2);
fill_quad(xO,y0,w,h,4,c);
gl_hline(Rnd(x0+w/2),Rnd(y0+h/2),Rnd(xO+w),c);
fill_quad(xO,yO,w,h,1,c);
break;
  }
}
break;
}
case 4: {
farc(xO,y0,w,h,c,t1,360*64,4);
switch (q2) {
  case 1: {
farc(xO,y0,w,h,c,0,t2,1);
break;
  }
case 2: {
farc(x0,y0,w,h,c,90*64,t2,2);
fill_quad(xO,y0,w,h,1,c);
break;
  }
case 3: {
farc(x0,y0,w,h,c,180*64,t2,3);
fill_quad(xO,y0,w,h,1,c);
fill_quad(xO,y0,w,h,2,c);
break;
  }
}
```c
if ( (((int)t1 / 64) % 360) <= (((int)t2 / 64)%360) )
  {  
    farc(x0,y0,w,h,c,t1,t2,q1);
  }
else {  
  switch (q1)
  {    
    case 1:
      { 
        farc(x0,y0,w,h,c,t1,90*64,1); 
        farc(x0,y0,w,h,c,0,t2,1); 
        fill_quad(x0,y0,w,h,2,c); 
        gl_hline(Rnd(x0),Rnd(y0+h/2),Rnd(x0+w/2),c); 
        fill_quad(x0,y0,w,h,1,c); 
        fill_quad(x0,y0,w,h,4,c); 
        break;
      }
    case 2:
      {  
        farc(x0,y0,w,h,c,t1,180*64,2); 
        farc(x0,y0,w,h,c,90*64,t2,2); 
        fill_quad(x0,y0,w,h,3,c); 
        gl_hline(Rnd(x0+w/2),Rnd(y0+h/2),Rnd(x0+w),c); 
        fill_quad(x0,y0,w,h,1,c); 
        fill_quad(x0,y0,w,h,4,c); 
        break;
      }
    case 3:
      {  
        farc(x0,y0,w,h,c,t1,270*64,3); 
        farc(x0,y0,w,h,c,180*64,t2,3); 
        fill_quad(x0,y0,w,h,1,c); 
        gl_hline(Rnd(x0+w/2),Rnd(y0+h/2),Rnd(x0+w),c); 
        fill_quad(x0,y0,w,h,2,c); 
        fill_quad(x0,y0,w,h,4,c); 
        break;
      }
    case 4:
      {  
        farc(x0,y0,w,h,c,t1,0,4); 
        farc(x0,y0,w,h,c,270*64,t2,4); 
        fill_quad(x0,y0,w,h,1,c); 
        fill_quad(x0,y0,w,h,2,c); 
        gl_hline(Rnd(x0),Rnd(y0+h/2),Rnd(x0+w/2),c); 
        fill_quad(x0,y0,w,h,3,c); 
        break;
      }
  }
}
}

void xorline(int xl, int yl, int x2, int y2, int w, int col, 
              unsigned char * pm, int wi, int h)
{
  XPoint p[4];
  float t,s,c;
  int xmin, ymin, xmax, ymax,k;
  unsigned char * temp;
  unsigned char * temp2;
  GraphicsContext *savedcontext;
  
  if (w==0 || w==1)
```
if (xl<x2)
    {  
        xmin=xl;  
        xmax=x2;  
    }
else
    {  
        xmin=x2;  
        xmax=x1;  
    }

if (yl<y2)
    {  
        ymin=yl;  
        ymax=y2;  
    }
else
    {  
        ymin=y2;  
        ymax=y1;  
    }

temp =malloc( (xmax-xmin)* (ymax-ymin)*BYTESPERPIXEL );  
temp2=malloc( (xmax-xmin)* (ymax-ymin)*BYTESPERPIXEL );  
gl_getbox(xmin, ymin, (xmax-xmin), (ymax-ymin), temp);  
savedcontext = gl_allocatecontext();  

gl_setcontextvirtual((xmax-xmin),(ymax-ymin),BYTESPERPIXEL,BITSPERPIXEL,temp2);  
gl_clearscreen(0);  
gl_line(xl-xmin, yl-ymin, x2-xmin, y2-ymin, col);  
gl_setcontext(savedcontext);  
for(k=0;k< (xmax-xmin) * (ymax-ymin) * BYTESPERPIXEL;k++)  
    *(temp + k)= *(temp + k) ^ *(temp2 + k);  
gl_putbox(xmin, ymin, (xmax-xmin), (ymax-ymin), temp);  
free(temp);  
free(temp2);  
}
#endif  

if (w==0 || w==1)
    {  
        temp2=malloc( wi * h * BYTESPERPIXEL );  
savedcontext=gl_allocatecontext();  
gl_getcontext(savedcontext);  
gl_setcontextvirtual(wi,h,BYTESPERPIXEL,BITSPERPIXEL,temp2);  
gl_clearscreen(0);  
gl_line(x1, y1, x2, y2, col);  
gl_setcontext(savedcontext);  
for(k=0;k< h * wi * BYTESPERPIXEL;k++)  
    *(pm + k)= *(pm + k) ^ *(temp2 + k);  
free(temp2);  
}
else
{
    if (x1==x2) t=PI/2;
    else t=atan( (y1-y2)/(x1-x2) );
    s=w*sin(t)/2;
    c=w*cos(t)/2;
    p[0].x=Rnd(x1+s);
    p[0].y=Rnd(y1-c);
    p[1].x=Rnd(x1-s);
    p[1].y=Rnd(y1+c);
    p[2].x=Rnd(x2-s);
    p[2].y=Rnd(y2+c);
    p[3].x=Rnd(x2+s);
    p[3].y=Rnd(y2-c);
    polygon(4,p,col,1);
}

void line(int xl, int yl, int x2, int y2, int w, int col)
{
    XPoint p[4];
    float t,s,c;
    if (w==0 || w==1)
    {
        gl_line(xl, yl, x2, y2, col);
    }
    else
    {
        if (x1==x2) t=PI/2;
        else t=atan( (y1-y2)/(x1-x2) );
        s=w*sin(t)/2;
        c=w*cos(t)/2;
        p[0].x=Rnd(x1+s);
        p[0].y=Rnd(y1-c);
        p[1].x=Rnd(x1-s);
        p[1].y=Rnd(y1+c);
        p[2].x=Rnd(x2-s);
        p[2].y=Rnd(y2+c);
        p[3].x=Rnd(x2+s);
        p[3].y=Rnd(y2-c);
        polygon(4,p,col,0);
    }
}

void polygon(int num, XPoint* p, int c, int xorflag)
{
    int i;
    if (num==3)
    { triangle(p[0].x, p[0].y,p[1].x, p[1].y,p[2].x, p[2].y, c,xorflag); }
    else
    { triangle(p[0].x, p[0].y, p[num-1].x, p[num-1].y, p[num-2].x, p[num-2].y, c,xorflag);
        polygon(num-1, p, c,xorflag);
    }
    return;
}

void xorhline(int xl, int y, int x2, int c)
{
    GraphicsContext *savedcontext;
    int k;
unsigned char * temp;
unsigned char * temp2;

savedcontext = gl_allocatecontext();
gl_getcontext(savedcontext);

temp = malloc( (x2-xl) * 1 * BYTESPERPIXEL);
temp2 = malloc( (x2-xl) * 1 * BYTESPERPIXEL);

for(k=0;k< (x2-xl) * 1 * BYTESPERPIXEL;k++)
  *(temp + k) = *(temp2 + k);

void triangle(int x0,int y0,int xl,int yl,int x2,int y2,int c,int xorflag)
{
  int i;
  int x[3], y[3];
  int min, max, mid;
  int xa, xb;

  x[0]=x0;
y[0]=y0;
x[1]=xl;
y[1]=yl;
x[2]=x2;
y[2]=y2;

  if (yl>y[1]) max=1;
  if (yl<y[min]) min=1;
  if (y2>y[1]) max=2;
  if (y2<y[min]) min=2;

  if (min==1 | max==0)
    if (min==1 | max==1)
      mid=2;
    else mid=1;
  else mid=0;

  for (i=y[min]+1; i<=y[mid]; i++){
    xa=(int) Rnd( (float) x[0] + ((float) (i-y[min]))*((float)(x[1]-
x[0]))/((float)(y[mid]-y[min]))));
    xb=(int) Rnd( (float) x[1] + ((float) (i-y[min]))*((float)(x[2]-
x[1]))/((float)(y[mid]-y[min]))));

    if (xorflag)
      if (xa<xb)
        xorhline(xa , i , xb ,c);
else xorhline(xb, i, xa, c);
}
else
{
    if (xa<xb)
        gl_hline(xa, i, xb, c);
    else gl_hline(xb, i, xa, c);
}
}

for (i=y[mid]+1; i<y[max]; i++) {
    xa=(int) Rnd((float) (x[mid] + ((float) (i-y[mid])) * ((float) (x[max]-x[mid])))/((float) (y[max]-y[mid]))));
    xb=(int) Rnd((float) (x[min] + ((float) (i-y[min])) * ((float) (x[max]-x[min])))/((float) (y[max]-y[min]))));

    if (xorflag)
    {
        if (xa<xb)
            xorhline(xa, i, xb, c);
        else xorhline(xb, i, xa, c);
    }
    else
    {
        if (xa<xb)
            gl_hline(xa, i, xb, c);
        else gl_hline(xb, i, xa, c);
    }
}
loat fullang(float x0, float y0, float x, float y)
{
    int q2;
    float t1;

    if (x==x0) t1=PI/2;
    else t1= atan((y0-y)/(x-x0));

    if ( x>x0 && y<=y0) q2=1;
    else if ( x<x0 && y<=y0) q2=2;
    else if ( x<x0 && y>y0) q2=3;
    else if ( x>x0 && y>=y0) q2=4;
    else q2=1;

    switch (q2){
    case 1:{break;}
    case 2:{t1+=PI ;break;}
    case 3:{t1+=PI ;break;}
    case 4:{t1+=2*PI ;break;}
    }

    return(t1);
}

float fullang2(float x0, float y0, float x, float y)
{
    int q2;
    float t1;

    if (x==x0) t1=PI/2;
    else t1= atan((y-y0)/(x-x0));

    if ( x>x0 && y<=y0) q2=1;
    else if ( x<x0 && y<=y0) q2=2;
    else if ( x<x0 && y>y0) q2=3;
    else if ( x>x0 && y>=y0) q2=4;
else q2=1;

switch (q2){
    case 1:{break;}
    case 2:{t1+=PI ;break;}
    case 3:{t1+=PI ;break;}
    case 4:{t1+=2*PI ;break;}
}

return(t1);

void miter(XPoint * pts, int w)
{
    XPoint p[4];
    XPoint fp[4];
    float t1,t2,t3,t4,t;
    float tla,t2a;
    double ml,m2,xf,yf;
    float a1,b1,a2,b2,x1,y1,x2,y2;
    XPoint a,b;
    int i,j;
    float s,c;
    int ql,q2;

    for (j=0;j<2;j++)
    {
        if (j==0) {
            p[0]=pts[0];
            p[1]=pts[1];
        }
        else {
            p[0]=pts[3];
            p[1]=pts[2];
            p[2]=pts[1];
        }

        if ( (p[0].x==p[1].x) && (p[0].y==p[1].y) )
        {
            if (p[1].x==p[2].x) t=PI/2;
            s=w*sin(t)/2;
            c=w*cos(t)/2;
            a.x=Rnd(p[1].x+s);
            a.y=Rnd(p[1].y-c);
            b.x=Rnd(p[1].x-s);
            b.y=Rnd(p[1].y+c);
        }
        else
        {
            tla=fullang(p[1].x,p[1].y,p[0].x,p[0].y);
            t2a=fullang(p[1].x,p[1].y,p[2].x,p[2].y);
            t1=fullang2(p[1].x,p[1].y,p[0].x,p[0].y);
            t2=fullang2(p[1].x,p[1].y,p[2].x,p[2].y);
            t3=fullang(p[1].x,p[1].y,p[1].x+w*sin(t1)/2,p[1].y-w*cos(t1)/2);
            t4=fullang(p[1].x,p[1].y,p[1].x-w*sin(t1)/2,p[1].y+w*cos(t1)/2);
            ml=fabs(t3-t2a);
            m2=fabs(t4-t2a);
            if (ml>PI) ml=2*PI-ml;
            if (m2>PI) m2=2*PI-m2;
            if ( ml > m2 )

        }
    }

}
{  
    a1=p[1].x+w*sin(t1)/2;
    b1=p[1].y-w*cos(t1)/2;
    a2=p[0].x+w*sin(t1)/2;
    b2=p[0].y-w*cos(t1)/2;
}  
else  
{  
    a1=p[1].x-w*sin(t1)/2;
    b1=p[1].y+w*cos(t1)/2;
    a2=p[0].x-w*sin(t1)/2;
    b2=p[0].y+w*cos(t1)/2;
}  
/**/  
t3=fullang(p[1].x,p[1].y,p[1].x+w*sin(t2)/2,p[1].y-w*cos(t2)/2);
t4=fullang(p[1].x,p[1].y,p[1].x-w*sin(t2)/2,p[1].y+w*cos(t2)/2);

ml=fabs(t3-tla);
m2=fabs(t4-tla);
if (ml>PI) ml=2*PI-ml;
if (m2>PI) m2=2*PI-m2;
if (ml > m2 )  
{  
    x1=p[1].x+w*sin(t2)/2;
    y1=p[1].y-w*cos(t2)/2;
    b.x=Rnd(p[1].x-w*sin(t2)/2);
    b.y=Rnd(p[1].y+w*cos(t2)/2);
    x2=p[2].x+w*sin(t2)/2;
    y2=p[2].y-w*cos(t2)/2;
}  
else  
{  
    x1=p[1].x-w*sin(t2)/2;
    y1=p[1].y+w*cos(t2)/2;
    b.x=Rnd(p[1].x+w*sin(t2)/2);
    b.y=Rnd(p[1].y-w*cos(t2)/2);
    x2=p[2].x-w*sin(t2)/2;
    y2=p[2].y+w*cos(t2)/2;
}  
if (x1==x2) ml=(double) 900000000;
else
  
    ml=(y2-y1)/(x2-x1);
if (a2==a1) m2= (double) 900000000;
else
    m2=(b2-b1)/(a2-a1);
if (m2!=ml)  
{  
    xf=(y1-ml*x1+m2*a1-b1)/(m2-ml);
    yf=m2*(xf-a1)+b1;
    a.x=Rnd(xf);
    a.y=Rnd(yf);
}  
else  
{  
    a.x=Rnd(x1);
    a.y=Rnd(y1);
}  
}  
if (j==0)  
{  
    fp[0].x=a.x;
```c
void lines(int num, XPoint* p, int w, int col)
{
    int i;
    XPoint pts[4];
    XPoint pts2[4];
    XPoint tmp;

    if (num<2) return;
    if (num==2) {
        line(p[0].x, p[0].y, p[1].x, p[1].y, w, col);
        return;
    }

    if (w==1)
        for (i=0; i<num-1; i++)
            gl_line(p[i].x, p[i].y, p[i+1].x, p[i+1].y, col);

    if ( (p[0].x==p[num-1].x) && (p[0].y==p[num-1].y) )
    {
        pts[0].x=p[num-2].x;
        pts[0].y=p[num-2].y;
        pts[1].x=p[0].x;
        pts[1].y=p[0].y;
        pts[2].x=p[1].x;
        pts[2].y=p[1].y;
        pts[3].x=p[2].x;
        pts[3].y=p[2].y;

        pts2[0].x=p[num-3].x;
        pts2[0].y=p[num-3].y;
        pts2[1].x=p[num-2].x;
        pts2[1].y=p[num-2].y;
        pts2[2].x=p[num-1].x;
        pts2[2].y=p[num-1].y;
        pts2[3].x=p[1].x;
        pts2[3].y=p[1].y;

    }
    else
    {
```
\begin{verbatim}
    pts[0].x=p[0].x;
    pts[0].y=p[0].y;
    pts[1].x=p[0].x;
    pts[1].y=p[0].y;
    pts[2].x=p[1].x;
    pts[2].y=p[1].y;
    pts[3].x=p[2].x;
    pts[3].y=p[2].y;

    pts2[0].x=p[num-3].x;
    pts2[0].y=p[num-3].y;
    pts2[1].x=p[num-2].x;
    pts2[1].y=p[num-2].y;
    pts2[2].x=p[num-1].x;
    pts2[2].y=p[num-1].y;
    pts2[3].x=p[num-1].x;
    pts2[3].y=p[num-1].y;

    miter(pts, w);
    polygon(4, pts, col0);

    #if 1
    tmp.x=pts[2].x;
    tmp.y=pts[2].y;
    pts[2].x=pts[3].x;
    pts[2].y=pts[3].y;
    pts[3].x=tmp.x;
    pts[3].y=tmp.y;
    polygon(4, pts, col0);
    #endif

    for (i=1;i<num-2;i++)
    {
        pts[0].x=p[i-1].x;
        pts[0].y=p[i-1].y;
        pts[1].x=p[i].x;
        pts[1].y=p[i].y;
        pts[2].x=p[i+1].x;
        pts[2].y=p[i+1].y;
        pts[3].x=p[i+2].x;
        pts[3].y=p[i+2].y;
        miter(pts, w);
        polygon(4, pts, col0);
    
    #if 1
    tmp.x=pts[2].x;
    tmp.y=pts[2].y;
    pts[2].x=pts[3].x;
    pts[2].y=pts[3].y;
    pts[3].x=tmp.x;
    pts[3].y=tmp.y;
    polygon(4, pts, col0);
    #endif
    }
    miter(pts2, w);
    polygon(4, pts2, col0);

    #if 1
\end{verbatim}
tmp.x=pts2[2].x;
tmp.y=pts2[2].y;
pts2[2].x=pts2[3].x;
pts2[2].y=pts2[3].y;
pts2[3].x=tmp.x;
pts2[3].y=tmp.y;
polygon(4, pts2, col, 0);
#endif
Appendix G  event.c

#include "my.h"
#include <unistd.h>
#include <sys/time.h>
#include <sys/types.h>
#include <linux/keyboard.h>

typedef struct {
    XEvent e;
    struct eventnode* next;
} eventnode;

eventnode * events=NULL;
eventnode * lastevent=NULL;
int numevents=0;

Window grabwin=NULL;
int grabbut=0;
int grabbor=0;

int m_x=0;
int m_y=0;

int m_but=0;
int m_cur_but=0;

int keys[NR_KEYS];
int state;
int caps_rel=0;

Window m_win;

char * mousebox;

int mcol;

int servertime;
int basetime;

int m_in_win(int x,int y,Window w);

Window cur_win(int x, int y)
{
    Window cur;
    Window check;

    cur=RootWindow(NULL,0);
    check=stacknext(cur);

    while(check != NULL)
    {
        if ( m_in_win(x,y,check) )
            cur=check;

        check=stacknext(check);
    }

    return(cur);
}
void send_enter(Window win, int in);
void send_button(Window win, int but, int press);
void send_expose(Window win);
void send_config(Window win);
void send_motion(Window win);
void send_key(Window win, int scancode, int press);

void update_state()
{
    int i = 0;
    i |= m_cur_but;
    if (keys[SCANCODE_LEFTSHIFT] || keys[SCANCODE_RIGHTSHIFT])
        i |= SHIFT;
    if (keys[SCANCODE_RIGHTCONTROL] || keys[SCANCODE_LEFTCONTROL])
        i |= CTRL;
    if (keys[SCANCODE_RIGHTALT] || keys[SCANCODE_LEFTALT])
        i |= ALT;
    if (keys[SCANCODE_CAPSLOCK])
        i |= CAPS;
    state = i;
}

void my_mouse_h(int button, int dx, int dy, int dz, int drx, int dry, int drz)
{
    Window cur;
    if (dx || dy)
    {
        gl_putbox(m_x, m_y, 10, 10, mousebox);
        m_x += dx;
        m_y += dy;
        if (m_x >= WIDTH) m_x = WIDTH - 1;
        if (m_x < 0) m_x = 0;
        if (m_y >= HEIGHT) m_y = HEIGHT - 1;
        if (m_y < 0) m_y = 0;
        gl_getbox(m_x, m_y, 10, 10, mousebox);
        draw_cursor();
    }

    cur = cur_win(m_x, m_y);
    m_cur_but = button;
    update_state();
    if (cur != m_win)
    {
        send_enter(cur, 1);
        send_enter(m_win, 0);
        m_win = cur;
    }
    if (dx || dy)
    {
        if (m_in_win(m_x, m_y, cur) == 1)
send_motion(cur);
)

if (button ^ m_but)
{
    if (button & MBUT1) send_button(cur, MBUT1, 1);
    else send_button(cur, MBUT1, 0);
    if (button & MBUT2) send_button(cur, MBUT2, 1);
    else send_button(cur, MBUT2, 0);
    if (button & MBUT3) send_button(cur, MBUT3, 1);
    else send_button(cur, MBUT3, 0);

    m_but=button;
}

void my_key_h (int scancode, int press)
{
    keys[scancode]=press;
    if (scancode==SCANCODE_CAPSLOCK)
    {
        if (press==0)
        {
            if (caps_rel==0)
            {
                keys[scancode]=1;
                caps_rel=1;
            }
        }
    }

    if (m_in_win(mx, my, m_win)==2) return;
    update_state();
    send_key(m_win, scancode, press);
}

void event_init()
{
    int i;
    for (i=0;i<NR_KEYS;i++)
        keys[i]=0;
    m_win=RootWindow(NULL,0);
    mousebox=malloc(10*10*BYTESPERPIXEL);
    gl_getbox(0,0,10,10,mousebox);
    mcol=gl_rgbcolor(255,255,255);
    servertime=0;
    basetime=0;  /************ time stuff */
#if 1
    if (keyboard_init()) {
        printf("Could not initialize keyboard.\n");
        exit(1);
    }

#endif


```c
void event_exit()
{
    keyboard_close();
    mouse_close();
}

void draw_cursor()
{

    gl_hline(m_x+2, m_y+1, m_x + 9, mcol);
    gl_line(m_x+2, m_y+1, m_x+9, mcol);
    gl_hline(m_x+1, m_y+2, m_x+9, mcol);
    gl_line(m_x+1, m_y+2, m_x+9, mcol);
    gl_line(m_x, m_y+2, m_x+8, mcol);
    gl_hline(m_x, m_y, m_x+9, 0);
    gl_line(m_x, m_y, m_x+9, 0);
    gl_line(m_x, m_y, m_x, m_y+9);
    
    int p_in_rect(int x, int y, XRectangle *r);
    int m_in_win(int x, int y, Window w)
    {
        XRectangle r;
        XRectangle r1, r2;
        Window pwin;
        winnode * p;
        winnode * rootnode;

        rootnode = (winnode *) RootWindow(NULL, 0)->node;
        r.x = w->x;
        r.y = w->y;
        r.width = w->w;
        r.height = w->h;

        p = (winnode *) w->node;
        r1.x = 0;
        r1.y = 0;
        r1.width = WIDTH;
        r1.height = HEIGHT;

        while ( p != rootnode )
        {
            pwin = (Window) p->win;

            if ( pwin->mapped == 0 )
            {
                return(0);
            }

            r2.x = pwin->x;
            r2.y = pwin->y;
            r2.width = pwin->w;
            r2.height = pwin->h;

            p = (winnode *) p->parent;
        }
    }
}

#endif

keyboard_seteventhandler(my_key_h);
mouse_seteventhandler(my_mouse_h);
```
rect_intersect(&rl,&r2,&rl);

if ( p_in_rect(x,y,&rl) ) return(1);
rl.x = w->x-w->bw;
rl.y = w->y-w->bw;
rl.width = w->w + 2*w->bw;
rl.height = w->h + 2*w->bw;
p= (winnode *) w->node;
if (p != rootnode)
p= (winnode *) p->parent;
while ( p != rootnode )
{
pwin = (Window) p->win;
r2.x=pwin->x;
r2.y=pwin->y;
r2.width=pwin->w;
r2.height=pwin->h;
p= (winnode *) p->parent;
rect_intersect(&rl,&r2,&rl);
}
if ( p_in_rect(x,y,&rl) ) return(2);
return(0);

void getevent() /* blocking */
{
    vga_waitevent(VGA_MOUSEEVENT | VGA_KEYEVENT, NULL, NULL, NULL, NULL);
}

void copyevent(XEvent *eret, int remove)
{
eventnode *e;
while (numevents==0)
    getevent();
e=events;
if (remove)
{
    events= (eventnode *) e->next;
    numevents--;
    if (numevents==0) lastevent=NULL;
}

eret->type = e->e.type;
switch(e->e.type)
{
case Expose:
{
eret->xexpose.window=e->e.xexpose.window;
eret->xexpose.x=e->e.xexpose.x;
eret->xexpose.y=e->e.xexpose.y;
eret->xexpose.width=e->e.xexpose.width;
eret->xexpose.height=e->e.xexpose.height;
}
break;
}
case ButtonPress:
case ButtonRelease:
{
eret->xbutton.window=e->e.xbutton.window;
eret->xbutton.x=e->e.xbutton.x;
eret->xbutton.y=e->e.xbutton.y;
eret->xbutton.time=e->e.xbutton.time;
eret->xbutton.state=e->e.xbutton.state;
eret->xbutton.button=e->e.xbutton.button;
break;
}
case KeyPress:
case KeyRelease:
{
eret->xkey.window=e->e.xkey.window;
eret->xkey.x=e->e.xkey.x;
eret->xkey.y=e->e.xkey.y;
eret->xkey.time=e->e.xkey.time;
eret->xkey.state=e->e.xkey.state;
eret->xkey.keycode=e->e.xkey.keycode;
break;
}
case EnterNotify:
case LeaveNotify:
{
eret->xcrossing.window=e->e.xcrossing.window;
eret->xcrossing.x=e->e.xcrossing.x;
eret->xcrossing.y=e->e.xcrossing.y;
eret->xcrossing.time=e->e.xcrossing.time;
eret->xcrossing.state=e->e.xcrossing.state;
break;
}
case MotionNotify:
{
eret->xmotion.window=e->e.xmotion.window;
eret->xmotion.time=e->e.xmotion.time;
break;
}
case ConfigureNotify:
{
eret->xconfigure.window=e->e.xconfigure.window;
eret->xconfigure.width=e->e.xconfigure.width;
eret->xconfigure.height=e->e.xconfigure.height;
break;
}
}
if (remove) free(e);

void XNextEvent(Display *display, XEvent *eret)
{
copyevent(eret, 1);
}

void XPeekEvent(Display *display, XEvent *eret)
{
copyevent(eret, 0);
}

int XEventsQueued(Display *display, int mode)
{
return (numevents);
Bool XQueryPointer(Display *display, Window w,
    Window * root_return, Window * child_return,
    int *root_x_return, int *root_y_return,
    int *win_x_return, int *win_y_return,
    unsigned int *mask_return)
{
    *win_x_return = m_x - w->x;
    *win_y_return = m_y - w->y;
    *mask_return = state;
}

void XFlush(Display *display)
{
    return;
}

void XSync(Display *display, Bool discard)
{
    return;
}

void putevent(eventnode *e)
{
    numevents++;
    if (numevents==1)
    {
        events=(lastevent=e);
        return;
    }
    lastevent->next = (struct eventnode*) e;
    lastevent=e;
}

int mytime()
{
    static int i=0;
    return ( i++);
}

void send_enter(Window win, int in)
{
    eventnode *en;
    if (win==RootWindow(NULL, 0)) return;
    en=malloc(sizeof(eventnode));
    en->next=NULL;
    if (in)
        en->e.type=EnterNotify;
    else en->e.type = LeaveNotify;
    en->e.xcrossing.window=win;
    en->e.xcrossing.state=state;
    en->e.xcrossing.x=m_x-win->x;
    en->e.xcrossing.y=m_y-win->y;
    en->e.xcrossing.time=mytime();
    putevent(en);
void really_send_button(Window win, int but, int press, int x, int y)
{
    eventnode *en;
    if (win==RootWindow(NULL,0)) return;
    en=malloc(sizeof(eventnode));
    en->next=NULL;
    if (press)
        en->e.type=ButtonPress;
    else en->e.type = ButtonRelease;
    en->e.xbutton.window=win;
    en->e.xbutton.state=state;
    en->e.xbutton.x=x;
    en->e.xbutton.y=y;
    en->e.xbutton.time=mytime();
    en->e.xbutton.button=but;
    putevent(en);
}

void send_button(Window win, int but, int press)
{
    eventnode *en;
    if (press)
    {
        if (grabwin==NULL)
        {
            if (m_in_win(m_x, m_y,win)==2) /* press on border */
            {
                XRaiseWindow(NULL, win);
                grabwin=win;
                grabbut=but;
                grabbor=1;
                return;
            }
            else /* press in win */
            {
                grabwin=win;
                grabbut=but;
                really_send_button(win, but, press,m_x-win->x, m_y-win->y);
                return;
            }
        }
        else /* button grab on */
        {
            if (grabwin==win)
            {
                if (grabbor)
                {
                    if (m_in_win(m_x, m_y,win)==2)
                        XRaiseWindow(NULL, win);
                    grabbut |= but;
                    return;
                }
                else
                {
                    really_send_button(win, but, press,m_x-win->x, m_y-win->y);
                    if (grabbut&MBUT1)
                        really_send_button(win,MBUT1,press, m_x-win->x,m_y-win->y);
                    if (grabbut&MBUT2)
                        really_send_button(win,MBUT2,press, m_x-win->x,m_y-win->y);
                }
            }
        }
    }
}
m_xWin->x, m_yWin->y);
if (grabbut&MBUT3)
  really_send_button(win, MBUT3, press,
  m_xWin->x, m_yWin->y);
grabbor=0;
grabbut |= but;
return;
}
else
{
  if (m_in_win(m_x, m_y, win)==1)
  {
    really_send_button(win, but, press, m_xWin->x, m_yWin->y);
    grabbut |= but;
    return;
  }
  else
  {
    XRaiseWindow(NULL, win);
    if (grabbut&MBUT1)
      really_send_button(win, MBUT1, 0, 0, 0);
    if (grabbut&MBUT2)
      really_send_button(win, MBUT2, 0, 0, 0);
    if (grabbut&MBUT3)
      really_send_button(win, MBUT3, 0, 0, 0);
    grabbut |= but;
    grabbor=1;
    return;
  }
}
else /* new press in non-grabbing window */
{
  if (grabbor==0)
  {
    if (grabbut&MBUT1)
      really_send_button(grabwin, MBUT1, 0, 0, 0);
    if (grabbut&MBUT2)
      really_send_button(grabwin, MBUT2, 0, 0, 0);
    if (grabbut&MBUT3)
      really_send_button(grabwin, MBUT3, 0, 0, 0);
  }
  if (m_in_win(m_x, m_y, win)==2)
  {XRaiseWindow(NULL, win);
    grabbut |= but;
    grabbor=1;
    return;}
else
  {
    if (grabbut&MBUT1)
      really_send_button(win, MBUT1, press, m_xWin->x, m_yWin->y);
    if (grabbut&MBUT2)
      really_send_button(win, MBUT2, press, m_xWin->x, m_yWin->y);
    if (grabbut&MBUT3)
      really_send_button(win, MBUT3, press, m_xWin->x, m_yWin->y);
    really_send_button(win, but, press, m_xWin->x, m_yWin->y);
    grabbor=0;
    grabbut |= but;
    return;
  }
}
else /* button release */
{
  grabbut ^= but;
if (grabbor==0)
  if ( win==RootWindow(NULL,0) || m_win(m_x, m_y, win)==2 )
    really_send_button(grabwin, but, press, 0,0);
  else really_send_button(win, but, press, m_x-win->x, m_y-win->y );
if (grabbut==0)
  { grabwin=NULL;
    grabbor=0;
  }
}

void send_expose(Window win)
{
  eventnode *en;
  en=malloc(sizeof(eventnode));
  en->next=NULL;
  en->e.type = Expose;
  en->e.xexpose.window=win;
  en->e.xexpose.width=win->w;
  en->e.xexpose.x=0;
  en->e.xexpose.y=0;
  en->e.xexpose.height=win->h;
  putevent(en);
}

void send_config(Window win)
{
  eventnode *en;
  if (win==RootWindow(NULL, 0)) return;
  en=malloc(sizeof(eventnode));
  en->next=NULL;
  en->e.type = ConfigureNotify;
  en->e.xconfigure.window=win;
  en->e.xconfigure.width=win->w;
  en->e.xconfigure.height=win->h;
  putevent(en);
}

void send_motion(Window win)
{
  eventnode *en;
  if (win==RootWindow(NULL, 0)) return;
  en=malloc(sizeof(eventnode));
  en->next=NULL;
  en->e.type = MotionNotify;
  en->e.xmotion.window=win;
  en->e.xmotion.time=mytime();
  putevent(en);
void send_key(Window win, int scancode, int press)
{
    eventnode *en;
    if (win==RootWindow(NULL, 0)) return;
    en=malloc(sizeof(eventnode));
    en->next=NULL;
    if (press)
        en->e.type=KeyPress;
    else en->e.type = KeyRelease;
    en->e.xkey.window=win;
    en->e.xkey.state=state;
    en->e.xkey.x=m_x-win->x;
    en->e.xkey.y=m_y-win->y;
    en->e.xkey.time=mytime();
    en->e.xkey.keycode=scancode;
    putevent(en);
}

int XLookupString(XKeyEvent *ev,char * buf,
    int bytes_buf,KeySym * ks,
    void *dummy)
{
    int code;
    int state;
    int s=0;
    char string[10];
    code = ev->keycode;
    state = ev->state;
    if ( (state & SHIFT) || (state & CAPS) )
        s=1;
    buf[0]=-l;
    switch (code)
    {
    case SCANCODE_A:
        {
            if (s) (*ks=XK_A; sprintf(buf, "%c","A");)
                else (*ks=XK_a; sprintf(buf, "%c","a");)
                break;}
    case SCANCODE_B:
        {
            if (s) (*ks=XK_B; sprintf(buf, "%c","B");)
                else (*ks=XK_b; sprintf(buf, "%c","b");)
                break;}
    case SCANCODE_C:
        {
            if (s) (*ks=XK_C; sprintf(buf, "%c","C");)
                else (*ks=XK_c; sprintf(buf, "%c","c");)
                break;}
    case SCANCODE_D:
        {
            if (s) (*ks=XK_D; sprintf(buf, "%c","D");)
                else (*ks=XK_d; sprintf(buf, "%c","d");)
                break;}
    }
case SCANCODE_E:
    if (s) {*ks=XK_E; sprintf(buf, "%c", 'E');}
    else {*ks=XK_e; sprintf(buf, "%c", 'e');}
    break;

case SCANCODE_F:
    if (s) {*ks=XK_F; sprintf(buf, "%c", 'F');}
    else {*ks=XK_f; sprintf(buf, "%c", 'f');}
    break;

case SCANCODE_G:
    if (s) {*ks=XK_G; sprintf(buf, "%c", 'G');}
    else {*ks=XK_g; sprintf(buf, "%c", 'g');}
    break;

case SCANCODE_H:
    if (s) {*ks=XK_H; sprintf(buf, "%c", 'H');}
    else {*ks=XK_h; sprintf(buf, "%c", 'h');}
    break;

case SCANCODE_I:
    if (s) {*ks=XK_I; sprintf(buf, "%c", 'I');}
    else {*ks=XK_i; sprintf(buf, "%c", 'i');}
    break;

case SCANCODE_J:
    if (s) {*ks=XK_J; sprintf(buf, "%c", 'J');}
    else {*ks=XK_j; sprintf(buf, "%c", 'j');}
    break;

case SCANCODE_K:
    if (s) {*ks=XK_K; sprintf(buf, "%c", 'K');}
    else {*ks=XK_k; sprintf(buf, "%c", 'k');}
    break;

case SCANCODE_L:
    if (s) {*ks=XK_L; sprintf(buf, "%c", 'L');}
    else {*ks=XK_l; sprintf(buf, "%c", 'l');}
    break;

case SCANCODE_M:
    if (s) {*ks=XK_M; sprintf(buf, "%c", 'M');}
    else {*ks=XK_m; sprintf(buf, "%c", 'm');}
    break;

case SCANCODE_N:
    if (s) {*ks=XK_N; sprintf(buf, "%c", 'N');}
    else {*ks=XK_n; sprintf(buf, "%c", 'n');}
    break;

case SCANCODE_O:
    if (s) {*ks=XK_O; sprintf(buf, "%c", 'O');}
    else {*ks=XK_o; sprintf(buf, "%c", 'o');}
    break;

case SCANCODE_P:
    if (s) {*ks=XK_P; sprintf(buf, "%c", 'P');}
    else {*ks=XK_p; sprintf(buf, "%c", 'p');}
    break;

case SCANCODE_Q:
    if (s) {*ks=XK_Q; sprintf(buf, "%c", 'Q');}
    else {*ks=XK_q; sprintf(buf, "%c", 'q');}
    break;

case SCANCODE_R:
    if (s) {*ks=XK_R; sprintf(buf, "%c", 'R');}
    else {*ks=XK_r; sprintf(buf, "%c", 'r');}

break;

break;
case SCANCODE_S:
    {
        if (s) (*ks=XK_S; sprintf(buf, "%c", 'S');)
        else (*ks=XK_s; sprintf(buf, "%c", 's');)
        break;
    }
case SCANCODE_T:
    {
        if (s) (*ks=XK_T; sprintf(buf, "%c", 'T');)
        else (*ks=XK_t; sprintf(buf, "%c", 't');)
        break;
    }
case SCANCODE_U:
    {
        if (s) (*ks=XK_U; sprintf(buf, "%c", 'U');)
        else (*ks=XK_u; sprintf(buf, "%c", 'u');)
        break;
    }
case SCANCODE_V:
    {
        if (s) (*ks=XK_V; sprintf(buf, "%c", 'V');)
        else (*ks=XK_v; sprintf(buf, "%c", 'v');)
        break;
    }
case SCANCODE_W:
    {
        if (s) (*ks=XK_W; sprintf(buf, "%c", 'W');)
        else (*ks=XK_w; sprintf(buf, "%c", 'w');)
        break;
    }
case SCANCODEX:
    {
        if (s) (*ks=XK_X; sprintf(buf, "%c", 'X');)
        else (*ks=XK_x; sprintf(buf, "%c", 'x');)
        break;
    }
case SCANCODEY:
    {
        if (s) (*ks=XK_Y; sprintf(buf, "%c", 'Y');)
        else (*ks=XK_y; sprintf(buf, "%c", 'y');)
        break;
    }
case SCANCODEZ:
    {
        if (s) (*ks=XK_Z; sprintf(buf, "%c", 'Z');)
        else (*ks=XK_z; sprintf(buf, "%c", 'z');)
        break;
    }
case SCANCODE_1:
    {
        if (s) (*ks=XK_exclam; sprintf(buf, "%c", '!');)
        else (*ks=XK_1; sprintf(buf, "%c", '1');)
        break;
    }
case SCANCODE_2:
    {
        if (s) (*ks=XK_at; sprintf(buf, "%c", '@');)
        else (*ks=XK_2; sprintf(buf, "%c", '2');)
        break;
    }
case SCANCODE_3:
    {
        if (s) (*ks=XK_numbersign; sprintf(buf, "%c", '#');)
        else (*ks=XK_3; sprintf(buf, "%c", '3');)
        break;
    }
case SCANCODE_4:
    {
        if (s) (*ks=XK_dollar; sprintf(buf, "%c", '$');)
        else (*ks=XK_4; sprintf(buf, "%c", '4');)
        break;
    }
case SCANCODE_5:
    {
        if (s) (*ks=XK_percent; sprintf(buf, "%c", '%');)
        else (*ks=XK_5; sprintf(buf, "%c", '5');)
        break;
    }
case SCANCODE_6:
    {
        if (s) (*ks=XK_asterisk; sprintf(buf, "%c", '*');)
        else (*ks=XK_6; sprintf(buf, "%c", '6');)
        break;
    }
else (*ks=XK_6; sprintf(buf, "\%c", '6');)
break;)
case SCANCODE_7:
    
    if (s) (*ks=XK_7; sprintf(buf, "\%c", '7');)
else (*ks=XK_7; sprintf(buf, "\%c", '7');)
break;)
case SCANCODE_8:
    
    if (s) (*ks=XK_8; sprintf(buf, "\%c", '8');)
else (*ks=XK_8; sprintf(buf, "\%c", '8');)
break;)
case SCANCODE_9:
    
    if (s) (*ks=XK_9; sprintf(buf, "\%c", '9');)
else (*ks=XK_9; sprintf(buf, "\%c", '9');)
break;)
case SCANCODE_0:
    
    if (s) (*ks=XK_0; sprintf(buf, "\%c", '0');)
else (*ks=XK_0; sprintf(buf, "\%c", '0');)
break;)
case SCANCODE_GRAVE:
    
    if (s) (*ks=XK_grave; sprintf(buf, "\%c", '\');)
else (*ks=XK_grave; sprintf(buf, "\%c", '\');)
break;)
case SCANCODE_MINUS:
    
    if (s) (*ks=XK_underscore; sprintf(buf, "\%c", '_');)
else (*ks=XK_underscore; sprintf(buf, "\%c", '_');)
break;)
case SCANCODE_EQUAL:
    
    if (s) (*ks=XK_equal; sprintf(buf, "\%c", '=');)
else (*ks=XK_equal; sprintf(buf, "\%c", '=');)
break;)
case SCANCODE_BACKSLASH:
    
    if (s) (*ks=XK_backslash; sprintf(buf, "\%c", '\');)
else (*ks=XK_backslash; sprintf(buf, "\%c", '\');)
break;)
case SCANCODE_SEMICOLON:
    
    if (s) (*ks=XK_semicolon; sprintf(buf, "\%c", ';');)
else (*ks=XK_semicolon; sprintf(buf, "\%c", ';');)
break;)
case SCANCODE_APOSTROPHE:
    
    if (s) (*ks=XK_apostrophe; sprintf(buf, "\%c", '');)
else (*ks=XK_apostrophe; sprintf(buf, "\%c", '');)
break;)
case SCANCODE_COMMA:
    
    if (s) (*ks=XK_comma; sprintf(buf, "\%c", ',');)
else (*ks=XK_comma; sprintf(buf, "\%c", ',');)
break;)
case SCANCODE_PERIOD:
    
    if (s) (*ks=XK_period; sprintf(buf, "\%c", '.');)
else (*ks=XK_period; sprintf(buf, "\%c", '.');)
if (s) {*ks=XK_greater; sprintf(buf, "c", '>');}
else {*ks=XK_period; sprintf(buf, "c", '.');}
break;

case SCANCODE_SLASH:
    {if (s) {*ks=XK_question; sprintf(buf, "c", '?');}
     else {*ks=XK_slash; sprintf(buf, "c", '/');}
     break;
}
case SCANCODE_SPACE:
    {*ks=XK_space; sprintf(buf, "%c", ' ');
break;
}

case SCANCODE_ESCAPE:
    {*ks=XK_Escape; sprintf(buf, "%c", XK_Escape);
break;
}
case SCANCODE_BACKSPACE:
    {*ks=XK_BackSpace; sprintf(buf, "%c", XK_BackSpace);
break;
}
case SCANCODE_TAB:
    {*ks=XK_Tab; sprintf(buf, "%c", XK_Tab);
break;
}
case SCANCODE_ENTER:
    {*ks=XK_Return; sprintf(buf, "%c", XK_Return);
break;
}
case SCANCODE_SCROLLLOCK:
    {*ks=XK_Scroll_Lock; sprintf(buf, "%c", XK_Scroll_Lock);
break;
}
case SCANCODE_REMOVE:
    {*ks=XK_Delete; sprintf(buf, "%c", XK_Delete);
break;
}
case SCANCODE_PRINTSCREEN:
    {*ks=XK_Sys_Req; sprintf(buf, "%c", XK_Sys_Req);
break;
}
case SCANCODE_BREAK:
    {*ks=XK_Pause; sprintf(buf, "%c", XK_Pause);
break;
}
case SCANCODE_LEFTCONTROL:
    {*ks=XK_ControlL;
break;
}
case SCANCODE_RIGHTCONTROL:
    {*ks=XK_Control_R;
break;
}
case SCANCODE_LEFTSHIFT:
    {*ks=XK_Shift_L;
break;
}
case SCANCODE_RIGHTSHIFT:
    {*ks=XK_Shift_R;
break;
}
case SCANCODE_LEFTALT:
    {*ks=XKAltL;
break;
}
case SCANCODE_RIGHTALT:
    {*ks=XKAlt_R;
break;
}
case SCANCODE_CAPSLOCK:
    *ks=XK_Caps_Lock;
    break;

case SCANCODE_F1:
    *ks=XK_F1;
    break;

case SCANCODE_F2:
    *ks=XK_F2;
    break;

case SCANCODE_F3:
    *ks=XK_F3;
    break;

case SCANCODE_F4:
    *ks=XK_F4;
    break;

case SCANCODE_F5:
    *ks=XK_F5;
    break;

case SCANCODE_F6:
    *ks=XK_F6;
    break;

case SCANCODE_F7:
    *ks=XK_F7;
    break;

case SCANCODE_F8:
    *ks=XK_F8;
    break;

case SCANCODE_F9:
    *ks=XK_F9;
    break;

case SCANCODE_F10:
    *ks=XK_F10;
    break;

case SCANCODE_F11:
    *ks=XK_F11;
    break;

case SCANCODE_F12:
    *ks=XK_F12;
    break;

case SCANCODE_KEYPADMULTIPLY:
    *ks=XK_KP_Multiply;
    sprintf(buf, "%c", '*');
    break;

case SCANCODE_KEYPAD0:
    *ks=XK_KP_0;
    sprintf(buf, "%c", '0');
    break;

case SCANCODE_KEYPAD1:
    *ks=XK_KP_1;
    sprintf(buf, "%c", '1');
    break;

case SCANCODE_KEYPAD2:
*ks=XK_KP_2;
sprintf(buf, "%c", '2');
break;}
case SCANCODE_KEYPAD3:
    { *ks=XK_KP_3;
      sprintf(buf, "%c", '3');
      break;}
case SCANCODE_KEYPAD4:
    { *ks=XK_KP_4;
      sprintf(buf, "%c", '4');
      break;}
case SCANCODE_KEYPAD5:
    { *ks=XK_KP_5;
      sprintf(buf, "%c", '5');
      break;}
case SCANCODE_KEYPAD6:
    { *ks=XK_KP_6;
      sprintf(buf, "%c", '6');
      break;}
case SCANCODE_KEYPAD7:
    { *ks=XK_KP_7;
      sprintf(buf, "%c", '7');
      break;}
case SCANCODE_KEYPAD8:
    { *ks=XK_KP_8;
      sprintf(buf, "%c", '8');
      break;}
case SCANCODE_KEYPAD9:
    { *ks=XK_KP_9;
      sprintf(buf, "%c", '9');
      break;}
case SCANCODE_KEYPADMINUS:
    { *ks=XK_KP_Subtract;
      sprintf(buf, "%c", '-');
      break;}
case SCANCODE_KEYPADPLUS:
    { *ks=XK_KP_Add;
      sprintf(buf, "%c", '+');
      break;}
case SCANCODE_KEYPADDIVIDE:
    { *ks=XK_KP_Divide;
      sprintf(buf, "%c", '/');
      break;}
case SCANCODE_KEYPADPERIOD:
    { *ks=XK_KP_Decimal;
      sprintf(buf, "%c", '.');
      break;}
case SCANCODE_KEYPADENTER:
    { *ks=XK_KP_Enter;
      break;}
case SCANCODE_NUMLOCK:
    { *ks=XK_Num_Lock;
      break;}
case SCANCODE_HOME:
    { *ks=XK_Home;
break;

break;

case SCANCODE_INSERT:
{
    *ks=XK_Insert;
    break;
}

case SCANCODE_CURSORBLOCKUP:
{
    *ks=XK_Up;
    break;
}

case SCANCODE_PAGEDOWN:
{
    *ks=XK_Page_Down;
    break;}

case SCANCODE_CURSORBLOCKDOWN:
{
    *ks=XK_Down;
    break;}

case SCANCODE_CURSORBLOCKLEFT:
{
    *ks=XK_Left;
    break;}

default: *ks=-1;
}

if ( (state & CTRL) || (state & ALT) )
    return(0);

if ( buf[0]==-1 ) return(0);

return(1);
Appendix H  font.c

#include "my.h"

TT_Engine engine;

typedef struct {
    char name[40];
    int psize;
    struct fontnode * next;
    XFontStruct *fs;
} fontnode;

fontnode *fonts=NULL;
fontnode *lastfont=NULL;

void fontinit()
{
    TT_Error error=1;
    error = TT_Init_FreeType( &engine );
    if (error)
        printf("crap");
    return;
}

void fontexit()
{
    TT_Done_FreeType(engine);
}

int XTextWidth(XFontStruct *fs,char * string,int count)
{
    int i,idx;
    int w=0;
    for (i=0;i<count;i++)
    {
        idx = TT_Char_Index( fs->char_map, string[i] );
        w += fs->per_char[idx].width;
    }
    return(w);
}

XFontStruct *XLoadQueryFont(int *ps,char * name)
{
    XFontStruct *fs;
    TT_Error error;
    TT_Face face;
    TT_Instance instance;
    TT_CharMap char_map;
    TT_Face_Properties properties;
    TT_Instance_Metrics imetrics;
    int i, n;
    short platform, encoding;
    fontnode *fn;
    fontnode *f;
    f=fonts;
printf("a\n");

while(f != NULL)
{
    if (*ps == f->psize)
        if (strncmp(f->name, name, 39) == 0)
            return(f->fs);
    f= (fontnode *) f->next;
}
printf("b\n");

fn=malloc(sizeof(fontnode));
printf("c\n");
strncpy(fn->name, name, 39);
fs=malloc(sizeof(XFontStruct));
printf("d\n");
fn->fs=fs;
fs->fn=fn;
printf("e\n");
fn->psize=*ps;
fn->next = NULL;
printf("f\n");

fflush(stdout);
for(i=0;i<1000;i++);
    fs->rendered=0;
printf("g\n");
if (fonts==NULL){
    printf("h\n");
    fonts=(lastfont=fn);
}
else {
    lastfont->next= (struct fontnode *)fn;
    lastfont=fn;
}
printf("i\n");
error=TT_Open_Face( engine, name, &fs->face );
if ( error == TT_Err_Could_Not_Open_File )
    printf( "Could not find/open.\n");
else if (error)
    printf( "Error while opening file, error code = .\n");
TT_Get_Face_Properties( fs->face, &fs->properties );
fs->num = fs->properties.num_Glyphs;
error = TT_New_Instance( fs->face, &fs->instance );
if ( error )
    printf( "Could not create instance for .\n" );

error = TT_Set_Instance_Resolutions( fs->instance, 96, 96 );
if ( error )
    printf( "Could not set device resolutions." );

/* TT_Get_Instance_Metrics( fs->instance, &fs->imetrics );*/

n = TT_Get_CharMap_Count( fs->face );
for ( i = 0; i < n; i++ )
{
    TT_Get_CharMap_ID( fs->face, i, &platform, &encoding );
    if ( (platform == 3 && encoding == 1) ||
         (platform == 0 && encoding == 0) )
    {
        TT_Get_CharMap( fs->face, i, &fs->char_map );
        i = n + 1;
    }
}

if ( i == n )
    TT_Get_CharMap( fs->face, 0, &fs->char_map );

fs->fid = fs;

void XDrawString( Display *display, Drawable d, GC gc, int x, int y,
                  char * string, int length )
{
    int w,
    int h,
    int xorg, yorg,
    int x0, y0;
    unsigned char * bitmap;
    GraphicsContext *savedcontext;
    int idx;
    XFontStruct *fs;
    unsigned char * bm;
    int i, j, k;
    int cnt;
    
    fs = gc->font;
    savedcontext = gl_allocatecontext();
    gl_getcontext(savedcontext);
    gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
                          BITSPERPIXEL, d->pmap);
    if(gc->clip_flag){
        x0 = gc->clip_x_origin;
        y0 = gc->clip_y_origin;
        gl_setclippingwindow(x0+gc->clip_mask.x,y0+gc->clip_mask.y,
                             x0+gc->clip_mask.x+gc->clip_mask.width,
                             y0+gc->clip_mask.y+gc->clip_mask.height);
    }

#if 0
    w = XTextWidth( fs, string, length );
    h = fs->ascent + fs->descent;
    xorg = x;
    yorg = y - fs->ascent;
    bitmap = malloc(w*h*BYTESPERPIXEL);
#endif
gl_getbox(xorg,yorg,w,h,bitmap);

gl_setcontextvirtual(w, h, BYTESPERPIXEL,
    BITSPERPIXEL, bitmap);
#endif

/* */
x0=x;
y0=y;

for (cnt=0;cnt<length;cnt++)
{
    idx = TT_Char_Index( fs->char_map, string[cnt] );
    bm = (unsigned char *) fs->per_char[idx].rm.bitmap;
    x0 += fs->per_char[idx].lbearing;
    y0 -= fs->per_char[idx].ascent;
    /* fucked up placement! */
    for (i=0;i<fs->per_char[idx].rm.rows;i++)
        for (j=fs->per_char[idx].rm.cols-1;j>=0;j--)
            for(k=0;k<8;k++)
                if ( bm[fs->per_char[idx].rm.cols*i+j] & 1<<k )
                    gl_setpixel(x0 + 8*(j) +7 - k,
                        y0+i, gc->foreground);
    /* 8*(fs->per_char[idx].rm.cols-j) */
    x0 -= fs->per_char[idx].lbearing;
    y0 += fs->per_char[idx].ascent;
    x0 += fs->per_char[idx].width;
}

/* */
#if 0
    gl_setcontextvirtual(d->w, d->h, BYTESPERPIXEL,
        BITSPERPIXEL, d->pmap);

    gl_putbox(xorg,yorg,w,h,bitmap);
#endif

gl_setcontext(savedcontext);

if(d->winflag)
    refresh(d);
if(gc->clip_flag) gl_disableclipping();
}

void setfontsize(XFontStruct* fs, int ps)
{
    TT_Glyph   glyph;
    TT_Glyph_Metrics metrics;

    int i, n,j,k;
    TT_Error   error;
    int idx;
    unsigned char * bm;
    TT_Raster_Map rm;
    int w,h;

107
int flags;
flags=TTLOAD_DEFAULT;
if (fs->rendered==1) return;

error = TT_New_Glyph( fs->face, &glyph );
if ( error )
    printf( "Could not create glyph container.\n" );
TT_Set_Instance_CharSize( fs->instance, ps*64 );
fs->per_char = malloc (fs->num * sizeof(XCharStruct));
fs->rendered=1;
for(i=0; i<fs->num; i++)
{
    TT_Load_Glyph( fs->instance, glyph, i, flags );
    TT_Get_Glyph_Metrics(glyph, &metrics);
    w = (metrics.bbox.xMax - metrics.bbox.xMin) / 64;
    h = (metrics.bbox.yMax - metrics.bbox.yMin) / 64;
    fs->per_char[i].rm.width = w;
    fs->per_char[i].rm.cols = (w+7) / 8;
    fs->per_char[i].rm.rows = h;
    fs->per_char[i].rm.flow = TT_Flow_Down;
    fs->per_char[i].rm.size = fs->per_char[i].rm.cols * fs->per_char[i].rm.rows;
    fs->per_char[i].rm.bitmap = malloc(fs->per_char[i].rm.size);
    TT_Get_Glyph_Bitmap( glyph, &fs->per_char[i].rm,
                           -metrics.bbox.xMin, -metrics.bbox.yMin);
    fs->per_char[i].width = metrics.advance/64;
    fs->per_char[i].lbearing = metrics.bearingX/64;
    fs->per_char[i].rbearing = metrics.bearingX/64 + w;
    fs->per_char[i].ascent = metrics.bearingY/64;
    fs->per_char[i].descent = h - metrics.bearingY/64;
}

fs->min_bounds.width = fs->per_char[0].width;
fs->min_bounds.lbearing = fs->per_char[0].lbearing;
fs->min_bounds.rbearing = fs->per_char[0].rbearing;
fs->min_bounds.ascent = fs->per_char[0].ascent;
fs->min_bounds.descent = fs->per_char[0].descent;

fs->max_bounds.width = fs->per_char[0].width;
fs->max_bounds.lbearing = fs->per_char[0].lbearing;
fs->max_bounds.rbearing = fs->per_char[0].rbearing;
fs->max_bounds.ascent = fs->per_char[0].ascent;
fs->max_bounds.descent = fs->per_char[0].descent;

for(i=1; i<fs->num; i++)
{
    if (fs->per_char[i].width > fs->max_bounds.width)
        fs->max_bounds.width = fs->per_char[i].width;
    if (fs->per_char[i].lbearing > fs->max_bounds.lbearing)
        fs->max_bounds.lbearing = fs->per_char[i].lbearing;
    if (fs->per_char[i].rbearing > fs->max_bounds.rbearing)
        fs->max_bounds.rbearing = fs->per_char[i].rbearing;
    if (fs->per_char[i].ascent > fs->max_bounds.ascent)
        fs->max_bounds.ascent = fs->per_char[i].ascent;
    if (fs->per_char[i].descent > fs->max_bounds.descent)
        fs->max_bounds.descent = fs->per_char[i].descent;
if (fs->per_char[i].width < fs->min_bounds.width)
  fs->min_bounds.width = fs->per_char[i].width;
if (fs->per_char[i].lbearing < fs->min_bounds.lbearing)
  fs->min_bounds.lbearing = fs->per_char[i].lbearing;
if (fs->per_char[i].rbearing < fs->min_bounds.rbearing)
  fs->min_bounds.rbearing = fs->per_char[i].rbearing;
if (fs->per_char[i].ascent < fs->min_bounds.ascent)
  fs->min_bounds.ascent = fs->per_char[i].ascent;
if (fs->per_char[i].descent < fs->min_bounds.descent)
  fs->min_bounds.descent = fs->per_char[i].descent;
}

fs->ascent=fs->max_bounds.ascent+3;
fs->descent=fs->max_bounds.descent+2;