University of Toronto, Faculty of Applied Science and Engineering
Department of Electrical and Computer Engineering

ECE 1387F - CAD for Digital Circuit Synthesis and Layout **Handout #5**

September 1999 J. Rose

Graphics Package for X11 and PostScript Displays

V. Betz (vaughn@eecg.utoronto.ca)

Where to Find the Source Code

The source code for this graphics package is on the EECG and ECF systems in the directory ~jayar/1387/graphics. The files in this directory are:

graphics.c: The source code for the graphics package.

graphics.h: The header file for the graphics package.

example.c: An example file showing how to use the graphics.

makefile: The makefile for the example program

These files are also available on the web page of the course in the directory:

www.eecg.toronto.edu/~jayar/courses/ece1387/graphics/

The sub-directory “windows” contains a graphics.c and graphics.h that can be compiled using Microsoft Visual C/C++.

General Use of this Graphics Package

Any source files which use graphics must include the line #include “graphics.h” and you must have both the graphics.c and graphics.h files in your source directory. When you compile you should use a command like:

gcc my_source.c graphics.c -o exe_name -lX11 -lm

Before any graphics can be drawn, your program must call init_graphics to set up the X display. Next you should call init_world to set up the coordinate system your program desires (you specify which coordinates correspond to the upper left and lower right corners of the screen). There are then two ways in which you can use this graphics package, interactively or non-interactively. Run the example program (graphics) to see how they look to the user. (To create the example program,

edit makefile to set the library paths, etc. to the appropriate values for your system, then type make).

Interactive Graphics

In this mode, your graphics will be drawn and the user can use on-screen arrow, zoom and window controls to focus on the areas of greatest interest. To use this mode, your program must have a routine which can redraw the entire picture. This routine should not set the world coordinates (no calls to init_world), since the graphics package changes the world coordinates in response to the user clicking on the pan and zoom buttons. This routine always draws the entire picture; the graphics package translates the points specified by you to the screen so that the view is properly zoomed and panned. The graphics are made interactive by calling the event_loop routine. This routine simply responds to user input and monitors the window state, calling your redraw routine to redraw the picture as necessary. When the user clicks on the proceed button, control exits the event_loop procedure and returns to your program. If you return from the event_loop and then wish to call it again later, you may want to first call init_world to set the world coordinates (since the user's panning and zooming may have changed them) and then call your redrawing routine (since the on-screen picture may be different from your new picture, and event_loop will not call your drawing routine until the user does something).

The buttons shown on-screen in this mode are:

Arrow keys: Pan half a window width or height in the specified direction.

Zoom In and Zoom Out: Blow up the view to 5/3 or reduce it to 3/5 of its original size, respectively.

Zoom Fit: Zooms all the way out so all your graphics fit into the display area. (Returns the view to the full coordinate range specified by the last call to init_world.)

Window: After clicking on this button, you can draw a box in the graphics area by clicking on its diagonally opposite corners. The program will zoom in on this box (while preserving the aspect ratio).

PostScript: This will create a PostScript file that corresponds exactly to the display you currently have on screen. The first time you click on this button it creates a file called pic1.ps; the second time it creates pic2.ps and so on.

Proceed: Return to the calling routine (allows the main program to compute some more and/or draw a different picture).

Exit: Abort the program.

Event_loop is called with two parameters. The first parameter is a pointer to a function that will be called whenever the user clicks on some point in the drawing area (button clicks in the menu areas are handled entirely by the graphics package). Whenever the user clicks a button within the drawing area the function you passed to event_loop is called and is passed the coordinates at which the user clicked. This allows you to respond to user mouse clicks, perhaps by changing the color of

the object they clicked on, etc. If you have no need to respond to mouse clicks on your graphics, simply create a dummy function that does nothing and pass this to event_loop.

The second parameter is a pointer to your screen redrawing function.

Message Area

There is small subwindow at the bottom of the main window which displays a single line of message text. This text is changed by calling update_message (char *msg). This message will not be changed in size as the user zooms in and out or pans the graphics, so it is useful for status messages.

Non-Interactive Graphics

In order to use the interactive graphics, your program must be capable of redrawing the entire picture it wants displayed. If this is impossible or inconvenient, you can simply draw the graphics piecemeal and call flushinput to ensure the X11 server draws them. You must call either event_loop or flushinput in order for your graphics to be displayed; otherwise the X server may buffer them forever. The buttons on the right side of the window that normally allow panning and zooming will be greyed out to show that they are not selectable.

To create PostScript output in non-interactive mode, simply call init_postscript, make your drawing calls normally, and then call close_postscript when you are done to close the output file and redirect graphics to the screen.

Compiling on Machines without X Windows

What happens if you incorporate graphics throughout your program and then decide you'd like to run the program on a system that doesn't support X Windows (e.g. Windows NT). Don't despair! Uncomment the line "#define NO_GRAPHICS" at the top of graphics.c and all the X Windows calls will be ripped out by the C preprocessor. You'll be able to compile your program and run it, but of course, the graphics will all be disabled.

Procedure Summary

*void init_graphics (char *window_name):* Opens and initializes the X display. Called once at the beginning of a program, before any drawing occurs. The *window_name* parameter will be displayed across the title bar of the window.

void close_graphics (void): Closes the X Display. Should be called once at the end of the program, after all drawing is finished.

*void init_postscript (char *fname):* Opens a file with the name *fname* (by convention you should specify a name ending in .ps) and directs all graphics output to this PostScript file until a close_postscript call is executed.

void close_postscript (void): Properly terminates and closes the PostScript file created by init_postscript, and sets the screen as the output device for subsequent drawing commands.

*void event_loop (void (*act_on_button) (float x, float y), void (*drawscreen) (void)):* Redraws the window if it has been obscured and modifies the world-coordinates to allow panning and zooming in response to user input. This routine is passed two "callback" functions. The drawscreen function must redraw the entire picture when called. The act_on_button function is called whenever the user clicks on the drawing area. This allows the "callback" function you have defined to look at the x and y coordinates at which the user clicked, and take some appropriate action.

void drawscreen (void): A user-supplied routine that redraws the entire picture. This must exist if the interactive graphics are to be used (and you need a dummy routine to keep the compiler happy even if you don't use the interactive graphics). The first line of drawscreen should be *clearscreen()* to erase the old graphics from the window.

*void update_message (char *msg):* Sets the message to be displayed in the message area to the string pointed to by msg.

void init_world (float xleft, float ytop, float xright, float ybottom): Specifies the world coordinates used for drawing. The four numbers define the coordinates of the edges of the window. You should always call init_world before event_loop (since previous calls of event_loop may have changed the world coordinates in response to the user's panning and zooming) and you should never call init_world in drawscreen (since this would disable the user's panning and zooming). If necessary, the graphics package will increase either the vertical or horizontal span of the world coordinates to make sure the aspect ratio is preserved (i.e. 1 vertical unit displays as the same length as 1 horizontal unit).

void flushinput (void): Forces previous drawing commands to be displayed, and empties the X event queue. Only needed during non-interactive drawing, as event_loop automatically forces the X server to update the display.

void setcolor (enum cindex): Sets the color for subsequent drawing commands. The colors are specified by enumerated constants, so they must always be in upper case. The colors available are: WHITE, BLACK, LIGHTGREY, DARKGREY, RED, YELLOW, GREEN, BLUE, CYAN, DARKGREEN, and MAGENTA. On black and white displays you should use only the BLACK and WHITE colors -- if you specify other colors the X server will map them to either black or white. Default: BLACK.

void setlinestyle (enum linestyle): Sets the linestyle used by subsequent drawline and drawrect calls. The choices available are SOLID or DASHED. Default: SOLID.

void setlinewidth (int linewidth): Sets the width of lines drawn by subsequent drawline and drawrect calls, in pixels. A width of zero requests the thinnest line (1 pixel wide) that the device can draw *and* indicates that lines can be drawn using a fast (and slightly less accurate) algorithm. Default: 0.

void setfontsize (int point_size): Sets the font size used by subsequent drawtext calls, in points. Default: 10.

void drawline (float x1, float y1, float x2, float y2): Draw a line connecting (*x1, y1*) to (*x2, y2*). The on-screen location of these points is determined by the current world coordinates.

void drawrect (float x1, float y1, float x2, float y2): Draws the outline of a rectangle with the diagonally-opposed corners specified by (*x1, y1*) and (*x2, y2*), in world coordinates.

void drawarc (float xcen, float ycen, float rad, float startang, float angextent): Draws a circular arc centered at (*xcen, ycen*) with radius *rad* (all in world coordinates). *Startang* specifies the staring angle of the arc, and *angextent* is the angle swept out from the beginning to the end of the arc. Both angles are specified in degrees. *Startang* is specified relative to the positive x-axis of the screen (a horizontal line from the center of the screen to its right side) and increases in the counterclockwise direction. Likewise a positive *angextent* specifies a counterclockwise arc, while a negative *angextent* describes a clockwise arc.

void fillarc (float xcen, float ycen, float rad, float startang, float angextent): Fills a circular segment (pie shape) centered at *(xcen, ycen)* with radius *rad* (all in world coordinates). *Startang* specifies the starting angle of the circular segment, and *angextent* is the angle swept out from the beginning to the end of the circular segment. Both angles are in degrees. See drawarc for details on the coordinate system assumed by startang and angextent.

void fillrect (float x1, float y1, float x2, float y2): Draws the filled rectangle specified by the diagonally-opposed (*x1, y1*) and (*x2, y2*) corners in world coordinates.

*void fillpoly (s_point *points, int npoints):* Fills in the polygon specified. The s_point type is defined to be *struct {float x; float y;},* so the points array specifies the outline of the polygon, and *npoints* indicates the number of points making up the polygon boundary. These points will be connected with straight lines; the last point will be connected to the first point to close the figure, and it will be filled. The *MAXPTS* defined constant in graphics.h determines the maximum number of points allowed in the points array and is currently set to 100. If you need to draw very complex polygons with over 100 boundary points, increase this number.

*void drawtext (float xcen, float ycen, char *text, float boundx):* Draws the null-terminated text string pointed to by **text* centered at (*xcen, ycen*) in world coordinates. The text will not be drawn if it will be more than *boundx* units (world coordinates) from its left to its right side. This prevents text messages from being drawn when the picture is very zoomed-out if they would obscure too much detail. If you want the text displayed no matter how far the user zooms out, specify a very large *boundx* (e.g. 1.e10).

void clearscreen (void): Clears the graphics window or PostScript page (erases all the current geometry).

The following two functions are used only if you want to create new buttons in the menu area or destroy some of the existing ones. If you're happy with the buttons the graphics package provides by default, you don't need to ever use these functions.

*void create_button (char *prev_button_text, char *button_text, void (*button_func) (void (*drawscreen) (void))):* Creates a new button underneath the button containing prev_button_text. The text on the button is given by button_text, and the function pointer passed to create button will be

called whenever the button is pressed. This callback function takes one argument -- a function pointer which points at the screen redrawing function.

*void destroy_button (char *button_text):* Destroys the button with the specified button_text (i.e. name).

Updates

I add new features to my graphics package as I need them. Periodically I will update the files on my web page to make these enhancements available to everybody. If you want me to email you when new updates are available send an email to vaughn@eecg.utoronto.ca. Any feedback on your experience with this code is also very welcome!