COLOR

Introduction

This document describes software for driving color displays. In order to run COLOR, you need either a Sun (3 or 4) with CG4 color hardware and display, a Dorado (Xerox 1132) with attached color display, or a Dandelion (Xerox 1108) with attached BusMaster and color display.

The color software which is distributed among a number of files can be divided into a machine independent group of files that all users can usefully load and a machine dependent group containing files that work for particular combinations of hardware.

The machine independent color graphics code is stored in the library files LLCOLOR.LCOM and COLOR.LCOM. LOADing COLOR.LCOM causes LLCOLOR.LCOM to be LOADed.

The machine dependent portions of Xerox Lisp color software is stored in files such as MAIKOCOLOR.LCOM, DORADOCOLOR.LCOM, or COLORNNCC.LCOM. The user LOADs one of these files according to what kind of machine and color card the user is using.

The Sun color driver resides in the file MAIKOCOLOR.LCOM which loads LLCOLOR.LCOM and COLOR.LCOM. The CG4 device suppports 8 bpp at 1152 by 900 resolution. The user must be running ldecolor, the special color capable emulator. The physical display monitor is shared by both the monochrome and color screens (described below) .

The Dorado color driver resides in the file DORADOCOLOR.LCOM which loads LLCOLOR.LCOM and COLOR.LCOM. The Dorado color board supports four or eight bits per pixel (bpp) at 640 by 480 resolution. (The board supports 24 bpp also, but Xerox Lisp doesn’t yet.)

The Dandelion color drivers reside in the files DANDELIONUFO.LCOM, DANDELIONUFO4096.LCOM, and COLORNNCC.LCOM, one package for each of three different kinds of boards. The user should load one of these packages on a Dandelion attached to a BusMaster and color display. The DANDELIONUFO and DANDELIONUFO4096 packages drive 4 bpp at 640 by 400 resolution color boards used inside Xerox which have been made obsolete by COLORNNCC. The COLORNNCC package drives an 8 bpp color at 512 by 480 resolution board, the Revolution 512 x 8, made by Number Nine Computer Corporation. The Revolution 512 x 8 is available both inside and outside Xerox through Number Nine.

Hardware Displays and Software Screens

On some workstations (such as the Dorado and Dandelion), there may be physically two separate displays. On most Suns, there is a single physical display, which additionally may be shared by two Unix devices. One device is monochrome (b/w), and the other is color.

To support the various hardware configurations and external display devices, the software has a special datatype, a "screen". There are two distinct instances of screens, a b/w screen, and a color screen. A screen represents and controls a physical hardware display, and contains windows, icons, and tracks the mouse.

On workstations with physically two separate hardware displays, each display is represented by a corresponding screen data structure. On workstations with a single hardware display, the display is shared by both the b/w screen and the color screen.

In all cases, before initialization only the b/w screen (and thus display) is visible and active. After initialization both screens are active (can contain screen images), although on single displays, only one screen is visible at a time. Since each screen logically controls a display, we will henceforth use the terms "screen" and "display" interchangeably. Screens are discussed in greater detail below.

Turning the Color Display Software On and Off

The color display software can be turned on and off. While the color display software is on, the memory used for the color display screen bitmap is locked down, and a small amount of processing time is used to drive the color display.

(COLORDISPLAYP) [Function]

returns T if the color display is on; otherwise it returns NIL.

(COLORDISPLAY*ONOFF TYPE***)** [Function]

turns off the color display if *ONOFF* is ’OFF. If *ONOFF* is ’ON, it turns on the color display allocating memory for the color screen bitmap. *TYPE* should be one of ’MAIKOCOLOR, ’DORADOCOLOR, ’DANDELIONUFO, ’DANDELIONUFO4096, or ’COLORNNCC. The usual sequence of events for the user is to LOAD the software needed to drive a particular color card and then to call COLORDISPLAY with the appropriate *TYPE* to turn the software on. For example,

```
(LOAD ’COLOR.LCOM)
(LOAD ’COLORNNCC.LCOM)
(COLORDISPLAY ’ON ’REV512X8)
```

will turn on the software needed to drive the Number Nine Computer Corporation’s Revolution 512 x 8 card with 1108 and BusMaster.

Besides initializing or reinitializing a color card that has been powered off, COLORDISPLAY allocates memory for the color screen bitmap. Turning on the color display requires allocating and locking down the memory necessary to hold the color display screen bitmap. Turning off the color display frees this memory.

Colors

The number of bits per pixel determines the number of different colors that can be displayed at one time. When there are 4 bpp, 16 colors can be displayed at once. When there are 8 bpp, 256 colors can be displayed at once. A table called a *color map* determines what color actually appears for each pixel value. A color map gives the color in terms of how much of the three primary colors (red, green, and blue) is displayed on the screen for each possible pixel value.

A color can be represented as a number, an atom, or a triple of numbers. Colors are ultimately given their final interpretation into how much red, blue, and green they represent through a color map. A color map maps a color number ([0 . . . 2^{nbits}-1]) into the intensities of the three color guns (primary colors red, green, and blue). Each entry in the color map has eight bits for each of the primary colors, allowing 256 levels per primary or 2^{24} possible colors (not all of which are distinct to the human eye). Within Xerox Lisp programs, colors can be manipulated as numbers, red-green-blue triples, names, or hue-lightness-saturation triples. Any function that takes a color accepts any of the different representations.

If a number is given, it is the color number used in the operation. It must be valid for the color bitmap used in the operation. (Since all of the routines that use a color need to determine its number, it is fastest to use numbers for colors. COLORNUMBERP, described below, provides a way to translate into numbers from the other representations.)

Red Green Blue Triples

A red green blue (RGB) triple is a list of three numbers between 0 and 255. The first element gives the intensity for red, the second for green, and the third for blue. When an RGB triple is used, the current color map is searched to find the color with the correct intensities. If none is found, an error is generated. (That is, no attempt is made by the system to assign color numbers to intensities automatically.) An example of an RGB triple is (255 255 255), which gives the color white.

RGB [Record]

is a record that is defined as (RED GREEN BLUE); it can be used to manipulate RGB triples.

COLORNAMES [Association list]

maps names into colors. The CDR of the color name's entry is used as the color corresponding to the color name. This can be any of the other representations. (Note: It can even be another color name. Loops in the name space such as would be caused by putting '(RED . CRIMSON) and '(CRIMSON . RED) on COLORNAMES are *not* checked for by the system.) Some color names are available in the initial system and are intended to allow color programs written by different users to coexist. These are:

Name	RGB	Number in default color maps	
BLACK	(0 0 0)	15	255
BLUE	(0 0 255)	14	252
GREEN	(0 255 0)	13	227
CYAN	(0 255 255)	12	224
RED	(255 0 0)	3	31
MAGENTA	(255 0 255)	2	28
YELLOW	(255 255 0)	1	3
WHITE	(255 255 255)	0	0

Hue Lightness Saturation Triples

A hue lightness saturation triple is a list of three numbers. The first number (HUE) is an integer between 0 and 355 and indicates a position in degrees on a color wheel (blue at 0, red at 120, and green at 240). The second (LIGHTNESS) is a real number between zero and one that indicates how much total intensity is in the color. The third (SATURATION) is a real number between zero and one that indicates how disparate the three primary levels are.

HLS [Record]

is a record defined as (HUE LIGHTNESS SATURATION); it is provided to manipulate HLS triples. Example: the color blue is represented in HLS notation by (0 .5 1.0).

(COLORNUMBERP *COLOR BITSPERPIXEL NOERRFLG***)**[Function]

returns the color number (offset into the screen color map) of *COLOR*. *COLOR* is one of the following:

· A positive number less than the maximum number of colors,

· A color name,

· AN RGB triple, or

· An HLS triple.

If *COLOR* is one of the above and is found in the screen color map, its color number in the screen color map is returned. If not, an error is generated unless NOERRFLG is non-NIL, in which case NIL is returned.

(RGBP *X***)** [Function]

returns *X* if *X* is an RGB triple; NIL otherwise.

(HLSP *X***)** [Function]

returns *X* if *X* is an HLS triple; NIL otherwise.

Color Maps

The screen color map holds the information about what color is displayed on the color screen for each pixel value in the color screen bitmap. The values in the current screen color map may be changed, and this change is reflected in the colors displayed at the next vertical retrace (approximately 1/30 of a second). The color map can be changed to obtain dramatic effects.

(SCREENCOLORMAP *NEWCOLORMAP***)** [Function]

reads and sets the color map that is used by the color display. If *NEWCOLORMAP* is non-NIL, it should be a color map, and **SCREENCOLORMAP** sets the system color map to be that color map. The value returned is the value of the screen color map before **SCREENCOLORMAP** was called. If *NEWCOLORMAP* is NIL, the current screen color map is returned without change.

(CMYCOLORMAP CYANBITS MAGENTABITS YELLOWBITS BITSPERPIXEL**)** [Function]

Returns a color map that assumes the BITSPERPIXEL bits are to be treated as three separate color planes with CYANBITS bits being in the cyan plane, MAGENTABITS bits being in the magenta plane, and YELLOWBITS bits being in the yellow plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 0 and black is 255.

(RGBCOLORMAP REDBITS GREENBITS BLUEBITS BITSPERPIXEL**)** [Function]

Returns a color map that assumes the BITSPERPIXEL bits are to be treated as three separate color planes with REDBITS bits being in the red plane, GREENBITS bits being in the green plane, and BLUEBITS bits being in the blue plane. Within each plane, the colors are uniformly distributed over the intensity range 0 to 255. White is 255 and black is 0.

(GRAYCOLORMAP BITSPERPIXEL**)** [Function]

Returns a color map containing shades of gray. White is 0 and black is 255.

(COLORMAPCREATE *INTENSITIES BITSPERPIXEL***)** [Function]

creates a color map for a screen that has *BITSPERPIXEL* bits per pixel. If *BITSPERPIXEL* is NIL, the number of bits per pixel is taken from the current color display setting. *INTENSITIES* specifies the initial colors that should be in the map. If *INTENSITIES* is not NIL, it should be a list of color specifications other than color numbers, e.g., the list of RGB triples returned by the function INTENSITIESFROMCOLOR MAP.

(INTENSITIESFROMCOLORMAP *COLORMAP***)** [Function]

returns a list of the intensity levels of *COLORMAP* (default is (SCREENCOLORMAP)) in a form accepted by COLORMAPCREATE. This list can be written on file and thus provides a way of saving color map specifications.

(COLORMAPCOPY *COLORMAP BITSPERPIXEL***)** [Function]

returns a color map that contains the same color intensities as *COLORMAP* if *COLORMAP* is a color map. Otherwise, it returns a color map with default color values.

(MAPOFACOLOR *PRIMARIES***)** [Function]

returns a color map that is different shades of one or more of the primary colors. For example, (MAPOFACOLOR '(RED GREEN BLUE)) gives a color map of different shades of gray; (MAPOFACOLOR 'RED) gives different shades of red.

Changing Color Maps

The following functions are provided to access and change the intensity levels in a color map.

(SETCOLORINTENSITY *COLORMAP COLORNUMBER COLORSPEC***)** [Function]

sets the primary intensities of color number *COLORNUMBER* in the color map *COLORMAP* to the ones specified by *COLORSPEC. COLORSPEC* can be either an RGB triple, an HLS triple, or a color name. The value returned is NIL.

(COLORLEVEL *COLORMAP COLORNUMBER PRIMARY NEWLEVEL***)** [Function]

sets and reads the intensity level of the primary color *PRIMARY* (RED, GREEN, or BLUE) for the color number *COLORNUMBER* in the color map *COLORMAP*. If *NEWLEVEL* is a number between 0 and 255, it is set. The previous value of the intensity of *PRIMARY* is returned.

(ADJUSTCOLORMAP *PRIMARY DELTA COLORMAP***)** [Function]

adds *DELTA* to the intensity of the *PRIMARY* color value (RED, GREEN, or BLUE) for every color number in *COLORMAP*.

(ROTATECOLORMAP *STARTCOLOR THRUCOLOR***)** [Function]

rotates a sequence of colors in the SCREENCOLORMAP. The rotation moves the intensity values of color number *STARTCOLOR* into color number *STARTCOLOR*+1, the intensity values of color number *STARTCOLOR*+1 into color number *STARTCOLOR*+2, etc., and *THRUCOLOR's* values into *STARTCOLOR*.

(EDITCOLORMAP *VAR NOQFLG***)** [Function]

allows interactive editing of a color map. If *VAR* is an atom whose value is a color map, its value is edited. Otherwise a new color map is created and edited. The color map being edited is made the screen color map while the editing takes place so that its effects can be observed. The edited color map is returned as the value. If *NOQFLG* is NIL and the color display is on, you are asked if you want a test pattern of colors. A yes response causes the function SHOWCOLORTESTPATTERN to be called, which displays a test pattern with blocks of each of the possible colors.

You are prompted for the location of a color control window to be placed on the black-and-white display. This window allows the value of any of the colors to be changed. The number of the color being edited is in the upper left part of the window. Six bars are displayed. The right three bars give the color intensities for the three primary colors of the current color number. The left three bars give the value of the color's Hue, Lightness, and Saturation parameters. These levels can be changed by positioning the mouse cursor in one of the bars and pressing the left mouse button. While the left button is down, the value of that parameter tracks the *Y* position of the cursor. When the left button is released, the color tracking stops. The color being edited is changed by pressing the middle mouse button while the cursor is in the interior of the edit window. This brings up a menu of color numbers. Selecting one sets the current color to the selected color.

The color being edited can also be changed by selecting the menu item "PickPt." This switches the cursor onto the color screen and allows you to select a point from the color screen. It then edits the color of the selected point.

To stop the editing, move the cursor into the title of the editing window and press the middle button. This brings up a menu. Select Stop to quit.

Color Bitmaps

A color bitmap is actually a bitmap that has more than one bit per pixel. To test whether a bitmap is a color bitmap, the function BITSPERPIXEL can be used.

(BITSPERPIXEL *BITMAP***)** [Function]

returns the bits per pixel of *BITMAP*; if this does not equal one, *BITMAP* is a color bitmap.

In multiple-bit-per-pixel bitmaps, the bits that represent a pixel are stored contiguously. BITMAPCREATE is passed a *BITSPERPIXEL* argument to create multiple-bit-per-pixel bitmaps.

(BITMAPCREATE *WIDTH HEIGHT BITSPERPIXEL***)** [Function]

creates a color bitmap that is *WIDTH* pixels wide by *HEIGHT* pixels high allowing *BITSPERPIXEL* bits per pixel. Currently any value of *BITSPERPIXEL* except one, four, eight, or NIL (defaults to one) causes an error.

A four-bit-per-pixel color screen bitmap uses approximately 76K words of storage, and an eight-bit-per-pixel one uses approximately 153K words. There is only one such bitmap. The following function provides access to it.

(COLORSCREENBITMAP) [Function]

returns the bitmap that is being or will be displayed on the color display. This is NIL if the color display has never been turned on (see COLORDISPLAY below).

Screens, Screenpositions, and Screenregions

In addition to positions and regions, the user needs to be aware of screens, screenpositions, and screenregions in the presence of multiple screens.

Screens

SCREEN [Datatype]

There are generally two screen datatype instances in existence when working with color. This is because the user is attached to two displays, a black and white display and a color display.

(MAINSCREEN) [Function]

returns the screen datatype instance that represents the black and white screen. This will be something like {SCREEN}#74,24740.

(COLORSCREEN) [Function]

returns the screen datatype instance that represents the color screen. Screens appear as part of screenpositions and screenregions, serving as the extra information needed to make clear whether a particular position or region should be viewed as lying on the black and white display or the color display.

(SCREENBITMAP *SCREEN***)** [Function]

returns the bitmap destination of *SCREEN*. If *SCREEN*=NIL, returns the black and white screen bitmap.

Screenpositions

SCREENPOSITION [Record]

Somewhat like a position, a screenposition denotes a point in an X,Y coordinate system on a particular screen. Screenpositions have been defined according to the following record declaration:

```
(RECORD SCREENPOSITION (SCREEN . POSITION)
        (SUBRECORD POSITION))
```

A SCREENPOSITION is an instance of a record with fields XCOORD, YCOORD, and SCREEN and is manipulated with the standard record package facilities. For example, (create SCREENPOSITION XCOORD _ 10 YCOORD _ 20 SCREEN _ (COLORSCREEN)) creates a screenposition representing the point (10,20) on the color display. The user can extract the position of a screenposition by fetching its POSITION. For example, (fetch (SCREENPOSITION POSITION) of SP12).

Screenregions

SCREENREGION [Record]

Somewhat like a region, a screenregion denotes a rectangular area in a coordinate system. Screenregions have been defined according to the following record declaration:

```
(RECORD SCREENREGION (SCREEN . REGION)
        (SUBRECORD REGION))
```

Screenregions are characterized by the coordinates of their bottom left corner and their width and height. A SCREENREGION is a record with fields LEFT, BOTTOM, WIDTH, HEIGHT, and SCREEN. It can be manipulated with the standard record package facilities. There are access functions for the REGION record that return the TOP and RIGHT of the region. The user can extract the region of a screenregion by fetching its REGION. For example, (fetch (SCREENREGION REGION) of SR8).

Screenposition and Screenregion Prompting

The following functions can be used by programs to allow the user to interactively specify screenpositions or screenregions on a display screen.

(GETSCREENPOSITION *WINDOW CURSOR***)** [Function]

Similar to GETPOSITION. Returns a SCREENPOSITION that is specified by the user. GETSCREENPOSITION waits for the user to press and release the left button of the mouse and returns the cursor screenposition at the time of release. If *WINDOW* is a WINDOW, the screenposition will be on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position will be in screen coordinates.

(GETBOXSCREENPOSITION *BOXWIDTH BOXHEIGHT ORGX ORGY WINDOW PROMPTMSG***)** [Function]

Similar to GETBOXPOSITION. Returns a SCREENPOSITION that is specified by the user. Allows the user to position a "ghost" region of size *BOXWIDTH* by *BOXHEIGHT* on a screen, and returns the SCREENPOSITION of the lower left corner of the screenregion chosen. A ghost region is locked to the cursor so that if the cursor is moved, the ghost region moves with it. The user can change to another corner by holding down the right button. With the right button down, the cursor can be moved across a screen or to other screens without effect on the ghost region frame. When the right button is released, the mouse will snap to the nearest corner, which will then become locked to the cursor. (The held corner can be changed after the left or middle button is down by holding both the original button and the right button down while the cursor is moved to the desired new corner, then letting up just the right button.) When the left or middle button is pressed and released, the lower left corner of the screenregion chosen at the time of release is returned. If *WINDOW* is a WINDOW, the screenposition will be on the same screen as *WINDOW* and in the coordinate system of *WINDOW*'s display stream. If *WINDOW* is NIL, the position will be in screen coordinates.its lower left corner in screen coordinates.

(GETSCREENREGION *MINWIDTH MINHEIGHT OLDREGION NEWREGIONFN NEWREGIONFNARG INITCORNERS***)** [Function]

Similar to GETREGION. Returns a SCREENREGION that is specified by the user. Lets the user specify a new screenregion and returns that screenregion. GETSCREENREGION prompts for a screenregion by displaying a four-pronged box next to the cursor arrow at one corner of a "ghost" region: . If the user presses the left button, the corner of a "ghost" screenregion opposite the cursor is locked where it is. Once one corner has been fixed, the ghost screenregion expands as the cursor moves.

To specify a screenregion: (1) Move the ghost box so that the corner opposite the cursor is at one corner of the intended screenregion. (2) Press the left button. (3) Move the cursor to the screenposition of the opposite corner of the intended screenregion while holding down the left button. (4) Release the left button.

Before one corner has been fixed, one can switch the cursor to another corner of the ghost screenregion by holding down the right button. With the right button down, the cursor changes to a "forceps" () and the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor will snap to the nearest corner of the ghost screenregion.

After one corner has been fixed, one can still switch to another corner. To change to another corner, continue to hold down the left button and hold down the right button also. With both buttons down, the cursor can be moved across a screen or to other screens without effect on the ghost screenregion frame. When the right button is released, the cursor will snap to the nearest corner, which will become the moving corner. In this way, the screenregion may be moved all over a screen and to other screens, before its size and screenposition is finalized.

The size of the initial ghost screenregion is controlled by the *MINWIDTH*, *MINHEIGHT*, *OLDREGION*, and *INITCORNERS* arguments.

(GETBOXSCREENREGION *WIDTH HEIGHT ORGX ORGY WINDOW PROMPTMSG***)** [Function]

Similar to GETBOXREGION. Returns a SCREENREGION that is specified by the user. Performs the same prompting as GETBOXSCREENPOSITION and returns the SCREENREGION specified by the user instead of the SCREENPOSITION of its lower left corner.

Color Windows and Menus

The Xerox Lisp window system provides both interactive and programmatic constructs for creating, moving, reshaping, overlapping, and destroying windows in such a way that a program can use a window in a relatively transparent fashion (see page X.XX). Menus are a special type of window provided by the window system, used for displaying a set of items to the user, and having the user select one using the mouse and cursor. The menu facility also allows users to create and use menus in interactive programs (see page X.XX). As of the LUTE release of Xerox Lisp, it is possible for the user to create and use windows and menus on the color display.

(CREATEW *REGION TITLE BORDERSIZE NOOPENFLG***)** [Function]

Creates a new window. *REGION* indicates where and how large the window should be by specifying the exterior screenregion of the window. In a user environment with multiple screens, such as a black and white screen and color screen both connected to the same machine, there is a new special problem in indicating which screen the *REGION* is supposed to be a region of. This problem is solved by allowing CREATEW to take screenregion arguments as *REGION*. For example,

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
                                        SCREEN _
(COLORSCREEN)
                           LEFT _ 20
                           BOTTOM _ 210
                           WIDTH _ 290
                           HEIGHT _ 170)
                  "FOO WINDOW"))
```

creates a window titled "FOO WINDOW" on the color screen. To create a window on the black and white screen, the user should use SCREEN _ (MAINSCREEN) in the CREATE SCREENREGION expression. Note that it is still perfectly legal to pass in a *REGION* that is a region, not a screenregion, to CREATEW, but it is preferable to be passing screenregions rather than regions to CREATEW. This is because when *REGION* is a region, *REGION* is disambiguated in a somewhat arbitrary manner that may not always turn out to be what the user was hoping for.

When *REGION* is a region, *REGION* is disambiguated by coercing *REGION* to be a screenregion on the screen which currently contains the cursor. This is so that software calling CREATEW with regions instead of screenregions tends to do the right thing in a user environment with multiple screens.

(WINDOWPROP *WINDOW PROP NEWVALUE***)** [NoSpread Function]

If *PROP*='SCREEN, then WINDOWPROP returns the screen *WINDOW* is on. If *NEWVALUE* is given, (even if given as NIL), with *PROP*='SCREEN, then WINDOWPROP will generate an error. Any other *PROP* name is handled in the usual way.

(OPENWINDOWS *SCREEN***)** [Function]

Returns a list of all open windows on *SCREEN* if *SCREEN* is a screen datatype such as (MAINSCREEN) or (COLORSCREEN). If *SCREEN*=NIL, then *SCREEN* will default to the screen containing the cursor. If *SCREEN*=T, then a list of all open windows on all screens is returned.

Color Fonts

The user can create color fonts and specify in the font profile that certain color fonts be used when printing in color.

Color Font Creation

The user can create and manipulate color fonts through the same functions that are used to create and manipulate black and white fonts. This is made possible in some cases by there being new ways to call familiar font functions.

(FONTCREATE *FAMILY SIZE FACE ROTATION DEVICE NOERRORFLG CHARSET***)** [Function
]

In addition to creating black and white fonts, FONTCREATE can be used to create color fonts. For example,

```
(FONTCREATE 'GACHA 10
                        '(BOLD REGULAR REGULAR YELLOW
BLUE)
                0 '8DISPLAY)
```

will create an 8 bit per pixel font with blue letters on a yellow background. The user indicates the color and bits per pixel of the font by the FACE and DEVICE arguments passed to FONTCREATE. DEVICE='8DISPLAY means to create an 8bpp font and DEVICE='4DISPLAY means to create a 4bpp font. A color font face is a 5 tuple,

(WEIGHT SLOPE EXPANSION BACKCOLOR FORECOLOR)

whereas a black and white font face is just a 3 tuple,

(WEIGHT SLOPE EXPANSION)

The FORECOLOR is the color of the characters of the font and the BACKCOLOR is the color of the background behind the characters that gets printed along with the characters. Both BACKCOLOR and FORECOLOR are allowed to a color name, color number, or any other legal color representation. A color font face can also be represented as a LITATOM. A three character atom such as MRR or any of the special atoms STANDARD, ITALIC, BOLD, BOLDITALIC can optionally be continued by hyphenating on BACKCOLOR and FORECOLOR suffixes. For example, MRR-YELLOW-BLUE

BOLD-YELLOW-RED

ITALIC-90-200

BRR-100-53

are acceptable color font faces. Hence,

```
(FONTCREATE 'GACHA 10 'BOLD-YELLOW-BLUE 0 '8DISPLAY)
```

will create a color font. LITATOM FACE arguments fall into one of the following patterns:

wse	wse-backcolor-forecolor
STANDARD	STANDARD-backcolor-forecolor
ITALIC	ITALIC-backcolor-forecolor
BOLD	BOLD-backcolor-forecolor
BOLDITALIC	BOLDITALIC-backcolor-forecolor

where w=B, M, or L; s=I or R; e=R, C, or E; backcolor=a color name or color number; and forecolor=a color name or color number.

(FONTPROP *FONT PROP***)** [Function]

Returns the value of the *PROP* property of font *FONT*. Besides black and white font properties, the following font properties are recognized:

FORECOLOR The color of the characters of the font, represented as a color number. This is the color in which the characters of the font will print.

BACKCOLOR The color of the background of the characters of the font, represented as a color number. This is the color in which the the background of characters of the font will print. A font with red characters on a yellow background would have a red FORECOLOR and a yellow BACKCOLOR.

Color Font Profiles

Font profiles are the facility PRETTYPRINT uses to print different elements (user functions, system functions, clisp words, comments, etc.) in different fonts to emphasize (or deemphasize) their importance, and in general to provide for a more pleasing appearance. The user can specify that different colors of fonts be used for different kinds of elements when printing in color. A well chosen font profile will allows user to DEDIT functions, PP functions, and SEE source files in color, for example. A **FONTPROFILE** such as

```
((DEFAULTFONT 1 (GACHA 10)
        (GACHA 8)
        (TERMINAL 8)
        (4DISPLAY (GACHA 10 MRR-WHITE-RED))
        (8DISPLAY (GACHA 10 MRR-WHITE-RED)))
 (BOLDFONT 2 (HELVETICA 10 BRR)
      (HELVETICA 8 BRR)
      (MODERN 8 BRR)
      (4DISPLAY (HELVETICA 10 BRR-WHITE-MAGENTA))
```

```
            (8DISPLAY (HELVETICA 10 BRR-WHITE-MAGENTA)))
    (LITTLEFONT 3 (HELVETICA 8)
             (HELVETICA 6 MIR)
             (MODERN 8 MIR)
             (4DISPLAY (HELVETICA 8 MRR-WHITE-GREEN))
             (8DISPLAY (HELVETICA 8 MRR-WHITE-GREEN)))
    (BIGFONT 4 (HELVETICA 12 BRR)
           (HELVETICA 10 BRR)
           (MODERN 10 BRR)
           (4DISPLAY (HELVETICA 12 BRR-WHITE-BLUE))
           (8DISPLAY (HELVETICA 12 BRR-WHITE-BLUE)))
    (USERFONT BOLDFONT)
    (COMMENTFONT LITTLEFONT)
    (LAMBDAFONT BIGFONT)
    (SYSTEMFONT)
    (CLISPFONT BOLDFONT)
    ...)
```

would have comments print in green and clisp words print in blue while ordinairy atoms would print in red.

Not all combinations of fonts will be aesthetically pleasing and the user may have to experiment to find a compatible set.

The user should indicate what font is to be used for each font class by calling the function FONTPROFILE:

(FONTPROFILE *PROFILE*) [Function]

Sets up the font classes as determined by *PROFILE*, a list of elements which defines the correspondence between font classes and specific fonts. Each element of *PROFILE* is a list of the form:

(*FONTCLASS FONT# DISPLAYFONT PRESSFONT INTERPRESSFONT* **(***OTHERDEVICE1 OTHERFONT1***)** ... **(***OTHERDEVICEn OTHERFONTn***))**

FONTCLASS is the font class name and *FONT#* is the font number for that class. *DISPLAYFONT*, *PRESSFONT*, and *INTERPRESSFONT* are font specifications (of the form accepted by FONTCREATE) for the fonts to use when printing to the black and white display and to Press and Interpress printers respectively. The appearance of color fonts can be affected by including an **(***OTHERDEVICEi OTHERFONTi***)** entry where *OTHERDEVICEi* is either 4DISPLAY or 8DISPLAY for a 4 bits per pixel or 8 bits per pixel color font and *OTHERFONTi* is a color font specification such as (GACHA 10 MRR-WHITE-RED).

FONTPROFILE [Variable]

This is the *variable* used to store the current font profile, in the form accepted by the *function* FONTPROFILE. Note that simply editing this value will not change the fonts used for the various font classes; it is necessary to execute (FONTPROFILE FONTPROFILE) to install the value of this variable.

Using Color

The current color implementation allows display streams to operate on color bitmaps. The two functions DSPCOLOR and DSPBACKCOLOR set the color in which a stream draws when the user defaults a color argument to a drawing function.

(DSPCOLOR *COLOR STREAM***)** [Function]

sets the foreground color of a stream. It returns the previous foreground color. If *COLOR* is NIL, it returns the current foreground color without changing anything. The default foreground color is MINIMUMCOLOR=0, which is white in the default color maps.

(DSPBACKCOLOR *COLOR STREAM***)** [Function]

sets the background color of a stream. It returns the previous background color. If *COLOR* is NIL, it returns the current background color without changing anything. The default background color is (MAXIMUMCOLOR BITSPERPIXEL)=15 or 255, which is black in the default color maps.

The BITBLT, line-drawing routines, and curve-drawing routines routines know how to operate on a color-capable stream. Following are some notes about them.

BITBLTing in Color

If BITBLTing from a color bitmap onto another color bitmap of the same bpp, the operations PAINT, INVERT, and ERASE are done on a bit level, not on a pixel level. Thus painting color 3 onto color 10 results in color 11.

If BITBLTing from a black-and-white bitmap onto a color bitmap, the one bits appear in the DSPCOLOR, and the zero bits in DSPBACKCOLOR. BLTing from black-and-white to color is fairly expensive; if the same bitmap is going to be put up several times in the same color, it is faster to create a color copy and then BLT the color copy.

If the source type is TEXTURE and the destination bitmap is a color bitmap, the *Texture* argument is taken to be a color. Thus to fill an area with the color BLUE assuming COLORSTR is a stream whose destination is the color screen, use (BITBLT NIL NIL NIL COLORSTR 50 75 100 200 ’TEXTURE ’REPLACE ’BLUE).

Drawing Curves and Lines in Color

For the functions DRAWCIRCLE, DRAWELLIPSE, and DRAWCURVE, the notion of a brush has been extended to include a color. A BRUSH is now (BRUSHSHAPE BRUSHSIZE BRUSHCOLOR). Also, a brush can be a bitmap (which can be a color bitmap).

Line-drawing routines take a color argument which is the color the line is to appear in if the destination of the display stream is a color bitmap.

(DRAWLINE *X1 Y1 X2 Y2 WIDTH OPERATION STREAM COLOR***)** [Function]

(DRAWTO *X Y WIDTH OPERATION STREAM COLOR***)** [Function]

(RELDRAWTO *X Y WIDTH OPERATION STREAM COLOR***)** [Function]

(DRAWBETWEEN *POS1 POS2 WIDTH OPERATION STREAM COLOR***)** [Function]

If the *COLOR* argument is NIL, the DSPCOLOR of the stream is used.

Printing in Color

Printing only works in REPLACE mode. The characters have a background color and a foreground color determined by the font face of the font the characters are being printed with.

Example of printing to an 8bpp color screen:

```
(SETQ FOO (CREATEW (CREATE SCREENREGION
                                                SCREEN _
(COLORSCREEN)
                                  LEFT _ 20
                                  BOTTOM _ 210
                                  WIDTH _ 290
                                  HEIGHT _ 170)
                          "FOO WINDOW"))
(DSPFONT (FONTCREATE 'GACHA
                           10
                           'MRR-YELLOW-GREEN
                           0
                           '8DISPLAY)
          FOO)
(PRINT 'HELLO FOO)         ; will print in green against a yellow
background.
```

Operating the Cursor on the Color Screen

The cursor can be moved to the color screen. The cursor can be moved to the color screen by sliding the cursor off the left or right edge of the black and white screen on to the color screen or by calling function CURSORPOSITION or CURSORSCREEN.

(CURSORPOSITION *NEWPOSITION - -***)** [Function]

NEWPOSITION can be a position or a screenposition.

(CURSORSCREEN *SCREEN XCOORD YCOORD***)** [Function]

Moves the cursor to the screenposition determined by *SCREEN*, *XCOORD*, and *YCOORD*. *SCREEN* should be the value of either (COLORSCREEN) or (MAINSCREEN).

While on the color screen, the cursor is placed by doing BITBLTs in software rather than with microcode and hardware as with the black and white cursor. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. The speed of the color cursor compares well with that of the black and white cursor but there can be a noticeable flicker when there is much input/output to the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared giving the appearance that there is never more than one cursor at a given time.

Miscellaneous Color Functions

(COLORIZEBITMAP *BITMAP 0COLOR 1COLOR BITSPERPIXEL***)**[Function]

creates a color bitmap from a black and white bitmap. The returned bitmap has color number *1COLOR* in those pixels of *BITMAP* that were one and *0COLOR* in those pixels of *BITMAP* that were zero. This provides a way of producing a color bitmap from a black and white bitmap.

(UNCOLORIZEBITMAP *BITMAP COLORMAP***)** [Function]

creates a black and white bitmap from a color bitmap.

(SHOWCOLORTESTPATTERN *BARSIZE***)** [Function]

displays a pattern of colors on the color display. This is useful when editing a color map. The pattern has squares of the 16 possible colors laid out in two rows at the top of the screen. Colors 0 through 7 are in the top row, and colors 8 through 15 are in the next row. The bottom part of the screen is filled with bars of *BARSIZE* width with consecutive color numbers. The pattern is designed so that every color has a border with every other color (unless *BARSIZE* is too large to allow room for every color—about 20).