# BITMAPFNS

(READBINARYBITMAP *WIDTH HEIGHT FILE*) [Function]
reads a series of bytes from *FILE* and creates a *WIDTH* times *HEIGHT* bit map with contents. Note that each scanline of the bit map is rounded up to the nearest multiple of 16 bits (two bytes).

(WRITEBINARY BITMAP *BITMAP FILE*) [Function]
writes out *BITMAP* to *FILE* in format read by READBINARYBITMAP. Please note that READBINARYBITMAP must be supplied with width and height.

(WRITEBM *FILE BITMAP*) [Function]
writes *BITMAP* on *FILE* first preceding with width and height (in binary) such that it can be read in with READBM.

(READBM *FILE*) [Function]
reads width, height, and then appropriate size bit map.

(WRITEBMLST *FILE LST*) [Function]
writes a list of bit maps on *FILE*.

(READBMLST *FILE*) [Function]
reads a list of bit maps.

The following functions open and close *FILE*.

(READPRESS *PRESSFILE*) [Function]
reads press file *PRESSFILE* and returns a bit map. Can only handle press files generated by PRESSBITMAP and a couple of other utilities. Has no smarts, and is not easily extended.

(WINDOWBM *BITMAP POSITION*) [Function]
creates and returns a window containing image of *BITMAP*. Will be at POSITION or (GETPOSITION).

# COLOR

## Introduction

This document describes the Interlisp-D facilities for using the color display.

The machine-independent color graphics code is stored on the library files LLCOLOR.DCOM and COLOR.DCOM; COLOR loads LLCOLOR. In order to run it, you need a Dorado (Xerox 1132) with attached color display. The Dorado color driver resides in the file DORADOCOLOR.DCOM, which loads after COLOR.DCOM.[1] The Dorado color board supports four or eight bits per pixel (bpp) at 640 by 480 resolution. (Actually the board supports 24 bpp also, but Interlisp-D doesn't yet.)

A color display's image can be thought of as a paint-by-number painting where the number of a pixel is its value. 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 mapping 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. In the following sections, the notions of "color map" and "color" are described.

## Color Bit Maps

A color bit map is actually a bit map that has more than one bit per pixel. To test whether a bit map is a color bit map, 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 bit map.

In multiple-bit-per-pixel bit maps, the bits that represent a pixel are stored contiguously. BITMAPCREATE was extended to create multiple-bit-per-pixel bit maps.

(BITMAPCREATE *Width Height BitsPerPixel*) [Function]

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

(COLORSCREENBITMAP) [Function]

returns the bit map 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).

WHOLECOLORDISPLAY [Variable]

is a global variable set to a REGION that covers the entire color display screen.

COLORSCREENWIDTH [Variable]

and

COLORSCREENHEIGHT [Variable]

are global variables whose values are the dimensions of the color display.

## Color Representations

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 Interlisp-D 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 bit map 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.) Several 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 map |
|---|---|---|
| BLACK | (0 0 0) | 0 |
| BLUE | (0 0 255) | 1 |
| GREEN | (0 255 0) | 2 |
| CYAN | (0 255 255) | 3 |
| RED | (255 0 0) | 4 |
| MAGENTA | (255 0 255) | 5 |
| YELLOW | (255 255 0) | 6 |
| WHITE | (255 255 255) | 7 |

## 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 bit map. 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.

(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. If *Intensities* is NIL, the value of the variable \DEFAULTCOLORINTENSITIES is used if *BitsPerPixel* is four; the value of the variable \DEFAULT8BITCOLORINTENSITIES if *BitsPerPixel* is eight.

(COLORMAPP *ColorMap? BitsPerPixel*) [Function]

returns *ColorMap?* if it is a color map that has *BitsPerPixel* bits per pixel; NIL otherwise. If *BitsPerPixel* is NIL, it returns *ColorMap?* if it is either a 4 bpp or an 8 bpp color 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.

(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.

(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.

## How to Change 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 PrimaryColor NewLevel*) [Function]

sets and reads the intensity level of the primary color *PrimaryColor* (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 *PrimaryColor* 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 *ColorMap StartColor ThruColor*) [Function]

rotates a sequence of colors in *ColorMap*. 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 COLORMAPP. If *Var* is an atom whose value is a COLORMAPP, 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.

## How to Turn the Color Display On and Off

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

(COLORDISPLAYP) [Function]

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

(COLORDISPLAY *ColorMapIfOn BitsPerPixel ClearScreenFlg*) [Function]

turns off the color display if *ColorMapIfOn* is NIL. If *ColorMapIfOn* is non-NIL, it turns on the color display allocating *BitsPerPixel* bits per pixel. If *ColorMapIfOn* is of type COLORMAPP or 8BITCOLORMAP, it is used as the screen color map. If *ClearScreenFlg* is non-NIL, all of the bits in the color screen are set to zero. Returns the previous color map, if any.

Turning on the color display requires allocating and locking down the memory necessary to hold the color display screen bit map and the system color map. Turning off the color display frees this memory.

## How to Use Color

The current color implementation allows display streams to operate on color bit maps. The two functions DSPCOLOR and DSPBACKCOLOR set the color in which a stream prints or draws.

(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 seven, which is white in the default color map.

(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 zero, which is black in the default color map.

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

## How to BITBLT

If BLTing from a color bit map onto another color bit map 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 BLTing from a black-and-white bit map onto a color bit map, the one bits appear in the DSPCOLOR, and the zero bits in DSPBACKCOLOR. Currently, REPLACE is the only operation that

supports BLTing from black-and-white to color. This operation is fairly expensive; if the same bit map is going to be put up several times in the same color, it is faster to create a color copy then BLT the color copy.

If the source type is TEXTURE and the destination bit map is a color bit map, 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).

## How to Draw Curves and Lines

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

The line-drawing routines have been extended to take another argument, which is the color the line is to appear in if the destination of the display stream is a color bit map.

(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.

## How to Print to the Color Screen

Printing only works in REPLACE mode. The characters have a background of DSPBACKCOLOR and a foreground color of DSPCOLOR. The first time a character is printed in a new color, the color images corresponding to the current font are calculated and cached. Thus the first character may take a while to appear, but succeeding characters print quickly.

Example of printing to the color screen:

```
(COLORDISPLAY T 4)          ; turn the display on in 4 bpp mode.

(SETQ FOO (DSPCREATE (COLORSCREENBITMAP))

                ; create a stream whose destination is the color screen

(DSPCOLOR 'RED FOO)
```

```
(PRINT 'HELLO FOO)                    ; will print in red.
```

## How to Move the Cursor on the Color Screen

The cursor can be moved to the color screen. While on the color screen, the cursor is placed using XOR mode; thus with some color maps it may be hard to see. It is automatically taken down whenever an operation is performed that changes any bits on the color screen. While the cursor is on the color screen, the black-and-white cursor is cleared.

(CHANGECURSORSCREEN *ScreenBitMap*) [Function]

moves the cursor onto the specified screen. *ScreenBitMap* must be the value of either (COLORSCREENBITMAP) or (SCREENBITMAP). The value returned is the screen bit map that the cursor was on before CHANGECURSORSCREEN was called.

## Miscellaneous Color Functions

The following functions provide some common operations on color bit maps and display streams.

(COLORFILL *Region ColorNumber Stream Operation*) [Function]

fills a region in the color bit map with a color.

(COLORFILLAREA *Left Bottom Width Height ColorNumber Stream Operation*) [Function]

fills an area in the color bit map with a color.

(COLORIZEBITMAP *BitMap 0Color 1Color NBits*) [Function]

creates a color bit map from a bit map. The returned bit map 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 bit map from a black-and-white bit map.

## Demonstration Programs

These are on file on COLORDEMO.DCOM.

(COLORDEMO) [Function]

brings up a menu of color demonstration programs. The system cycles through the entries on the menu automatically, allowing

each to run for a small, fixed amount of time (typically 40 seconds). Selecting one of the entries in the menu causes it to start that program.

(COLORDEMO1) [Function]

runs the Interlisp-D logo demonstration until a button is pressed, then adds color kinetic. The middle mouse button brings up a menu that allows changing the speed of rotation or editing the color map. The left button rotates the color map in the kinetic area.

(COLORDEMO2 *Size*) [Function]

puts up a test pattern of size *Size*, then rotates the color map. The speed of rotation of the color map is determined by the *Y* position of the cursor. The middle button brings up a menu that allows editing of the color map or changing the color map to a map of different shades of one color.

(COLORKINETIC *Region FirstColor LastColor*) [Function]

runs a color version of the standard *Kinetic* demo on a color display using colors *FirstColor* through *Last Color*.

(TUNNEL *Speed*) [Function]

draws a series of concentric rectangles of increasing size in increasing color numbers. *Speed* determines the size of the rectangles. This can then be "run" by calling ROTATEIT, which is described below.

(MINESHAFT *N OutFlg*) [Function]

draws a series of concentric rectangles of size *N* in increasing color numbers. *OutFlg* determines whether the color numbers increase or decrease. This can then be "run" by calling ROTATEIT, which is described below.

(WELL *N*) [Function]

draws a series of concentric circles on the color screen in increasing color numbers. The circles are of size *N*. This can then be "run" by calling ROTATEIT, which is described below.

(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).

(ROTATEIT *BeginColor EndColor Wait*) [Function]

goes into an infinite loop rotating the screen color map. The colors between *BeginColor* (default zero) and *EndColor* (default

maximum color) are rotated. If *Wait* is given, (DISMISS *Wait*) is called each time the color map is changed. This provides an easy way of "animating" screen images.

(COLORPOLYDEMO *ColorStream*) [Function]

is on the file COLORPOLYGONS.DCOM. It runs a version of the *Polygons* program on the color screen.

End Notes

1. The only user-accessible item in this package is the variable DORADO.COLORDISPLAY.LEFTMARGIN, which controls the period of time the display controller should wait before turning on the color guns. This is normally set to 56; if this value causes odd results with your monitor, try setting it a little higher or lower and reinitializing the display. (You can reinitialize the display by calling COLORDISPLAY twice in succession.)