

\$SPAD/src/lib spadcolors.c

The Axiom Team

July 29, 2014

**Abstract**

# Contents

1	License	3
---	---------	---

# 1 License

```
/*
Copyright (c) 1991-2002, The Numerical Algorithms Group Ltd.
All rights reserved.
```

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of The Numerical Algorithms Group Ltd. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

```
THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED
TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A
PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
*/
```

— \* —

```
#include "spadcolors.h"

#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "spadcolors.h1"
#include "util.h1"

#if 0
```

```

int colors[100];
#endif

static unsigned long    pixels[smoothConst+1];

/*
 * make sure you define a global variable like int *spadColors; in the main
 * program
 */

/*
 * code taken from Foley and Van Dam "Fundamentals of Interactive Computer
 * Graphics"
 */

RGB
HSVtoRGB(HSV hsv)
{
    RGB rgb;
    float h, f, p, q, t;
    int i;

    rgb.r = 0.0;
    rgb.g = 0.0;
    rgb.b = 0.0;
    if (hsv.s == 0.0) {
        rgb.r = rgb.g = rgb.b = hsv.v;
        return (rgb);
    }
    else {
        if (hsv.h == 360.0) {
            hsv.h = 0.0;
        }
        h = hsv.h / 60;
        i = floor(h);
        f = h - i;
        p = hsv.v * (1 - hsv.s);
        q = hsv.v * (1 - (hsv.s * f));
        t = hsv.v * (1 - (hsv.s * (1 - f)));
        switch (i) {
            case 0:
                rgb.r = hsv.v;
                rgb.g = t;
                rgb.b = p;
                break;
            case 1:
                rgb.r = q;

```

```

        rgb.g = hsv.v;
        rgb.b = p;
        break;
    case 2:
        rgb.r = p;
        rgb.g = hsv.v;
        rgb.b = t;
        break;
    case 3:
        rgb.r = p;
        rgb.g = q;
        rgb.b = hsv.v;
        break;
    case 4:
        rgb.r = t;
        rgb.g = p;
        rgb.b = hsv.v;
        break;
    case 5:
        rgb.r = hsv.v;
        rgb.g = p;
        rgb.b = q;
        break;
    }
    return (rgb);
}

}

float
value(float n1, float n2, float hue)
{
    float v;

    if (hue > 360.0)
        hue -= 360.0;
    if (hue < 0.0)
        hue += 360.0;
    if (hue < 60.0) {
        v = n1 + (n2 - n1) * hue / 60.0;
    }
    else {
        if (hue < 180.0)
            v = n2;
        else {
            if (hue < 240.0)
                v = n1 + (n2 - n1) * (240.0 - hue) / 60.0;
            else
                v = n1;
        }
    }
}

```

}

{

```

* int makeColors(dsply,scrn,colorMap,total_Shades)
*
* This routine tries to allocate an adequate color
* map to be used by all the AXIOM applications
* that are to be run under X Windows that use
* colors that may be user-definable (e.g. viewports,
* HyperTeX, etc). All these application should call
* this routine and then access the colors with the
* the returned color map.
*
* For example, the following creates the map and
* then sets the foreground color for a GC:
*
*
*   i = makeColors(d,s,&cmap,&spadColors,&ts);
*   XSetForegroundColor(d,gc,spadColors[3]);
*
* where
*
* spadColors is of type (unsigned long *)
* i (the return value) is the total number of colors
* allocated.
*
* ts is the total number of shades for each hue
*
*
* KF 6/14/90 (modification)
*
* makeColors creates color table once only.

```

```

* hiya is of type static.
*****/

int
makeColors(Display *dsply, int scrn, Colormap *colorMap,
           unsigned long **colorIndex, int *total_Shades)
{
    int h, s;
    static unsigned long *hiya; /* keep colortable around for next time */
    HSV hsv;
    RGB rgb;
    XColor color;
    int okay = yes;             /* is true (1) so long as XAllocColor is
                                * working ok. if 0, then we ran out of room
                                * on the color table. */

    int colorNum;

    /* shade5 definition */

    static float saturations[5] = {0.90, 0.80, 0.74, 0.50, 0.18};
    static float values[5] = {0.38, 0.58, 0.75, 0.88, 0.94};

    /* static float values[5]      = {0.34, 0.52, 0.80, 0.88, 0.94}; */

    /*      fprintf(stderr,"makeColors called\n");*/

    /* printf("making new colors...\n"); */
    *total_Shades = totalShadesConst;

    /* space for color table */
    hiya = (unsigned long *) saymem("spadcolors30.c",
                                   totalHuesConst * (*total_Shades) + 2, sizeof(unsigned long));
    *colorIndex = hiya;

    for (h = 0, colorNum = 0; okay && h < 60; h += (hueStep - 6)) {
        for (s = 0; okay && s < *total_Shades; s++) {
            hsv.h = h;
            hsv.s = saturations[s];
            hsv.v = values[s];
            rgb = HSVtoRGB(hsv);
            color.red    = rgb.r * ((1<<16)-1);
            color.green  = rgb.g * ((1<<16)-1);
            color.blue   = rgb.b * ((1<<16)-1);
            color.flags = DoRed | DoGreen | DoBlue;
            /*
            fprintf(stderr,"%f\t%f\t%f\n",rgb.r,rgb.g,rgb.b);
            fprintf(stderr,"%d\t%d\t%d\n",
                    color.red,color.green,color.blue);
            */
        }
    }
}

```

```

        if ((okay = XAllocColor(dsply, *colorMap, &color)))
            hiya[colorNum++] = color.pixel; /* hiya points to table */
    }
    /* for s */
}
    /* for h */
for (h = 60; okay && h < 180; h += 20) {
    for (s = 0; okay && s < *total_Shades; s++) {
        hsv.h = h;
        hsv.s = saturations[s];
        hsv.v = values[s];
        rgb = HSVtoRGB(hsv);

        color.red    = rgb.r * ((1<<16)-1);
        color.green  = rgb.g * ((1<<16)-1);
        color.blue   = rgb.b * ((1<<16)-1);
        color.flags = DoRed | DoGreen | DoBlue;
        /*
        fprintf(stderr,"%f\t%f\t%f\n",rgb.r,rgb.g,rgb.b);
        fprintf(stderr,"%d\t%d\t%d\n",
            color.red,color.green,color.blue);
        */

        if ((okay = XAllocColor(dsply, *colorMap, &color)))
            hiya[colorNum++] = color.pixel;
    }
}

for (h = 180; okay && h <= 300; h += hueStep) {
    for (s = 0; okay && s < *total_Shades; s++) {
        hsv.h = h;
        hsv.s = saturations[s];
        hsv.v = values[s];
        rgb = HSVtoRGB(hsv);

        color.red    = rgb.r * ((1<<16)-1);
        color.green  = rgb.g * ((1<<16)-1);
        color.blue   = rgb.b * ((1<<16)-1);
        color.flags = DoRed | DoGreen | DoBlue;
        /*
        fprintf(stderr,"%f\t%f\t%f\n",rgb.r,rgb.g,rgb.b);
        fprintf(stderr,"%d\t%d\t%d\n",
            color.red,color.green,color.blue);
        */

        if ((okay = XAllocColor(dsply, *colorMap, &color)))
            hiya[colorNum++] = color.pixel;
    }
}

hiya[colorNum++] = BlackPixel(dsply, scrn);
hiya[colorNum++] = WhitePixel(dsply, scrn);

```



```

    if (colorNum < (totalShadesConst * totalHuesConst + 2)) {
        free(*colorIndex);
        fprintf(stderr,
            "    > Warning: cannot allocate all the necessary colors");
        fprintf(stderr, " - switching to monochrome mode\n");
        *colorIndex = (unsigned long *)
            saymem("while allocating the colormap for AXIOM ",
                2, sizeof(unsigned long));
        (*colorIndex)[0] = BlackPixel(dsply, scrn);
        (*colorIndex)[1] = WhitePixel(dsply, scrn);
        return (-1);
    }

    return (colorNum);
}

#ifdef OLD
/*****
KF 6/14/90
INPUT: display dsply, screen scrn
OUTPUT: a pointer to the permutation color vector (permIndex)
PURPOSE: when called for the first time, this procedure creates a
         permutation vector of the color table spadColor. It
         returns the pointer to this vector for subsequent calls.

*****/

int
makePermVector(Display *dsply, int scrn, unsigned long **permIndex)
{
    static int firstTime = yes;
    unsigned long *spadColorsToo;
    static unsigned long *pIndex;
    Colormap cmap;
    int num_colors;
    int i, ts;

    if (firstTime) {

        /* initialization */

        cmap = DefaultColormap(dsply, scrn);    /* what are other cmaps?? */
        pIndex = (unsigned long *)
            saymem("makePermVector", Colorcells, sizeof(unsigned long));

        /* get spadColors table */

        if ((num_colors =
            makeColors(dsply, scrn, &cmap, &spadColorsToo, &ts)) < 0) {
            printf("num_colors < 0!!\n");

```

```

        exit(-1);
    }

    /* initialize unused slots in permutation vector */

    for (i = 0; i < spadColorsToo[0]; i++)
        pIndex[i] = 0;
    for (i = num_colors; i < Colorcells; i++)
        pIndex[i] = 0;

    /* make permutation vector */

    for (i = 0; i < num_colors; i++)
        pIndex[spadColorsToo[i]] = i;

    firstTime = no;
}

*permIndex = pIndex;
return (Colorcells);
}

#endif

/*****
 * int makeNewColorMap(dsply,colorMap,smoothHue)      *
 *                                                    *
 * This routine tries to allocate an adequate color  *
 * map to be used by the AXIOM smooth shading       *
 * application that is to be run under X Windows.   *
 * The colors are allocated from available space in  *
 * the colorMap and returned in the array pixels.    *
 * The size of the array is determined by smoothConst *
 * which is the number of shades desired. The colors *
 * returned are variations in lightness of the hue   *
 * smoothHue indicated on the control panel Colormap. *
 *                                                    *
 * If smoothConst colors can be allocated the value  *
 * 1 is returned, otherwise returns 0                *
 *                                                    *
 *****/

int
makeNewColorMap(Display *dsply, Colormap colorMap, int smoothHue)

{

    int count, i;
    float lightness;

```

```

    RGB rgb;
    XColor xcolor;
    HLS hls;

    count = 0;
    /* i = 0 .. smoothConst */
    for (i = 0; i < (smoothConst + 1); i++) {
        /* lightnes = 0.0 .. 1.0 */
        lightness = (float) (i) / (float) (smoothConst);
        hls.h = (float) smoothHue;
        hls.l = lightness;
        hls.s = saturation;
        rgb = HLStoRGB(hls);

        xcolor.red    = rgb.r * ((1<<16)-1);
        xcolor.green  = rgb.g * ((1<<16)-1);
        xcolor.blue   = rgb.b * ((1<<16)-1);
        xcolor.flags = DoRed | DoGreen | DoBlue;
        /*
        fprintf(stderr,"%f\t%f\t%f\n",rgb.r,rgb.g,rgb.b);
        fprintf(stderr,"%d\t%d\t%d\n",
            xcolor.red,xcolor.green,xcolor.blue);
        */
        if (XAllocColor(dsply, colorMap, &xcolor)) {
            pixels[count] = xcolor.pixel;
            count++;
        }
    }
    /* count says how many succeeded */
    if (count != (smoothConst+1) ) {

        /* we have failed to get all of them - free the ones we got */

        FreePixels(dsply,colorMap,count);
        return (0);
    }
    return (1);
}

/*****
* unsigned long XPixelColor(num)
*
*
* XPixelColor is a straight forward function that
* merely returns the XColor value desired within
* the pixels array.  For smooth shading, given an
* intensity from 0..1, scaling by the number of
* values in the array will return the location in
* pixels[] of the desired color for that intensity.
*
*****/

```

```

*
*
*****/

unsigned long
XPixelColor(int num)

{
    if (num < 0)
        num = 0;
    return (pixels[num]);
}

/*****
* FreePixels(dsply,colorMap,num)
*
* FreePixels is a call to XFreeColors which frees
* previously allocated colors for the indicated
* colorMap. If it cannot free the number of colors
* given by num a BadAccess error will crash the
* viewport process. This should ONLY be used if
* it can be guaranteed that there will be num colors
* to free in colorMap. return 0 == success
*
*****/

void
FreePixels(Display *dsply, Colormap colorMap, int num)
{
    XFreeColors(dsply, colorMap, pixels, num, 0);
}

/*****
* int AllocCells(dsply,colorMap,smoothHue)
*
* Use either makeNewColormap() OR AllocCells().
* This routine tries to allocate an adequate color
* map to be used by the AXIOM smooth shading
* application that is to be run under X Windows.
* The colors are allocated from available space in
* the colorMap and returned in the array pixels.
* The size of the array is determined by smoothConst
* which is the number of shades desired. The colors
* returned are variations in lightness of the hue
* smoothHue indicated on the control panel Colormap.
*
*****/

```

```

* It is different from makeNewColormap() in that      *
* the cells are read/write, and if it cannot alloc   *
* all the colors desired it doesn't allocate any.    *
*                                                     *
*****/

```

```

int
AllocCells(Display *dsply, Colormap colorMap, int smoothHue)

```

This routine used to have the following code block. However this code block makes no sense. To see why you need to know that an XColor object looks like:

```

/*
 * Data structure used by color operations
 */
typedef struct {
    unsigned long pixel;
    unsigned short red, green, blue;
    char flags; /* do_red, do_green, do_blue */
    char pad;
} XColor;

```

This routine used to set the values of all of the elements of the XColor struct except [[pixel]]. This is usually done to specify a desired color in RGB values. To try to get a pixel value close to that color you call XAllocColor. This routine sets up the desired color values but it never asks for the pixel (which is really an index into the colormap of the nearest color) value that corresponds to the desired color. In fact it uses pixel without ever giving it a value. I've rewritten that code.

```

{
    unsigned long plane_masks[1];
    int i, count;
    float lightness;
    RGB rgb;
    XColor xcolor;
    HLS hls;

    count = 0;
    for (i = 0; i < (smoothConst + 1); i++) {
        lightness = (float) (i) / (float) (smoothConst);
        hls.h = (float) smoothHue;
        hls.l = lightness;
        hls.s = saturation;
        rgb = HLStoRGB(hls);
        xcolor.red    = rgb.r * ((1<<16)-1);
        xcolor.green  = rgb.g * ((1<<16)-1);
        xcolor.blue   = rgb.b * ((1<<16)-1);
    }
}

```

```

        xcolor.flags = DoRed | DoGreen | DoBlue;
        /*
        fprintf(stderr,"%f\t%f\t%f\n",rgb.r,rgb.g,rgb.b);
        fprintf(stderr,"%d\t%d\t%d\n",
                xcolor.red,xcolor.green,xcolor.blue);
        */
        pixels[i] = xcolor.pixel;
        count++;
    }
    if (XAllocColorCells(dsply, colorMap, False,
                        plane_masks, 0, pixels, smoothConst + 1)) {
        return (smoothConst + 1);
    }
    else {
        return (0);
    }
}

```

```

        ____ * ____

{
    unsigned long plane_masks[1];
    int i, count;
    float lightness;
    RGB rgb;
    XColor xcolor;
    HLS hls;

    count = 0;
    for (i = 0; i < (smoothConst + 1); i++) {
        lightness = (float) (i) / (float) (smoothConst);
        hls.h = (float) smoothHue;
        hls.l = lightness;
        hls.s = saturation;
        rgb = HLStoRGB(hls);
        xcolor.red    = rgb.r * ((1<<16)-1);
        xcolor.green  = rgb.g * ((1<<16)-1);
        xcolor.blue   = rgb.b * ((1<<16)-1);
        xcolor.flags = DoRed | DoGreen | DoBlue;
        /*
        fprintf(stderr,"%f\t%f\t%f\n",rgb.r,rgb.g,rgb.b);
        fprintf(stderr,"%d\t%d\t%d\n"
                ,xcolor.red,xcolor.green,xcolor.blue);
        */
    }
}

```

Here I've modified the code to actually as for the pixel (colormap index) that most closely matches our requested RGB values.

\_\_\_\_ \* \_\_\_\_

```

        if (XAllocColor(dsply, colorMap, &xcolor)) {
            pixels[count] = xcolor.pixel;
            count++;
        }
    }
    /* count says how many succeeded */
    if (count != (smoothConst+1) ) {
        /* we have failed to get all of them - free the ones we got */
        FreePixels(dsply,colorMap,count);
        return (0);
    }
    if (XAllocColorCells(dsply, colorMap, False,
                        plane_masks, 0, pixels, smoothConst + 1)) {
        return (smoothConst + 1);
    }
    else {
        return (0);
    }
}

```

---

## References

- [1] nothing