Paper 234-28

Working with RGB and HLS Color Coding Systems in SAS® Software

Perry Watts, Independent Consultant, Elkins Park, PA

Abstract

Full-color displays have become commonplace with the advent of the Web. Typically they are rendered from desktop computers equipped with 16-bit color palettes capable of generating 65K unique colors. Thus a chart or scale is needed for making informed color selections for graphics applications. In SAS, the programmer has access to both the RGB (red/green/blue) and HLS (hue/light/saturation) coding systems. However, different algorithms are required for building charts and scales in the two systems.

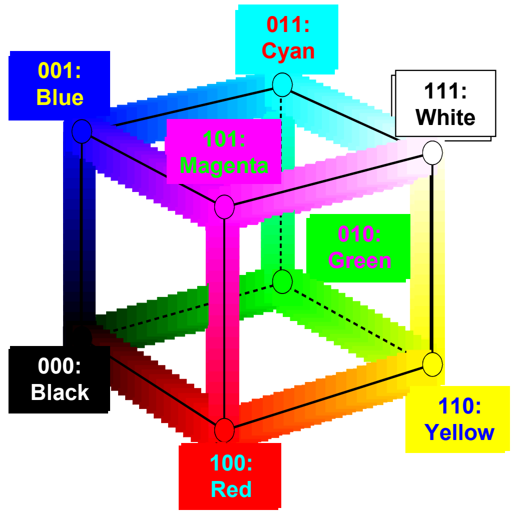
Before the technical aspects of chart and scale construction are presented in the paper, color spaces are defined to show how the full gamut of codes is mapped in both coding systems. For RGB, codes map to a cube, and a double-ended cone accommodates the Tektronix HLS system. With the color spaces as a foundation, instructions are then provided for building color charts with ODS and scales from PROC GPLOT. Both the charts and scales were introduced in the SUGI 27 paper *Using ODS and the Macro Facility to Construct Color Charts and Scales for SAS® Software Applications*.

RGB: The Color System for Screen Output

The RGB (red/green/blue) Color Space

Originally defined for color TV, the RGB system is the only system that is Web compatible (Weinman, 86). This means RGB codes are used in GIF, JPG and HTML files. In Figure 1, the color space for this screen-based system is depicted as a cube. While there are eight vertices in the cube, all are defined by combining the primary colors: red, green, and blue. Cyan(0,1,1), for example, contains a mixture of green and blue but no red, and red with (1,0,0) contains neither green nor blue - only red.

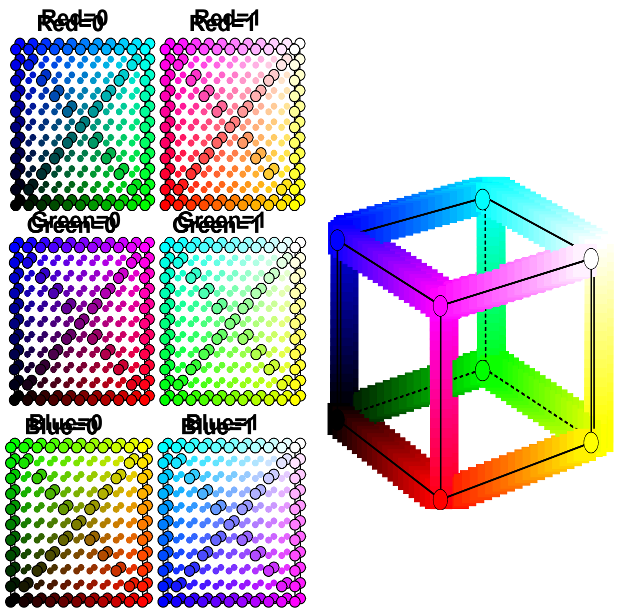
Figure 1. RGB color space depicted as a cube. Coordinate ranges 0-1 directly map to color ranges 0-255.



The cube in Figure 1 is constructed from an application of the G3D procedure that uses an ANNOTATE data set for coloring edges and vertices. The vertices are "dots" and the edges are marker font "U's" expressed as small parallelograms connected to form a "line" of color. Only one parameter is altered to traverse the cube. To go from red(1,0,0) to yellow(1,1,0), for example, green is added in incremental amounts along the base line. There is also a one-to-one correspondence between the X-Y-Z coordinates(0-1) and the RGB code values(0-255). Since SAS uses hexadecimal notation, yellow plotted at (1,1,0) translates to (255,255,0) or CXFFFF00 for actual coloring.

The faces of the cube can be represented in 2-D space with an application of the GPLOT procedure. In Figure 2, one color coordinate is held constant and the other two are varied systematically to color all six sides of the RGB cube.

Figure 2. All faces of the RGB cube are defined by setting the value of the third component to 0 or 1.



Look at the top face in Figure 1. It corresponds to the second face rotated 90° in row 3 of Figure 2. Note that the value for the blue or third component at all four vertices is set to 1. The title "Blue=1" in Figure 2 should now make sense.

To plot the diagonals in the RGB cube, all three components need to be systematically varied between two opposing vertices. The gray "line" connecting the black (0,0,0) and white (1,1,1) vertices in Figure 3 is created with an application of the LINES macro below:

```
%macro Lines(SR,SG,SB,TR,TG,TB,n,ds); ❶
 data &ds;
 length function font value color $8 text $12;
 retain function 'label' text 'U'
        style 'marker';
 retain xsys ysys zsys '2';
 %do i=0 %to %eval(&n-1);
  %let r=%sysevalf(&sr+(&tr.- &sr.)*&i/&n.); ❷
```

```
    %let r=%sysfunc(round(&r.));
    %let g=%sysevalf(&sg+(&tg-&sg.)*&i/&n.);
    %let g=%sysfunc(round(&g.));
    %let b=%sysevalf(&sb+(&tb-&sb.)*&i/&n.);
    %let b=%sysfunc(round(&b.));
     r=&r; g=&g; b=&b;
     x=r/255; y=g/255; z=b/255;
     color="%rgbhex(&r,&g,&b)"; output; ❸
    %end;
    stop;
  run;
%mend lines;
%Lines(0,0,0,255,255,255,51,anno1); ❶
```

❶ A macro is used to draw the four diagonal RGB color lines in Figure 3. Values of 0 or 255 are assigned to the Source and Target endpoint codes. 255/5 or 51 points are plotted per line.

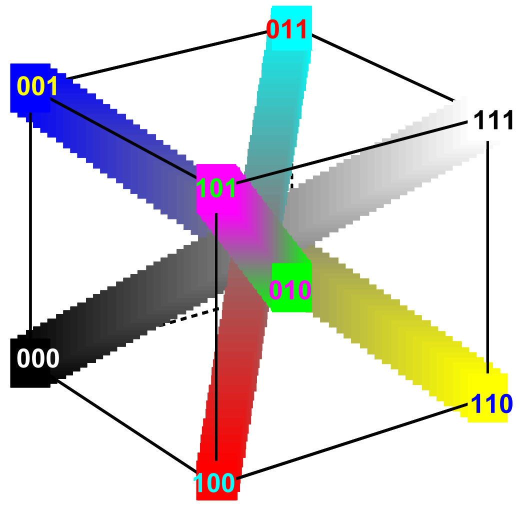
❷ Correct hues for intermediate points in the line are calculated with an application of component-wise linear interpolation. The interpolation formula without macro variables and functions is:

$$r = sr + (tr - sr) \times {}^{i}\!/_{n}$$

where ***r*** is the interpolated value for the red component, ***sr*** and ***tr*** represent the source and destination component values, ***i*** is the iteration number, and ***n*** represents the number of iterations in the loop.

❸ RGBHEX is a user-defined macro function that converts the decimal values for the RGB code components to a hexadecimal string preceded by the characters 'CX'.

Figure 3. Complement colors result by joining opposing vertices in RGB space. The labels are complements too!



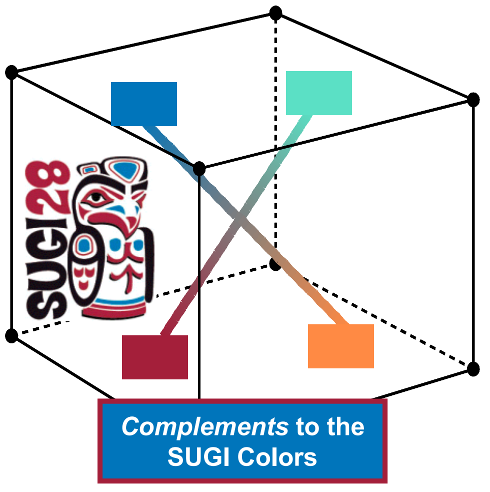
The complement colors depicted in Figure 3 are not restricted to the main diagonals in RGB space. In fact all colors have complements defined as:

Color' = (r',g',b') = ((255-r),(255-g),(255-b))

Figure 4 shows complement colors for the non-diagonal SUGI 28 colors.

Being able to distinguish between complements is a function of their distance in RGB space. That is why their use for labeling is only partially successful in Figure 8 whereas the black and white labels are consistently more visible in Figure 13.

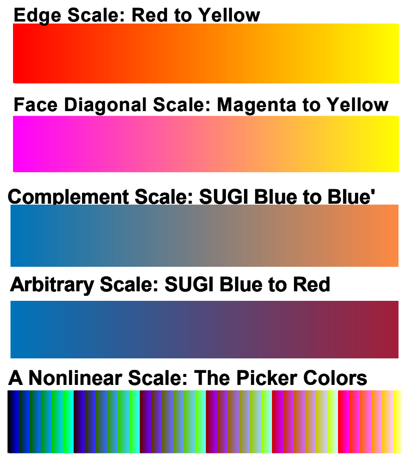
Figure 4. Color complements are not restricted to the vertices of the RGB cube.



RGB ColorScales

A color scale with gradually changing hues can be generated by connecting two end points in the RGB color space with the same component-wise linear interpolation technique used on the main diagonals. The first four scales in Figure 5 are interpolated. The fifth scale for the browser-safe "picker" colors discussed on page 3 is not. Instead, each of the 216 RGB codes in the picker scale is treated as a separate point and simply plotted in its presented order. The lack of linearity is readily apparent with the visible bands of color. What is especially remarkable though is a periodicity that defied visual inspection of any browser-safe chart shown on the Web. A pattern repeats itself every 36 colors.

Figure 5. RGB Color Scales



RGB Color Charts

1)Constructing a SAS color chart[*]

The SAS program that processes the downloaded text file of the 280 predefined colors from V8 *SAS OnlineDoc* uses

[*] Prefabricated SAS color charts are also available. See the *Annotated Web Listings* section in this paper.

the macro SELECTRGB with customized "traffic lighting" in ODS to add color blocks to the output from PROC Report. The annotated output in HTML format is called a "color chart".

SELECTRGB is listed in full below:

```
%macro selectRGB;
  select (_C1_);
  %do i = 1 %to &NColors;
     when (&i.) CALL DEFINE(_COL_,"STYLE",
       "STYLE=[ BACKGROUND=&&RgbN&i.]"); ❶
  %end;
  otherwise;
  end;
%mend selectRGB;
```

❶ Macro variable `&&RGBN&i` is created in a Data _NULL_ step that processes all 280 colors. It contains an RGB code for a particular color.

Macro variables must be used for background color assignments, because the STYLE option in the ODS DEFINE statement calls for unquoted constant text, not a SAS variable. Here is how the color chart in Figure 6 is derived from PROC REPORT in ODS:

```
ods listing close;
ods html
  body='c:\SUGI28\SASColorChrtUnique.html'
  style=SUGI28Colors; ❶
  proc report data=X nowindows headline
       headskip split='*' ls=100 ps=40;
    columns CNum cName RGB HLS; ❷
    define  cNum    / display width=2
                      format=missf. 'C'; ❸
    define cName    / display width=10
                      'SAS*Color Name';
    define RGB      / display width=8 'RGB';
    define HLS      / display width=8 'HLS';
    compute CNum;
      %selectRGB;
    endcomp;
  run;
ods html close;
ods listing;
```

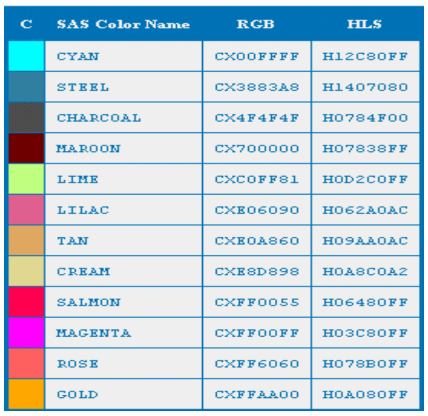
❶ The SUGI28COLORS style assigns the dark turquoise blue from Figure 4 to the table foreground region. The background is a very light shade of gray.

❷ The variable CNUM is column #1, the argument of the SELECT statement in the SELECTRGB macro.

❸ CNUM is set to missing with the MISSF format so that only the background color is displayed.

❹ The SELECTRGB macro is invoked from a compute block.

Figure 6. A color chart of SAS defined unique colors.



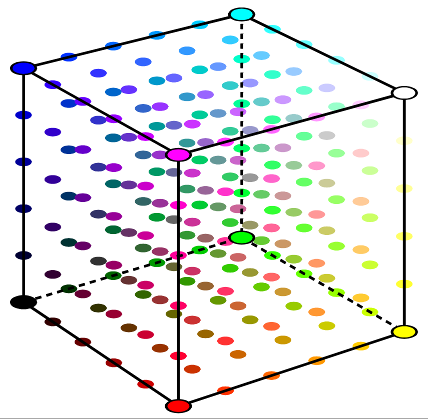
C	SAS Color Name	RGB	HLS
🟦	CYAN	CX00FFFF	H12C80FF
🟦	STEEL	CX3883A8	H1407080
⬛	CHARCOAL	CX4F4F4F	H0784F00
🟫	MAROON	CX700000	H07838FF
🟩	LIME	CXC0FF81	H0D2C0FF
🟪	LILAC	CXE06090	H062A0AC
🟧	TAN	CXE0A860	H09AA0AC
🟨	CREAM	CXE8D898	H0A8C0A2
🟥	SALMON	CXFF0055	H06480FF
🟪	MAGENTA	CXFF00FF	H03C80FF
🟥	ROSE	CXFF6060	H078B0FF
🟧	GOLD	CXFFAA00	H0A080FF

2) A Browser-Safe "picker" Color Chart

The browser-safe palette is a collection of 216 "picker" colors that are consistently rendered across all browsers, operating systems, and computer platforms (Weinman, p124). Web authors cited in the reference section generally feel that the importance of the browser-safe palette has diminished as a result of improvements in computer technology. However, LeRoy Bessler, noted SAS/GRAPH specialist, urges caution suggesting that all color selections for Web design be restricted to the palette so that the designer will be absolutely sure of what the viewer sees. Oddly enough, though, only the color black is browser-safe in the ODS HTML default-style template.

Regardless of the controversy, the browser-safe palette is important and merits its own color chart. Creating one, however, turns out to be a challenge, because the colors do not fall into neat groupings of gradually changing hues. While there is a pattern in the browser-safe scale in Figure 5, the color modulation can hardly be called "gradual". Recall too, that a color scale is linear containing at most a few lines that go through the RGB color space. In contrast, the points in Figure 7 are uniformly distributed in the RGB cube, and there is a very large number of ways to connect them.

Figure 7. Distribution of browser-safe colors in the RGB color space.



Generally speaking, the same steps used for the predefined SAS colors are needed for creating a browser-safe color chart:

1) Create a SAS data set with identifying information - color name, code, etc.
2) Write the SELECTRGB macro so that it operates on the proper columns of data.
3) Run PROC Report from inside ODS to generate a color chart enhanced with color annotations.

Since browser-safe colors are not named, placing multiple colors on a single row of output results in a more compact color chart. The color samples can also be enlarged and labeled with 8-character RGB codes that are colored in contrasting hues for greater visibility.

Source Code Summary:

There is no need for downloading data. Instead the browser-safe colors can easily be calculated, since they consist of all six-digit combinations for the three code components (6^3 = 216). Below is the SAS source code that creates the control-in data set used for generating the

PICKERF format and the SELECTRGB macro variables used in PROC Report:

```
  data cntlin(keep=fmtname type start label);
   retain fmtname 'PickerF' type 'N';
   array hexnums {6}$
        ('00','33','66','99','CC','FF');
   array decnums{6} (0,51,102,153,204,255);❶
   length HexPVal HexPCVal $8;❷
   substr(HexPVal,1,2)='CX';
   substr(HexPCVal,1,2)='CX';
   do i=1 to dim(hexnums);
    do j=1 to dim(hexnums);
     do k=1 to dim(hexnums);
      n+1;
      substr(HexPVal,3,2)=hexnums[i];
      substr(HexPVal,5,2)=hexnums[j];
      substr(HexPVal,7,2)=hexnums[k];
      cr=255-decnums[i]; ❶
      cg=255-decnums[j];
      cb=255-decnums[k];
      if cr EQ cg EQ cb then❸
       do;
        if cr gt 128 then
         do;
          cr=255; cg=255; cb=255; *white txt;
         end;
        else
         do;
          cr=0; cg=0; cb=0;        *black txt;
         end;
       end;
      substr(HexPCVal,3,2)=put(cr,hex2.);
      substr(HexPCVal,5,2)=put(cg,hex2.);
      substr(HexPCVal,7,2)=put(cb,hex2.);
      start=n; label=HexPVal;
      call symput('CN'||left(put(n,3.)),
         left(trim(HexPVal)||trim(HexPCVal)));❹
      output;
     end;
    end;
   end;
   stop;
  run;
```

❶ The points are distributed uniformly in RGB space, because all the values are multiples of 51.

❷ P and PC stand for "Picker" and "Picker Complement". Complements are used to label the RGB codes. Note that some of the labels in Figure 8 aren't too visible.

❸ When complement components are equal in value the resulting color is gray and possibly not visible. The complement, therefore, is recast in black or white depending upon the value of the original Picker color (also gray). None of the colors in Figure 8, however, are gray.

❹ Macro variables CN1-CN216 contain values for the browser-safe colors and their complements.

A matrix data set, not shown, simply rearranges the linear input into a desired row, column configuration. For example, the vector (1,2,3,4,5,6) becomes { (1,2,3), (4,5,6) } with two rows and three columns. The SELECTRGB macro increases in complexity to accommodate the matrix just created:

```
%macro selectrgb;
%do j=1 %to &maxcol;
 compute CNum&j; ❶
 select (CNum&j);
 %do i = 1 %to 216;
  when (&i.) do;
   CALL DEFINE(_COL_,"STYLE", ❶
    "STYLE=[ BACKGROUND= %substr(&&CN&i.,1,8) ❷
             FOREGROUND= %substr(&&CN&i.,9) ❷
             FONT_FACE='Arial' FONT_SIZE=2
                       FONT_WEIGHT=Bold]");
  end;
 %end;
  otherwise;
 end;
 endcomp;
%end;
%mend selectrgb;
```

❶ While the SELECTRGB macro is still invoked from PROC REPORT, the COMPUTE statement is placed inside the macro so that the _COL_ clause in CALL DEFINE will work as intended.

❷ %SUBSTR is used to parse the macro variable into foreground and background colors.

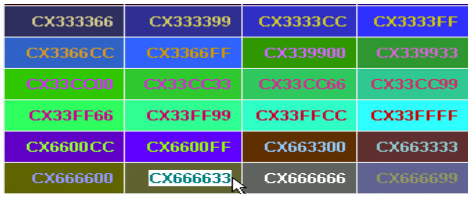
Fortunately the code for PROC Report is very simple by comparison, but it too must be enclosed in a macro. It is included without comment below:

```
  ods listing close;
  ods html body='c:\N02Col\HTM\ColPick.html';
   /*Get a Matrix Report*/
   %macro Mreport;
    proc report data=matrix nowindows
         headline headskip ls=80 ps=75;
    columns cnum1-cnum&maxcol;
    %do i=1 %to &maxcol;
      define cnum&i / display
             width=8 format=PickerF. " ";
    %end;
    %selectRGB;
    run;
   %mend Mreport;
   %Mreport;
  ods html close;
  ods listing;
```

Usage:

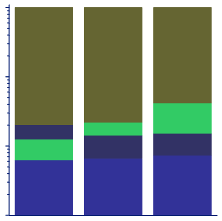
A color chart configured like the one for the browser-safe colors is a useful tool for graphics design. Besides being able to visually inspect actual colors prior to selection, it is also possible to copy and paste a desired RGB code from an HTML file directly into a SAS program. In Figure 8, the selection process is rendered pictorially:

Figure 8. Selecting colors for display from a Browser-Safe Color Chart.



CX333366	CX333399	CX3333CC	CX3333FF
CX3366CC	CX3366FF	CX339900	CX339933
CX33CC00	CX33CC33	CX33CC66	CX33CC99
CX33FF66	CX33FF99	CX33FFCC	CX33FFFF
CX6600CC	CX6600FF	CX663300	CX663333
CX666600	CX666633	CX666666	CX666699

```
data adjusted;
   set ssummary end=last;
   select (srchtype);
     when ('B')  color="CX333366";
     when ('BE') color="CX333399";
     when ('SE') color="CX33CC66";
     when ('S')  color="CX666633";
   end;
run;
```



HLS: The System for Color Scales

An alternative color-coding system supported by SAS software is the Tektronix HLS system for *hue*, *lightness*, and *saturation*. These terms are not as intuitive as *red, green,* and *blue*, so definitions from Web Source #1 are provided below:

Hue:
the attribute of color by means of which a color is perceived to be red, yellow, green, blue, purple, etc. Pure white, black, and grays possess no hue (Range 0 to 360°)

Lightness:
(1) the attribute of color perception by which a non-self-luminous body is judged to reflect more or less light. (2) the attribute by which a perceived color is judged to be equivalent to one of a series of grays ranging from black to white (Range 0 to 255).

Saturation:
the attribute of color perception that expresses the degree of departure from the gray of the same lightness. All grays have zero saturation. Commonly used as a synonym for chroma especially in graphic arts (Range 0 to 255).

Lightness and saturation may still be somewhat confusing. An alternative definition for lightness is the amount of white contained within a color whereas the amount of gray defines a color's saturation. Fully saturated colors contain no gray. All grays including black and white are not saturated.

The HLS Color Space

The HLS color space is depicted as a double-ended cone shown in Figure 9. Figures 10 to 12 show how this color space lends itself so easily to color scale construction.

Figure 9. The Tektronix HLS Color System maps as a double-ended cone. The diagram on the right is from *SAS Online Doc*.

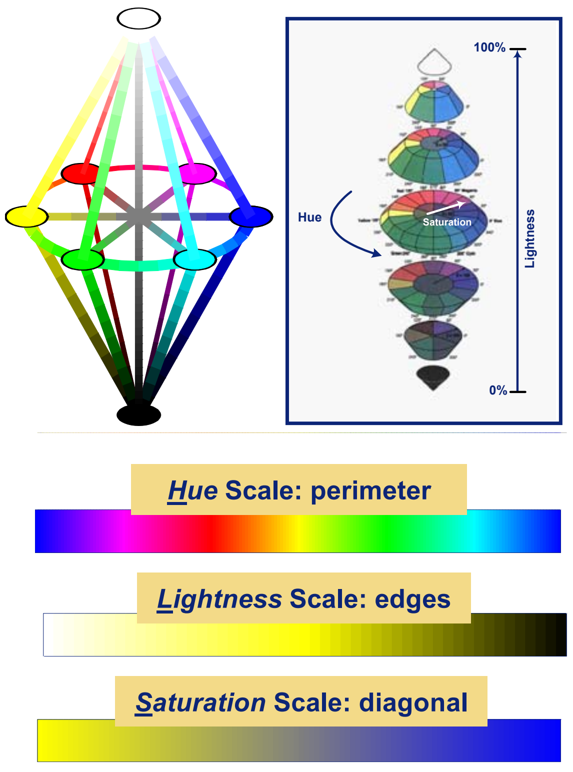


Figure 10. Light and Saturation Space mappings.

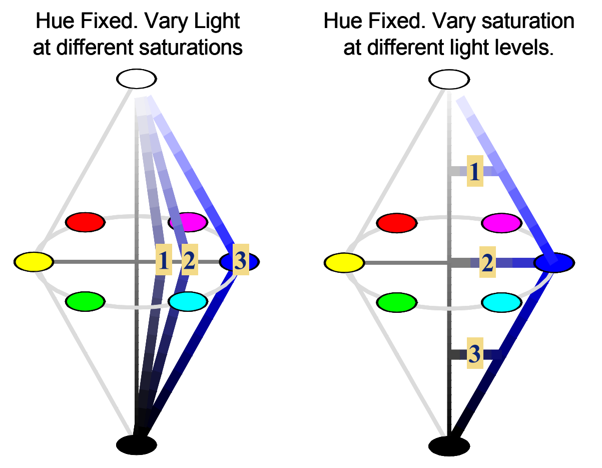

Hue Fixed. Vary Light at different saturations
Hue Fixed. Vary saturation at different light levels.
1
2
3
1
2
3

Figure 11. Hue and Saturation Space mappings.

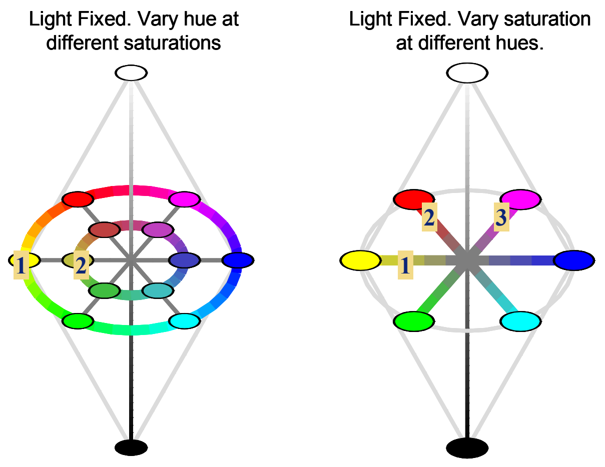
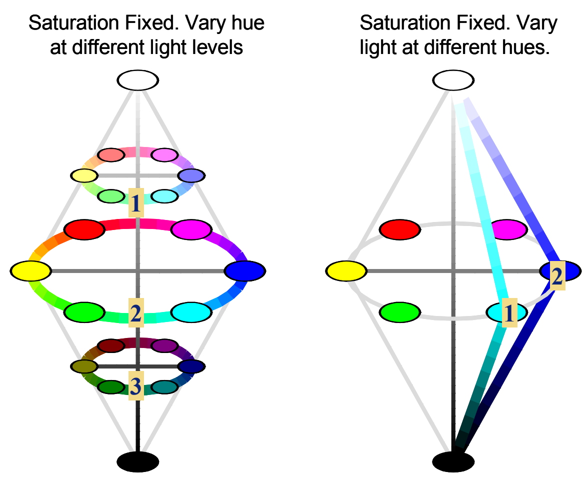

Light Fixed. Vary hue at different saturations
Light Fixed. Vary saturation at different hues.
1
2
1
2
3

Figure 12. Hue and Light Space mappings.


Saturation Fixed. Vary hue at different light levels
Saturation Fixed. Vary light at different hues.
1
2
3
1
2

Color scales are easy to construct in the HLS coding system. Unlike the RGB scales where all components are manipulated simultaneously with component-wise interpolation, two components can be held constant while the value of the third is altered incrementally in a looping structure. Below is SAS code from an ANNOTATE data set for the hue scale pictured in Figures 9 and 11:

```
%let S=255; %let L=128;❶
%do H=0 %to 355 %by 5;
  ylo=10; yhi=20; xlft=&H; xrt=xlft+5;
  %bar(xlft,ylo,xrt,yhigh,
      %hlshex(&H,&L,&S),0,solid); ❷
  output;
%end;
```

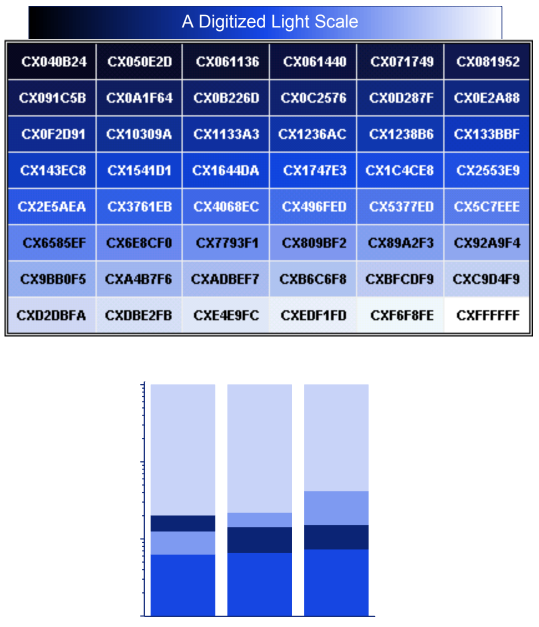
❶ Values for saturation and light are held constant. The brightest possible hue scale is being constructed.

❷The ANNOTATE %BAR macro simplifies coding but only takes constant values as parameters. The conversion macro HLSHEX is listed in Appendix A. It converts decimal loop values to hexadecimals SAS understands.

A Color Chart from a Lightness Scale

A color chart derived from a lightness scale can be very useful in those situations where related, but clearly distinguishable colors are needed for a graphic. In Figure 13 a blue lightness color scale is displayed along with its corresponding color chart. A bar chart comparing format resolution times with selections from the color chart is also shown.

Figure 13. 48 evenly spaced values from 20 to 255 are used for constructing the digitized lightness scale.



A Digitized Light Scale

CX040B24	CX050E2D	CX061136	CX061440	CX071749	CX081952
CX091C5B	CX0A1F64	CX0B226D	CX0C2576	CX0D287F	CX0E2A88
CX0F2D91	CX10309A	CX1133A3	CX1236AC	CX1238B6	CX133BBF
CX143EC8	CX1541D1	CX1644DA	CX1747E3	CX1C4CE8	CX2553E9
CX2E5AEA	CX3761EB	CX4068EC	CX496FED	CX5377ED	CX5C7EEE
CX6585EF	CX6E8CF0	CX7793F1	CX809BF2	CX89A2F3	CX92A9F4
CX9BB0F5	CXA4B7F6	CXADBEF7	CXB6C6F8	CXBFCDF9	CXC9D4F9
CXD2DBFA	CXDBE2FB	CXE4E9FC	CXEDF1FD	CXF6F8FE	CXFFFFFF

The same algorithm used for generating the browser-safe color chart in HTML is used for the digitized color chart in Figure 13. This means that the discrete values in the light scale have to be translated into RGB. The translation macro, HLSTORGB listed in Appendix A performs the job. This macro is adapted from psuedo-code in Foley and Van Dam's book, *Fundamentals of Interactive Computer Graphics*. With RGB values supplied by HLSTORGB the input data set DIGITIZE can be processed by the SELECTRGB macro to assign colors to cells in an HTML file.

```
%let xh=346; %let xs=208;
%macro convert;
 data digitize(keep=cnum rgbhex);
  length rgbhex $8;
  %do i=20 %to 255 %by 5;
   cnum+1;
   rgbhex="%RGBHex(%HLStoRGB(&xh,&i,&xs))"; ❶
   output;
  %end;
 run;
%mend convert;
%convert;
```

❶ This is an example of a nested user-defined macro function call. The inner macro, HLSTORGB, converts a decimal HLS triple to an RGB triple, and then RGBHEX returns a hexadecimal value for the RGB decimal. For a complete discussion of user-defined macro functions see Art Carpenter's SUGI 27 paper cited in the reference section.

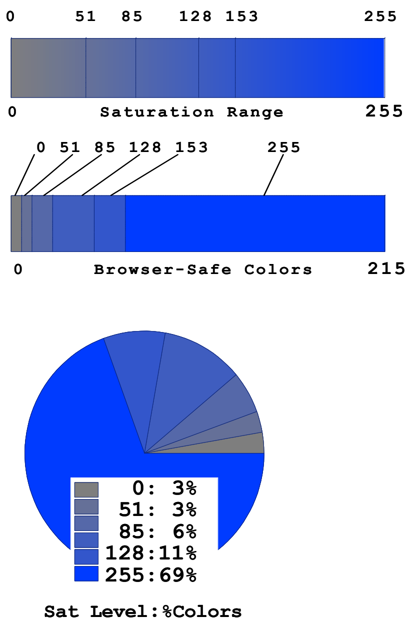
Saturation Scale Legends support a Pie Chart

Lynda Weinman, web author cited in the reference section, voices objections to the set of hues selected for the browser-safe palette:

> The browser-safe palette was developed by programmers with no design sense, I assure you. That's because a designer would have never picked these colors. Mostly, the palette contains far less light and dark colors than I wish it did, and is heavy on highly saturated colors and low on muted, tinted or toned colors.

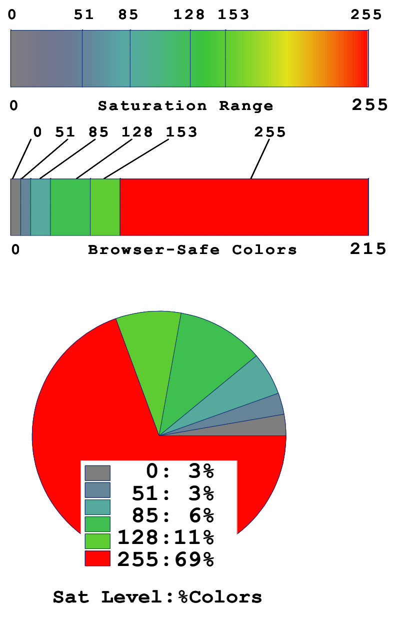
A macro function RGBTOSAT that returns an HLS Saturation value for an RGB code is used to test Weinman's assertion about the saturation levels for browser-safe colors. This macro can be found with the other Foley and Van Dam conversion macros in Appendix A. It is used here to calculate saturation levels for all the browser-safe colors. The output summarized graphically in Figures 14 and 15 corroborates Weinman's assertion.

Figure 14. Distribution of saturation levels for browser safe colors is shown with two saturation scales that define the context for the pie chart.



The gradual increase in saturation may not be so visible in the first color scale in Figure 14. Figure 15 below provides an alternative where both saturation and hue are changed simultaneously. Hue starts at 15 (blue) and an arc is traced clockwise to 120 (red). 255 unique hue values are contained within the arc - one for each value of saturation.

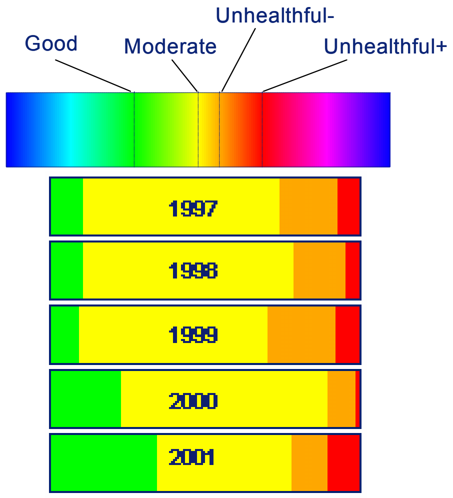
Figure 15. Hue is changed along with saturation to substantiate the claim that the browser-safe colors are highly saturated.



Hue Scale Displays for Ordinal Data

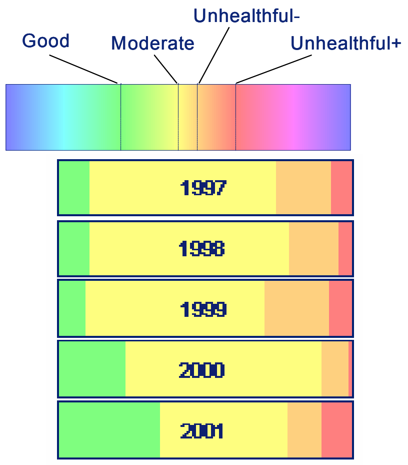
To round out the picture about HLS color scales, ozone bars are shown stacked to summarize levels of the pollutant in the Philadelphia region during the summer months from 1997 to 2001. Again a fully defined hue scale serves as a legend to provide a context for the ozone levels. Note that the scale is actually reversed. The value for green (240) is greater than red (120).

Figure 16. Summer Ozone Levels for a five- year period in the Philadelphia region.



Saturation is not the only parameter that can be modified to produce softer, muted colors. The color scale in Figure 17 is identical to the one shown in Figure 16 except that lightness has increased from 128 to 192. The result is a more muted display.

Figure 17. Same ozone levels depicted with lighter colors.



While not analyzed graphically, lightness scores for the browser-safe colors have the following distribution:

```
                                  Cum        Cum
 lite     Freq       Pct         Freq        Pct
 ------------------------------------------------
    0        1       0.46           1       0.46
   26        6       2.78           7       3.24
   51       13       6.02          20       9.26
   77       24      11.11          44      20.37
  102       37      17.13          81      37.50
  128       54      25.00         135      62.50
  153       37      17.13         172      79.63
  179       24      11.11         196      90.74
  204       13       6.02         209      96.76
  230        6       2.78         215      99.54
  255        1       0.46         216      100.0
```

Only 20 out of the 216 colors are lighter than the ones shown in Figure 17, and 25 percent of the browser-safe colors have a lightness value of 128: the brightest setting in the HLS system.

Summary and Conclusions

RGB and HLS color-coding systems are reviewed in depth. Color spaces are depicted graphically, and an effort is made to show how distinctive color charts and scales are developed in both systems. Insight about the browser-safe palette is gained by examining its rendition in the RGB and HLS color spaces, and graphics displays show how colors can be softened and tinted by changing their saturation and lightness levels.

Annotated Web Citations

1) http://www.hypersolutions.org/pages/colorDef.html
Provides *hue, lightness*, and *saturation* definitions used in the paper. Additional terms such as *color, tint* and *dithering* are also defined.

2) http://home.att.net/~B-P.TRUSCIO/COLOR.htm
copyright by R.Truscio. Excellent graphics for additive coloring (RGB) and subtractive coloring (CMY**).**

3) http://www.colormatters.com/colortheory.html
Color wheel displays, primary, secondary, and tertiary colors are featured.

4) http://hort.ifas.ufl.edu/TEACH/floral/color.htm
Additional color wheel displays are shown.

5)http://www.webreview.com/1999/08_06/designers/08_06_99_3.shtml
Discusses how color works: RGB triplet, browser-safe colors etc.

6)http://wp.netscape.com/computing/webbuilding/studio/feature19981111-5.html
Good discussion of RGB and HLS color spaces. Good depiction of Hue and Saturation scales. Describes a tool for selecting a picker color.

7) http://www.lynda.com/hex.html
Lynda Weinman's discusses the web-safe color dilemma reversing the position she took in the book cited in the reference section. She says now the palette's importance has been outstripped by gains in hardware technology.

8) http://www.webtemplates.com/colors/index.html
A web-safe color picker is featured. When a particular color is selected, you are taken to a separate page to view its HEX code.

9)http://www.webreview.com/1999/08_06/designers/08_06_99_3.shtml
Discusses the hexadecimal numbering system along with the RGB coding system.

10) http://www.websitetips.com/color/
Excellent general "gateway" site containing links to color charts, articles and tutorials about color, articles and tools about color blindness, color psychology, color and design, browser-safe colors organized by hue and value, and the browser-safe controversy.

11) Websites featuring ready-made SAS color charts:
http://www.nesug.org link to publications and then to the title of this paper in the source code listing section.
http://www.devenezia.com/docs/SAS/sas-colors.html

References

Bessler, LeRoy. *Inform and Influence with Image and Data: Communication-effective Web Design for ODS, SAS®, and SAS/GRAPH®.* Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002.

Carpenter, Arthur. *Macro Functions: How to Make Them - How to Use Them*. Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002.

Foley, J.D. and A. Van Dam. *Fundamentals of Interactive Computer Graphics*. Reading, MA: Addison-Wesley Publishing Company, 1983.

Haworth, Lauren E., *Output Delivery System: The Basics*. Cary, NC: SAS Institute Inc., 2001.

SAS Institute Inc. *SAS Online Doc.* Cary, NC: SAS Institute Inc., 1999.

SAS Institute Inc. *SAS/GRAPH® Software: Reference, Version 8, Volume 1.* Cary, NC: SAS Institute Inc., 1999.

Watts, Perry. *Using ODS and the Macro Facility to Construct Color Charts and Scales for SAS® Software Applications.* Proceedings of the Twenty-Seventh SAS® User Group International Conference, Cary, NC: SAS Institute Inc., 2002.

Weinman, Lynda. *Designing Web Graphics.3: How to Prepare Images and Media for the Web, Third Edition*. Indianapolis, Indiana. New Riders Publishing, 1999.

Acknowledgements

The author would like to thank Samuel Litwin for his explanations of the RGB and HLS conversion algorithms in Foley and Van Dam's book. She is also grateful to Dr. Litwin for reviewing the manuscript.

Contact Information

Perry Watts, Independent Consultant
wattsp@dca.net

Appendix A: Conversion Macros

The programs below for the conversion macros are named so that they can be stored in an AUTOCALL LIBRARY. They can be downloaded from the NESUG-15 CD along with two additional programs that create a color chart and a color scale. Complete HTML listings of the color charts are also available on the CD.

```
/*  -----------------------------------------------------------------------
    Program   :  HLSHex.sas
    Purpose   :  Convert SAS Decimal HLS color to HEX which SAS understands
    Input     :  Three decimal numbers representing a color in terms of
                 its hue, lightness, and saturation.
    Output    :  a SAS HLS color Code Hhhhllss where hhh,ll,ss are
                 three hexadecimal numbers.
    ------------------------------------------------------------------- */
    %macro HLSHex(hhh,ll,ss);
      %sysfunc(compress(H%sysfunc(putn(&hhh,hex3.))
                          %sysfunc(putn(&ll,hex2.))
                          %sysfunc(putn(&ss,hex2.))))
    %mend HLSHex;


/*  -----------------------------------------------------------------------
    Program   :  HLStoRGB.sas
    Purpose   :  Convert an HLS decimal code to its RGB counterpart.
    Algorithm:   Foley, J.D. and A. Van Dam.
                 "Fundamentals of Interactive Computer Graphics".
                 Reading, MA: Addison-Wesley Publishing Company, 1983,
                 page 619.
    Input     :  Three decimal numbers for hue, light, and saturation
    Output    :  A character string representing three decimal RGB digits
                 separated by commas: rrr,ggg,bbb
    ------------------------------------------------------------------- */
    %macro HLStoRGB(h,l,s);
        %let hue=%sysevalf(&h - 120);
        %let light =%sysevalf(&l/255);
        %let sat=%sysevalf(&s/255);
        %if &light le 0.5 %then %let m2=%sysevalf(&light*(1+&sat.));
        %else %let m2=%sysevalf(&light+&sat.-&light*&sat);
        %let m1 = %sysevalf(2 * &light. - &m2.);
        %if &sat eq 0 %then %do;
          %let r=&l; %let g=&l; %let b=&l;
        %end;
        %else %do;
          %let rhue=%eval(&hue+120);
          %if &rhue gt 360 %then %let rhue= %eval(&rhue.-360);
          %if &rhue lt 0 %then %let rhue= %eval(&rhue.+360);
          %if &rhue lt 60 %then %let r = %sysevalf((&m1+(&m2-&m1)*&rhue./60)*255);
          %else %if &rhue lt 180 %then %let r=%sysevalf(255*&m2.);
          %else %if &rhue lt 240 %then %let r=%sysevalf((&m1+(&m2-&m1)*(240-&rhue)/60)*255);
          %else %let r=%sysevalf(255*&m1);

          %let ghue=&hue;
          %if &ghue gt 360 %then %let ghue= %eval(&ghue.-360);
          %if &ghue lt 0 %then %let ghue= %eval(&ghue.+360);
          %if &ghue lt 60 %then %let g = %sysevalf((&m1+(&m2-&m1)*&ghue./60)*255);
          %else %if &ghue lt 180 %then %let g=%sysevalf(255*&m2.);
          %else %if &ghue lt 240 %then %let g=%sysevalf((&m1+(&m2-&m1)*(240-&ghue)/60)*255);
          %else %let g=%sysevalf(255*&m1);

          %let bhue=%eval(&hue-120);
          %if &bhue gt 360 %then %let bhue= %eval(&bhue.-360);
          %if &bhue lt 0 %then %let bhue= %eval(&bhue.+360);
          %if &bhue lt 60 %then %let b = %sysevalf((&m1+(&m2-&m1)*&bhue./60)*255);
          %else %if &bhue lt 180 %then %let b=%sysevalf(255*&m2.);
          %else %if &bhue lt 240 %then %let b=%sysevalf((&m1+(&m2-&m1)*(240-&bhue)/60)*255);
          %else %let b=%sysevalf(255*&m1);
        %end;
        %let rr=%sysfunc(putn(&r,3.));
        %let gg=%sysfunc(putn(&g,3.));
        %let bb=%sysfunc(putn(&b,3.));
        &rr.,&gg.,&bb.

     %mend HLStoRGB;
```

```
/*  ------------------------------------------------------------------------
    Program   :  RGBHex.sas
    Purpose   :  Convert Decimal RGB color to HEX which SAS understands
    Input     :  Three decimal RGB codes.
    Output    :  a SAS RGB color Code CXrrggbb where rrggbb are
                 three hexadecimal numbers.
    ---------------------------------------------------------------------- */
    %macro RGBHex(rr,gg,bb);
     %sysfunc(compress(CX%sysfunc(putn(&rr,hex2.))
                         %sysfunc(putn(&gg,hex2.))
                         %sysfunc(putn(&bb,hex2.))))
    %mend RGBHex;

  /*  ------------------------------------------------------------------------
    Program   :  RGBtoHue.sas
    Purpose   :  Calculate hue in HLS from an RGB code.
    Algorithm:  Foley, J.D. and A. Van Dam.
                 "Fundamentals of Interactive Computer Graphics".
                 Reading, MA: Addison-Wesley Publishing Company, 1983, p618.
    Input     :  Three decimals for red, green and blue.
    Output    :  One decimal number for hue
    ---------------------------------------------------------------------- */
   %macro RGBtoHue(r,g,b);
       %let red = %sysevalf(&r/255);
       %let green =%sysevalf(&g/255);
       %let blue=%sysevalf(&b/255);
       %let mmax=%sysfunc(max(&red,&green,&blue));
       %let mmin=%sysfunc(min(&red,&green,&blue));
       %if &mmax eq &mmin %then %let hue=0;
       %else %do;
         %let rc=%sysevalf((&mmax-&red)/(&mmax-&mmin));
         %let gc=%sysevalf((&mmax-&green)/(&mmax-&mmin));
         %let bc=%sysevalf((&mmax-&blue)/(&mmax-&mmin));
         %if &red eq &mmax %then %let hue=%sysevalf(&bc-&gc);
         %else %if &green eq &mmax %then %let hue=%sysevalf(2+&rc-&bc);
         %else %if &blue eq &mmax %then %let hue=%sysevalf(4+&gc-&rc);
         %let hue=%sysevalf(&hue*60);
         /*if hue lt 0*/
         %if %index(&hue,'-') gt 0 %then %let hue=%sysevalf(&hue+360);
         %let hue=%sysfunc(round(&hue+120));
         %if &hue ge 360 %then %let hue=%eval(&hue-360);
       %end;
       %sysfunc(putn(&hue,3.))
    %mend RGBtoHue;
/*   ----------------------------------------------------------------------------------
    Program   :  RGBtoLum.sas (see header comments for RGBtoHue.sas - gets a light val)
     ---------------------------------------------------------------------------- */
   %macro RGBtoLUM(r,g,b);
       %let red = %sysevalf(&r/255);
       %let green =%sysevalf(&g/255);
       %let blue=%sysevalf(&b/255);
       %let mmax=%sysfunc(max(&red,&green,&blue));
       %let mmin=%sysfunc(min(&red,&green,&blue));
       %let ll=%sysevalf((&mmax+&mmin)/2);
       %let lite=%sysfunc(round(%sysevalf(&ll*255)));
       %sysfunc(putn(&lite,3.))
    %mend RGBtoLUM;
/*  -----------------------------------------------------------------------------------
    Program   :  RGBtoSat.sas (see header comments for RGBtoHue.sas - gets a saturation val)
    -------------------------------------------------------------------------------- */
    %macro RGBtoSat(r,g,b);
        %let red = %sysevalf(&r/255);
        %let green =%sysevalf(&g/255);
        %let blue=%sysevalf(&b/255);
        %let mmax=%sysfunc(max(&red,&green,&blue));
        %let mmin=%sysfunc(min(&red,&green,&blue));
        %let ll=%sysevalf((&mmax+&mmin)/2);
        %if &mmax eq &mmin %then %let sat=0;
        %else %do;
          %if %sysfunc(putn(&ll,3.1)) le 0.5 %then
            %let sat=%sysfunc(round((%sysevalf((&mmax-&mmin))/%sysevalf((&mmax+&mmin)))*255));
          %else
            %let sat=%sysfunc(round((%sysevalf((&mmax-&mmin))/%sysevalf((2-&mmax-&mmin)))*255));
        %end;
        %sysfunc(putn(&sat,3.))
     %mend RGBtoSat;
```