HSP Color Model — Alternative to HSV (HSB) and HSL

After googling the internet for information about color systems that give the kind of sweet, clean results as Adobe Photoshop's RGB-to-Greyscale conversion, and not finding any, I decided to create my own, "**HSP**" color system. (I'd be surprised if I'm the original inventor; if you know of a pre-existing name for this system, please do contact me.)

In the HSV (also called HSB) system, the brightness of a color is its V component. That component is defined simply as the maximum value of any of the three RGB components of the color — the other two RGB components are ignored when determining V. This, I think, cannot a very good way to determine the brightness of the color.

In the HSL system, the L component is a 50-50 average of the maximum RGB value and the minimum RGB value. (The middle RGB value is ignored.) That sounds better than HSV, but still doesn't seem like a particularly good way to do it, IMHO.

By playing around with Photoshop's RGB-to-Greyscale mode conversion, I determined that it is doing something very close to this:

$$\textbf{brightness} = \textbf{sqrt}(.299\ R^2 + .587\ G^2 + .114\ B^2)$$

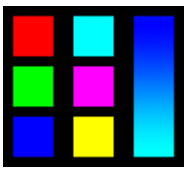
(Note: The values I came up with by playing with Photoshop were actually .241, .691, and .068, but I have since been informed that the values .299, .587, and .114 are more accurate. All of the image samples below were generated with the 241 values, except for the image explicitly labelled 299.)

I decided to call this system **HSP**, where the **P** stands for Perceived brightness.

The three constants (.299, .587, and .114) represent the different degrees to which each of the primary (RGB) colors affects human perception of the overall brightness of a color. Notice that they sum to 1.

Interestingly, Photoshop 6 appears to use both HSL and HSP, depending on how you remove color from an image. If you use the saturation slider, HSL seems to be used, but if you change the mode of the image to Greyscale, then HSP — or something a whole lot like it — appears to be employed. (I don't know about more recent versions of Photoshop.)

Let's try out HSP and see how it compares:


Original


HSV (V)


HSL (L)


HSP (P)

This simple color pallette of red, green, blue, cyan, magenta, and yellow, plus a blue-to-cyan gradient, painfully exposes the limitations of HSV and HSL in their attempts to gauge the brightness of a color. You really have to use HSP to get good results. Why Adobe chose to use HSL in their saturation slider is beyond me — perhaps it's a speed issue, and they wanted the picture to change smoothly in real-time

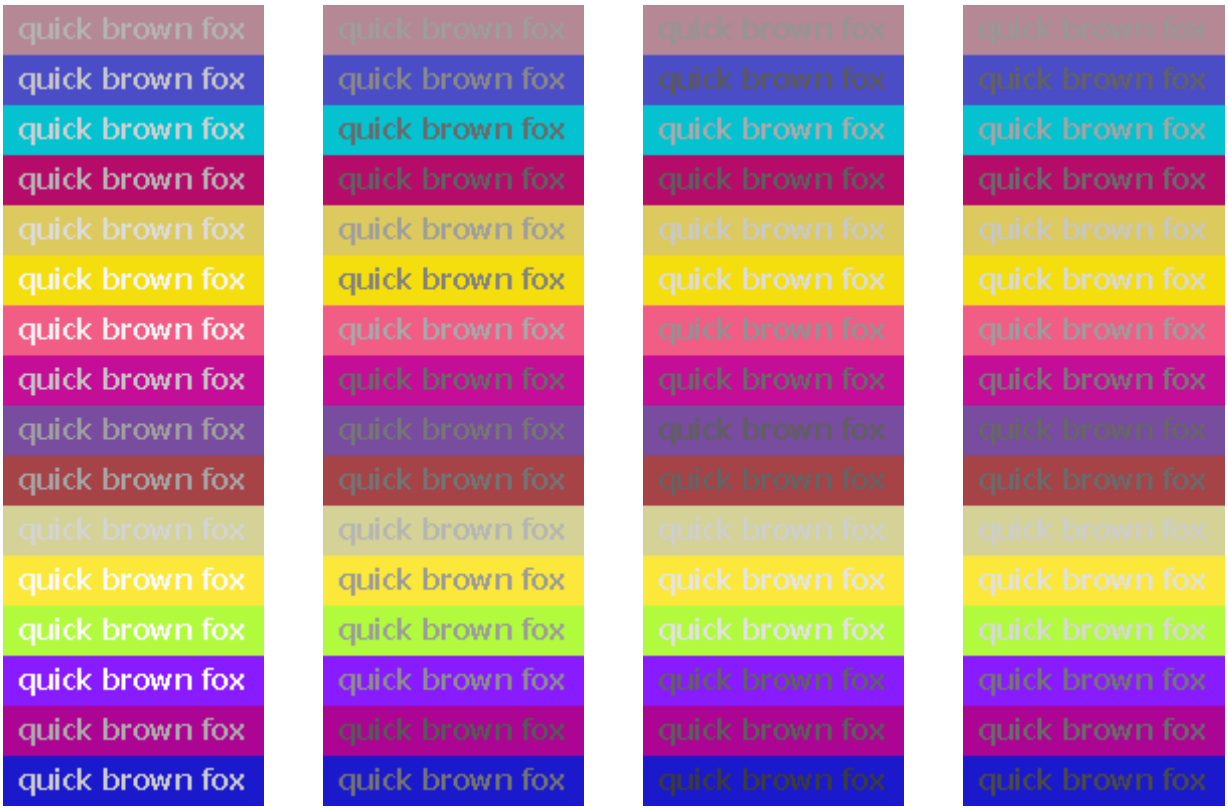
while you drag the slider.

HSV (V) HSV (V)

HSL (L) HSL (L)

HSP (P) HSP (P)

Notice in the above, split-screen images, the border between the colored region and the black-and-white region looks much less obvious in the HSP rendition than in either of the other two systems. This indicates that HSP renders a black-and-white image that looks more like the color original to the human eye.



HSV (V) HSL (L) HSP (P) HSP 299 (P)

In the above samples, the text is written in grey using the value delivered by the indicated color system. Notice that in the HSP system, the text is very hard to read, indicating that the grey delivered by that system is extremely similar (as interpreted by human vision) to the corresponding RGB color from which it was generated.

C Code Sample

```
#define  Pr  .299
#define  Pg  .587
#define  Pb  .114


//  public domain function by Darel Rex Finley, 2006
//
//  This function expects the passed-in values to be on a scale
//  of 0 to 1, and uses that same scale for the return values.
//
//  See description/examples at alienryderflex.com/hsp.html

void RGBtoHSP(
double  R, double  G, double  B,
double *H, double *S, double *P) {

  //  Calculate the Perceived brightness.
  *P=sqrt(R*R*Pr+G*G*Pg+B*B*Pb);

  //  Calculate the Hue and Saturation.  (This part works
  //  the same way as in the HSV/B and HSL systems???.)
  if       (R==G && R==B) {
    *H=0.; *S=0.; return; }
  if       (R>=G && R>=B) {   //  R is largest
    if     (B>=G) {
      *H=6./6.-1./6.*(B-G)/(R-G); *S=1.-G/R; }
    else          {
      *H=0./6.+1./6.*(G-B)/(R-B); *S=1.-B/R; }}
  else if (G>=R && G>=B) {   //  G is largest
    if     (R>=B) {
      *H=2./6.-1./6.*(R-B)/(G-B); *S=1.-B/G; }
    else          {
```

```
      *H=2./6.+1./6.*(B-R)/(G-R); *S=1.-R/G; }}
  else                  {   //  B is largest
    if    (G>=R) {
      *H=4./6.-1./6.*(G-R)/(B-R); *S=1.-R/B; }
    else         {
      *H=4./6.+1./6.*(R-G)/(B-G); *S=1.-G/B; }}}



//  public domain function by Darel Rex Finley, 2006
//
//  This function expects the passed-in values to be on a scale
//  of 0 to 1, and uses that same scale for the return values.
//
//  Note that some combinations of HSP, even if in the scale
//  0-1, may return RGB values that exceed a value of 1.  For
//  example, if you pass in the HSP color 0,1,1, the result
//  will be the RGB color 2.037,0,0.
//
//  See description/examples at alienryderflex.com/hsp.html

void HSPtoRGB(
double  H, double  S, double  P,
double *R, double *G, double *B) {

  double  part, minOverMax=1.-S ;

  if (minOverMax>0.) {
    if      ( H<1./6.) {   //  R>G>B
      H= 6.*( H-0./6.); part=1.+H*(1./minOverMax-1.);
      *B=P/sqrt(Pr/minOverMax/minOverMax+Pg*part*part+Pb);
      *R=(*B)/minOverMax; *G=(*B)+H*((*R)-(*B)); }
    else if ( H<2./6.) {   //  G>R>B
      H= 6.*(-H+2./6.); part=1.+H*(1./minOverMax-1.);
      *B=P/sqrt(Pg/minOverMax/minOverMax+Pr*part*part+Pb);
      *G=(*B)/minOverMax; *R=(*B)+H*((*G)-(*B)); }
    else if ( H<3./6.) {   //  G>B>R
      H= 6.*( H-2./6.); part=1.+H*(1./minOverMax-1.);
      *R=P/sqrt(Pg/minOverMax/minOverMax+Pb*part*part+Pr);
      *G=(*R)/minOverMax; *B=(*R)+H*((*G)-(*R)); }
    else if ( H<4./6.) {   //  B>G>R
      H= 6.*(-H+4./6.); part=1.+H*(1./minOverMax-1.);
      *R=P/sqrt(Pb/minOverMax/minOverMax+Pg*part*part+Pr);
      *B=(*R)/minOverMax; *G=(*R)+H*((*B)-(*R)); }
    else if ( H<5./6.) {   //  B>R>G
      H= 6.*( H-4./6.); part=1.+H*(1./minOverMax-1.);
      *G=P/sqrt(Pb/minOverMax/minOverMax+Pr*part*part+Pg);
      *B=(*G)/minOverMax; *R=(*G)+H*((*B)-(*G)); }
    else               {   //  R>B>G
      H= 6.*(-H+6./6.); part=1.+H*(1./minOverMax-1.);
      *G=P/sqrt(Pr/minOverMax/minOverMax+Pb*part*part+Pg);
      *R=(*G)/minOverMax; *B=(*G)+H*((*R)-(*G)); }}
  else {
    if      ( H<1./6.) {   //  R>G>B
      H= 6.*( H-0./6.); *R=sqrt(P*P/(Pr+Pg*H*H)); *G=(*R)*H; *B=0.; }
    else if ( H<2./6.) {   //  G>R>B
      H= 6.*(-H+2./6.); *G=sqrt(P*P/(Pg+Pr*H*H)); *R=(*G)*H; *B=0.; }
    else if ( H<3./6.) {   //  G>B>R
      H= 6.*( H-2./6.); *G=sqrt(P*P/(Pg+Pb*H*H)); *B=(*G)*H; *R=0.; }
    else if ( H<4./6.) {   //  B>G>R
      H= 6.*(-H+4./6.); *B=sqrt(P*P/(Pb+Pg*H*H)); *G=(*B)*H; *R=0.; }
    else if ( H<5./6.) {   //  B>R>G
      H= 6.*( H-4./6.); *B=sqrt(P*P/(Pb+Pr*H*H)); *R=(*B)*H; *G=0.; }
    else               {   //  R>B>G
      H= 6.*(-H+6./6.); *R=sqrt(P*P/(Pr+Pb*H*H)); *B=(*R)*H; *G=0.; }}}
```