

# HWB—A More Intuitive Hue-Based Color Model

*Alvy Ray Smith and Eric Ray Lyons*<sup>1</sup>

Submitted October 10, 1995

Revised January 29, 1996

**ABSTRACT:** The two most common color selector models, other than RGB (Red-Green-Blue), are the hue-based HSV (Hue-Saturation-Value) and HSL (Hue-Saturation-Lightness) color models. It is shown that both of these models are flawed. A closely related model, HWB (Hue-Whiteness-Blackness), is introduced that avoids the flaws, is slightly faster to compute, and is very easy to teach to new users: Choose a hue. Lighten it with white. Darken it with black. That whitening is not the complement of blackening is explained.

The HSV [Smith78] and HSL [GSPC79] color space models are well-known hue-based representations of the RGB color space of computer graphics. They are cylindrical coordinate systems equivalent to the RGB rectangular coordinate system. They are convenient alternatives to RGB because they offer a more natural or intuitive way of specifying colors. For example, a new user of a color computer graphics application often has trouble specifying pink or brown with RGB controls. In a hue-based system, pink is simply the hue red with white added; red is chosen with the *H* hue control then desaturated toward white with the *S* saturation control. Brown is just as easy: A red hue is chosen with the *H* control and then it is darkened with the *V* value control in HSV or the *L* lightness control in HSL.

Textbooks [e.g., Foley90] present the HSV and HSL models as of more or less equal usefulness. It is simple to show that the HSV model is slightly superior to the HSL from a user interface viewpoint [Smith79], but both models suffer from certain awkwardnesses to be explained below. We introduce a new hue-based model (HWB) that is superior to both on all counts, while being just as computationally inexpensive—slightly cheaper actually.

The popular Windows system for personal computers provides an HSL color selector that displays an *H* vs *S* color array next to an *L* ramp that passes from white at the top, to the (*H*, *S*, .5) color using the *H* and *S* from the color array, and then on down to black at the bottom. Users soon notice the following awkward aspects of HSL:

*HSL has a floating definition of S:* The meaning of the *S* saturation control changes as the *L* lightness (luminosity) control changes. They are mathematically orthogonal but not semantically orthogonal. For *L* equal 1 (full lightness), lowering *S* desaturates a hue toward white. For *L* equal 0 (no lightness), *S* desaturates toward black. In general, *S* desaturates a

---

<sup>1</sup> The authors were with Altamira Software Corporation, Mill Valley, CA when this paper was originally written, May 5, 1992. Dr. Smith is now with Microsoft Corporation, Redmond, WA and Mr. Lyons is self-employed in Mill Valley, CA. Correspondence should be sent to Alvy Ray Smith, 15600 NE 8<sup>th</sup> St.#B1/155, Bellevue, WA 98008.

hue toward the gray that  $L$  represents.<sup>2</sup> In particular, the full hues (at  $L = .5$ ) desaturate to half gray.

*HSL has an extrema problem:* An important special position on the  $L$  control is the difficult-to-locate center point rather than one of its two easily located extreme points. This is the special point corresponding to the unlightened and undarkened tint chosen with the other two controls. For a mechanical slider, a physical detent could be provided for the center point to make it easier to find, but software sliders typically do not provide this luxury.

HSV fares only a little better:

*HSV has a floating definition of S:* The HSV saturation control always desaturates to a gray that depends on  $V$ . The only intuitive advantage it has over HSL is that the full hues ( $V = 1$ ) desaturate toward full gray (white) instead of half gray, the half-valued hues ( $V = .5$ ) desaturate toward half gray, and so forth.

*HSV does not have an extrema problem:* The easily located top extreme of the  $V$  value control is always the tint selected by the  $H$  and  $S$  controls.

## The HWB Model

We now introduce the Hue-Whiteness-Blackness (HWB) model that maintains the good feature of the HSV model (easily located important slider positions) but adds the feature that neither HSV nor HSL provides (a fixed meaning for the saturation (whiteness) control, regardless of the other two controls). Most importantly, it is very easy to explain to new users.

The  $H$  hue of the HWB model is exactly that of the HSV and HSL models. Geometrically, the hue dimension is represented by a hexagon with RGB primaries located at every other vertex and the so-called secondary primaries at the intervening vertices. To new users, it can be explained that changing hue corresponds to moving around the familiar color circle seen in paint stores.

The  $W$  whiteness control varies from 0 to 1, with 1 being always full white and 0 always being the color *shade* (a mixture of a pure hue with black) chosen with the other two controls.

The  $B$  blackness control varies from 0 to 1, with 1 being always full black and 0 always being the color *tint* (a mixture of a pure hue with white) chosen with the other two controls.

The use of HWB is simply described to even the newest user: Choose a hue. Darken it with blackness. Lighten it with whiteness. More carefully stated it is this: Choose a hue. Blacken it, or a tint of it, with blackness. Whiten it, or a shade of it, with whiteness. See Fig. 1 for a demonstration. A slightly reddish blue hue is selected by sliding the horizontal slider ( $\nabla$ ) along the color circle (unrolled into a straight line). Three situations are shown:

---

<sup>2</sup> Windows compounds the problem by displaying a constant midvalue gray at the low extreme of the  $S$  control (in an  $H$  vs  $S$  color array), regardless of the current value of  $L$ .

Whiteness (only) is added to the hue with the right vertical slider ( $\triangleleft$ ) to create a tint of it. Blackness (only) is added to it with the left vertical slider ( $\triangleright$ ) to create a shade of blue. Both whiteness and blackness are added to create a *tone* of the given hue—i.e., an arbitrary color.

[See first attached color plate.]

**Figure 1. HWB color selection. (ul) Selection of a pure hue. (ur) Adding whiteness to it for a tint. (ll) Adding blackness instead for a shade. (lr) Adding both for a tone.**

In Fig. 1, the color ramp on the left is constant, showing all possible shades of all possible hues. A crosshair cursor always indicates the current shade of the selected hue. The whiteness ramp at the right varies, however. Its topmost color is always the shade indicated by the crosshair cursor on the left ramp.

The HWB model is very closely related to the HSV model for which transforms to and from RGB are well-known [Foley90]. Thus we provide transforms to and from HSV for HWB. The derivation is presented in Appendix A.

HSV to HWB	HWB to HSV
$H = H$	$H = H$
$W = (1 - S)V$	$S = 1 - \frac{W}{1 - B}$
$B = 1 - V$	$V = 1 - B$

## HWB To and From RGB

The full transforms of HWB to and from RGB are presented in Appendix B, but here we point out a particularly simple relationship of  $W$  to RGB:  $W = \min(R, G, B)$ . See Appendix A3 for the simple proof. So, if a color is represented by three bars for its relative amounts of RGB, then its HWB can be derived (almost) by inspection of the bars. The tallest bar is  $1 - B$ . The shortest bar is  $W$ . And  $H$  is the relative mixture of the two primaries that remain when the gray content of the RGB is removed. That is, remove the gray ( $W, W, W$ ) from the given RGB. The remaining contribution to the color must come from just two primaries. Their relative amounts determine  $H$ .

## A Note on Terminology

The HSV and HSL models also suffer from terminology problems. The first is ambiguous naming: PostScript [Adobe90] uses the HSV model but has renamed it HSB, changing the V for Value to B for Brightness. Windows [Microsoft90] uses the HSL model as its standard color selector but has changed the label of L from Lightness to Luminosity.

More importantly, neither the old or new terms are quite right. The term “saturation” is not commonly understood. The terms “value”, “brightness”, “lightness”, and “luminosity” either are not commonly understood, are incorrectly used, or are confused

with other terms that are better defined. “Value” is not commonly used outside artistic circles for the amount of shading (blackness) of a color. “Lightness” is defined in scientific circles as the perceived quantity of reflected light, not emitted light, so is incorrectly used. “Brightness” is the perceived quantity of emitted light whereas the parameter it is used to describe (the V of HSV) is only vaguely related to perception and definitely does not correlate with what a human means by “bright.” For example, in HSV pure blue and pure white have the same value, but white is certainly perceived as brighter than blue. “Luminosity” is both a well-defined optical quantity<sup>3</sup> and a loose equivalent (especially in Britain) to “brightness.” It is related to the L of HSL only in that both refer to emitted light and increase in the same direction. L is just an algorithmic derivation from RGB and not a measured physical or perceptual quantity. Pure blue and pure green have the same L value but green is physically more luminous (brighter) than blue on video and computer displays.

We propose to emerge from this naming morass by keeping only the term “hue” with which few seem to have trouble and replacing the other two dimension names with the very intuitive terms “blackness” and “whiteness” that describe (in HWB space) exactly their function. We considered alternatively the terms “shade” and “tint” that have the same meanings in artistic circles [Birren61], but which are not more widely understood unfortunately. We have elected to use the terms informally as above (blackness is mixing black with a tint, whiteness is mixing white with a shade).

We comment now on the fact that whiteness is not the complement of blackness; they are in fact mutually orthogonal. This is not surprising if one thinks of mixing a paint: One is free to mix in white or black paint completely independently and with obviously distinct results (cf. Fig. 1). The problem comes in confusion of this act with direct control of perceived brightness of the resulting color, an understandable error since adding black does tend to decrease brightness. But lowering the saturation of a color also decreases its brightness, and changing its hue does too (e.g., in video, from green to blue).

There are ways to define a direct brightness correlate to a given color. In video applications, for example, the well-defined “luminance” of American television can be extracted from the RGB (hence, by transforms, the HWB, HSV, or HSL) coordinates of a color by the well-known formula  $.3R + .59G + .11B$ , a function of all three dimensions (in any of the four systems).

Incidentally, we have intentionally misused the perceived brightness terminology in this paper. We have said to instruct new users to lighten a hue with white and darken it with black. The pedantically correct terminology is to whiten a hue with white and blacken it with black, which sounds tautological. We have been so successful in training new users of HWB with the incorrect but intuitive terminology that we do not argue against it in this situation.

It is worth mentioning that HWB, HSV, and HSL are not perception-based models, despite the perception-based names of their dimensions. They are simply convenient

---

<sup>3</sup> The ratio of luminous flux at a given wavelength to the total radiant flux at that same wavelength. Luminous flux is the rate of flow of light per unit time and is given in “lumens” in the International System. The *luminosity function* for the human “standard observer” was measured and defined by the CIE color standards committee in 1931.

algorithmic transformations of RGB. Thus they are not to be confused with such systems as Munsell, CIE, and Ostwald that are carefully based on measurements of real human color perception [Hunter 75]. In some of these complex systems, it is possible to isolate the brightness correlate into a single dimension. HWB, HSV, and HSL are intended to resemble those systems only. They feature very simple non-trigonometric computations and a certain intuitiveness, but their third dimensions are only indirect brightness controls.

## The HWB Color Solid

The RGB color space corresponds to a unit cube, the HSV space to a singly-ended hexagonal cone (a cone of hexagonal cross section), and the HSL to a doubly-ended hexagonal cone [Foley90]. The HWB model is just a recoordination of the HSV model, so the HWB color solid is also a singly-ended hexagonal cone, or hexcone.

In the HSV color solid, the loci of constant S are concentric hexagonal cones sharing a common vertex and a common axis. See Fig. 2. In the HWB solid, the loci of constant W are also concentric hexagonal cones, but they are nested cones with sides parallel to those of the HSV cone itself. Thus they do not share a common vertex nor a common axis (though their axes are collinear). The loci of constant V and B in the respective models are the same: hexagonal slices orthogonal to the axis of the cone.

With a slight stretch of the imagination, the HWB color solid is spherical rather than cylindrical. The white point is the center of the “sphere” (hemisphere actually). H is one of the angles, B is the other, and W (actually the complement of W) measures distance along the spherical radius determined by the two “angles.” This hemisphere has a hexagonal cross section and straight sides, however.

[See second attached color plate.]

**Figure 2. HWB (HSV) color solid. (ul) Constant H loci for HWB and HSV. (ur) Constant B loci for HWB and S loci for HSV. (ll) Constant W loci for HWB—cf., dashed lines in (ul). (lr) Constant S loci for HSV—cf., dashed lines in (ul).**

## Application Note

The HWB model was used to design the color selector for our commercial imaging product, Altamira Composer [Altamira93]. When implementing a user interface to HWB controls, we found it more natural to reverse the senses of W and B. See Fig. 1. Thus a whiteness slider oriented vertically would have the shade, to be mixed with white, at the top and white at the bottom. This means W is 0 at the top and 1 at the bottom, whereas it is typical to put 1 at the top of a vertical slider and 0 at the bottom. Similarly for B.

## References

[Adobe90] **PostScript Language Reference Manual**, Adobe Systems Incorporated, Addison-Wesley Publishing Company, New York, second edition, 1990, pp. 182, 304, 506, 746.

- [Altamira93] **Altamira Composer Owner's Manual**, Altamira Software Corporation, Mill Valley, CA, 1993, pp. 3-12 - 3-15.
- [Birren61] Faber Birren, **Creative Color: A Dynamic Approach for Artists and Designers**, Van Nostrand Reinhold Company, New York, 1961, pp. 12-16.
- [Foley90] James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes, **Computer Graphics, Principles and Practice**, Addison-Wesley Publishing Company, New York, second edition, 1990, pp. 590-596.
- [GSPC79] Graphics Standards Planning Committee, *Status Report of the Graphics Standards Planning Committee*, **Computer Graphics**, Vol. 13, No. 3, August 1979.
- [Hunter75] Richard S. Hunter, **The Measurement of Appearance**, John Wiley & Sons, New York, 1975. An excellent source for color theory.
- [Smith78] Alvy Ray Smith, *Color Gamut Transform Pairs*, **Computer Graphics**, Vol. 12, No. 3, August 1978, pp. 12-19. (SIGGRAPH 78 Conference Proceedings). Reprinted in **Tutorial: Computer Graphics**, edited by John C. Beatty and Kellogg S. Booth, IEEE Computer Society Press, Silver Spring, Maryland, second edition, 1982, pp. 376-383.
- [Smith79] Alvy Ray Smith, *Color Model Objections and Counterproposals*, Technical Memo No. 11, New York Institute of Technology, Computer Graphics Lab, August 1979. Submitted to the Graphics Standards Planning Committee, 1979. (Also issued as tutorial notes at SIGGRAPH 80 and 81.)
- [Microsoft90] **User's Guide for the Microsoft Windows Operating System, Windows Version 3.1**, Microsoft Corporation, Redmond, WA, 1990, p. 146.

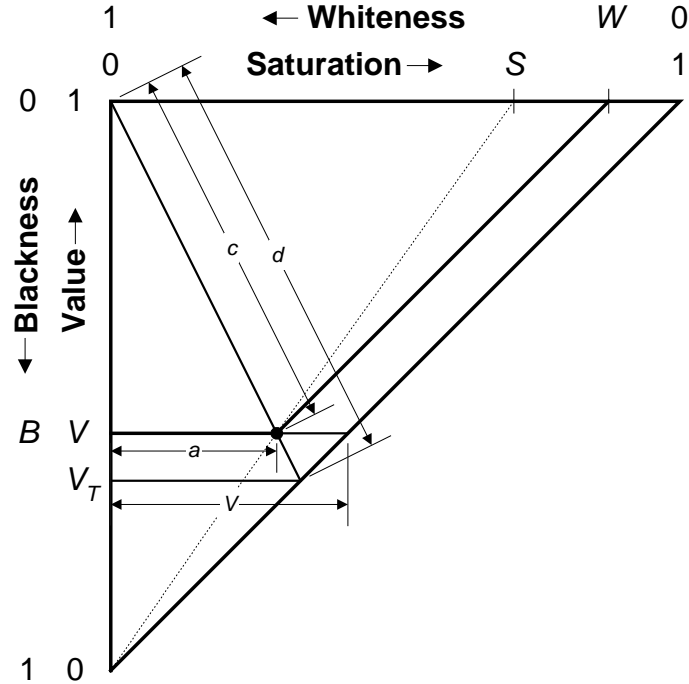
## Appendix A: Derivations

### A1. HWB from HSV

We derive  $B'$  and  $W'$ , the complements of  $B$  and  $W$ . Thus we can avoid the nuisance of having geometrical lengths measured in opposing directions. Since  $V$  in the HSV model has always represented mixing with black, we see that  $V$  and  $B'$  are identical.

Fig. 3 pictures a constant  $H$  slice in the HSV (hence HWB) hexagonal cone. Value varies from 0 to 1 along the left edge (and so does  $B'$ ). The black dot is the color for which we are deriving coordinates. Its value is  $V$ . Its saturation  $S$  is defined to be the ratio of the radius  $a$  to the maximum possible radius for  $V$ .  $V_T$  is the value of the shade that is to be whitened by moving the color point along the diagonal shown of length  $d$  to the white point (the upper left point, which is the center of the top face of the HSV cone). Whiteness (complement)  $W'$  is the ratio of  $c$  to  $d$ . Here we can see the key difference between HSV and HWB. For HSV, desaturation occurs along the cone radius from the point  $V$  on the cone axis through the color. For HWB, desaturation occurs along the diagonal through the white point and the color. Where this diagonal intersects the HSV cone corresponds to the pure shade that is to be desaturated along the diagonal with white. We have labeled the value of this pure shade  $V_T$ , since it corresponds to the blackness of the shade that exists at the top of the whiteness slider control where  $W'$  is 1 (again it is really the complement of

the blackness there). The following derivation is based principally on the fact that the diagram is a 45-degree right triangle.



**Figure 3. Constant H plane in HSV (equivalently HWB) hexcone color solid. Color at dot has coordinates S and V in HSV (W and B in HWB).**

Since the radius containing the color point is of maximum length  $V$  (because the right triangle has equal length sides) and from the definition of  $S$  as the ratio of  $a$  to the maximum possible radius there, we have

$$a = SV .$$

The maximum radius at value  $V_T$  is  $V_T$ , again because of the 45-degree right triangle. From this and triangle proportions

$$\frac{a}{V_T} = \frac{1-V}{1-V_T} = \frac{c}{d} .$$

But the desired whiteness (complement) is  $W' = c/d$ , giving these two equations:

$$W' = \frac{SV}{V_T} \quad \text{and} \quad W' = \frac{1-V}{1-V_T} .$$

Eliminating  $V_T$  gives

$$W' = 1 - (1-S)V$$

$$W = (1-S)V$$

as claimed. Notice that  $W$  is the length  $V - a$  in Fig. 3. We have all along used the fact that  $B' = V$  so

$$B = 1 - V .$$

It is sometimes convenient to know the formula for blackness  $B_T$  corresponding to value  $V_T$ . For example, in Fig. 1, we needed it to compute the color at the top of the whiteness slider. Notice that  $B_T = 1$  defines the gray axis of the HWB hexcone. Rearranging the first equation above for  $W'$  gives  $V_T = SV/W'$ . So

$$B_T = 1 - \frac{SV}{1 - W}$$

where  $W$  is given in terms of  $S$  and  $V$  above. Substitution gives

$$B_T = \frac{1 - V}{1 - W} = \frac{1 - V}{1 - (1 - S)V} .$$

Note that  $HWB_T$  is also a valid coordinate system for a color, but slightly less efficiently computed than HWB.

## A2. HSV from HWB

A trivial rearrangement of the formulas for HWB in terms of HSV above.

## A3. HWB to and from RGB

Hue for HWB is exactly the hue for HSV but we reformulate it here. The [Smith78] computation for HSV hue from RGB is, for  $v = \max(R, G, B)$  and  $x = \min(R, G, B)$ :

```

if(R == v)    h = (G == x) ? 5 + (v-B)/(v-x) : 1 - (v-G)/(v-x);    else
if(G == v)    h = (B == x) ? 1 + (v-R)/(v-x) : 3 - (v-B)/(v-x);    else
              h = (R == x) ? 3 + (v-G)/(v-x) : 5 - (v-R)/(v-x);

```

This C fragment computes hue  $h$  on  $[0, 6]$  for all other variables on  $[0, 1]$ . We simplify this computation first by substituting  $v$  with its actual value in the logic and then by inverting the maximum and minimum tests:

```

if(R == x)    h = (B == v) ? 3 + (B-G)/(v-x) : 3 - (G-B)/(v-x);    else
if(G == x)    h = (R == v) ? 5 + (R-B)/(v-x) : 5 - (B-R)/(v-x);    else
              h = (G == v) ? 1 + (G-R)/(v-x) : 1 - (R-G)/(v-x);

```

which simplifies to:

```

if(R == x)    h = 3 - (G-B)/(v-x);    else
if(G == x)    h = 5 - (B-R)/(v-x);    else
              h = 1 - (R-G)/(v-x);

```

This can be shown equal to the [Foley90] form

```

if(R == v)    h = (G-B)/(v-x);    else
if(G == v)    h = 2 + (B-R)/(v-x);    else
              h = 4 + (R-G)/(v-x);
if(h < 0) h += 6;

```

by substituting, in the first **if** statement above, the actual values for  $v$  and  $x$  and rearranging for 0, 2, and 4 (modulo 6) rather than 1, 3, and 5. Notice that taking pure red to be the  $h = 0$  hue is completely arbitrary.

Since our derivations depend on them, we give the full conversion routines for HSV to and from RGB below for convenience. These are adapted from [Smith78] and [Foley90] but given in a form like the routines for HWB to and from RGB in Appendix B for comparison. These routines are written in C (with C++ comments):

```
#define RETURN_HSV(h, w, v) {HSV.H = h; HSV.S = s; HSV.V = v; return HSV;}
#define RETURN_RGB(r, g, b) {RGB.R = r; RGB.G = g; RGB.B = b; return RGB;}
#define UNDEFINED -1
```

```
// Theoretically, hue 0 (pure red) is identical to hue 6 in these transforms. Pure
// red always maps to 6 in this implementation. Therefore UNDEFINED can be
// defined as 0 in situations where only unsigned numbers are desired.
```

```
typedef struct {float R, G, B;} RGBType;
typedef struct {float H, S, V;} HSVType;
```

```
HSVType
```

```
RGB_to_HSV( RGBType RGB ) {
```

```
    // RGB are each on [0, 1]. S and V are returned on [0, 1] and H is
    // returned on [0, 6]. Exception: H is returned UNDEFINED if S=0.
```

```
    float R = RGB.R, G = RGB.G, B = RGB.B, v, x, f;
```

```
    int i;
```

```
    HSVType HSV;
```

```
    x = min(R, G, B);
```

```
    v = max(R, G, B);
```

```
    if(v == x) RETURN_HSV(UNDEFINED, 0, v);
```

```
    f = (R == x) ? G - B : ((G == x) ? B - R : R - G);
```

```
    i = (R == x) ? 3 : ((G == x) ? 5 : 1);
```

```
    RETURN_HSV(i - f/(v - x), (v - x)/v, v);
```

```
}
```

```
RGBType
```

```
HSV_to_RGB( HSVType HSV ) {
```

```
    // H is given on [0, 6] or UNDEFINED. S and V are given on [0, 1].
```

```
    // RGB are each returned on [0, 1].
```

```
    float h = HSV.H, s = HSV.S, v = HSV.V, m, n, f;
```

```
    int i;
```

```
    RGBType RGB;
```

```
    if(h == UNDEFINED) RETURN_RGB(v, v, v);
```

```
    i = floor(h);
```

```
    f = h - i;
```

```
    if(!(i&1)) f = 1 - f; // if i is even
```

```
    m = v * (1 - s);
```

```

    n = v * (1 - s * f);
    switch(i) {
        case 6:
            case 0: RETURN_RGB(v, n, m);
            case 1: RETURN_RGB(n, v, m);
            case 2: RETURN_RGB(m, v, n);
            case 3: RETURN_RGB(m, n, v);
            case 4: RETURN_RGB(n, m, v);
            case 5: RETURN_RGB(v, m, n);
    }
}

```

We prove, as promised, that  $W = \min(R, G, B)$ :  $S$  in HSV is derived from RGB (see routine RGB\_to\_HSV above) by  $X = \min(R, G, B)$  and  $S = (V - X)/V$ . But  $W = (1 - S)V = X$ .

$H$  of HWB is the same as  $H$  of HSV, and  $B$  of HWB is  $1 - V = 1 - \max(R, G, B)$ . This completes the HWB from RGB transform derivation.

For the reverse transform first note that  $m$  in the HSV\_to\_RGB routine above is exactly  $W = (1 - S)V$ . Then the expression for  $n$  in that routine can be shown to equal that in Appendix B, in the routine HWB\_to\_RGB, by simple substitution and reversing the sense of  $f$ . Again  $H$  and  $B$  are handled trivially, thus completing the derivation of RGB from HWB. We have simplified the RGB from HSV routine slightly from that of [Foley90] (and [Smith78]) by using the odd-even test on  $f$  to eliminate one expression,  $t = v * (1 - s * (1 - f))$ , entirely.

## Appendix B: The HWB-RGB Transform Pair

The following routines are written in C (with C++ comments):

```

#define RETURN_HWB(h, w, b) {HWB.H = h; HWB.W = w; HWB.B = b; return HWB;}
#define RETURN_RGB(r, g, b) {RGB.R = r; RGB.G = g; RGB.B = b; return RGB;}
#define UNDEFINED -1

// Theoretically, hue 0 (pure red) is identical to hue 6 in these transforms. Pure
// red always maps to 6 in this implementation. Therefore UNDEFINED can be
// defined as 0 in situations where only unsigned numbers are desired.

typedef struct {float R, G, B;} RGBType;
typedef struct {float H, W, B;} HWBType;

HWBType
RGB_to_HWB( RGBType RGB ) {
    // RGB are each on [0, 1]. W and B are returned on [0, 1] and H is
    // returned on [0, 6]. Exception: H is returned UNDEFINED if W == 1 - B.

    float R = RGB.R, G = RGB.G, B = RGB.B, w, v, b, f;
    int i;
    HWBType HWB;

```

```

    w = min(R, G, B);
    v = max(R, G, B);
    b = 1 - v;
    if(v == w) RETURN_HWB(UNDEFINED, w, b);
    f = (R == w) ? G - B : ((G == w) ? B - R : R - G);
    i = (R == w) ? 3 : ((G == w) ? 5 : 1);
    RETURN_HWB(i - f / (v - w), w, b);
}

RGBType
HWB_to_RGB( HWBType HWB ) {
    // H is given on [0, 6] or UNDEFINED. W and B are given on [0, 1].
    // RGB are each returned on [0, 1]. NB, W+B <= 1 for valid RGB.

    float h = HWB.H, w = HWB.W, b = HWB.B, v, n, f;
    int i;
    RGBType RGB;

    v = 1 - b;
    if(h == UNDEFINED) RETURN_RGB(v, v, v);
    i = floor(h);
    f = h - i;
    if(i & 1) f = 1 - f;           // if i is odd
    n = w + f * (v - w);         // linear interpolation between w and v
    switch(i) {
        case 6:
        case 0: RETURN_RGB(v, n, w);
        case 1: RETURN_RGB(n, v, w);
        case 2: RETURN_RGB(w, v, n);
        case 3: RETURN_RGB(w, n, v);
        case 4: RETURN_RGB(n, w, v);
        case 5: RETURN_RGB(v, w, n);
    }
}

```

Check for errata at <http://www.acm.org/jgt/papers/SmithLyons96/>

Note added 29 October 2011: The url above is no longer valid. This is what it said:

[quote]

#### Source Code

Downloadable C code that implements the color transforms described in the paper: [HWB to RGB transforms](#), and the closely related [HSV to RGB transforms](#).

Revision history:

- 31 Jan 1996
  - Initial version.
- 28 Mar 1997
  - HSV to RGB: Corrected bug in test if i is even (see erratum below).
- 27 Oct 1997
  - HSV to RGB: deleted spurious line  $b = 1-v$ ;

### Errata

On page 17, in the source code for the routine HSV\_to\_RGB(), the line

```
if (i^1) f = 1 - f; // if i is even
```

Should be:

```
if (!(i&1)) f + 1 - f; // if i is even
```

This correction has been made in the downloadable source code listed above.  
[end quote]

On 29 October 2011 this url worked: <http://jgt.akpeters.com/papers/SmithLyons96/> .

N.B. I have changed the code in this paper at the top of p. 11 (as numbered here) to reflect the Errata above. The latest versions of the algorithms are found now at [HWB to RGB transforms](#) and [HSV to RGB transforms](#).

