Article

Browse Code

Stats

Revisions

Alternatives

Comments (107)

Tagged as

.NET1.1
.NET2.0
VS.NET2003
C#2.0
Windows
Dev
Design
Intermediate

Related Articles

SAPrefs - Netscape-like Preferences Dialog

XNA Snooker Club

WPF: A* search

Windows 7 / VS2010 demo app

Window Tabs (WndTabs) Add-In for DevStudio

Manipulating colors in .NET - Part 1

Guillaume Leparmentier, 3 Jun 2007 CPOL

★★★★★ 4.96 (244 votes)

Rate this: ☆☆☆☆☆

Understand and use color models in .NET

Download demo project (.NET 1.1) - 58.8 KB
Download C# source files (.NET 1.1) - 110.0 KB
Download C# source files (.NET 2.0) - 111.2 KB
Download VB source files (.NET 2.0) - 115.7 KB

Info

Best C# article of May 2007

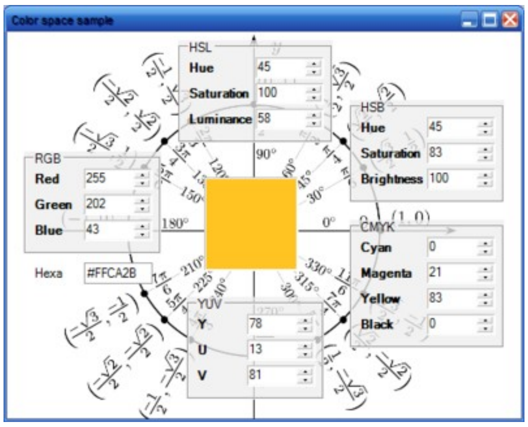
First Posted	3 Jun 2007
Views	276,384
Downloads	8,028
Bookmarked	396 times

Research

Four Strategies To Reduce Your Open Source Risk

Contents

An application example converting from one color model to another.



Color space sample

HSL: Hue 45, Saturation 100, Luminance 58

HSB: Hue 45, Saturation 83, Brightness 100

RGB: Red 255, Green 202, Blue 43

Hexa #FFCA2B

CMYK: Cyan 0, Magenta 21, Yellow 83, Black 0

YUV: Y 78, U 13, V 81

Introduction

Why an article on "colors"? It's the same question I asked myself before writing this series.

The fact is, in .NET, there are only two color formats that can be used: the RGB color model and the HSB color model. Those two are encapsulated in the `Color` structure of the `System.Drawing` namespace.

It is largely sufficient for simple uses, like changing the background color of a component, but insufficient if we want to develop graphic tools (or something which implies conversion between color formats).

I've started looking for other formats, like CMYK, when I was learning how to add cool design-time support to my custom controls.
But, I quickly understood the utilities I was coding would be more than useful in other projects, like my SVG editor.

That's why I publish this series of 3 articles. The proposed article content will be as follows:

- Part 1: This one, will be an introduction into color spaces, and the most used ones.
- Part 2: Will be about dedicated controls, or how to select/define a color with custom

components. I will share a lot of my custom controls for this purpose.
- Part 3: Will be about how to use those color controls at design-time. It will be a tutorial on how to create your own editors.

Let's start with some definitions

"**Color** (or colour, see spelling differences) is the visual perceptual property corresponding in humans to the categories called red, yellow, white, etc. Color derives from the spectrum of light (distribution of light energy versus wavelength) interacting in the eye with the spectral sensitivities of the light receptors. Color categories and physical specifications of color are also associated with objects, materials, light sources, etc., based on their physical properties such as light absorption, reflection, or emission spectra."

Color definition, Wikipedia.

"**Colorimetry** is the science that describes colors in numbers, or provides a physical color match using a variety of measurement instruments. Colorimetry is used in chemistry, and in industries such as color printing, textile manufacturing, paint manufacturing and in the food industry."

Colorimetry definition, Wikipedia.

Then, how can we display colors as numbers? the answer: color models.

I - Color models

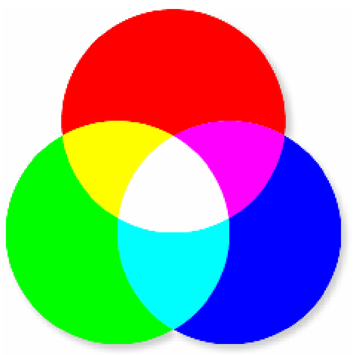
A - RGB (Red Green Blue)

The **RGB** (**R**ed, **G**reen, **B**lue) color model is the most known, and the most used every day. It defines a color space in terms of three components:

- **R**ed, which ranges from 0-255
- **G**reen, which ranges from 0-255
- **B**lue, which ranges from 0-255

The **RGB** color model is an additive one. In other words, **R**ed, **G**reen and **B**lue values (known as the three primary colors) are combined to reproduce other colors.
For example, the color "Red" can be represented as [R=255, G=0, B=0], "Violet" as [R=238, G=130, B=238], etc.

Its common graphic representation is the following image:



In .NET, the `Color` structure use this model to provide color support through R, G and B properties.

Collapse | Copy Code

```
Console.WriteLine(String.Format("R={0}, G={1}, B={2}",
    Color.Red.R, Color.Red.G, Color.Red.B);
Console.WriteLine(String.Format("R={0}, G={1}, B={2}",
    Color.Cyan.R, Color.Cyan.G, Color.Cyan.B);
Console.WriteLine(String.Format("R={0}, G={1}, B={2}",
    Color.White.R, Color.White.G, Color.White.B);
Console.WriteLine(String.Format("R={0}, G={1}, B={2}",
    Color.SteelBlue.R, Color.SteelBlue.G, Color.SteelBlue.B);

// etc...
```

but this is not its only usage. For this reason, we can define a dedicated RGB structure for further coding, as shown below:

Collapse | Copy Code

```
/// <summary>
/// RGB structure.
/// </summary>
public struct RGB
```

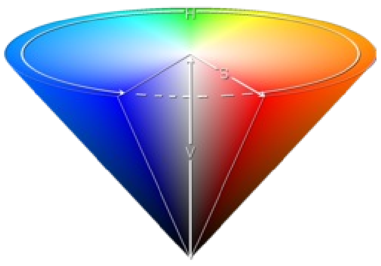
```
{
    /// <summary>
    /// Gets an empty RGB structure;
    /// </summary>
    public static readonly RGB Empty = new RGB();

    private int red;
    private int green;
    private int blue;

    public static bool operator ==(RGB item1, RGB item2)
    {
        return (
            item1.Red == item2.Red
            && item1.Green == item2.Green
            && item1.Blue == item2.Blue
            );
    }

    public static bool operator !=(RGB item1, RGB item2)
    {
        return (
            item1.Red != item2.Red
            || item1.Green != item2.Green
            || item1.Blue != item2.Blue
            );
    }

    /// <summary>
    /// Gets or sets red value.
    /// </summary>
    public int Red
    {
        get
        {
            return red;
        }
        set
        {
                red = (value>255)? 255 : ((value<0)?0 : value);
        }
    }

    /// <summary>
    /// Gets or sets red value.
    /// </summary>
    public int Green
    {
        get
        {
            return green;
        }
        set
        {
            green = (value>255)? 255 : ((value<0)?0 : value);
        }
    }

    /// <summary>
    /// Gets or sets red value.
    /// </summary>
    public int Blue
    {
        get
        {
            return blue;
        }
        set
        {
            blue = (value>255)? 255 : ((value<0)?0 : value);
        }
    }

    public RGB(int R, int G, int B)
    {
        this.red = (R>255)? 255 : ((R<0)?0 : R);
        this.green = (G>255)? 255 : ((G<0)?0 : G);
        this.blue = (B>255)? 255 : ((B<0)?0 : B);
    }

    public override bool Equals(Object obj)
    {
        if(obj==null || GetType()!=obj.GetType()) return false;

        return (this == (RGB)obj);
    }

    public override int GetHashCode()
    {
        return Red.GetHashCode() ^ Green.GetHashCode() ^ Blue.GetHashCode();
    }
}
```

B - HSB color space

The **HSB** (**H**ue, **S**aturation, **B**rightness) color model defines a color space in terms of three constituent components:

- **H**ue : the color type (such as red, blue, or yellow).
  - Ranges from 0 to 360° in most applications. (each value corresponds to one color : 0 is red, 45 is a shade of orange and 55 is a shade of yellow).
- **S**aturation : the intensity of the color.
  - Ranges from 0 to 100% (0 means no color, that is a shade of grey between black and white; 100 means intense color).
  - Also sometimes called the "purity" by analogy to the **colorimetric** quantities excitation purity.
- **B**rightness (or **V**alue) : the brightness of the color.
  - Ranges from 0 to 100% (0 is always black; depending on the saturation, 100 may be white or a more or less saturated color).

Its common graphic representation is the following image:



The **HSB** model is also known as **HSV** (**H**ue, **S**aturation, **V**alue) model. The **HSV** model was created in 1978 by Alvy Ray Smith. It is a nonlinear transformation of the RGB color space. In other words, color is not defined as a simple combination (addition/substraction) of primary colors but as a mathematical transformation.

Note: **HSV** and **HSB** are the same, but **HSL** is different.

All this said, a **HSB** structure can be :

⊟ Collapse | Copy Code

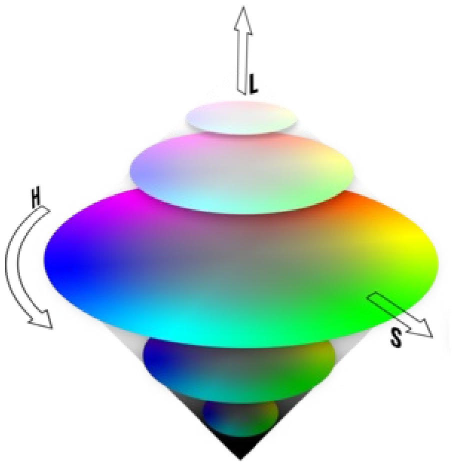
```
/// <summary>
/// Structure to define HSB.
/// </summary>
public struct HSB
{
    /// <summary>
    /// Gets an empty HSB structure;
    /// </summary>
    public static readonly HSB Empty = new HSB();

    private double hue;
    private double saturation;
    private double brightness;

    public static bool operator ==(HSB item1, HSB item2)
    {
        return (
            item1.Hue == item2.Hue
            && item1.Saturation == item2.Saturation
            && item1.Brightness == item2.Brightness
            );
    }

    public static bool operator !=(HSB item1, HSB item2)
    {
        return (
            item1.Hue != item2.Hue
            || item1.Saturation != item2.Saturation
            || item1.Brightness != item2.Brightness
            );
    }

    /// <summary>
    /// Gets or sets the hue component.
    /// </summary>
    public double Hue
    {
        get
        {
            return hue;
        }
        set
        {
            hue = (value>360)? 360 : ((value<0)?0:value);
        }
    }

    /// <summary>
    /// Gets or sets saturation component.
    /// </summary>
    public double Saturation
    {
        get
        {
            return saturation;
        }
        set
        {
            saturation = (value>1)? 1 : ((value<0)?0:value);
        }
    }

    /// <summary>
    /// Gets or sets the brightness component.
    /// </summary>
    public double Brightness
    {
        get
        {
            return brightness;
        }
        set
        {
            brightness = (value>1)? 1 : ((value<0)? 0 : value);
        }
    }

    /// <summary>
    /// Creates an instance of a HSB structure.
    /// </summary>
    /// <param name="h">Hue value.</param>
    /// <param name="s">Saturation value.</param>
    /// <param name="b">Brightness value.</param>
    public HSB(double h, double s, double b)
    {
        hue = (h>360)? 360 : ((h<0)?0:h);
        saturation = (s>1)? 1 : ((s<0)?0:s);
        brightness = (b>1)? 1 : ((b<0)?0:b);
    }

    public override bool Equals(Object obj)
    {
        if(obj==null || GetType()!=obj.GetType()) return false;

        return (this == (HSB)obj);
    }

    public override int GetHashCode()
    {
        return Hue.GetHashCode() ^ Saturation.GetHashCode() ^
            Brightness.GetHashCode();
    }
}
```

C - HSL color space

The **HSL** color space, also called **HLS** or **HSI**, stands for:

- **H**ue : the color type (such as red, blue, or yellow).
    - Ranges from 0 to 360° in most applications (each value corresponds to one color : 0 is red, 45 is a shade of orange and 55 is a shade of yellow).
- **S**aturation : variation of the color depending on the lightness.
    - Ranges from 0 to 100% (from the center of the black&white axis).
- **L**ightness (also Luminance or Luminosity or Intensity).
    - Ranges from 0 to 100% (from black to white).

Its common graphic representation is the following image:



HSL is similar to **HSB**. The main difference is that **HSL** is symmetrical to lightness and darkness. This means that:

- In **HSL**, the Saturation component always goes from fully saturated color to the equivalent gray (in **HSB**, with B at maximum, it goes from saturated color to white).
- In **HSL**, the Lightness always spans the entire range from black through the chosen hue to white (in **HSB**, the B component only goes half that way, from black to the chosen hue).

For my part, **HSL** offers a more accurate (even if it's not absolute) color approximation than **HSB**.

All this said, a **HSL** structure can be:

Collapse | Copy Code

```
/// <summary>
/// Structure to define HSL.
/// </summary>
public struct HSL
{
    /// <summary>
    /// Gets an empty HSL structure;
    /// </summary>
    public static readonly HSL Empty = new HSL();

    private double hue;
    private double saturation;
    private double luminance;

    public static bool operator ==(HSL item1, HSL item2)
    {
        return (
            item1.Hue == item2.Hue
            && item1.Saturation == item2.Saturation
            && item1.Luminance == item2.Luminance
            );
    }

    public static bool operator !=(HSL item1, HSL item2)
    {
        return (
            item1.Hue != item2.Hue
            || item1.Saturation != item2.Saturation
            || item1.Luminance != item2.Luminance
            );
    }

    /// <summary>
    /// Gets or sets the hue component.
    /// </summary>
    public double Hue
    {
        get
        {
            return hue;
        }
        set
        {
            hue = (value>360)? 360 : ((value<0)?0:value);
        }
    }

    /// <summary>
    /// Gets or sets saturation component.
    /// </summary>
    public double Saturation
    {
        get
        {
            return saturation;
        }
        set
        {
            saturation = (value>1)? 1 : ((value<0)?0:value);
        }
    }

    /// <summary>
    /// Gets or sets the luminance component.
    /// </summary>
    public double Luminance
    {
        get
        {
            return luminance;
        }
        set
        {
            luminance = (value>1)? 1 : ((value<0)? 0 : value);
        }
    }

    /// <summary>
    /// Creates an instance of a HSL structure.
    /// </summary>
    /// <param name="h">Hue value.</param>
    /// <param name="s">Saturation value.</param>
    /// <param name="l">Lightness value.</param>
    public HSL(double h, double s, double l)
    {
        this.hue = (h>360)? 360 : ((h<0)?0:h);
        this.saturation = (s>1)? 1 : ((s<0)?0:s);
        this.luminance = (l>1)? 1 : ((l<0)?0:l);
    }

    public override bool Equals(Object obj)
    {
        if(obj==null || GetType()!=obj.GetType()) return false;

        return (this == (HSL)obj);
    }

    public override int GetHashCode()
    {
        return Hue.GetHashCode() ^ Saturation.GetHashCode() ^
            Luminance.GetHashCode();
    }
}
```

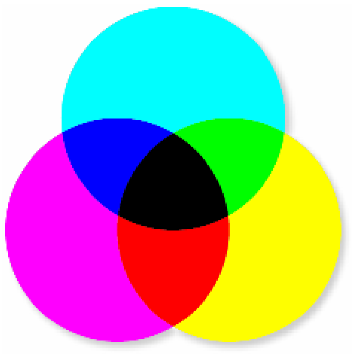
D - CMYK color space

The **CMYK** color space, also known as **CMJN**, stands for:

- **C**yan.
  - Ranges from 0 to 100% in most applications.
- **M**agenta.
  - Ranges from 0 to 100% in most applications.
- **Y**ellow.
  - Ranges from 0 to 100% in most applications.
- blac**K**.
  - Ranges from 0 to 100% in most applications.

It is a subtractive color model used in color printing. **CMYK** works on an optical illusion that is based on light absorption.
The principle is to superimpose three images; one for cyan, one for magenta and one for yellow; which will reproduce colors.

Its common graphic representation is the following image:



Like the **RGB** color model, **CMYK** is a combination of primary colors (cyan, magenta, yellow and black). It is, probably, the only thing they have in common.
CMYK suffers from a lack of color shades that causes holes in the color spectrum it can reproduce. That's why there are often differencies when someone convert a color between **CMYK** to **RGB**.

Why using this model? Why black is used? you can tell me... Well it's only for practical purpose. Wikipedia said:

- To improve print quality and reduce moiré patterns,
- Text is typically printed in black and includes fine detail (such as serifs); so to reproduce text using three inks would require an extremely precise alignment for each three components image.
- A combination of cyan, magenta, and yellow pigments don't produce (or rarely) pure black.
- Mixing all three color inks together to make black can make the paper rather wet when not using dry toner, which is an issue in high speed printing where the paper must dry extremely rapidly to avoid marking the next sheet, and poor quality paper such as newsprint may break if it becomes too wet.
- Using a unit amount of black ink rather than three unit amounts of the process color inks can lead to significant cost savings (black ink is often cheaper).

Let's come back to our reality. A **CMYK** structure can be:

⊟ Collapse | Copy Code

```
/// <summary>
/// Structure to define CMYK.
/// </summary>
public struct CMYK
{
    /// <summary>
    /// Gets an empty CMYK structure;
    /// </summary>
    public readonly static CMYK Empty = new CMYK();

    private double c;
    private double m;
    private double y;
    private double k;

    public static bool operator ==(CMYK item1, CMYK item2)
    {
        return (
            item1.Cyan == item2.Cyan
            && item1.Magenta == item2.Magenta
            && item1.Yellow == item2.Yellow
            && item1.Black == item2.Black
            );
    }

    public static bool operator !=(CMYK item1, CMYK item2)
    {
        return (
            item1.Cyan != item2.Cyan
            || item1.Magenta != item2.Magenta
            || item1.Yellow != item2.Yellow
            || item1.Black != item2.Black
            );
    }

    public double Cyan
    {
        get
        {
            return c;
        }
        set
        {
            c = value;
            c = (c>1)? 1 : ((c<0)? 0 : c);
        }
    }

    public double Magenta
    {
        get
        {
            return m;
        }
        set
        {
            m = value;
            m = (m>1)? 1 : ((m<0)? 0 : m);
        }
    }

    public double Yellow
    {
        get
        {
            return y;
        }
        set
        {
            y = value;
            y = (y>1)? 1 : ((y<0)? 0 : y);
        }
    }

    public double Black
    {
        get
        {
            return k;
        }
        set
        {
            k = value;
            k = (k>1)? 1 : ((k<0)? 0 : k);
        }
    }

    /// <summary>
    /// Creates an instance of a CMYK structure.
    /// </summary>
    public CMYK(double c, double m, double y, double k)
    {
        this.c = c;
        this.m = m;
        this.y = y;
        this.k = k;
    }

    public override bool Equals(Object obj)
    {
        if(obj==null || GetType()!=obj.GetType()) return false;

        return (this == (CMYK)obj);
    }

    public override int GetHashCode()
    {
        return Cyan.GetHashCode() ^
          Magenta.GetHashCode() ^ Yellow.GetHashCode() ^ Black.GetHashCode();
    }

}
```

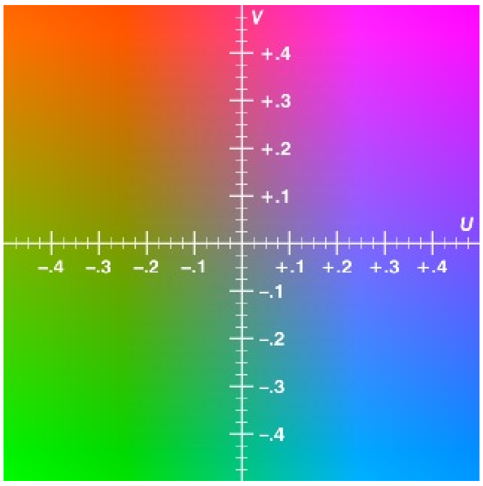
E - YUV color space

The **YUV** model defines a color space in terms of one luma and two chrominance components. The **YUV** color model is used in the PAL, NTSC, and SECAM composite color video standards. **YUV** models human perception of color more closely than the standard RGB model used in computer graphics hardware.

The **YUV** color space stands for:

- **Y**, the luma component, or the brightness.
    - Ranges from 0 to 100% in most applications.
- **U** and **V** are the chrominance components (blue-luminance and red-luminance differences components).
    - Expressed as factors depending on the **YUV** version you want to use.

A graphic representation is the following image:



A **YUV** structure can be:

⊟ Collapse | Copy Code

```
/// <summary>
/// Structure to define YUV.
/// </summary>
public struct YUV
{
    /// <summary>
    /// Gets an empty YUV structure.
    /// </summary>
    public static readonly YUV Empty = new YUV();

    private double y;
    private double u;
    private double v;

    public static bool operator ==(YUV item1, YUV item2)
    {
        return (
            item1.Y == item2.Y
            && item1.U == item2.U
            && item1.V == item2.V
            );
    }

    public static bool operator !=(YUV item1, YUV item2)
    {
        return (
            item1.Y != item2.Y
            || item1.U != item2.U
            || item1.V != item2.V
            );
    }

    public double Y
    {
        get
        {
            return y;
        }
        set
        {
            y = value;
            y = (y>1)? 1 : ((y<0)? 0 : y);
        }
    }

    public double U
    {
        get
        {
            return u;
        }
        set
        {
            u = value;
            u = (u>0.436)? 0.436 : ((u<-0.436)? -0.436 : u);
        }
    }

    public double V
    {
        get
        {
            return v;
        }
        set
        {
            v = value;
            v = (v>0.615)? 0.615 : ((v<-0.615)? -0.615 : v);
        }
    }

    /// <summary>
    /// Creates an instance of a YUV structure.
    /// </summary>
    public YUV(double y, double u, double v)
    {
        this.y = (y>1)? 1 : ((y<0)? 0 : y);
        this.u = (u>0.436)? 0.436 : ((u<-0.436)? -0.436 : u);
        this.v = (v>0.615)? 0.615 : ((v<-0.615)? -0.615 : v);
    }

    public override bool Equals(Object obj)
    {
        if(obj==null || GetType()!=obj.GetType()) return false;

        return (this == (YUV)obj);
    }

    public override int GetHashCode()
    {
        return Y.GetHashCode() ^ U.GetHashCode() ^ V.GetHashCode();
    }

}
```

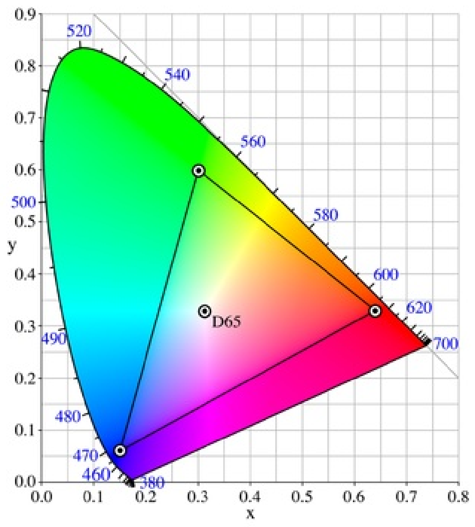
F - CIE XYZ color space

In opposition to the previous models, the **CIE XYZ** model defines an absolute color space. It is also known as the **CIE 1931 XYZ** color space and stands for:

- **X**, which can be compared to red
    - Ranges from 0 to 0.9505
- **Y**, which can be compared to green
    - Ranges from 0 to 1.0
- **Z**, which can be compared to blue
    - Ranges from 0 to 1.089

Before trying to explain why I include this color space in this article, you have to know that it's one of the first standards created by the International Commission on Illumination (CIE) in 1931. It is based on direct measurements of the human eye, and serves as the basis from which many other color spaces are defined.

A graphic representation is the following image:



A **CIE XYZ** structure can be:

Collapse | Copy Code

```
/// <summary>
/// Structure to define CIE XYZ.
/// </summary>
public struct CIEXYZ
{
    /// <summary>
    /// Gets an empty CIEXYZ structure.
    /// </summary>
    public static readonly CIEXYZ Empty = new CIEXYZ();
    /// <summary>
    /// Gets the CIE D65 (white) structure.
    /// </summary>
    public static readonly CIEXYZ D65 = new CIEXYZ(0.9505, 1.0, 1.0890);


    private double x;
    private double y;
    private double z;

    public static bool operator ==(CIEXYZ item1, CIEXYZ item2)
    {
        return (
            item1.X == item2.X
            && item1.Y == item2.Y
            && item1.Z == item2.Z
            );
    }

    public static bool operator !=(CIEXYZ item1, CIEXYZ item2)
    {
        return (
            item1.X != item2.X
            || item1.Y != item2.Y
            || item1.Z != item2.Z
            );
    }

    /// <summary>
    /// Gets or sets X component.
    /// </summary>
    public double X
    {
        get
        {
            return this.x;
        }
        set
        {
            this.x = (value>0.9505)? 0.9505 : ((value<0)? 0 : value);
        }
    }

    /// <summary>
    /// Gets or sets Y component.
    /// </summary>
    public double Y
    {
        get
        {
            return this.y;
        }
        set
        {
            this.y = (value>1.0)? 1.0 : ((value<0)?0 : value);
        }
    }

    /// <summary>
    /// Gets or sets Z component.
    /// </summary>
    public double Z
    {
        get
        {
            return this.z;
        }
        set
        {
            this.z = (value>1.089)? 1.089 : ((value<0)? 0 : value);
        }
    }

    public CIEXYZ(double x, double y, double z)
    {
        this.x = (x>0.9505)? 0.9505 : ((x<0)? 0 : x);
        this.y = (y>1.0)? 1.0 : ((y<0)? 0 : y);
        this.z = (z>1.089)? 1.089 : ((z<0)? 0 : z);
    }

    public override bool Equals(Object obj)
    {
        if(obj==null || GetType()!=obj.GetType()) return false;

        return (this == (CIEXYZ)obj);
    }

    public override int GetHashCode()
    {
        return X.GetHashCode() ^ Y.GetHashCode() ^ Z.GetHashCode();
    }

}
```

Well! why do I have to include this model?

I have made a quick research to include **Cie L*a*b*** color model in this article, and I find that a conversion to an absolute color space is required before converting to **L*a*b***. The model used in the conversion principle is **Cie XYZ**. So, I've included it and now everyone can understand "what are those XYZ values" used further in the article.

G - CIE L*a*b* color space

"A Lab color space is a color-opponent space with dimension L for luminance and a and b for the color-opponent dimensions, based on nonlinearly-compressed **CIE XYZ** color space coordinates."

As said in the previous definition, **CIE L*a*b*** color space, also know as **CIE 1976** color space, stands for:

- **L***, the luminance
- **a***, the red/green color-opponent dimension
- **b***, the yellow/blue color-opponent dimension

The **L*a*b*** color model has been created to serve as a device independent model to be used as a reference. It is based directly on the **CIE 1931 XYZ** color space as an attempt to linearize the perceptibility of color differences.

The non-linear relations for L*, a*, and b* are intended to mimic the logarithmic response of the eye, coloring information is referred to the color of the white point of the system.

A **CIE L*a*b*** structure can be:

⊟ Collapse | Copy Code

```
/// <summary>
/// Structure to define CIE L*a*b*.
/// </summary>
public struct CIELab
{
    /// <summary>
    /// Gets an empty CIELab structure.
    /// </summary>
    public static readonly CIELab Empty = new CIELab();

    private double l;
    private double a;
    private double b;


    public static bool operator ==(CIELab item1, CIELab item2)
    {
        return (
            item1.L == item2.L
            && item1.A == item2.A
            && item1.B == item2.B
            );
    }

    public static bool operator !=(CIELab item1, CIELab item2)
    {
        return (
            item1.L != item2.L
            || item1.A != item2.A
            || item1.B != item2.B
            );
    }


    /// <summary>
    /// Gets or sets L component.
    /// </summary>
    public double L
    {
        get
        {
            return this.l;
        }
        set
        {
            this.l = value;
        }
    }

    /// <summary>
    /// Gets or sets a component.
    /// </summary>
    public double A
    {
        get
        {
            return this.a;
        }
        set
        {
            this.a = value;
        }
    }

    /// <summary>
    /// Gets or sets a component.
    /// </summary>
    public double B
    {
        get
        {
            return this.b;
        }
        set
        {
            this.b = value;
        }
    }

    public CIELab(double l, double a, double b)
    {
        this.l = l;
        this.a = a;
        this.b = b;
        this.b = b;
    }

    public override bool Equals(Object obj)
    {
        if(obj==null || GetType()!=obj.GetType()) return false;

        return (this == (CIELab)obj);
    }

    public override int GetHashCode()
    {
        return L.GetHashCode() ^ a.GetHashCode() ^ b.GetHashCode();
    }

}
```

There are still many other formats like RYB and CcMmYK. I still don't intend to create a "color framework", but if you have other ideas...

II - Conversion between models

A - RGB conversions

Converting **RGB** color to any other model is the basis in conversion algorithms. It implies a normalisation of red, green and blue : value ranges now from `[0..255]` to `[0..1]`.

a - RGB to HSB

The conversion principle is the one below:

H ? [0, 360]

S, V, R, G, B ? [0, 1]

$$H = \begin{cases} \text{undefined}, & \text{if } MAX = MIN \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 0^\circ, & \text{if } MAX = R \\ & \text{and } G \geq B \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 360^\circ, & \text{if } MAX = R \\ & \text{and } G < B \\ 60^\circ \times \frac{B-R}{MAX-MIN} + 120^\circ, & \text{if } MAX = G \\ 60^\circ \times \frac{R-G}{MAX-MIN} + 240^\circ, & \text{if } MAX = B \end{cases}$$

$$S = \begin{cases} 0, & \text{if } MAX = 0 \\ 1 - \frac{MIN}{MAX}, & \text{otherwise} \end{cases}$$

V = MAX

Well! Interesting! But what's the C# equivalent? Here it is.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts RGB to HSB.
/// </summary>
public static HSB RGBtoHSB(int red, int green, int blue)
{
    // normalize red, green and blue values
    double r = ((double)red/255.0);
    double g = ((double)green/255.0);
    double b = ((double)blue/255.0);

    // conversion start
    double max = Math.Max(r, Math.Max(g, b));
    double min = Math.Min(r, Math.Min(g, b));

    double h = 0.0;
    if(max==r && g>=b)
    {
        h = 60 * (g-b)/(max-min);
    }
    else if(max==r && g < b)
    {
        h = 60 * (g-b)/(max-min) + 360;
    }
    else if(max == g)
    {
        h = 60 * (b-r)/(max-min) + 120;
    }
    else if(max == b)
    {
        h = 60 * (r-g)/(max-min) + 240;
    }

    double s = (max == 0)? 0.0 : (1.0 - (min/max));

    return new HSB(h, s, (double)max);
}
```

b - RGB to HSL

The conversion principle is the one below:

H ? [0, 360]

S, L, R, G, B ? [0, 1]

$$H = \begin{cases} \text{undefined} & \text{if } MAX = MIN \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 0^\circ, & \text{if } MAX = R \\ & \text{and } G \geq B \\ 60^\circ \times \frac{G-B}{MAX-MIN} + 360^\circ, & \text{if } MAX = R \\ & \text{and } G < B \\ 60^\circ \times \frac{B-R}{MAX-MIN} + 120^\circ, & \text{if } MAX = G \\ 60^\circ \times \frac{R-G}{MAX-MIN} + 240^\circ, & \text{if } MAX = B \end{cases}$$

$$S = \begin{cases} 0 & \text{if } L = 0 \text{ or } MAX = MIN \\ \frac{MAX-MIN}{MAX+MIN} = \frac{MAX-MIN}{2L}, & \text{if } 0 < L \leq \frac{1}{2} \\ \frac{MAX-MIN}{2-(MAX+MIN)} = \frac{MAX-MIN}{2-2L}, & \text{if } L > \frac{1}{2} \end{cases}$$

L = ½(*MAX* + *MIN*)

The C# equivalent is:

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts RGB to HSL.
/// </summary>
/// <param name="red">Red value, must be in [0,255].</param>
/// <param name="green">Green value, must be in [0,255].</param>
/// <param name="blue">Blue value, must be in [0,255].</param>
public static HSL RGBtoHSL(int red, int green, int blue)
{
    double h=0, s=0, l=0;

    // normalize red, green, blue values
    double r = (double)red/255.0;
    double g = (double)green/255.0;
    double b = (double)blue/255.0;

    double max = Math.Max(r, Math.Max(g, b));
    double min = Math.Min(r, Math.Min(g, b));

    // hue
    if(max == min)
    {
        h = 0; // undefined
    }
    else if(max==r && g>=b)
    {
        h = 60.0*(g-b)/(max-min);
    }
    else if(max==r && g<b)
    {
        h = 60.0*(g-b)/(max-min) + 360.0;
    }
    else if(max==g)
    {
        h = 60.0*(b-r)/(max-min) + 120.0;
    }
    else if(max==b)
    {
        h = 60.0*(r-g)/(max-min) + 240.0;
    }

    // luminance
    l = (max+min)/2.0;

    // saturation
    if(l == 0 || max == min)
    {
        s = 0;
    }
    else if(0<l && l<=0.5)
    {
        s = (max-min)/(max+min);
    }
    else if(l>0.5)
    {
        s = (max-min)/(2 - (max+min)); //(max-min > 0)?
    }

    return new HSL(
        Double.Parse(String.Format("{0:0.##}", h)),
        Double.Parse(String.Format("{0:0.##}", s)),
        Double.Parse(String.Format("{0:0.##}", l))
        );
}
```

Note: You have probably noticed `String.Format("{0:0.##}", h)`... It's the .NET solution for keeping the same rounding behavior. If you don't understand what I mean, try the sample code below:

⊟ Collapse | Copy Code

```
Console.WriteLine(Math.Round(4.45, 1)); // returns 4.4.
Console.WriteLine(Math.Round(4.55, 1)); // returns 4.6.
```

You didn't notice a problem? Ok, rouding 4.45 should have returned 4.5 and not 4.4. The solution is using `String.Format()` which always applies "round-to-even" method.

c - RGB to CMYK

The conversion principle is the one below :

R, G, B ? [0, 1]

$t_{C'M'Y'} = \{1 - R, 1 - G, 1 - B\}$

$K = \min\{C', M', Y'\}$

$t_{CMYK} = \{0, 0, 0, 1\} \quad if\ K = 1$

$t_{CMYK} = \{ (C' - K)/(1 - K), (M' - K)/(1 - K), (Y' - K)/(1 - K), K \}$ *otherwise*

The C# equivalent is:

Collapse | Copy Code

```
/// <summary>
/// Converts RGB to CMYK.
/// </summary>
/// <param name="red">Red vaue must be in [0, 255]. </param>
/// <param name="green">Green vaue must be in [0, 255].</param>
/// <param name="blue">Blue vaue must be in [0, 255].</param>
public static CMYK RGBtoCMYK(int red, int green, int blue)
{
    // normalizes red, green, blue values
    double c = (double)(255 - red)/255;
    double m = (double)(255 - green)/255;
    double y = (double)(255 - blue)/255;

    double k = (double)Math.Min(c, Math.Min(m, y));

    if(k == 1.0)
    {
        return new CMYK(0,0,0,1);
    }
    else
    {
        return new CMYK((c-k)/(1-k), (m-k)/(1-k), (y-k)/(1-k), k);
    }
}
```

d - RGB to YUV (YUV444)

The conversion principle is the one below :

R, G, B, Y ? [0, 1]

U ? [-0.436, 0.436]

V ? [-0.615, 0.615]

$t_{YUV} = \{ (0.299\ R + 0.587\ G + 0.114\ B), (- 0.14713\ R + 0.28886\ G + 0.436\ B), (0.615\ R + 0.51499\ G + 0.10001\ B) \}$

The C# equivalent is :

Collapse | Copy Code

```
/// <summary>
/// Converts RGB to YUV.
/// </summary>
/// <param name="red">Red must be in [0, 255].</param>
/// <param name="green">Green must be in [0, 255].</param>
/// <param name="blue">Blue must be in [0, 255].</param>
public static YUV RGBtoYUV(int red, int green, int blue)
{
    YUV yuv = new YUV();

    // normalizes red, green, blue values
    double r = (double)red/255.0;
    double g = (double)green/255.0;
    double b = (double)blue/255.0;

    yuv.Y = 0.299*r + 0.587*g + 0.114*b;
    yuv.U = -0.14713*r -0.28886*g + 0.436*b;
    yuv.V = 0.615*r -0.51499*g -0.10001*b;

    return yuv;
}
```

e - RGB to web color

Haaa! Something I can explain.

As you probably already know, web colors can be defined in two ways: for example, "red" can be defined as `rgb(255,0,0)` or `#FF0000`.

The explanation of the second form is simple :

- "#" character tells that the format is the hexadecimal one.
- The last 6 characters define 3 pairs: one for "Red", one for "Green" and one for "Blue".
- Each pair is a hexadecimal value (base 16) of a value which ranges from 0 to 255.

So, you can divide each color component by 16 and replace numbers superior to 9 by theirs hexadecimal value (eg. 10 = A, 11 = B, etc.)... but the best way is to use `String.Format()` habilities.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts a RGB color format to an hexadecimal color.
/// </summary>
/// <param name="r">Red value.</param>
/// <param name="g">Green value.</param>
/// <param name="b">Blue value.</param>
public static string RGBToHex(int r, int g, int b)
{
    return String.Format("#{0:x2}{1:x2}{2:x2}", r, g, b).ToUpper();
}
```

f - RGB to XYZ

The conversion principle is the one below:

R, G, B ? [0, 1]

a = 0.055 and ? ˜ 2.2

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.4124 & 0.3576 & 0.1805 \\ 0.2126 & 0.7152 & 0.0722 \\ 0.0193 & 0.1192 & 0.9505 \end{bmatrix} \begin{bmatrix} g(R_{srgb}) \\ g(G_{srgb}) \\ g(B_{srgb}) \end{bmatrix}$$

where

$$g(K) = \begin{cases} \left(\frac{K+a}{1+a}\right)^{\gamma}, & K > 0.04045 \\ \frac{K}{12.92}, & otherwise \end{cases}$$

The C# equivalent is :

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts RGB to CIE XYZ (CIE 1931 color space)
/// </summary>
public static CIEXYZ RGBtoXYZ(int red, int green, int blue)
{
    // normalize red, green, blue values
    double rLinear = (double)red/255.0;
    double gLinear = (double)green/255.0;
    double bLinear = (double)blue/255.0;

    // convert to a sRGB form
    double r = (rLinear > 0.04045)? Math.Pow((rLinear + 0.055)/(
        1 + 0.055), 2.2) : (rLinear/12.92) ;
    double g = (gLinear > 0.04045)? Math.Pow((gLinear + 0.055)/(
        1 + 0.055), 2.2) : (gLinear/12.92) ;
    double b = (bLinear > 0.04045)? Math.Pow((bLinear + 0.055)/(
        1 + 0.055), 2.2) : (bLinear/12.92) ;

    // converts
    return new CIEXYZ(
        (r*0.4124 + g*0.3576 + b*0.1805),
        (r*0.2126 + g*0.7152 + b*0.0722),
        (r*0.0193 + g*0.1192 + b*0.9505)
        );
}
```

g - RGB to L*a*b*

As I said before, converting to the **CIE L*a*b** color model is a little bit tricky: we need to convert to **CIE XYZ** before trying to have **L*a*b*** values.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts RGB to CIELab.
/// </summary>
public static CIELab RGBtoLab(int red, int green, int blue)
{
    return XYZtoLab( RGBtoXYZ(red, green, blue) );
}
```

The conversion between **XYZ** and **L*a*b*** is given below.

B - HSB conversions

a - HSB to RGB

The conversion principle is the one below :

H ? [0, 360]

S, V, R, G, B ? [0, 1]

$H_i = [H / 60] \bmod 6$

$f = (H / 60) - H_i$

$p = V (1 - S)$

$q = V (1 - f S)$

$t = V (1 - (1 - f) S)$

if $H_i = 0$? $R = V$, $G = t$, $B = p$
if $H_i = 1$? $R = q$, $G = V$, $B = p$
if $H_i = 2$? $R = p$, $G = V$, $B = t$
if $H_i = 3$? $R = p$, $G = q$, $B = V$
if $H_i = 4$? $R = t$, $G = p$, $B = V$
if $H_i = 5$? $R = V$, $G = p$, $B = q$

The C# equivalent? Here it is.

Collapse | Copy Code

```
/// <summary>
/// Converts HSB to RGB.
/// </summary>
public static RGB HSBtoRGB(double h, double s, double b)
{
    double r = 0;
    double g = 0;
    double b = 0;

    if(s == 0)
    {
        r = g = b = b;
    }
    else
    {
        // the color wheel consists of 6 sectors. Figure out which sector
        // you're in.
        double sectorPos = h / 60.0;
        int sectorNumber = (int)(Math.Floor(sectorPos));
        // get the fractional part of the sector
        double fractionalSector = sectorPos - sectorNumber;

        // calculate values for the three axes of the color.
        double p = b * (1.0 - s);
        double q = b * (1.0 - (s * fractionalSector));
        double t = b * (1.0 - (s * (1 - fractionalSector)));

        // assign the fractional colors to r, g, and b based on the sector
        // the angle is in.
        switch(sectorNumber)
        {
            case 0:
                r = b;
                g = t;
                b = p;
                break;
            case 1:
                r = q;
                g = b;
                b = p;
                break;
            case 2:
                r = p;
                g = b;
                b = t;
                break;
            case 3:
                r = p;
                g = q;
                b = b;
                break;
            case 4:
                r = t;
                g = p;
                b = b;
                break;
            case 5:
                r = b;
                g = p;
                b = q;
                break;
        }
    }

    return new RGB(
        Convert.ToInt32( Double.Parse(String.Format("{0:0.00}", r*255.0)) ),
        Convert.ToInt32( Double.Parse(String.Format("{0:0.00}", g*255.0)) ),
        Convert.ToInt32( Double.Parse(String.Format("{0:0.00}", b*255.0)) )
    );
}
```

b - HSB to HSL

Conversion principle is quite simple (but not accurate): convert to **RGB** and then to **HSL**.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts HSB to HSL.
/// </summary>
public static HSL HSBtoHSL(double h, double s, double b)
{
    RGB rgb = HSBtoRGB(h, s, b);

    return RGBtoHSL(rgb.Red, rgb.Green, rgb.Blue);
}
```

c - HSB to CMYK

Nothing new : conversion principle is to convert to **RGB** and then to **CMYK**.

Collapse | Copy Code

```
/// <summary>
/// Converts HSB to CMYK.
/// </summary>
public static CMYK HSBtoCMYK(double h, double s, double b)
{
    RGB rgb = HSBtoRGB(h, s, b);

    return RGBtoCMYK(rgb.Red, rgb.Green, rgb.Blue);
}
```

d - HSB to YUV

Nothing new : conversion principle is to convert to **RGB** and then to **YUV**.

Collapse | Copy Code

```
/// <summary>
/// Converts HSB to CMYK.
/// </summary>
public static YUV HSBtoYUV(double h, double s, double b)
{
    RGB rgb = HSBtoRGB(h, s, b);

    return RGBtoYUV(rgb.Red, rgb.Green, rgb.Blue);
}
```

C - HSL conversions

a - HSL to RGB

The conversion principle is the one below :

H ? [0, 360]

S, L, R, G, B ? [0, 1]

if $L < 0.5$? $Q = L \times (1 + S)$
if $L = 0.5$? $Q = L + S - (L \times S)$

$P = 2 \times L - Q$

$H_k = H / 360$

$T_r = H_k + 1/3$

$T_g = H_k$

$T_b = H_k - 1/3$

For each c = R,G,B :

if $T_c < 0$? $T_c = T_c + 1.0$
if $T_c > 1$? $T_c = T_c - 1.0$

if $T_c < 1/6$? $T_c = P + ((Q - P) \times 6.0 \times T_c)$
if $1/6 = T_c > 1/2$? $T_c = Q$
if $1/2 = T_c > 2/3$? $T_c = P + ((Q - P) \times (2/3 - T_c) \times 6.0)$
else $T_c = P$

The C# equivalent? Here it is.

Collapse | Copy Code

```
/// <summary>
/// Converts HSL to RGB.
/// </summary>
/// <param name="h">Hue, must be in [0, 360].</param>
/// <param name="s">Saturation, must be in [0, 1].</param>
/// <param name="l">Luminance, must be in [0, 1].</param>
public static RGB HSLtoRGB(double h, double s, double l)
{
    if(s == 0)
    {
        // achromatic color (gray scale)
        return new RGB(
            Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
                l*255.0)) ),
            Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
                l*255.0)) ),
            Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
                l*255.0)) )
            );
    }
    else
    {
        double q = (l<0.5)?(l * (1.0+s)):(l+s - (l*s));
        double p = (2.0 * l) - q;

        double Hk = h/360.0;
        double[] T = new double[3];
        T[0] = Hk + (1.0/3.0);    // Tr
        T[1] = Hk;                // Tb
        T[2] = Hk - (1.0/3.0);    // Tg

        for(int i=0; i<3; i++)
        {
            if(T[i] < 0) T[i] += 1.0;
            if(T[i] > 1) T[i] -= 1.0;

            if((T[i]*6) < 1)
            {
                T[i] = p + ((q-p)*6.0*T[i]);
            }
            else if((T[i]*2.0) < 1) //(1.0/6.0)<=T[i] && T[i]<0.5
            {
                T[i] = q;
            }
            else if((T[i]*3.0) < 2) // 0.5<=T[i] && T[i]<(2.0/3.0)
            {
                T[i] = p + (q-p) * ((2.0/3.0) - T[i]) * 6.0;
            }
            else T[i] = p;
        }

        return new RGB(
            Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
                T[0]*255.0)) ),
            Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
                T[1]*255.0)) ),
            Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
                T[2]*255.0)) )
            );
    }
}
```

b - HSL to HSB

Nothing new: conversion principle is to convert to **RGB** and then to **HSB**.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts HSL to HSB.
/// </summary>
public static HSB HSLtoHSB(double h, double s, double l)
{
    RGB rgb = HSLtoRGB(h, s, l);

    return RGBtoHSB(rgb.Red, rgb.Green, rgb.Blue);
}
```

c - HSL to CMYK

Nothing new: conversion principle is to convert to **RGB** and then to **CMYK**.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts HSL to CMYK.
/// </summary>
public static CMYK HSLtoCMYK(double h, double s, double l)
{
    RGB rgb = HSLtoRGB(h, s, l);

    return RGBtoCMYK(rgb.Red, rgb.Green, rgb.Blue);
}
```

d - HSL to YUV

Nothing new: conversion principle is to convert to **RGB** and then to **CMYK**.

Collapse | Copy Code

```
/// <summary>
/// Converts HSL to YUV.
/// </summary>
public static YUV HSLtoYUV(double h, double s, double l)
{
    RGB rgb = HSLtoRGB(h, s, l);

    return RGBtoYUV(rgb.Red, rgb.Green, rgb.Blue);
}
```

D - CMYK conversions

a - CMYK to RGB

The conversion principle is the one below:

$$t_{RGB} = \{ (1 - C) \times (1 - K) , (1 - M) \times (1 - K), (1 - Y) \times (1 - K)\}$$

The C# equivalent? Here it is.

Collapse | Copy Code

```
/// <summary>
/// Converts CMYK to RGB.
/// </summary>
public static Color CMYKtoRGB(double c, double m, double y, double k)
{
    int red = Convert.ToInt32((1-c) * (1-k) * 255.0);
    int green = Convert.ToInt32((1-m) * (1-k) * 255.0);
    int blue = Convert.ToInt32((1-y) * (1-k) * 255.0);

    return Color.FromArgb(red, green, blue);
}
```

b - CMYK to HSL

Nothing new: conversion principle is to convert to **RGB** and then to **HSL**.

Collapse | Copy Code

```
/// <summary>
/// Converts CMYK to HSL.
/// </summary>
public static HSL CMYKtoHSL(double c, double m, double y, double k)
{
    RGB rgb = CMYKtoRGB(c, m, y, k);

    return RGBtoHSL(rgb.Red, rgb.Green, rgb.Blue);
}
```

c - CMYK to HSB

Nothing new: conversion principle is to convert to **RGB** and then to **HSB**.

Collapse | Copy Code

```
/// <summary>
/// Converts CMYK to HSB.
/// </summary>
public static HSB CMYKtoHSB(double c, double m, double y, double k)
{
    RGB rgb = CMYKtoRGB(c, m, y, k);

    return RGBtoHSB(rgb.Red, rgb.Green, rgb.Blue);
}
```

d - CMYK to YUV

Nothing new: conversion principle is to convert to **RGB** and then to **YUV**.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts CMYK to YUV.
/// </summary>
public static YUV CMYKtoYUV(double c, double m, double y, double k)
{
    RGB rgb = CMYKtoRGB(c, m, y, k);

    return RGBtoYUV(rgb.Red, rgb.Green, rgb.Blue);
}
```

E - YUV conversions

a - YUV to RGB

The conversion principle is the one below:

R, G, B, Y ? $[0, 1]$

U ? $[-0.436, 0.436]$

V ? $[-0.615, 0.615]$

$t_{RGB} = \{ (Y + 1.13983\ V), (Y - 0.39466\ U - 0.58060\ V), (Y + 2.03211\ U) \}$

The C# equivalent is :

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts YUV to RGB.
/// </summary>
/// <param name="y">Y must be in [0, 1].</param>
/// <param name="u">U must be in [-0.436, +0.436].</param>
/// <param name="v">V must be in [-0.615, +0.615].</param>
public static RGB YUVtoRGB(double y, double u, double v)
{
    RGB rgb = new RGB();

    rgb.Red = Convert.ToInt32((y + 1.139837398373983740*v)*255);
    rgb.Green = Convert.ToInt32((
        y - 0.39465170435897035 15*u - 0.58059860666749768 01*v)*255);
    rgb.Blue = Convert.ToInt32((y + 2.03211009174311926 6*u)*255);

    return rgb;
}
```

b - YUV to HSL

Nothing new: conversion principle is to convert to **RGB** and then to **HSL**.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts YUV to HSL.
/// </summary>
/// <param name="y">Y must be in [0, 1].</param>
/// <param name="u">U must be in [-0.436, +0.436].</param>
/// <param name="v">V must be in [-0.615, +0.615].</param>
public static HSL YUVtoHSL(double y, double u, double v)
{
    RGB rgb = YUVtoRGB(y, u, v);

    return RGBtoHSL(rgb.Red, rgb.Green, rgb.Blue);
}
```

c - YUV to HSB

Nothing new: conversion principle is to convert to **RGB** and then to **HSB**.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts YUV to HSB.
/// </summary>
/// <param name="y">Y must be in [0, 1].</param>
/// <param name="u">U must be in [-0.436, +0.436].</param>
/// <param name="v">V must be in [-0.615, +0.615].</param>
public static HSB YUVtoHSB(double y, double u, double v)
{
    RGB rgb = YUVtoRGB(y, u, v);

    return RGBtoHSB(rgb.Red, rgb.Green, rgb.Blue);
}
```

d - YUV to CMYK

Nothing new: conversion principle is to convert to **RGB** and then to **CMYK**.

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts YUV to CMYK.
/// </summary>
/// <param name="y">Y must be in [0, 1].</param>
/// <param name="u">U must be in [-0.436, +0.436].</param>
/// <param name="v">V must be in [-0.615, +0.615].</param>
public static CMYK YUVtoCMYK(double y, double u, double v)
{
    RGB rgb = YUVtoRGB(y, u, v);

    return RGBtoCMYK(rgb.Red, rgb.Green, rgb.Blue);
}
```

F - XYZ conversions

a - XYZ to RGB

The conversion principle is the one below:

$$a = 0.055$$

$$\begin{bmatrix} R_{linear} \\ G_{linear} \\ B_{linear} \end{bmatrix} = \begin{bmatrix} 3.2410 & -1.5374 & -0.4986 \\ -0.9692 & 1.8760 & 0.0416 \\ 0.0556 & -0.2040 & 1.0570 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

then

$$C_{srgb} = \begin{cases} 12.92 C_{linear}, & C_{linear} <= 0.0031308 \\ (1+a) C_{linear}^{1/2.4} - a, & C_{linear} > 0.0031308 \end{cases}$$

The C# equivalent is :

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts CIEXYZ to RGB structure.
/// </summary>
public static RGB XYZtoRGB(double x, double y, double z)
{
    double[] Clinear = new double[3];
    Clinear[0] = x*3.2410 - y*1.5374 - z*0.4986; // red
    Clinear[1] = -x*0.9692 + y*1.8760 - z*0.0416; // green
    Clinear[2] = x*0.0556 - y*0.2040 + z*1.0570; // blue

    for(int i=0; i<3; i++)
    {
        Clinear[i] = (Clinear[i]<=0.0031308)? 12.92*Clinear[i] : (
            1+0.055)* Math.Pow(Clinear[i], (1.0/2.4)) - 0.055;
    }

    return new RGB(
        Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
            Clinear[0]*255.0)) ),
        Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
            Clinear[1]*255.0)) ),
        Convert.ToInt32( Double.Parse(String.Format("{0:0.00}",
            Clinear[2]*255.0)) )
        );
}
```

b - XYZ to L*a*b*

The conversion principle is the one below:

$$L^* = 116\, f(Y/Y_n) - 16$$
$$a^* = 500\,[f(X/X_n) - f(Y/Y_n)]$$
$$b^* = 200\,[f(Y/Y_n) - f(Z/Z_n)]$$

where

$f(t) = t^{1/3}$ for $t > 0.008856$

$f(t) = 7.787\,t + 16/116$ otherwise

Xn, Yn and Zn are the CIE XYZ tristimulus values of the reference white point.

The C# equivalent is:

Collapse | Copy Code

```
/// <summary>
/// XYZ to L*a*b* transformation function.
/// </summary>
private static double Fxyz(double t)
{
    return ((t > 0.008856)? Math.Pow(t, (1.0/3.0)) : (7.787*t + 16.0/116.0));
}

/// <summary>
/// Converts CIEXYZ to CIELab.
/// </summary>
public static CIELab XYZtoLab(double x, double y, double z)
{
    CIELab lab = CIELab.Empty;

    lab.L = 116.0 * Fxyz( y/CIEXYZ.D65.Y ) -16;
    lab.A = 500.0 * (Fxyz( x/CIEXYZ.D65.X ) - Fxyz( y/CIEXYZ.D65.Y) );
    lab.B = 200.0 * (Fxyz( y/CIEXYZ.D65.Y ) - Fxyz( z/CIEXYZ.D65.Z) );

    return lab;
}
```

G - L*a*b* conversions

a - L*a*b* to XYZ

The conversion principle is the one below:

$d = 6/29$

$$f_y \overset{\text{def}}{=} (L^* + 16)/116$$
$$f_x \overset{\text{def}}{=} f_y + a^*/500$$
$$f_z \overset{\text{def}}{=} f_y - b^*/200$$

if $f_y > \delta$ then $Y = Y_n f_y^3$ else $Y = (f_y - 16/116)3\delta^2 Y_n$

if $f_x > \delta$ then $X = X_n f_x^3$ else $X = (f_x - 16/116)3\delta^2 X_n$

if $f_z > \delta$ then $Z = Z_n f_z^3$ else $Z = (f_z - 16/116)3\delta^2 Z_n$

The C# equivalent is:

Collapse | Copy Code

```
/// <summary>
/// Converts CIELab to CIEXYZ.
/// </summary>
public static CIEXYZ LabtoXYZ(double l, double a, double b)
{
    double delta = 6.0/29.0;

    double fy = (l+16)/116.0;
    double fx = fy + (a/500.0);
    double fz = fy - (b/200.0);

    return new CIEXYZ(
        (fx > delta)? CIEXYZ.D65.X * (fx*fx*fx) : (fx - 16.0/116.0)*3*(
            delta*delta)*CIEXYZ.D65.X,
        (fy > delta)? CIEXYZ.D65.Y * (fy*fy*fy) : (fy - 16.0/116.0)*3*(
            delta*delta)*CIEXYZ.D65.Y,
        (fz > delta)? CIEXYZ.D65.Z * (fz*fz*fz) : (fz - 16.0/116.0)*3*(
            delta*delta)*CIEXYZ.D65.Z
        );
}
```

b - L*a*b* to RGB

Nothing really new, the principle is to convert to **XYZ** and then to **RGB** :

⊟ Collapse | Copy Code

```
/// <summary>
/// Converts CIELab to RGB.
/// </summary>
public static RGB LabtoRGB(double l, double a, double b)
{
    return XYZtoRGB( LabtoXYZ(l, a, b) );
}
```

Using the code

Well, after showing you the conversion algorithms, maybe there is nothing more I can tell you.

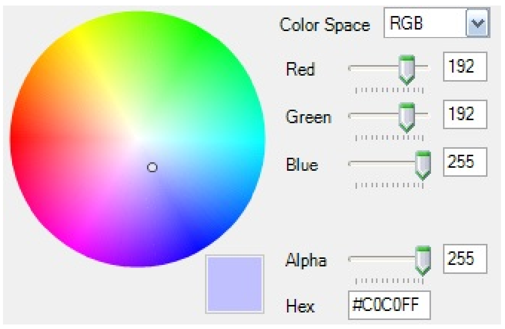
In fact, there are many other useful methods in `ColorSpaceHelper`. You will find:

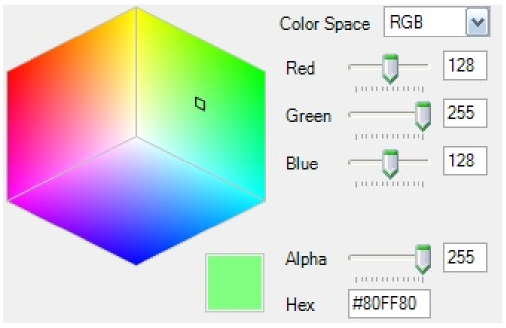
- Average color implementation (`ColorSpaceHelper.GetColorDistance()`).
- Wheel color generation (`ColorSpaceHelper.GetWheelColors()`) with 32bit support (alpha).
- Light spectrum color generation for (`ColorSpaceHelper.GetSpectrumColors()`) with 32bit support (alpha).
- Conversion to and from web colors (`ColorSpaceHelper.HexToColor()`).
- Conversion to and from `System.Drawing.Color` and the other structures.

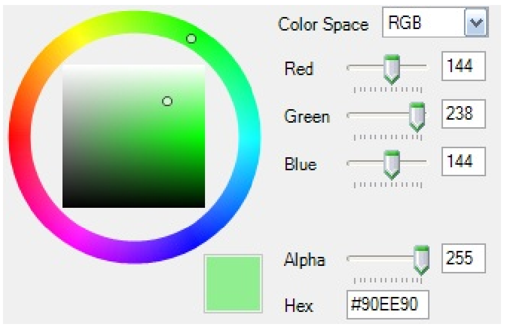
Points of Interest

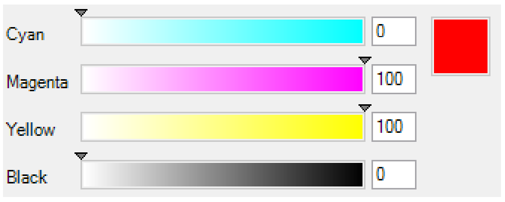
Like I said before, there are many formats. This article gives you an example of how using common color models (**RGB**, **HSL**, **HSB**, etc.) perhaps will help you in your own projects.

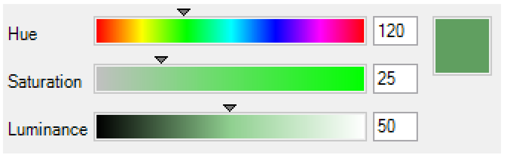
Here is a preview of what we can do (and what we will see in the next article) with `ColorSpaceHelper`:


Color Space: RGB
Red: 192
Green: 192
Blue: 255
Alpha: 255
Hex: #C0C0FF


Color Space RGB
Red 128
Green 255
Blue 128
Alpha 255
Hex #80FF80


Color Space RGB
Red 144
Green 238
Blue 144
Alpha 255
Hex #90EE90


Cyan 0
Magenta 100
Yellow 100
Black 0


Hue 120
Saturation 25
Luminance 50

History

- 08 June 2007
    - Added GraGra_33 VB .NET version.
- 07 June 2007
    - Added **CIE XYZ** color space definition, and conversion methods in `ColorSpaceHelper`.
    - Added **CIE L*a*b*** color space definition, and conversion methods in `ColorSpaceHelper`.
- 03 June 2007 - Initial article.

License

Share

About the Author

Guillaume Leparmentier

Engineer
France

IT consultant and Project Manager in Paris, specialized in software engineering/design.

He spends most of his time in meetings
He would love to have more time to develop all those ideas/concepts he has in mind.

Comments and Discussions

You must Sign In to use this message board.

Search Comments Go

☑ Profile popups Spacing Relaxed Noise Medium Layout Open All Per page 50 Update

First Prev Next

Just what I have been looking for, thanks :) Jab_RTS 23-Jul-14 12:07

Has to be the best full description I have seen, thanks.

Jim

Sign In · View Thread · Permalink

Small bug in RGBtoHSB() Member 8419326 27-Jan-14 21:15

First of all, great stuff. it realy helped me on my philps hue light project

i did notice a small issue while testing,
if reg & green & blue are all the same value
the code will return new HSB(NaN, NaN, max);
since it will cause a 0/0

to fix this, you can do a
if(r == g && g == b) return new HSB(0, 0, max);

Sign In · View Thread · Permalink

Re: Small bug in RGBtoHSB() Paul Effect[the real one] 28-Sep-14 2:35

Thank you!

Sign In · View Thread · Permalink

doesn't work markentingh 9-Nov-13 21:50

When trying to use ColorHelper.GetBlendColor, the R, G, & B values return 1. I believe your code uses values between 0 and 1, when the color object in VB.net (4.5) uses values between 0 and 255.

Here is my debug info

color1 = {Color [A=255, R=238, G=251, B=255]}
color2 = {Color [A=255, R=153, G=232, B=255]}
blended color = {Color [A=255, R=1, G=1, B=1]}

Sign In · View Thread · Permalink

Excellent - thank you! One question I have... Hans Loepfe 22-Oct-13 14:03

Hello Guillaume,
thank you for your comprehensive insight into the various models for displaying color(s).
You seem to be the right person to ask how to determine the average brightness of a grayscale image (sRGB)?

To begin we have a linear RAW image (Nikon NEF), 16-bit, sRGB, gamma 0.454545. This image contains the same as preview image 570 x 375 pixels, 8-bit, sRGB, gamma 0.454545. That preview image gets extracted and converted to grayscale (rec709luma).
How can the average or overall brightness of that preview image be determined?
Hope this request will ever reach you.
Looking forward to receiving your reply
With kind regards, Hans.

Sign In · View Thread · Permalink

Brilliant cinamon 20-Sep-13 11:13

I work in the graphics & printing business with a technical job and this is the best explanation of colour models for computers!
5 out of 5!

Sign In · View Thread · Permalink

Luminace zeebedee 12-Jun-13 8:01

Hi, Gr8 article & code.

I am trying to work out by using Saturation and Luminance whether an image has been taken at night time or day time. When i have a lot of white light from say a large moth close to a camera the luminance (overall) goes up making my code thinks it is daytime. Is HSL the best color model to use in this case? Should I look at another one to use?

Thanks

Sign In · View Thread · Permalink 5.00/5 (1 vote)

please fix this code [modified] yurtan 17-Mar-13 1:41

```
public static RGB HSBtoRGB(double h, double s, double b)
{                                                      ^
    double r = 0;                                      |
    double g = 0;                                      |
    double b = 0;  <---------one variable for two different parameters
    .......
   and so on
```

modified 17-Mar-13 8:13am.

Sign In · View Thread · Permalink

Quick (maybe obvious) note on CMYK xirisjohn 13-Feb-13 11:40

Great article!

Of course when converting to CMYK, there is always an extra degree of freedom, mathematically speaking. In your RGB to CMYK formula you show the normal assumption that K is min(r,g,b), but people should be aware that there is "more than one right answer". The mathematical result may not correspond to the values from some software used to make the color separation plates used in offset printing press setup, for example. Some graphic designers may prefer always having "more K" to get what they perceive to be a "better" result.

Sign In · View Thread · Permalink

My vote of 5 Kanasz Robert 6-Nov-12 0:10

great one

Sign In · View Thread · Permalink

Text File Export kittell 30-Oct-12 7:36

I love this project, as it has saved me a lot of time.

I apologize if someone else has already posted this but I had a need to see the other values with the HEX as the source so I added a button and the following code:

```
private void button1_Click(object sender, EventArgs e)
   {
      Color ConvertedColor = ColorTranslator.FromHtml(hexBox.Text);
      redUD.Value = ConvertedColor.R;
      greenUD.Value = ConvertedColor.G;
      blueUD.Value = ConvertedColor.B;

      using (StreamWriter sw = File.AppendText("ColorConversion.txt"))
         {
            sw.WriteLine("Color Conversion - " + DateTime.Now.ToString());
            sw.WriteLine("HEX: " + hexBox.Text);
            sw.WriteLine("RGB: " + redUD.Value.ToString() + "," +
greenUD.Value.ToString() + "," + blueUD.Value.ToString());
            sw.WriteLine("HSL: " + hueUD.Value.ToString() + "," +
satUD.Value.ToString() + "," + lumUD.Value.ToString());
            sw.WriteLine("HSB: " + hUD.Value.ToString() + "," + sUD.Value.ToString()
+ "," + bUD.Value.ToString());
            sw.WriteLine("CMYK: " + cyanUD.Value.ToString() + "," +
magentaUD.Value.ToString() + "," + yellowUD.Value.ToString() + "," +
blackUD.Value.ToString());
            sw.WriteLine("YUV: " + yUD.Value.ToString() + "," + uUD.Value.ToString()
+ "," + vUD.Value.ToString());

         }
}
```

The content of the text file looks like this

```
Color Conversion - 10/30/2012 1:32:11 PM
HEX: #c47233
RGB: 196,114,51
HSL: 26,59,48
HSB: 26,74,77
CMYK: 4,44,75,20
YUV: 52,28,84
```

As the RGB has the update process setup based on changed values this will update the rest of the values as if I manually set the RGB.

Please feel free to use/improve my code.

David Kittell
Kittell.net

Sign In · View Thread · Permalink

sRGB and standard Color class [modified] GlebKudr 5-Aug-12 4:34

If you need to convert from standard aRGB Color class in .net, you should notice that Alpha component from this is not an alpha component from sRGB standard.

Killed a whole evening to realize this

To deal with Alpha in Color class you should convert it to simple RGB removing background "alpha" component first.

```
        public static Color RemoveAlpha(Color argbColor, Color background)
        {
            if (argbColor.A == 255)
                return argbColor;

            var alpha = argbColor.A / 255.0;
            var diff = 1.0 - alpha;
            return Color.FromArgb(255,
                (byte)(argbColor.R * alpha + background.R * diff),
                (byte)(argbColor.G * alpha + background.G * diff),
                (byte)(argbColor.B * alpha + background.B * diff));
        }
```

modified 6-Aug-12 13:10pm.

Sign In · View Thread · Permalink

My vote of 5 **BoneSoft** **11-May-12 7:07**

Fantastic work. Thank you thank you!

Sign In · View Thread · Permalink

My vote of 5 **RichM** **8-May-12 5:14**

I could never have figured out CIELAB by myself.

Sign In · View Thread · Permalink

Great work! **ColinCren** **26-Mar-12 17:13**

For a research topic I need to do conversations between some of these color spaces. I'm glad I had your code available to me so I didn't have to re-tread ground already covered. This saved me a ton of time. Thank you so much.

Sign In · View Thread · Permalink

my vote 5 **Recan_Miracle** **26-Feb-12 20:51**

Very useful. Thanks a million.

Sign In · View Thread · Permalink

My vote of 5 **xyzabc123321** **5-Feb-12 15:29**

Nice!

Sign In · View Thread · Permalink

My vote of 5 **ZiniDaTiKazem2** **11-Jan-12 12:51**

He has celarly explained a great number of very complex color systems converions.

Sign In · View Thread · Permalink

bug in XYZ to RGB **EvgenyTr** **12-Nov-11 6:46**

Clinear[1] = -x*0.9692 + y*1.8760 - z*0.0416; // green

should be Clinear[1] = -x*0.9692 + y*1.8760 **+** z*0.0416; // green

Sign In · View Thread · Permalink 5.00/5 (1 vote)

Color Distance

Eric Ouellet 8-Sep-11 8:32

Hello,

I'm trying to create a color generator for a graphic control where there could be any number a plots drawn.
For each new plot, I want to have the most different color possible while keeping existing plot color.

I took a look at ColorSpaceHelper:GetColorDistance and see that you do a simple calc. But my readings give me the impression that it is better to use CIELab to get a good color space of human perception of color (and a more accurate distance between colors). I wonder if it is not better to calc distance based on CIELab color and if so, how I can achieved that with you library if possible ?

Thanks,
Eric

Sign In · View Thread · Permalink

Re: Color Distance

Rob2412 3-Oct-11 3:42

Here is the code for CIE 2000 color distance:

```
/// <summary>
        /// Returns the color difference (distance) between a sample color 
CIELap(2) and a reference color CIELap(1)
        /// <para>in accorance with CIE 2000 alogorithm.</para>
        /// </summary>
        /// <param name="lab1">CIELap reference color.</param>
        /// <param name="lab2">CIELap sample color.</param>
        /// <returns>Color difference.</returns>
        public static float ColorDifference(CIELab lab1, CIELab lab2)
        {
            double p25 = Math.Pow(25, 7);

            double C1 = Math.Sqrt(lab1.A * lab1.A + lab1.B * lab1.B);
            double C2 = Math.Sqrt(lab2.A * lab2.A + lab2.B * lab2.B);
            double avgC = (C1 + C2) / 2F;

            double powAvgC = Math.Pow(avgC, 7);
            double G = (1 - Math.Sqrt(powAvgC / (powAvgC + p25))) / 2D;

            double a_1 = lab1.A * (1 + G);
            double a_2 = lab2.A * (1 + G);

            double C_1 = Math.Sqrt(a_1 * a_1 + lab1.B * lab1.B);
            double C_2 = Math.Sqrt(a_2 * a_2 + lab2.B * lab2.B);
            double avgC_ = (C_1 + C_2) / 2D;

            double h1 = (Atan(lab1.B, a_1) >= 0 ? Atan(lab1.B, a_1) : Atan(lab1.B, 
a_1) + 360F);
            double h2 = (Atan(lab2.B, a_2) >= 0 ? Atan(lab2.B, a_2) : Atan(lab2.B, 
a_2) + 360F);

            double H = (h1 - h2 > 180D ? (h1 + h2 + 360F) / 2D : (h1 + h2) / 2D);

            double T = 1;
            T -= 0.17 * Cos(H - 30);
            T += 0.24 * Cos(2 * H);
            T += 0.32 * Cos(3 * H + 6);
            T -= 0.20 * Cos(4 * H - 63);

            double deltah = 0;
            if (h2 - h1 <= 180)
                deltah = h2 - h1;
            else if (h2 <= h1)
                deltah = h2 - h1 + 360;
            else
                deltah = h2 - h1 - 360;

            double avgL = (lab1.L + lab2.L) / 2F;
            double deltaL_ = lab2.L - lab1.L;
            double deltaC_ = C_2 - C_1;
            double deltaH_ = 2 * Math.Sqrt(C_1 * C_2) * Sin(deltah / 2);

            double SL = 1 + (0.015 * Math.Pow(avgL - 50, 2)) / Math.Sqrt(20 + 
Math.Pow(avgL - 50, 2));
```

```
            double SC = 1 + 0.045 * avgC_;
            double SH = 1 + 0.015 * avgC_ * T;

            double exp = Math.Pow((H - 275) / 25, 2);
            double teta = Math.Pow(30, -exp);

            double RC = 2D * Math.Sqrt(Math.Pow(avgC_, 7) / (Math.Pow(avgC_, 7) +
p25));
            double RT = -RC * Sin(2 * teta);

            double deltaE = 0;
            deltaE = Math.Pow(deltaL_ / SL, 2);
            deltaE += Math.Pow(deltaC_ / SC, 2);
            deltaE += Math.Pow(deltaH_ / SH, 2);
            deltaE += RT * (deltaC_ / SC) * (deltaH_ / SH);
            deltaE = Math.Sqrt(deltaE);

            return (float)deltaE;
        }

        /// <summary>
        /// Returns the angle in degree whose tangent is the quotient of the two
specified numbers.
        /// </summary>
        /// <param name="y">The y coordinate of a point.</param>
        /// <param name="x">The x coordinate of a point.</param>
        /// <returns>Angle in degree.</returns>
        private static double Atan(double y, double x)
        {
            return Math.Atan2(y, x) * 180D / Math.PI;
        }

        /// <summary>
        /// Returns the cosine of the specified angle in degree.
        /// </summary>
        /// <param name="d">Angle in degree</param>
        /// <returns>Cosine of the specified angle.</returns>
        private static double Cos(double d)
        {
            return Math.Cos(d * Math.PI / 180);
        }

        /// <summary>
        /// Returns the sine of the specified angle in degree.
        /// </summary>
        /// <param name="d">Angle in degree</param>
        /// <returns>Sine of the specified angle.</returns>
        private static double Sin(double d)
        {
            return Math.Sin(d * Math.PI / 180);
        }
```

Sign In · View Thread · Permalink

Re: Color Distance

Eric Ouellet 3-Oct-11 6:42

Thanks a lot Rob,

In fact I found an algorithm and translated in c# but I think is is based on "Delta E 1994".
Are you aware if your algorithm is better and if the changes are significant ?

This is the algo I have found/used in my post:
Random Color Generator

```
// ******************************************************************
// EO: Based on: http://www.easyrgb.com/index.php?X=DELT&H=04#text4
// Based on: Delta E 1994
public static double GetDistanceBetween(CIELab lab1, CIELab lab2)
{
 const double whtL = 1;
 const double whtC = 1;
 const double whtH = 1;               //Weighting factors depending
          //on the application (1 = default)

 double xC1 = Math.Sqrt(Math.Pow(lab1.a, 2.0) + Math.Pow(lab1.b, 2.0));
 double xC2 = Math.Sqrt(Math.Pow(lab2.a, 2.0) + Math.Pow(lab2.b, 2.0));
 double xDL = lab2.l - lab1.l;
 double xDC = xC2 - xC1;
 double xDE = Math.Sqrt(
  ((lab1.l - lab2.l) * (lab1.l - lab2.l))
  + ((lab1.a - lab2.a) * (lab1.a - lab2.a))
  + ((lab1.b - lab2.b) * (lab1.b - lab2.b)));

 double xDH;
 if (Math.Sqrt( xDE ) > ( Math.Sqrt( Math.Abs( xDL ) ) + Math.Sqrt( Math.Abs(
xDC ) ) ) )
 {
  xDH = Math.Sqrt((xDE*xDE) - (xDL*xDL) - (xDC*xDC));
 }
 else
 {
  xDH = 0;
 }

 double xSC = 1 + (0.045*xC1);
 double xSH = 1 + (0.015*xC1);
 xDL /= whtL;
 xDC /= whtC*xSC;
 xDH /= whtH*xSH;
 double deltaE94 = Math.Sqrt(Math.Pow(xDL, 2.0) + Math.Pow(xDC, 2.0) +
Math.Pow(xDH, 2.0));
 return deltaE94;
}
```

Sign In · View Thread · Permalink 5.00/5 (1 vote)

Re: Color Distance

Member 10716758 3-Jun-14 10:09

I would like to mention my library: Colourful

It currently supports these color spaces (and conversions between each other):

RGB (with working spaces!)
CIE XYZ
CIE xyY
CIE Lab
CIE LCh(ab)
CIE Luv
CIE LCh(uv)
Hunter Lab

It also has more advanced features like Delta-E color difference, color temperature computation (CCT) and chromatic adaptation.

Check it out, it is on NuGet also. 😊

Sign In · View Thread · Permalink

Re: Color Distance

Eric Ouellet 17-Jun-14 6:17

Thanks a lot for that information !

It will also serve more peoples !!!

I will add it in my article to ensure people will get it because it seems a lot better

Sign In · View Thread · Permalink

Re: Color Distance

Eric Ouellet 3-Oct-11 6:54

I implemented it and tested it in my little "Tips and tricks" (Random Color Generator[^]).
It seems to work very fine. I suspect that it is better than what I have used Cie1994. I will update my article to use you algorithm.

Thanks a lot !!!
Éric

Sign In · View Thread · Permalink

Re: Color Distance

Rob2412 3-Oct-11 11:47

Eric,
I have not tested the "older" CIE standards, just read in Wikipedia about corrections from one version to the other and coded using the step by step formulas from:
http://www.brucelindbloom.com/index.html?Eqn_Lab_to_XYZ.html

Somewhere on Bruce webpage is a color distance calculator, there you can test the different CIE standards.

I just saw in my code that there are some float identifiers. Just change 'F' to 'D' then all is consistent.

Rob

Sign In · View Thread · Permalink

Re: Color Distance

Eric Ouellet 4-Oct-11 3:26

Thanks a lot.

It's already updated on my "Tips and Tricks".
I updated the code to use your function (and also changed float to double).
I kept my old code as reference if anybody would want to use it or test with it.

Thanks a lot !
Eric

Sign In · View Thread · Permalink

Re: Color Distance

RichM 8-May-12 5:44

For any color newbz like myself, the difference between black and white, which I assume is the greatest distance between colors, comes out to 100.0 with this code. Thanks again Rob!

Sign In · View Thread · Permalink

My vote of 5

Eric Ouellet **8-Sep-11 8:06**

Great !

Sign In · View Thread · Permalink

Color seperation

joedimagio1945 **20-May-11 2:43**

Hi - I have a quick question. Is there a way we can load an image and do a color separation dynamically?
e.g. I might have check boxes for CMYK on the left and the image on the right and choosing any combination of colors, the respective image shows on the right. Is this possible?

Also, we come across Pantone colors. How are these handled?

Sign In · View Thread · Permalink

RGB2Lab and Lab2RGB **Darren Schroeder** **29-Apr-11 4:27**

When I execute the following code the RGB values come back different than the original, why?

```
CIELab lab = ColorSpaceHelper.RGBtoLab(120, 193, 231);
RGB rgb = ColorSpaceHelper.LabtoRGB(lab);
```

The r g b values returned are r=128 g=186 b=233. I don't understand why they're not 120,193,231. Is this a rounding issue? or just something funky with converting color spaces?

Sign In · View Thread · Permalink

Re: RGB2Lab and Lab2RGB Guillaume Leparmentier 2-May-11 9:03

Hi Darren,

Actually this issue is up to a conversion error in ColorSpaceHelper, in XYZtoRGB() method.
I was using a different gamma correction when I convert from and to CIE XYZ (to convert to a CIE Lab, we first convert to CIE XYZ).

To fix this issue, even if it's not perfect, you'll need to use the same gamma correction.

Change line 1379 in ColorSpaceHelper with :

```
Clinear[i] = (Clinear[i]<=0.0031308)? 12.92*Clinear[i] :
(1+0.055)* Math.Pow(Clinear[i], (1.0/2.2)) - 0.055;
```

The correct correction is the one in bold, which is 2.2.
It won't fix rounding issues, but will let you have a more consistant conversion.

I'm currently working on a update of this lib. It's not for tomorrow but, it'll fix every weird things you may have found (+ many new things)

Cheers,
Guillaume

Sign In · View Thread · Permalink

Re: RGB2Lab and Lab2RGB Rob2412 20-Sep-11 9:29

Hi Guillaume,
I noted that you are using D65 illuminant tristimulus and the corresponding matrix to convert CIE Lab <> CIE XYZ <> RGB.
Your formulas are correct, apart from method RGBtoXYZ(). It should read Math.Pow((... + 0.055)/(1 + 0.055), **2.4**) !!
Please see here: http://www.brucelindbloom.com/index.html?Eqn_Lab_to_XYZ.html

I have programmed a similar library to read Adobe color book (acb) files. Adobe color books contain CIE Lab color spaces.
After a bit playing a D50 illuminant tristimulus and a Bradford-adapted D50 matrix provide for a spot on conversion from Lab to sRGB.

For your convenience here are the matrices:

D50 Bradford-adapted RGB to XYZ [M]
0.4360747 0.3850649 0.1430804
0.2225045 0.7168786 0.0606169
0.0139322 0.0971045 0.7141733

D50 Bradford-adapted XYZ to RGB [M]-1
3.1338561 -1.6168667 -0.4906146
-0.9787684 1.9161415 0.0334540
0.0719453 -0.2289914 1.4052427

D50 tristimulus:
X 0.96422
Y 1.00000
Z 0.82521

You might want to consider these enhancements for an update.

Rob

Re: RGB2Lab and Lab2RGB

RichM 8-May-12 4:23

Thank you Rob2412!

These D50 numbers give a much closer conversion from lab to RGB and back. Also, for anyone else wanting to convert lab to RGB there is a bug pointed out by EvgenyTr above with a flipped sign in the XYZtoRGB function. I will repeat it here:

```
Clinear[1] = -x*0.9692 + y*1.8760 - z*0.0416; // green

should be Clinear[1] = -x*0.9692 + y*1.8760 + z*0.0416; // green
```

Fixing this bug plus Rob's numbers gives good results for flipping Lab to Rgb and back.

Hi, Great Article

roboticEDAR **28-Mar-11 11:02**

Hi, The preview for part 2 looks great did you ever get around to finishing it ?
It would be great to see the next part

Thanks.

Sign In · View Thread · Permalink

Amazing! no.reason 6-Feb-11 4:35

I've been looking for an article about converting colors for ages! This is top quality! Thanks a lot!

Sign In · View Thread · Permalink

help mouass 25-Jun-10 13:52

hello

would you please help me to find the right formules to convert image from rgb to hsi?.

thank you

Sign In · View Thread · Permalink 2.00/5 (1 vote)

Thanks! igbu 26-Apr-10 23:36

Great Work! Thank you for the Article!

Sign In · View Thread · Permalink

Quality Article Mike Rossouw 1-Apr-10 5:58

Your article on colour (I live in the UK so colour = color) is a really great contribution. Warms the hert that the internet can host the considered efforts of people like you. Well done!

Sign In · View Thread · Permalink

Where could i find the Part 2, and Part 3 ? sakumira 2-Jan-10 6:02

It's really awesome article.

I have been looking around, however i cannot find the part 2, and part 3.

Where could i find them?

I love freecode

Sign In · View Thread · Permalink

Absolutely & Top wonderful article!!! Thanx! Member 3932606 30-Oct-09 20:04

I am inpatient in waiting to continue this serie of articles and fix to this article. If I can mention some idea of more color models let me say "CMYKOG" and "CMYKOGxy" which becames more and more popular for hexachrome (and more) printing. With obvious expirience of author I belive that would be possible to programm in C# or VBasic.
Once more thank you and keep working this great job.
If I can help - I would like, but unfortunatelly can't. I dont know to program and also I dont know much about separations. This is why I am here. To learn something.

Sign In · View Thread · Permalink

OUTSTANDING! ShinjiR 27-Oct-09 9:59

One of the most complete articles I have read!!!
THANK YOU!!

Sign In · View Thread · Permalink

Very useful Oleg V. Polikarpotchkin 1-Jun-09 22:25

Salute Guillaume,
Excellent article! Have the 5.
Can you provide me with the references to Part II and III. Have you published them?
Best regards,
Oleg V. Polikarpotchkin

---

ovp

Sign In · View Thread · Permalink

Rounding method Ron Waller 3-May-09 18:51

Great article. Very helpful.

But I think it would be better to use the Round method from the Math class instead of converting the number to string then parsing it back to a number.

I am converting HSL to Color (RGB) and back, and round HSL doubles to 10 digit precision and integer-round before casting RGB values to bytes. When converting from Color to HSL and back to Color I get the same RGB values I started with.

Using the Round method is probably faster as well.

Sign In · View Thread · Permalink 5.00/5 (3 votes)

Re: Rounding method CoperNick 31-Jul-12 5:57

Right, those roundings are strange:

```
double r;
//...
Double.Parse(String.Format("{0:0.00}", r*255.0))
```

Sign In · View Thread · Permalink

Awesome Article Scotley 27-Apr-09 17:50

After scouring the internet for hours, I find this artical right under my nose.

It contains the perfect information that I needed to learn about colour spaces.
Thanks heaps for putting in so much time to teach us.

- Scotley

Sign In · View Thread · Permalink

Very useful haroldlai 12-Mar-09 19:52

It is very useful. Cheers. it is very clear.
I only have to modify a little bit to provide LAB to RGB service.
Cheers.

Sign In · View Thread · Permalink

How do you convert to/from RYB DEK46656 28-Nov-08 18:01

I have not had a chance to thoroughly read through the article, but just from my research into "this question" I can tell you have much more here than I've seen in similar types of articles.

Anyway, what I am trying to do is convert between RYB and RGB. Do you have any details on that, or can you point me in the direction of where I can see how it is done?

Thanks...

Sign In · View Thread · Permalink

Looking for a way to select colors that appear complementary or "not in conflict" visually; also to lighten and darken any color accurately stmarcus 28-Oct-08 7:56

Is there any research published that allows me to:

(1) Select any group of colors automatically, given a certain lightness or darkness, so that when the colors are drawn on a map, they do not conflict? One imagines that complementary colors would work, but what happens when we want 100 colors, all different enough so they do not appear similar? The application here is geographical mapping.

(2) I have been trying to lighten and darken colors programmatically, but the process is much more complex than one can imagine, for instance, this color:

R = 50
G = 100
B = 0

... if we lighten by increasing the numbers proportionally, we might get this color:

R = 100
G = 200
B = 0

... the Blue is zero, so doubling it is still zero. The resulting color does not look like a lighter version of the original color.

There are many other cases where the numbers are all mid-range (between 50 and 100), but increasing their size tends to drift the color off course anyway.

The same dilemmas arise when darkening colors by reducing their values.

This appears to be a real science. Any ideas as to how to solve this problem?

Thank you!

Sign In · View Thread · Permalink 1.50/5 (2 votes)

Awesome! Génial! Thanks! **Inner Swirl** **4-Oct-08 6:53**

Best Color Space article for .Net around!

Sign In · View Thread · Permalink

Last Visit: 31-Dec-99 18:00 Last Update: 26-Oct-14 0:16 Refresh 1 2 3 Next »

General News Suggestion Question Bug Answer Joke Rant Admin