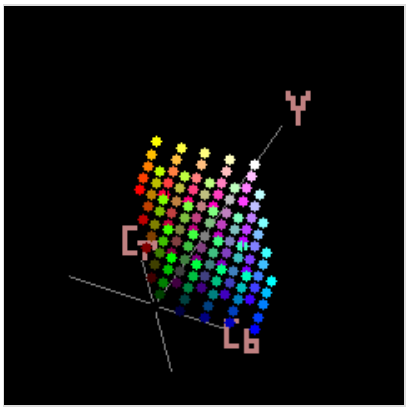
YCbCr

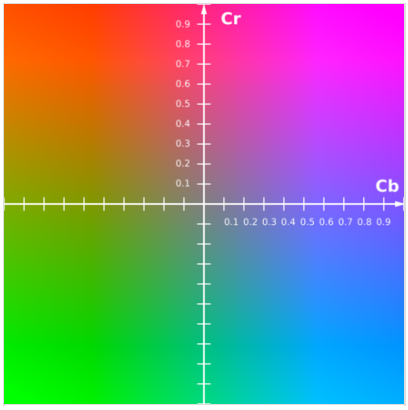
YCbCr, **Y′CbCr**, or **Y Pb/Cb Pr/Cr**, also written as $\mathbf{YC_BC_R}$ or $\mathbf{Y'C_BC_R}$, is a family of color spaces used as a part of the color image pipeline in digital video and photography systems.



A visualization of YCbCr color space

Y′ is the luma component and C_B and C_R are the blue-difference and red-difference chroma components. Luma Y′ (with prime) is distinguished from luminance Y, meaning that light intensity is nonlinearly encoded based on gamma corrected RGB primaries.

Y′CbCr color spaces are defined by a mathematical coordinate transformation from an associated RGB primaries and white point. If the underlying RGB color space is absolute, the Y′CbCr color space is an absolute color space as well; conversely, if the RGB space is ill-defined, so is Y′CbCr. The transformation is defined in equations 32, 33 in ITU-T H.273.[1]



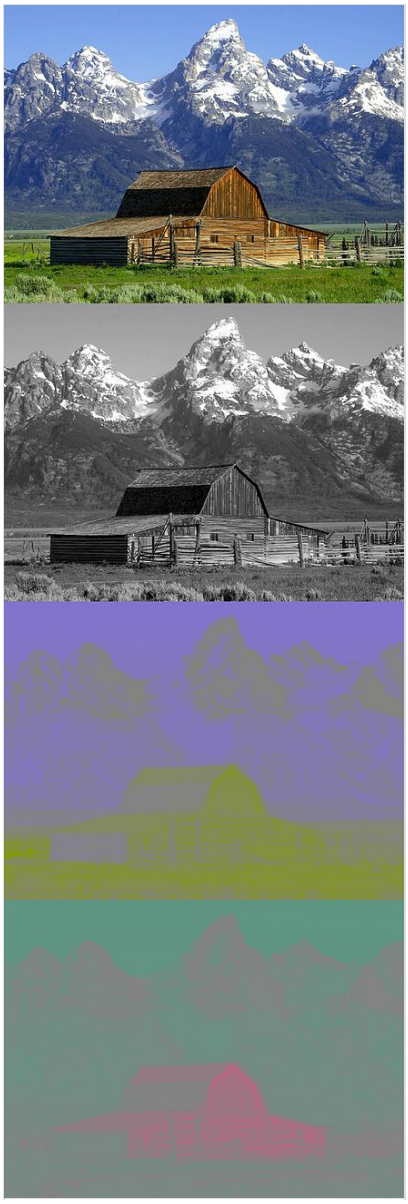
The CbCr plane at constant luma Y′=0.5

Rationale

Black and white television was in wide use before color television. Due to the number of existing TV sets and cameras, some form of backwards compatibility was desired for the new color broadcasts. French engineer Georges Valensi developed and patented a system for transmitting RGB color as luma and chroma signals in 1938. This would allow existing black and white televisions to process only the luma information and ignore the chroma, essentially packaging a black and white video within the color video. Because of this backwards compatibility the system based on Valensi's idea was called compatible color. In the same way, a black and white broadcast could be received by a color television without any additional processing circuitry. To preserve existing broadcast frequency allocations, the new chroma information was given lower bandwidth than the luma information. This is possible because humans are more sensitive to the black-and-white information (see image example to the right). This is called chroma subsampling.

YCbCr and Y′CbCr are a practical approximation to color processing and perceptual uniformity, where the primary colors corresponding roughly to red, green and blue are processed into perceptually meaningful information. By doing this, subsequent image/video processing, transmission and storage can do operations and introduce errors in perceptually meaningful ways. Y′CbCr is used to separate out a luma signal (Y′) that can be stored with high resolution or

transmitted at high bandwidth, and two chroma components (C_B and C_R) that can be bandwidth-reduced, subsampled, compressed, or otherwise treated separately for improved system efficiency.



A color image and its Y′, C_B and C_R components. The Y′ image is essentially a greyscale copy of the main image.

CbCr

YCbCr is sometimes abbreviated to **YCC**. Typically the terms Y′CbCr, YCbCr, YPbPr and YUV are used interchangeably, leading to some confusion. The main difference is that YPbPr is used with analog images and YCbCr with digital images, leading to different scaling values for U_{max} and V_{max} (in YCbCr both are $\frac{1}{2}$) when converting to/from YUV. Y′CbCr and YCbCr differ due to the values being gamma corrected or not.

The equations below give a better picture of the common principles and general differences between these formats.

RGB conversion

R'G'B' to Y′PbPr

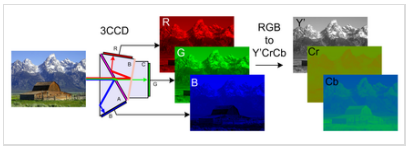
Y′CbCr signals (prior to scaling and offsets to place the signals into digital form) are called YPbPr, and are created from the corresponding gamma-adjusted RGB (red, green and blue) source using three defined constants K_R, K_G, and K_B as follows:

$$Y' = K_R \cdot R' + K_G \cdot G' + K_B \cdot B'$$
$$P_B = \frac{1}{2} \cdot \frac{B' - Y'}{1 - K_B}$$
$$P_R = \frac{1}{2} \cdot \frac{R' - Y'}{1 - K_R}$$

where K_R, K_G, and K_B are ordinarily derived from the definition of the corresponding RGB space, and required to satisfy $K_R + K_G + K_B = 1$.

The equivalent matrix manipulation is often referred to as the "color matrix":

$$\begin{bmatrix} Y' \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} K_R & K_G & K_B \\ -\frac{1}{2} \cdot \frac{K_R}{1-K_B} & -\frac{1}{2} \cdot \frac{K_G}{1-K_B} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} \cdot \frac{K_G}{1-K_R} & -\frac{1}{2} \cdot \frac{K_B}{1-K_R} \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}$$



RGB to YCbCr conversion

And its inverse:

$$\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 2-2\cdot K_R \\ 1 & -\frac{K_B}{K_G}\cdot(2-2\cdot K_B) & -\frac{K_R}{K_G}\cdot(2-2\cdot K_R) \\ 1 & 2-2\cdot K_B & 0 \end{bmatrix} \begin{bmatrix} Y' \\ P_B \\ P_R \end{bmatrix}$$

Here, the prime (′) symbols mean gamma correction is being used; thus R′, G′ and B′ nominally range from 0 to 1, with 0 representing the minimum intensity (e.g., for display of the color black) and 1 the maximum (e.g., for display of the color white). The resulting luma (Y) value will then have a nominal range from 0 to 1, and the chroma (P_B and P_R) values will have a nominal range from -0.5 to +0.5. The reverse conversion process can be readily derived by inverting the above equations.

Y′PbPr to Y′CbCr

When representing the signals in digital form, the results are scaled and rounded, and offsets are typically added. For example, the scaling and offset applied to the Y′ component per specification (e.g. MPEG-2[2]) results in the value of 16 for black and the value of 235 for white when using an 8-bit representation. The standard has 8-bit digitized versions of C_B and C_R scaled to a different range of 16 to 240. Consequently, rescaling by the fraction (235-16)/(240-16) = 219/224 is sometimes required when doing color matrixing or processing in YCbCr space, resulting in quantization distortions when the subsequent processing is not performed using higher bit depths.

The scaling that results in the use of a smaller range of digital values than what might appear to be desirable for representation of the nominal range of the input data allows for some "overshoot" and "undershoot" during processing without necessitating undesirable clipping. This "headroom" and "toeroom"[3] can also be used for extension of the nominal color gamut, as specified by xvYCC.

The value 235 accommodates a maximum overshoot of (255 - 235) / (235 - 16) = 9.1%, which is slightly larger than the theoretical maximum overshoot (Gibbs' Phenomenon) of about 8.9% of the maximum (black-to-white) step. The toeroom is smaller, allowing only 16 / 219 = 7.3% overshoot, which is less than the theoretical maximum overshoot of 8.9%. In addition, because values 0 and 255 are reserved in HDMI, the room is actually slightly less.

Y′CbCr to xvYCC

Since the equations defining Y′CbCr are formed in a way that rotates the entire nominal RGB color cube and scales it to fit within a (larger) YCbCr color cube, there are some points within the Y′CbCr color cube that *cannot* be represented in the corresponding RGB domain (at least not within the nominal RGB range). This causes some difficulty in determining how to correctly interpret and display some Y′CbCr signals. These out-of-range Y′CbCr values are used by xvYCC to encode colors outside the BT.709 gamut.

ITU-R BT.601 conversion

The form of Y′CbCr that was defined for standard-definition television use in the ITU-R BT.601 (formerly CCIR 601) standard for use with digital component video is derived from the corresponding RGB space (ITU-R BT.470-6 System M primaries) as follows:

$$K_R = 0.299$$
$$K_G = 0.587$$
$$K_B = 0.114$$

From the above constants and formulas, the following can be derived for ITU-R BT.601.

Analog YPbPr from analog R'G'B' is derived as follows:

$$\begin{aligned} Y' &= 0.299 \cdot R' + 0.587 \cdot G' + 0.114 \cdot B' \\ P_B &= -0.168736 \cdot R' - 0.331264 \cdot G' + 0.5 \cdot B' \\ P_R &= 0.5 \cdot R' - 0.418688 \cdot G' - 0.081312 \cdot B' \end{aligned}$$

Digital Y′CbCr (8 bits per sample) is derived from analog R'G'B' as follows:

$$\begin{aligned} Y' &= 16 + (65.481 \cdot R' + 128.553 \cdot G' + 24.966 \cdot B') \\ C_B &= 128 + (-37.797 \cdot R' - 74.203 \cdot G' + 112.0 \cdot B') \\ C_R &= 128 + (112.0 \cdot R' - 93.786 \cdot G' - 18.214 \cdot B') \end{aligned}$$

or simply componentwise

$$(Y', C_B, C_R) = (16, 128, 128) + (219 \cdot Y, 224 \cdot P_B, 224 \cdot P_R)$$

The resultant signals range from 16 to 235 for Y′ (Cb and Cr range from 16 to 240); the values from 0 to 15 are called *footroom*, while the values from 236 to 255 are called *headroom*. The same quantisation ranges, different for Y and Cb, Cr also apply to BT.2020 and BT.709.

Alternatively, digital Y′CbCr can derived from digital R'dG'dB'd (8 bits per sample, each using the full range with zero representing black and 255 representing white) according to the following equations:

$$\begin{aligned} Y' &= 16 + \frac{65.481 \cdot R'_D}{255} + \frac{128.553 \cdot G'_D}{255} + \frac{24.966 \cdot B'_D}{255} \\ C_B &= 128 - \frac{37.797 \cdot R'_D}{255} - \frac{74.203 \cdot G'_D}{255} + \frac{112.0 \cdot B'_D}{255} \\ C_R &= 128 + \frac{112.0 \cdot R'_D}{255} - \frac{93.786 \cdot G'_D}{255} - \frac{18.214 \cdot B'_D}{255} \end{aligned}$$

In the formula below, the scaling factors are multiplied by $\frac{256}{255}$. This allows for the value 256 in the denominator, which can be calculated by a single bitshift.

$$\begin{aligned} Y' &= 16 + \frac{65.738 \cdot R'_D}{256} + \frac{129.057 \cdot G'_D}{256} + \frac{25.064 \cdot B'_D}{256} \\ C_B &= 128 - \frac{37.945 \cdot R'_D}{256} - \frac{74.494 \cdot G'_D}{256} + \frac{112.439 \cdot B'_D}{256} \\ C_R &= 128 + \frac{112.439 \cdot R'_D}{256} - \frac{94.154 \cdot G'_D}{256} - \frac{18.285 \cdot B'_D}{256} \end{aligned}$$

If the R'd G'd B'd digital source includes footroom and headroom, the footroom offset 16 needs to be subtracted first from each signal, and a scale factor of $\frac{255}{219}$ needs to be included in the equations.

The inverse transform is:

$$
\begin{aligned}
R'_D &= \frac{298.082 \cdot Y'}{256} + \frac{408.583 \cdot C_R}{256} - 222.921 \\
G'_D &= \frac{298.082 \cdot Y'}{256} - \frac{100.291 \cdot C_B}{256} - \frac{208.120 \cdot C_R}{256} + 135.576 \\
B'_D &= \frac{298.082 \cdot Y'}{256} + \frac{516.412 \cdot C_B}{256} - 276.836
\end{aligned}
$$

The inverse transform without any roundings (using values coming directly from ITU-R BT.601 recommendation) is:

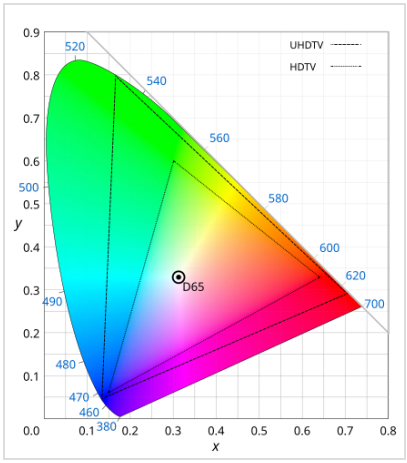
$$
\begin{aligned}
R'_D &= \frac{255}{219} \cdot (Y' - 16) + \frac{255}{224} \cdot 1.402 \cdot (C_R - 128) \\
G'_D &= \frac{255}{219} \cdot (Y' - 16) - \frac{255}{224} \cdot 1.772 \cdot \frac{0.114}{0.587} \cdot (C_B - 128) - \frac{255}{224} \cdot 1.402 \cdot \frac{0.299}{0.587} \cdot (C_R - 128) \\
B'_D &= \frac{255}{219} \cdot (Y' - 16) + \frac{255}{224} \cdot 1.772 \cdot (C_B - 128)
\end{aligned}
$$

This form of Y′CbCr is used primarily for older standard-definition television systems, as it uses an RGB model that fits the phosphor emission characteristics of older CRTs.

ITU-R BT.709 conversion

A different form of Y′CbCr is specified in the ITU-R BT.709 standard, primarily for HDTV use. The newer form is also used in some computer-display oriented applications, as sRGB (though the matrix used for sRGB form of YCbCr, sYCC, is still BT.601). In this case, the values of Kb and Kr differ, but the formulas for using them are the same. For ITU-R BT.709, the constants are:

$$
\begin{aligned}
K_B &= 0.0722 \\
K_R &= 0.2126 \\
(K_G &= 1 - K_B - K_R = 0.7152)
\end{aligned}
$$



Rec. 709 compared with Rec. 2020

This form of Y′CbCr is based on an RGB model that more closely fits the phosphor emission characteristics of newer CRTs and other modern display equipment. The conversion matrices for BT.709 are these:

$$
\begin{bmatrix} Y' \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.1146 & -0.3854 & 0.5 \\ 0.5 & -0.4542 & -0.0458 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix}
$$

$$
\begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.5748 \\ 1 & -0.1873 & -0.4681 \\ 1 & 1.8556 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ C_B \\ C_R \end{bmatrix}
$$

The definitions of the R', G', and B' signals also differ between BT.709 and BT.601, and differ within BT.601 depending on the type of TV system in use (625-line as in PAL and SECAM or 525-line as in NTSC), and differ further in other specifications. In different designs there are differences in the definitions of the R, G, and B chromaticity coordinates, the reference white point, the supported gamut range, the exact gamma pre-compensation functions for deriving R', G' and B' from R, G, and B, and in the scaling and offsets to be applied during conversion from R'G'B' to Y′CbCr. So proper conversion of Y′CbCr from one form to the other is not just a matter of inverting

one matrix and applying the other. In fact, when Y′CbCr is designed ideally, the values of K_B and K_R are derived from the precise specification of the RGB color primary signals, so that the luma (Y′) signal corresponds as closely as possible to a gamma-adjusted measurement of luminance (typically based on the CIE 1931 measurements of the response of the human visual system to color stimuli).[4]

ITU-R BT.2020 conversion

The ITU-R BT.2020 standard uses the same gamma function as BT.709. It defines:[5]

- Non-constant luminance Y'CbCr, similar to the previous entries, except with different K_B and K_R.
- Constant luminance Y'cCbcCrc, a formulation where Y' is the gamma-codec version of the true luminance.[5]

For both, the coefficients derived from the primaries are:

$$\begin{aligned} K_B &= 0.0593 \\ K_R &= 0.2627 \\ (K_G &= 1 - K_B - K_R = 0.6780) \end{aligned}$$

For NCL, the definition is classical: $Y' = 0.2627R' + 0.6780G' + 0.0593B'$; $Cb = (B' - Y')/1.8814$; $Cr = (R' - Y')/1.4746$. The encoding conversion can, as usual, be written as a matrix.[5] The decoding matrix for BT.2020-NCL is this with 14 decimal places:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.4746 \\ 1 & -0.16455312684366 & -0.57135312684366 \\ 1 & 1.8814 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ C_B \\ C_R \end{bmatrix}$$

The smaller values in the matrix are not rounded; they are precise values. For systems with limited precision (8 or 10 bit, for example) a lower precision of the above matrix could be used, for example, retaining only 6 digits after decimal point.[6]

The CL version, YcCbcCrc, codes:[5]

- $Y'c = (0.2627R + 0.6780G + 0.0593B)'$. This is the gamma function applied to the true luminance calculated from linear RGB.
- $Cbc = (B' - Y'c)/(-2Nb)$ if $B' < Y'c$ otherwise $(B' - Y'c)/(2Pb)$. Nb and Pb are the theoretical minimum and maximum of $(B' - Y'c)$ corresponding to the gamut. The rounded "practical" values are $Pb = 0.7910$, $Nb = -0.9702$. The full derivation can be found in the recommendation.[5]
- $Crc = (R' - Y'c)/(-2Nr)$ if $R' < Y'c$ otherwise $(R' - Y'c)/(2Pr)$. Again, Pr and Nr are theoretical limits. The rounded values are $Pr = 0.4969$, $Nr = -0.8591$.

The CL function can be used when preservation of luminance is of primary importance (see: Chroma subsampling § Gamma luminance error), or when "there is an expectation of improved coding efficiency for delivery." The specification refers to Report ITU-R BT.2246 on this matter.[5] BT.2246 states that CL has improved compression efficiency and luminance preservation, but NCL will be more familiar to a staff that has previously handled color mixing and other production practices in HDTV YCbCr.[7]

BT.2020 does not define PQ and thus HDR, it is further defined in SMPTE ST 2084 and BT.2100. BT.2100 will introduce the use of IC_TC_P, a semi-perceptual colorspace derived from linear RGB with good hue linearity. It is "near-constant luminance".[8]

SMPTE 240M conversion

The SMPTE 240M standard (used on the MUSE analog HD television system) defines YCC with these coefficients:

$$K_B = 0.087$$
$$K_R = 0.212$$

The coefficients are derived from SMPTE 170M primaries and white point, as used in 240M standard.

JPEG conversion

JFIF usage of JPEG supports a modified Rec. 601 Y′CbCr where Y′, C_B and C_R have the full 8-bit range of [0...255].[9] Below are the conversion equations expressed to six decimal digits of precision. (For ideal equations, see ITU-T T.871.[10]) Note that for the following formulae, the range of each input (R,G,B) is also the full 8-bit range of [0...255].

$$\begin{aligned} Y' &= 0 + (0.299 \cdot R'_D) + (0.587 \cdot G'_D) + (0.114 \cdot B'_D) \\ C_B &= 128 - (0.168736 \cdot R'_D) - (0.331264 \cdot G'_D) + (0.5 \cdot B'_D) \\ C_R &= 128 + (0.5 \cdot R'_D) - (0.418688 \cdot G'_D) - (0.081312 \cdot B'_D) \end{aligned}$$

And back:

$$\begin{aligned} R'_D &= Y' + 1.402 \cdot (C_R - 128) \\ G'_D &= Y' - 0.344136 \cdot (C_B - 128) - 0.714136 \cdot (C_R - 128) \\ B'_D &= Y' + 1.772 \cdot (C_B - 128) \end{aligned}$$

The above conversion is identical to sYCC when the input is given as sRGB, except that IEC 61966-2-1:1999/Amd1:2003 only gives four decimal digits.

JPEG also defines a "YCCK" format from Adobe for CMYK input. In this format, the "K" value is passed as-is, while CMY are used to derive YCbCr with the above matrix by assuming $R = 1 - C$, $G = 1 - M$, and $B = 1 - Y$. As a result, a similar set of subsampling techniques can be used.[11]

Coefficients for BT.470-6 System B, G primaries

$$K_B = 0.0713$$
$$K_R = 0.2220$$

These coefficients are not in use and were never in use.[12]

Chromaticity-derived luminance systems

H.273 also describes constant and non-constant luminance systems which are derived strictly from primaries and white point, so that situations like sRGB/BT.709 default primaries of JPEG that use BT.601 matrix (that is derived from BT.470-6 System M) do not happen.

Numerical approximations

Prior to the development of fast SIMD floating-point processors, most digital implementations of RGB → Y′UV used integer math, in particular fixed-point approximations. Approximation means that the precision of the used numbers (input data, output data and constant values) is limited, and thus a precision loss of typically about the last binary digit is accepted by whoever makes use of that option in typically a trade-off to improved computation speeds.

Y′ values are conventionally shifted and scaled to the range [16, 235] (referred to as studio swing or "TV levels") rather than using the full range of [0, 255] (referred to as full swing or "PC levels"). This practice was standardized in SMPTE-125M in order to accommodate signal overshoots ("ringing") due to filtering.[13] U and V values, which may be positive or negative, are summed with 128 to make them always positive, giving a studio range of 16–240 for U and V. (These ranges are important in video editing and production, since using the wrong range will result either in an image with "clipped" blacks and whites, or a low-contrast image.)

Approximate 8-bit matrices for BT.601

These matrices round all factors to the closest 1/256 unit. As a result, only one 16-bit intermediate value is formed for each component, and a simple right-shift with rounding `(x + 128) >> 8` can take care of the division.[13]

For studio-swing:

$$\begin{bmatrix} Y' \\ C_B \\ C_R \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 66 & 129 & 25 \\ -38 & -74 & 112 \\ 112 & -94 & -18 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 16 \\ 128 \\ 128 \end{bmatrix}$$

For full-swing:

$$\begin{bmatrix} Y' \\ C_B \\ C_R \end{bmatrix} = \frac{1}{256} \begin{bmatrix} 77 & 150 & 29 \\ -43 & -84 & 127 \\ 127 & -106 & -21 \end{bmatrix} \begin{bmatrix} R' \\ G' \\ B' \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

Google's Skia used to use the above 8-bit full-range matrix, resulting in a slight greening effect on JPEG images encoded by Android devices, more noticeable on repeated saving. The issue was fixed in 2016, when the more accurate version was used instead. Due to SIMD optimizations in libjpeg-turbo, the accurate version is actually faster.[14]

Packed pixel formats and conversion

RGB files are typically encoded in 8, 12, 16 or 24 bits per pixel. In these examples, we will assume 24 bits per pixel, which is written as RGB888. The standard byte format is simply `r0, g0, b0, r1, g1, b1, ...`.

YCbCr Packed pixel formats are often referred to as "YUV". Such files can be encoded in 12, 16 or 24 bits per pixel. Depending on subsampling, the formats can largely be described as 4:4:4, 4:2:2, and 4:2:0p. The apostrophe after the Y is often omitted, as is the "p" (for planar) after YUV420p. In terms of actual file formats, 4:2:0 is the most common, as the data is more reduced, and the file extension is usually ".YUV". The relation between data rate and sampling (A:B:C) is defined by the ratio between Y to U and V channel.[15][16] The notation of "YUV" followed by three numbers is vague: the three numbers could refer to the subsampling (as is done in "YUV420"), or it could refer to bit depth in each channel (as is done in "YUV565"). The unambiguous way to refer to these formats is via the FourCC code.[17]

To convert from RGB to YUV or back, it is simplest to use RGB888 and 4:4:4. For 4:1:1, 4:2:2 and 4:2:0, the bytes need to be converted to 4:4:4 first.

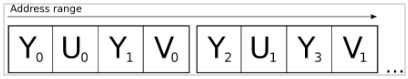
4:4:4

4:4:4 is straightforward, as no pixel-grouping is done: the difference lies solely in how many bits each channel is given, and their arrangement. The basic `YUV3` scheme uses 3 bytes per pixel, with the order `y0, u0, v0, y1, u1, v1` (using "u" for Cb and "v" for Cr; the same applies to content below).[17] In computers, it is more common to see a `AYUV` format, which adds an alpha channel and goes `a0, y0, u0, v0, a1, y1, u1, v1`, because groups of 32-bits are easier to deal with.[15]

4:2:2

4:2:2 groups 2 pixels together horizontally in each conceptual "container". Two main arrangements are:[16]

- YUY2: also called YUYV, runs in the format `y0, u, y1, v`.
- UYVY: the byte-swapped reverse of YUY2, runs in the format `u, y0, v, y1`.



YUY2 format

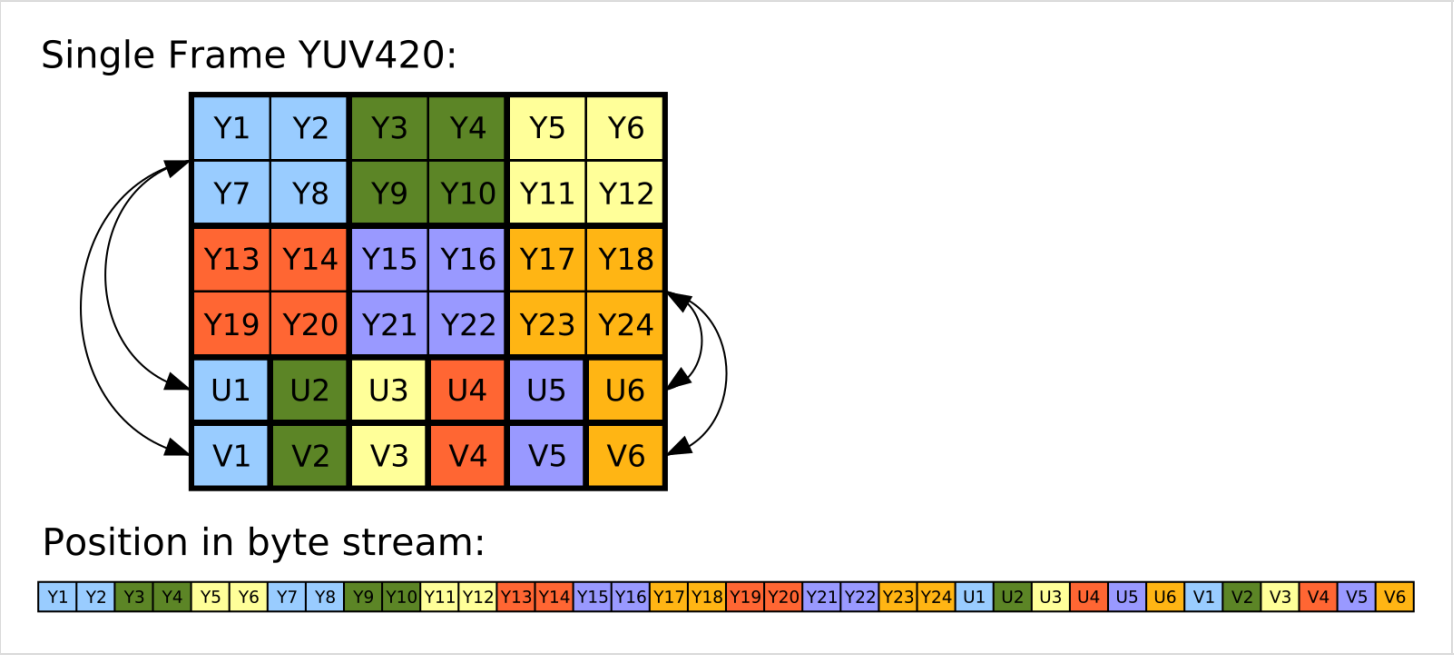
4:1:1

4:1:1 is rarely used. Pixels are in horizontal groups of 4.[16]

4:2:0

4:2:0 is very commonly used. The main formats are IMC2, IMC4, YV12, and NV12.[16] All of these four formats are "planar", meaning that the Y, U, and V values are grouped together instead of interspersed. They all occupy 12 bits per pixel, assuming a 8-bit channel.

- IMC2 first lays the full images out in Y. It then arranges each line of chroma in the order of V_0 ... V_n, U_0 ... U_n, where *n* is the number of chroma samples per line, equal to half the width of Y.
- IMC4 is similar to IMC2, except it runs in U_0 ... U_n, V_0 ... V_n.
- I420 is a simpler design and is more commonly used. The entire image in Y is written out, followed by the image in U, then by the whole image in V.



I420 layout

- YV12 follows the same general design as I420, only the order between the U and V images is flipped.[18]
- NV12 is possibly the most commonly-used 8-bit 4:2:0 format. It is the default for Android camera preview.[19] The entire image in Y is written out, followed by interleaved lines that go U_0, V_0, U_1, V_1, etc.

There are also "tiled" variants of planar formats.[20]

References

1. "H.273: Coding-independent code points for video signal type identification" (https://www.itu.int/rec/T-REC-H.273/en). *www.itu.int*. Retrieved 2025-01-06.
2. e.g. the MPEG-2 specification, ITU-T H.262 2000 E pg. 44
3. "MFNominalRange (mfobjects.h) - Win32 apps" (https://docs.microsoft.com/en-US/windows/win32/api/mfobjects/ne-mfobjects-mfnominalrange). *docs.microsoft.com*. Retrieved 10 November 2020.
4. Charles Poynton, *Digital Video and HDTV*, Chapter 24, pp. 291–292, Morgan Kaufmann, 2003.
5. "BT.2020 : Parameter values for ultra-high definition television systems for production and international programme exchange" (https://www.itu.int/rec/R-REC-BT.2020/en). International Telecommunication Union. June 2014. Retrieved 2014-09-08.
6. "ITU-T H Suppl. 18" (https://www.itu.int/ITU-T/recommendations/rec.aspx?rec=13441&lang=ru). October 2017. hdl:11.1002/1000/13441 (https://hdl.handle.net/11.1002%2F1000%2F13441).
7. "BT.2246-8 (03/2023) The present state of ultra-high definition television" (https://www.itu.int/pub/R-REP-BT.2246-8-2023). *ITU*.
8. "Subsampling in ICtCp vs YCbCr" (https://web.archive.org/web/20181013172410/https://www.dolby.com/us/en/technologies/dolby-vision/ictcp_vs_ycbcr-subsampling.pdf) (PDF). Dolby Laboratories, Inc. Archived from the original (https://www.dolby.com/us/en/technologies/dolby-vision/ictcp_vs_ycbcr-subsampling.pdf) (PDF) on 13 October 2018.
9. JPEG File Interchange Format Version 1.02 (http://www.w3.org/Graphics/JPEG/jfif3.pdf)
10. *T.871: Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF)* (http://www.itu.int/rec/T-REC-T.871). ITU-T. September 11, 2012. Retrieved 2016-07-25.
11. See libjpeg-turbo documentation for: CS_YCCK (https://github.com/libjpeg-turbo/libjpeg-turbo/blob/6b0e2b04008

11. See libjpeg-turbo documentation for: CS_YCCK (https://github.com/libjpeg-turbo/libjpeg-turbo/blob/6b9e3b04008165260a13f77cf235170438d5adf8/java/org/libjpegturbo/turbojpeg/TJ.java#L402) 'YCCK (AKA "YCbCrK") is not an absolute colorspace but rather a mathematical transformation of CMYK designed solely for storage and transmission', cmyk_ycck_convert() (https://github.com/libjpeg-turbo/libjpeg-turbo/blob/6b9e3b04008165260a13f77cf235170438d5adf8/jccolor.c#L396); see
12. "EBU Tech 3237 Supplement 1" (https://tech.ebu.ch/docs/tech/tech3237s1.pdf) (PDF). p. 18. Retrieved 15 April 2021.
13. Jack, Keith (1993). *Video Demystified* (https://archive.org/details/videodemystified00jack). HighText Publications. p. 30. ISBN 1-878707-09-4.
14. "Use libjpeg-turbo for YUV->RGB conversion in jpeg encoder · google/skia@c7d01d3" (https://github.com/google/skia/commit/c7d01d3e1d3621907c27b283fb7f8b6e177c629d). *GitHub*.
15. msdn.microsoft.com, YUV Video Subtypes (http://msdn.microsoft.com/en-us/library/windows/desktop/dd391027(v=vs.85).aspx)
16. msdn.microsoft.com, Recommended 8-Bit YUV Formats for Video Rendering (http://msdn.microsoft.com/de-de/library/windows/desktop/dd206750(v=vs.85).aspx)
17. "2.7.1.1. Packed YUV formats — The Linux Kernel documentation" (https://docs.kernel.org/userspace-api/media/v4l/pixfmt-packed-yuv.html). *docs.kernel.org*.
18. "VideoLAN Wiki: YUV" (https://wiki.videolan.org/YUV#YV12). *wiki.videolan.org*.
19. fourcc.com YUV pixel formas (https://fourcc.org/yuv.php#NV21)
20. "2.7.1.2. Planar YUV formats — The Linux Kernel documentation" (https://docs.kernel.org/userspace-api/media/v4l/pixfmt-yuv-planar.html). *docs.kernel.org*.

External links

- Y′CbCr calculator (https://res18h39.netlify.app/color), including BT.1886
- Charles Poynton — Color FAQ (https://poynton.ca/ColorFAQ.html)
- Charles Poynton — Video engineering (https://www.poynton.ca/Poynton-video-eng.html)
- Color Space Visualization (http://www.couleur.org/index.php?page=transformations#YCbCr)
- YUV, YCbCr, YPbPr color spaces. (https://discoverybiz.net/enu0/faq/faq_YUV_YCbCr_YPbPr.html)
- YCbCr Definition (https://api.video/what-is/ycbcr)

Software resources for packed pixels:

- Kohn, Mike. Y′UV422 to RGB using SSE/Assembly (https://www.mikekohn.net/stuff/image_processing.php)
- libyuv (https://chromium.googlesource.com/libyuv/libyuv/)
- pixfc-sse (https://code.google.com/p/pixfc-sse/) – C library of SSE-optimized color format conversions
- YUV files (https://web.archive.org/web/20190220164028/http://www.sunrayimage.com/examples.html) – Sample / Demo YUV/RGB video files in many YUV formats, help you for the testing.