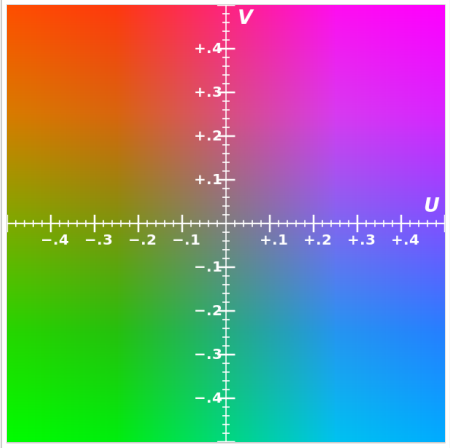
YUV

From Wikipedia, the free encyclopedia

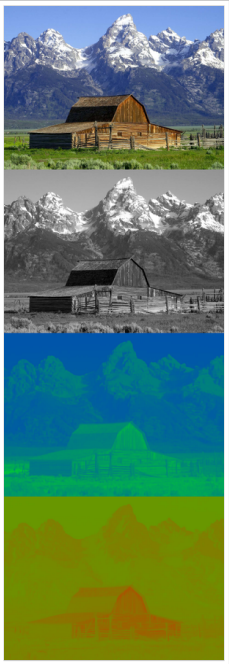
YUV is a color space typically used as part of a color image pipeline. It encodes a color image or video taking human perception into account, allowing reduced bandwidth for chrominance components, thereby typically enabling transmission errors or compression artifacts to be more efficiently masked by the human perception than using a "direct" RGB-representation. Other color spaces have similar properties, and the main reason to implement or investigate properties of Y′UV would be for interfacing with analog or digital television or photographic equipment that conforms to certain Y′UV standards.



Example of U-V color plane, Y′ value = 0.5, represented within RGB color gamut

The scope of the terms Y′UV, YUV, YCbCr, YPbPr, etc., is sometimes ambiguous and overlapping. Historically, the terms YUV and Y′UV were used for a specific *analog encoding* of color information in television systems, while YCbCr was used for *digital encoding* of color information suited for video *and* still-image compression and transmission such as MPEG and JPEG. Today, the term YUV is commonly used in the computer industry to describe *file-formats* that are encoded using YCbCr.

The Y′UV model defines a color space in terms of one luma (Y′) and two chrominance (UV) components. The Y′UV color model is used in the PAL and SECAM composite color video standards. Previous black-and-white systems used only luma (Y′) information. Color information (U and V) was added separately via a sub-carrier so that a black-and-white receiver would still be able to receive and display a color picture transmission in the receiver's native black-and-white format.



An image along with its Y′, U, and V components respectively

Y′ stands for the luma component (the brightness) and U and V are the chrominance (color) components; luminance is denoted by Y and luma by Y′ – the prime symbols (′) denote gamma compression,[1] with "luminance" meaning perceptual (color science) brightness, while "luma" is electronic (voltage of display) brightness.

The YPbPr color model used in analog component video and its digital version YCbCr used in digital video are more or less derived from it, and are sometimes called Y′UV. (C_B/P_B and C_R/P_R are deviations from grey on blue–yellow and red–cyan axes, whereas U and V are blue–luminance and red–luminance differences respectively.) The Y′IQ color space used in the analog NTSC television broadcasting system is related to it, although in a more complex way.

As for etymology, Y, Y′, U, and V are not abbreviations. The use of the letter Y for luminance can be traced back to the choice of X Y Z primaries. This lends itself naturally to the usage of the same letter in luma (Y′), which approximates a perceptually uniform correlate of luminance. Likewise, U and V were chosen to differentiate the U and V axes from those in other spaces, such as the x and y chromaticity space. See the equations below or compare the historical development of the math.[2][3][4]

Contents

History

Y′UV was invented when engineers wanted color television in a black-and-white infrastructure.[5] They needed a signal transmission method that was compatible with black-and-white (B&W) TV while being able to add color. The luma component already existed as the black and white signal; they added the UV signal to this as a solution.

The UV representation of chrominance was chosen over straight R and B signals because U and V are color difference signals. This meant that in a black and white scene the U and V signals would be zero and only the Y′ signal would need to be transmitted. If R and B were to have been used, these would have non-zero values even in a B&W scene, requiring all three data-carrying signals. This was important in the early days of color television, because holding the U and V signals to zero while connecting the black and white signal to Y′ allowed color TV sets to display B&W TV without the additional expense and complexity of special B&W circuitry. In addition, black and white receivers could take the Y′ signal and ignore the color signals, making Y′UV backward-compatible with all existing black-and-white equipment, input and output. It was necessary to assign a narrower bandwidth to the chrominance channel because there was no additional bandwidth available. If some of the luminance information arrived via the chrominance channel (as it would have if RB signals were used instead of differential UV signals), B&W resolution would have been compromised.[6]

Conversion to/from RGB

SDTV with BT.601

Y′UV signals are typically created from RGB (red, green and blue) source. Weighted values of R, G, and B are summed to produce Y′, a measure of overall brightness or luminance. U and V are computed as scaled differences between Y′ and the B and R values.

BT.601 defines the following constants:

$$
\begin{aligned}
W_R &= 0.299 \\
W_G &= 1 - W_R - W_B = 0.587 \\
W_B &= 0.114 \\
U_{\text{Max}} &= 0.436 \\
V_{\text{Max}} &= 0.615
\end{aligned}
$$

Y′UV is computed from RGB as follows:

$$
\begin{aligned}
Y' &= W_R R + W_G G + W_B B = 0.299R + 0.587G + 0.114B \\
U &= U_{\text{Max}} \frac{B - Y'}{1 - W_B} \approx 0.492(B - Y') \\
V &= V_{\text{Max}} \frac{R - Y'}{1 - W_R} \approx 0.877(R - Y')
\end{aligned}
$$

The resulting ranges of Y′, U, and V respectively are [0, 1], [$-U_{\text{Max}}$, U_{Max}], and [$-V_{\text{Max}}$, V_{Max}].

Inverting the above transformation converts Y′UV to RGB:

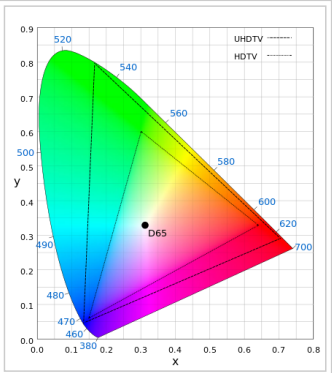
$$R = Y' + V\frac{1-W_R}{V_{\text{Max}}} = Y' + \frac{V}{0.877} = Y' + 1.14V$$
$$G = Y' - U\frac{W_B(1-W_B)}{U_{\text{Max}}W_G} - V\frac{W_R(1-W_R)}{V_{\text{Max}}W_G}$$
$$= Y' - \frac{0.232U}{0.587} - \frac{0.341V}{0.587} = Y' - 0.395U - 0.581V$$
$$B = Y' + U\frac{1-W_B}{U_{\text{Max}}} = Y' + \frac{U}{0.492} = Y' + 2.033U$$

Equivalently, substituting values for the constants and expressing them as matrices gives these formulas for BT.601:

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51499 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.13983 \\ 1 & -0.39465 & -0.58060 \\ 1 & 2.03211 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

HDTV with BT.709

For HDTV the ATSC decided to change the basic values for W_R and W_B compared to the previously selected values in the SDTV system. For HDTV these values are provided by Rec. 709. This decision further impacted on the matrix for the Y′UV↔RGB conversion so that its member values are also slightly different. As a result, with SDTV and HDTV there are generally two distinct Y′UV representations possible for any RGB triple: a SDTV-Y′UV and a HDTV-Y′UV one. This means in detail that when directly converting between SDTV and HDTV, the luma (Y′) information is roughly the same but the representation of the chroma (U & V) channel information needs conversion. Still in coverage of the CIE 1931 color space the Rec. 709 color space is almost identical to Rec. 601 and covers 35.9%.[7] In contrast to this UHDTV with Rec. 2020 covers a much larger area and would further see its very own matrix set for YUV/Y′UV.



HDTV Rec. 709 (quite close to SDTV Rec. 601) compared with UHDTV Rec. 2020

BT.709 defines these weight values:

$$W_R = 0.2126$$
$$W_B = 0.0722$$

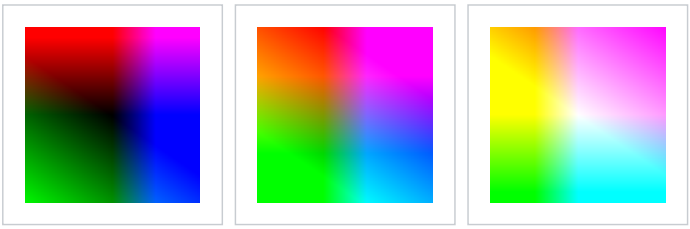
The conversion matrices & formulas for BT.709 are these:

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

Notes

- The weights used to compute Y′ (top row of matrix) are identical to those used in the Y′IQ color space.
- Equal values of red, green and blue (i.e. levels of gray) yield 0 for U and V. Black, RGB=(0, 0, 0), yields YUV=(0, 0, 0). White, RGB=(1, 1, 1), yields YUV=(1, 0, 0).
- These formulas are traditionally used in analog televisions and equipment; digital equipment such as HDTV and digital video cameras use Y′CbCr.

UV planes in a range of [-1,1]



Y′ value of 0 Y′ value of 0.5 Y′ value of 1

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.

In the following examples, the operator "$a \gg b$" denotes a right-shift of *a* by *b* bits. For clarification the variables are using two suffix characters: 'u' is used for the unsigned final representation and 't' is used for the scaled down intermediate value. The examples below are given for BT.601 only. The same principle can be used for doing functionally equivalent operations using values that do an acceptable match for data that follows the BT.709 or any other comparable standard.

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. The value 235 accommodates a maximum black-to-white overshoot of 255 - 235 = 20, or 20 / (235 - 16) = 9.1%, which is slightly larger than the theoretical maximum overshoot (Gibbs' Phenomenon) of about 8.9% of the maximum step. The toe-room is smaller, allowing only 16 / 219 = 7.3% overshoot, which is less than the theoretical maximum overshoot of 8.9%. This is why 16 is added to Y′ and why the Y′ coefficients in the basic transform sum to 220 instead of 255.[8] 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.)

Studio swing for BT.601

For getting the traditional 'studio swing' 8 bit representation of Y′UV for SDTV/BT.601 the following operations can be used:

1. Basic transform from 8 bit RGB to 16 bit values (Y′: unsigned, U/V: signed, matrix values got rounded so that the later on desired Y′ range of [16..235] and U/V range of [16..240] is reached):

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 66 & 129 & 25 \\ -38 & -74 & 112 \\ 112 & -94 & -18 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

2. Scale down (">>8") to 8 bit with rounding ("+128") (Y′: unsigned, U/V: signed):

$$\begin{aligned} Yt' &= (Y' + 128) \gg 8 \\ Ut &= (U + 128) \gg 8 \\ Vt &= (V + 128) \gg 8 \end{aligned}$$

3. Add an offset to the values to eliminate any negative values (all results are 8 bit unsigned):

$$\begin{aligned} Yu' &= Yt' + 16 \\ Uu &= Ut + 128 \\ Vu &= Vt + 128 \end{aligned}$$

Full swing for BT.601

For getting a 'full swing' 8 bit representation of Y′UV for SDTV/BT.601 the following operations can be used:

1. Basic transform from 8 bit RGB to 16 bit values (Y′: unsigned, U/V: signed, matrix values got rounded so that the later on desired Y′UV range of each [0..255] is reached whilst no overflow can happen):

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 76 & 150 & 29 \\ -43 & -84 & 127 \\ 127 & -106 & -21 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

2. Scale down (">>8") to 8 bit values with rounding ("+128") (Y′: unsigned, U/V: signed):

$$\begin{aligned} Yt' &= (Y' + 128) \gg 8 \\ Ut &= (U + 128) \gg 8 \\ Vt &= (V + 128) \gg 8 \end{aligned}$$

3. Add an offset to the values to eliminate any negative values (all results are 8 bit unsigned):

$$\begin{aligned} Yu' &= Yt' \\ Uu &= Ut + 128 \\ Vu &= Vt + 128 \end{aligned}$$

Luminance/chrominance systems in general

The primary advantage of luma/chroma systems such as Y′UV, and its relatives Y′IQ and YDbDr, is that they remain compatible with black and white analog television (largely due to the work of Georges Valensi). The Y′ channel saves all the data recorded by black and white cameras, so it produces a signal suitable for reception on old monochrome displays. In this case, the U and V are simply discarded. If displaying color, all three channels are used, and the original RGB information can be decoded.

Another advantage of Y′UV is that some of the information can be discarded in order to reduce bandwidth. The human eye has fairly little spatial sensitivity to color: the accuracy of the brightness information of the luminance channel has far more impact on the image detail discerned than that of the other two. Understanding this human shortcoming, standards such as NTSC and PAL reduce the bandwidth of the chrominance channels considerably. (Bandwidth is in the temporal domain, but this translates into the spatial domain as the image is scanned out.)

Therefore, the resulting U and V signals can be substantially "compressed". In the NTSC (Y′IQ) and PAL systems, the chrominance signals had significantly narrower bandwidth than that for the luminance. Early versions of NTSC rapidly alternated between particular colors in identical image areas to make them appear adding up to each other to the human eye, while all modern analogue and even most digital video standards use chroma subsampling by recording a picture's color information at reduced resolution. Only half the horizontal resolution compared to the brightness information is kept (termed 4:2:2 chroma subsampling), and often the vertical resolution is also halved (giving 4:2:0). The 4:x:x standard was adopted due to the very earliest color NTSC standard which used a chroma subsampling of 4:1:1 (where the horizontal color resolution is quartered while the vertical is full resolution) so that the picture carried only a quarter as much color resolution compared to brightness resolution. Today, only high-end equipment processing uncompressed signals uses a chroma subsampling of 4:4:4 with identical resolution for both brightness and color information.

The I and Q axes were chosen according to bandwidth needed by human vision, one axis being that requiring the most bandwidth, and the other (fortuitously at 90 degrees) the minimum. However, true I and Q demodulation was relatively more complex, requiring two analog delay lines, and NTSC receivers rarely used it.

However, this color space conversion is lossy, particularly obvious in crosstalk from the luma to the chroma-carrying wire, and vice versa, in analogue equipment (including RCA connectors to transfer a digital signal, as all they carry is analogue composite video, which is either YUV, YIQ, or even CVBS). Furthermore, NTSC and PAL encoded color signals in a manner that causes high bandwidth chroma and luma signals to mix with each other in a bid to maintain backward compatibility with black and white television equipment, which results in dot crawl and cross color artifacts. When the NTSC standard was created in the 1950s, this was not a real concern since the quality of the image was limited by the monitor equipment, not the limited-bandwidth signal being received. However today's modern television is capable of

displaying more information than is contained in these lossy signals. To keep pace with the abilities of new display technologies, attempts were made since the late 1970s to preserve more of the Y′UV signal while transferring images, such as SCART (1977) and S-Video (1987) connectors.

Instead of Y′UV, Y′CbCr was used as the standard format for (digital) common video compression algorithms such as MPEG-2. Digital television and DVDs preserve their compressed video streams in the MPEG-2 format, which uses a full Y′CbCr color space, although retaining the established process of chroma subsampling. The professional CCIR 601 digital video format also uses Y′CbCr at the common chroma subsampling rate of 4:2:2, primarily for compatibility with previous analog video standards. This stream can be easily mixed into any output format needed.

Y′UV is not an absolute color space. It is a way of encoding RGB information, and the actual color displayed depends on the actual RGB colorants used to display the signal. Therefore a value expressed as Y′UV is only predictable if standard RGB colorants are used (i.e. a fixed set of primary chromaticities, or particular set of red, green, and blue).

Furthermore, the range of colors and brightnesses (known as the color gamut) of RGB (whether it be BT.601 or Rec.709) is far smaller than the range of colors and brightnesses allowed by Y′UV. This can be very important when converting from Y′UV (or Y′CbCr) to RGB, since the formulas above can produce "invalid" RGB values – i.e., values below 0% or very far above 100% of the range (e.g. outside the standard 16-235 luma range (and 16-240 chroma range) for TVs and HD content, or outside 0-255 for standard definition on PCs). Unless these values are dealt with they will usually be "clipped" (i.e., limited) to the valid range of the channel affected. This changes the hue of the color, so it is therefore often considered better to desaturate the offending colors such that they fall within the RGB gamut.[9] Likewise, when RGB at a given bit depth is converted to YUV at the same bit depth, several RGB colors can become the same Y′UV color, resulting in information loss.

Relation with Y′CbCr

Y′UV is often used as a term for YCbCr. However, they are completely different formats with different scale factors.[10]

Nevertheless, the relationship between them in the standard case is simple. In particular, the Y channel is the same in both, both Cb and U are proportional to (B-Y), and both Cr and V are proportional to (R-Y).

Types of sampling

To get a digital signal, Y′UV images can be sampled in several different ways; see chroma subsampling.

Converting between Y′UV and RGB

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:

```
r0 = rgb[0];
g0 = rgb[1];
b0 = rgb[2];
r1 = rgb[3];
g1 = rgb[4];
b1 = rgb[5];
 ...
```

Y′UV files can be encoded in 12, 16 or 24 bits per pixel. The common formats are Y′UV444 (or YUV444), YUV411, Y′UV422 (or YUV422) and Y′UV420p (or YUV420). The apostrophe after the Y is often omitted, as is the "p" after YUV420p. In terms of actual file formats, YUV420 is the most common, as the data is more easily compressed, 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.[11][12]

To convert from RGB to YUV or back, it is simplest to use RGB888 and YUV444. For YUV411, YUV422 and YUV420, the bytes need to be converted to YUV444 first.

```
YUV444    3 bytes per pixel     (12 bytes per 4 pixels)
YUV422    4 bytes per 2 pixels  ( 8 bytes per 4 pixels)
YUV411    6 bytes per 4 pixels
YUV420p   6 bytes per 4 pixels, reordered
```

Y′UV444 to RGB888 conversion

The function [R, G, B] = Y′UV444toRGB888(Y′, U, V) converts Y′UV format to simple RGB format.

The RGB conversion formulae used for Y′UV444 format are also applicable to the standard NTSC TV transmission format of YUV420 (or YUV422 for that matter). For YUV420, since each U or V sample is used to represent 4 Y samples that form a square, a proper sampling method can allow the utilization of the exact conversion formulae shown below. For more details, please see the 420 format demonstration in the bottom section of this article.

These formulae are based on the NTSC standard:

$$\begin{aligned} Y' &= 0.299 \times R + 0.587 \times G + 0.114 \times B \\ U &= -0.147 \times R - 0.289 \times G + 0.436 \times B \\ V &= 0.615 \times R - 0.515 \times G - 0.100 \times B \end{aligned}$$

On older, non-SIMD architectures, floating point arithmetic is much slower than using fixed-point arithmetic, so an alternative formulation is:[13]

$$\begin{aligned} Y' &= ((66 \times R + 129 \times G + 25 \times B + 128) >> 8) + 16 \\ U &= ((-38 \times R - 74 \times G + 112 \times B + 128) >> 8) + 128 \\ V &= ((112 \times R - 94 \times G - 18 \times B + 128) >> 8) + 128 \end{aligned}$$

For the conversion from Y'UV to RGB, using the coefficients C, D and E and noting that clamp() denotes clamping a value to the range of 0 to 255, the following formulae provide the conversion from Y′UV to RGB (NTSC version):

$$\begin{aligned} C &= Y' - 16 \\ D &= U - 128 \\ E &= V - 128 \end{aligned}$$

$$\begin{aligned} R &= \text{clamp}((298 \times C + 409 \times E + 128) >> 8) \\ G &= \text{clamp}((298 \times C - 100 \times D - 208 \times E + 128) >> 8) \\ B &= \text{clamp}((298 \times C + 516 \times D + 128) >> 8) \end{aligned}$$

Note: The above formulae are actually implied for YCbCr. Though the term YUV is used here, it should be noted that YUV and YCbCr are not exactly the same in a strict manner.

The ITU-R version of the formulae is different:

$$\begin{aligned} Y &= 0.299 \times R + 0.587 \times G + 0.114 \times B + 0 \\ C_b &= -0.169 \times R - 0.331 \times G + 0.499 \times B + 128 \\ C_r &= 0.499 \times R - 0.418 \times G - 0.0813 \times B + 128 \end{aligned}$$

$$\begin{aligned} R &= \text{clamp}(Y + 1.402 \times (Cr - 128)) \\ G &= \text{clamp}(Y - 0.344 \times (Cb - 128) - 0.714 \times (Cr - 128)) \\ B &= \text{clamp}(Y + 1.772 \times (Cb - 128)) \end{aligned}$$

Integer operation of ITU-R standard for YCbCr(8 bits per channel) to RGB888:

$$\begin{aligned} C_r &= C_r - 128; \\ C_b &= C_b - 128; \\ R &= Y + C_r + (C_r >> 2) + (C_r >> 3) + (C_r >> 5) \\ G &= Y - ((C_b >> 2) + (C_b >> 4) + (C_b >> 5)) - ((C_r >> 1) + (C_r >> 3) + (C_r >> 4) + (C_r >> 5)) \\ B &= Y + C_b + (C_b >> 1) + (C_b >> 2) + (C_b >> 6) \end{aligned}$$

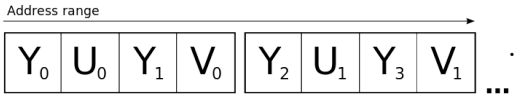
Y′UV422 to RGB888 conversion

Input: Read 4 bytes of Y′UV (u, y1, v, y2)
Output: Writes 6 bytes of RGB (R, G, B, R, G, B)

```
u  = yuv[0];
y1 = yuv[1];
v  = yuv[2];
y2 = yuv[3];
```

Using this information it could be parsed as regular Y′UV444 format to get 2 RGB pixels info:

```
rgb1 = Y′UV444toRGB888(y1, u, v);
rgb2 = Y′UV444toRGB888(y2, u, v);
```

Y′UV422 can also be expressed in YUY2 FourCC format code. That means 2 pixels will be defined in each macropixel (four bytes) treated in the image. 

Y′UV411 to RGB888 conversion

Input: Read 6 bytes of Y′UV
Output: Writes 12 bytes of RGB

```
// Extract YUV components
u  = yuv[0];
y1 = yuv[1];
y2 = yuv[2];
v  = yuv[3];
y3 = yuv[4];
y4 = yuv[5];
```

```
rgb1 = Y′UV444toRGB888(y1, u, v);
rgb2 = Y′UV444toRGB888(y2, u, v);
rgb3 = Y′UV444toRGB888(y3, u, v);
rgb4 = Y′UV444toRGB888(y4, u, v);
```

So the result is we are getting 4 RGB pixels values (4*3 bytes) from 6 bytes. This means reducing the size of transferred data to half, with a loss of quality.

Y′UV420p (and Y′V12 or YV12) to RGB888 conversion

Y′UV420p is a planar format, meaning that the Y′, U, and V values are grouped together instead of interspersed. The reason for this is that by grouping the U and V values together, the image becomes much more compressible. When given an array of an image in the Y′UV420p format, all the Y′ values come first, followed by all the U values, followed finally by all the V values.

The Y′V12 format is essentially the same as Y′UV420p, but it has the U and V data switched: the Y′ values are followed by the V values, with the U values last. As long as care is taken to extract U and V values from the proper locations, both Y′UV420p and Y′V12 can be processed using the same algorithm.

As with most Y′UV formats, there are as many Y′ values as there are pixels. Where X equals the height multiplied by the width, the first X indices in the array are Y′ values that correspond to each individual pixel. However, there are only one fourth as many U and V values. The U and V values correspond to each 2 by 2 block of the image, meaning each U and V entry applies to four pixels. After the Y′ values, the next X/4 indices are the U values for each 2 by 2 block, and the next X/4 indices after that are the V values that also apply to each 2 by 2 block.

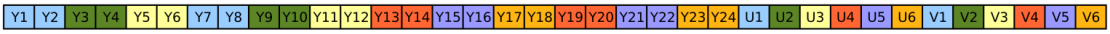
Translating Y′UV420p to RGB is a more involved process compared to the previous formats. Lookup of the Y′, U and V values can be done using the following method:

```
size.total = size.width * size.height;
y = yuv[position.y * size.width + position.x];
u = yuv[(position.y / 2) * (size.width / 2) + (position.x / 2) + size.total];
v = yuv[(position.y / 2) * (size.width / 2) + (position.x / 2) + size.total + (size.total / 4)];
rgb = Y′UV444toRGB888(y, u, v);
```

Here "/" means integer division.

Single Frame YUV420:

Position in byte stream:



As shown in the above image, the Y′, U and V components in Y′UV420 are encoded separately in sequential blocks. A Y′ value is stored for every pixel, followed by a U value for each 2×2 square block of pixels, and finally a V value for each 2×2 block. Corresponding Y′, U and V values are shown using the same color in the diagram above. Read line-by-line as a byte stream from a device, the Y′ block would be found at position 0, the U block at position x×y (6×4 = 24 in this example) and the V block at position x×y + (x×y)/4 (here, 6×4 + (6×4)/4 = 30).

Y′UV420sp (NV21) to RGB conversion (Android)

This format (NV21) is the standard picture format on Android camera preview. YUV 4:2:0 planar image, with 8 bit Y samples, followed by interleaved V/U plane with 8bit 2x2 subsampled chroma samples.[14]

C++ code used on Android to convert pixels of YUVImage:[15]

```
void YUVImage::yuv2rgb(uint8_t yValue, uint8_t uValue, uint8_t vValue,
        uint8_t *r, uint8_t *g, uint8_t *b) const {
    int rTmp = yValue + (1.370705 * (vValue-128));
    int gTmp = yValue - (0.698001 * (vValue-128)) - (0.337633 * (uValue-128));
    int bTmp = yValue + (1.732446 * (uValue-128));
    *r = clamp(rTmp, 0, 255);
    *g = clamp(gTmp, 0, 255);
    *b = clamp(bTmp, 0, 255);
}
```

References

1. Engineering Guideline EG 28, "Annotated Glossary of Essential Terms for Electronic Production," SMPTE, 1993.
2. CIELUV
3. CIE 1960 color space
4. Macadam, David L. (1 August 1937). "Projective Transformations of I. C. I. Color Specifications". *Journal of the Optical society of America*. **27** (8): 294–297. doi:10.1364/JOSA.27.000294. Retrieved 12 April 2014.
5. Maller, Joe. RGB and YUV Color (http://joemaller.com/fcp/fxscript_yuv_color.shtml), *FXScript Reference*
6. W. Wharton & D. Howorth, *Principles of Television Reception*, Pitman Publishing, 1971, pp 161-163
7. " "Super Hi-Vision" as Next-Generation Television and Its Video Parameters". Information Display. Retrieved 1 January 2013.
8. Keith Jack. *Video Demystified*. ISBN 1-878707-09-4.
9. Limiting of YUV digital video signals (BBC publication) Authors: V.G. Devereux http://downloads.bbc.co.uk/rd/pubs/reports/1987-22.pdf
10. Poynton, Charles (19 June 1999). "YUV and luminance considered harmful" (PDF). Retrieved 18 November 2016.
11. msdn.microsoft.com, Recommended 8-Bit YUV Formats for Video Rendering (http://msdn.microsoft.com/en-us/library/windows/desktop/dd391027(v=vs.85).aspx)
12. msdn.microsoft.com, YUV Video Subtypes (http://msdn.microsoft.com/de-de/library/windows/desktop/dd206750(v=vs.85).aspx)
13. https://msdn.microsoft.com/en-us/library/ms893078.aspx
14. fourcc.com YUV pixel formas (http://www.fourcc.org/yuv.php#NV21)
15. https://android.googlesource.com/platform/frameworks/av/+/master/media/libstagefright/yuv/YUVImage.cpp

External links

- RGB/Y′UV Pixel Conversion (http://www.fourcc.org/fccyvrgb.php)
- Explanation of many different formats in the Y′UV family (http://www.fourcc.org/yuv.php)

- Poynton, Charles. Video engineering (http://www.poynton.com/Poynton-video-eng.html)
- Kohn, Mike. Y′UV422 to RGB using SSE/Assembly (http://www.mikekohn.net/stuff/image_processing.php)
- YUV, YCbCr, YPbPr color spaces (http://discoverybiz.net/enu0/faq/faq_YUV_YCbCr_YPbPr.html)
- Color formats (http://www.equasys.de/colorformat.html) for image and video processing - Color conversion (http://www.equasys.de/colorconversion.html) between RGB, YUV, YCbCr and YPbPr
- How to convert RGB to YUV420P (http://www.erazer.org/how-to-convert-rgb-to-yuv420p)
- libyuv (https://code.google.com/p/libyuv/)
- pixfc-sse (https://code.google.com/p/pixfc-sse/) - C library of SSE-optimized color format conversions

Categories: Color space
