Y'UV 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, allowed 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.

The scope of the terms Y'UV, YUV, YCbCr, YPbPr, etc., is sometimes ambiguous and overlapping. Historically, the terms YUV and Y'UV was 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 NTSC, 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 receivers native black-and-white format.

Y' stands for the luma component (the brightness) and U and V are the chrominance (color) components. 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 (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), and are sometimes called Y'UV. The YIQ color space used in the analog NTSC television broadcasting system is related to it, although in a more complex way.

### Contents
- 1 History
- 2 Mathematical derivations and formulas
  - 2.1 BT.709 and BT.601
- 3 Numerical approximations
- 4 Luminance/chrominance systems in general
- 5 Confusion with Y'CbCr
- 6 Types of sampling
- 7 Converting from Y'UV to RGB
  - 7.1 Y'UV444
  - 7.2 Y'UV422
  - 7.3 Y'UV411
  - 7.4 Y'UV420p (and Y'V12)
- 8 References
- 9 External links

### History

Y'UV was invented when engineers wanted color television in a black-and-white infrastructure.[11] They needed a signal transmission method that was compatible with black-and-white (B&W) TV while being able to add color television. The luma component was already existing as the B&W signal. They added the UV signal as a solution.
The UV representation of chrominance was chosen over, say, straight R and B signals because U and V are color difference signals. This meant that in a B&W scene the U and V signals would be zero and only the Y' signal would be transmitted. If R and B were to have been used, these would have non-zero values even in a B&W scene. This was important in the early days of color television as many programs were still being made and transmitted in B&W and many TV receivers were B&W only. It was necessary to assign a narrower bandwidth to the chrominance channel (there was no additional bandwidth available) and having some of the luminance information arrive via the chrominance channel - an inevitable consequence of Y'RB - would have resulted in a loss of B&W resolution.[2]

**Mathematical derivations and formulas**

Y'UV signals are typically created from an original RGB (red, green and blue) source. The weighted values of R, G, and B are added together to produce a single Y' signal, representing the overall brightness, or luminance, of that spot. The U signal is then created by subtracting the Y' from the blue signal of the original RGB, and then scaling; V is created by subtracting the Y' from the red, and then scaling by a different factor. This can be accomplished easily with analog circuitry.

Mathematically, the analog version of Y'UV can be obtained from RGB with the following relationships

\[
\begin{align*}
W_R &= 0.299 \\
W_B &= 0.114 \\
W_G &= 1 - W_R - W_B = 0.587 \\
Y' &= W_R \times R + W_G \times G + W_B \times B \\
U &= 0.436 \times (B - Y')/(1 - W_B) \\
V &= 0.615 \times (R - Y')/(1 - W_R)
\end{align*}
\]

The U and V components can also be expressed in terms of raw R, G, and B, obtaining:

\[
\begin{bmatrix}
Y' \\
U \\
V
\end{bmatrix} =
\begin{bmatrix}
0.299 & 0.587 & 0.114 \\
-0.14713 & -0.28886 & 0.436 \\
0.615 & -0.51499 & -0.10001
\end{bmatrix}
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix}
\]

It is supposed, in all the previous equations, that \(R, G, B \in [0, 1]\).

As a consequence, the range of the transformed components is given by

\[
Y' \in [0, 1], \quad U \in [-0.436, 0.436], \quad V \in [-0.615, 0.615]
\]

The inverse relationship, from Y'UV to RGB, is given by

\[
\begin{bmatrix}
R \\
G \\
B
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 1.13983 \\
1 & -0.39465 & -0.58060 \\
1 & 2.03211 & 0
\end{bmatrix}
\begin{bmatrix}
Y' \\
U \\
V
\end{bmatrix}
\]

There are some points regarding the RGB transformation matrix:

- The top row is identical to that of the Y'IQ color space
- \(R, G, B \) correspond to \(Y', U, V\) respectively.

These formulae use the more traditional model of Y'UV, which is used for analog PAL equipment; digital PAL and digital NTSC HDTV do not use YUV but YCbcCr.

**BT.709 and BT.601**
When standardising high-definition video, the ATSC chose a different formula for the YCbCr than that used for standard-definition video. This means that when converting between SDTV and HDTV, the color information has to be altered, in addition to image scaling the video.

The formulas above reference Rec. 601. For HDTV, a slightly different matrix is used, where \( W_R \) and \( W_B \) in the above formula is replaced by Rec. 709:

\[
\begin{align*}
W_R &= 0.2126 \\
W_B &= 0.0722
\end{align*}
\]

**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. In the following examples, the operator "\( a >> b \)" denotes an integer division by a power of two, which is equivalent to a right-shift of \( a \) by \( b \) bits.

Traditional 8 bit representation of Y'UV with unsigned integers uses the following

**Basic Transform**

\[
\begin{align*}
Y' &= 66 \times R + 129 \times G + 25 \times B \\
U &= -38 \times R - 74 \times G + 112 \times B \\
V &= 112 \times R - 94 \times G - 18 \times B
\end{align*}
\]

**Scale down to 8 bits with rounding**

\[
\begin{align*}
Y &= (Y' + 128) >> 8 \\
U &= (U + 128) >> 8 \\
V &= (V + 128) >> 8
\end{align*}
\]

**Shift values**

\[
\begin{align*}
Y' &= + 16 \\
U &= + 128 \\
V &= + 128
\end{align*}
\]

Y' values are conventionally shifted and scaled to the range \([16, 235]\) rather than using the full range of \([0, 255]\). This confusing practice derives from the MPEG standards and explains why 16 is added to Y' and why the Y' coefficients in the basic transform sum to 220 instead of 255. U and V values, which may be positive or negative, are shifted by 128 to make them always positive.

In 16-bit (modulo 65,536) arithmetic, we have

\[
\begin{align*}
Y'' &= \min\{\text{abs}(r \times 2104 + g \times 4130 + b \times 802 + 4096 + 131072) >> 13, 235\} \\
U &= \min\{\text{abs}(r \times -1214 + g \times -2384 + b \times 3598 + 4096 + 1048576) >> 13, 240\} \\
V &= \min\{\text{abs}(r \times 3598 + g \times -3013 + b \times -585 + 4096 + 1048576) >> 13, 240\}
\end{align*}
\]

16 bit Y'UV to RGB conversion formulae

\[
\begin{align*}
r &= \min\{(9535 \times (y - 16) + 13074 \times (v - 128)) >> 13, 255\} \\
g &= \min\{(9535 \times (y - 16) - 6660 \times (v - 128) - 3203 \times (u - 128)) >> 13, 255\} \\
b &= \min\{(9535 \times (y - 16) + 16531 \times (u - 128)) >> 13, 255\}
\end{align*}
\]

**Luminance/chrominance systems in general**

The primary advantages of luminance/chrominance systems such as Y'UV, and its relatives Y'IQ and YDbDr, are that they remain compatible with black and white analog television (largely due to the work of Georges Valensi). The Y' channel saves nearly 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 color sensitivity: the accuracy of the brightness information of the luminance channel has far more impact on the image discerned than that of the other two. Understanding this human shortcoming, standards such as NTSC reduce the amount...
of data consumed by the chrominance channels considerably, leaving the brain to extrapolate much of the color. Therefore, the resulting U and V signals can be substantially compressed.

However, this color space conversion is lossy. 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 compressed 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 technology, attempts have been made to preserve more of the Y'UV signal while recording images, such as S-Video on VCRs.

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. The professional CCIR 601 uncompressed digital video format also uses Y'CbCr, 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).

Confusion with Y'CbCr

Y'UV is often and mistakenly used as the term for YCbCr. However, they are different formats. Y'UV is an analog system with scale factors different than the digital Y'CbCr system.[3]

In digital video/image systems, Y'CbCr is the most common way to express color in a way suitable for compression/transmission. The confusion stems from computer implementations and text-books erroneously using the term YUV where Y'CbCr would be correct.

Types of sampling

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

Converting from Y'UV to RGB

The function \([R, G, B] = Y'UV444toRGB888(Y', U, V)\) converts Y'UV format to simple RGB format and could be implemented using floating point arithmetic as:

**Y'UV444**

\[
Y' = 0.299 \times R + 0.587 \times G + 0.114 \times B \\
U = -0.1687 \times R - 0.3313 \times G + 0.5 \times B \\
V = 0.5 \times R - 0.4187 \times G - 0.0813 \times B \\
\]

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

\[
C = Y' \, \text{clip}(16) \\
D = U \, \text{clip}(128) \\
E = V \, \text{clip}(256) \\
\]

Using the previous coefficients and noting that clip() denotes clipping a value to the range of 0 to 255, the following formulas provide the conversion from Y'UV to RGB:

\[
R = \text{clip}(\lfloor 298 \times C + 409 \times E + 128 \rfloor >> 8) \\
G = \text{clip}(\lfloor 298 \times C - 100 \times D - 208 \times E + 128 \rfloor >> 8) \\
B = \text{clip}(\lfloor 298 \times C + 516 \times D + 128 \rfloor >> 8) \\
\]

**Y'UV422**
Input: Read 4 bytes of Y’UV (u, y1, v, y2 )

Using this information it could be parsed as regular Y’U444 format to get 2 RGB pixels info:

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

Y’UV422 can also be expressed in Y’U’Y2 FourCC format code. That means 2 pixels will be defined in each macropixel (four bytes) treated in the image.

**Y’UV411**

```c
// Extract YUV components
u = yuv[0];
y1 = yuv[1];
v = yuv[2];
y2 = 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 size of transferred data to half and with quite good loss of quality.

**Y’UV420p (and Y’V12)**

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 reversed: 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 rather involved process compared to the previous formats. Taking a 16 by 16 image for example, getting the RGB values for pixel (5, 7) where (0, 0) is the top left pixel would be done as follows. The character "/" implies integer division, meaning that if there is a remainder, it will be discarded.
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).

Raw Y'CbCr streams are often stored in files with extension ".yuv". Here are some freely available examples for research purposes. These are simply a sequence of Y'CbCr frames serialized into a byte stream.

References

1. ^ Maller, Joe. RGB and YUV Color, FXScript Reference

External links

- RGB/Y'UV Pixel Conversion
- Explanation of many different formats in the Y'UV family
- Poynton, Charles. Video engineering
- Kohn, Mike. Y'UV422 to RGB using SSE/Assembly


Categories: Color space
Hidden category: Articles needing additional references from August 2008