主要的采样格式有YCbCr 4:2:0、YCbCr 4:2:2、YCbCr 4:1:1和 YCbCr 4:4:4。其中YCbCr 4:1:1 比较常用，其含义为：每个点保存一个 8bit 的亮度值(也就是Y值), 每 2x2 个点保存一个 Cr 和Cb 值, 图像在肉眼中的感觉不会起太大的变化。所以, 原来用 RGB(R,G,B 都是 8bit unsigned) 模型, 4 个点需要 8x3=24 bites（如下图第一个图）. 而现在仅需要 8+(8/4)+(8/4)=12bites, 平均每个点占12bites(如下图第二个图)。这样就把图像的数据压缩了一半。

上边仅给出了理论上的示例，在实际数据存储中是有可能是不同的，下面给出几种具体的存储形式：

（1） YUV 4:4:4

YUV三个信道的抽样率相同，因此在生成的图像里，每个象素的三个分量信息完整（每个分量通常8比特），经过8比特量化之后，未经压缩的每个像素占用3个字节。

下面的四个像素为: [Y0 U0 V0] [Y1 U1 V1] [Y2 U2 V2] [Y3 U3 V3]

存放的码流为: Y0 U0 V0 Y1 U1 V1 Y2 U2 V2 Y3 U3 V3

（2） YUV 4:2:2

每个色差信道的抽样率是亮度信道的一半，所以**水平方向**的色度抽样率只是4:4:4的一半。对非压缩的8比特量化的图像来说，每个由两个水平方向相邻的像素组成的宏像素需要占用4字节内存**（亮度2个字节,两个色度各1个字节）。**。

下面的四个像素为: **[Y0 U0 V0] [Y1 U1 V1]** [Y2 U2 V2] [Y3 U3 V3]

存放的码流为: **Y0 U0 Y1 V1**Y2 U2 Y3 V3

映射出像素点为：[Y0 U0 V1] [Y1 U0 V1] [Y2 U2 V3] [Y3 U2 V3]

（3） YUV 4:1:1

4:1:1的色度抽样，是在**水平方向上对色度进行4:1抽样**。对于低端用户和消费类产品这仍然是可以接受的。对非压缩的8比特量化的视频来说，每个由4个水平方向相邻的像素组成的宏像素需要占用6字节内存**（亮度4个字节,两个色度各1个字节）。**

下面的四个像素为: **[Y0 U0 V0] [Y1 U1 V1] [Y2 U2 V2] [Y3 U3 V3]**

存放的码流为: **Y0 U0 Y1 Y2 V2 Y3**

映射出像素点为：[Y0 U0 V2] [Y1 U0 V2] [Y2 U0 V2] [Y3 U0 V2]

（4）YUV4:2:0

4:2:0并不意味着只有Y,Cb而没有Cr分量。**它指得是对每行扫描线来说，只有一种色度分量以2:1的抽样率存储。相邻的扫描行存储不同的色度分量**， 也就是说，如果一行是4:2:0的话，下一行就是4:0:2，再下一行是4:2:0...以此类推。对每个色度分量来说，水平方向和竖直方向的抽样率都是 2:1，所以可以说色度的抽样率是4:1。对非压缩的8比特量化的视频来说，每个由2x2个2行2列相邻的像素组成的宏像素需要占用6字节内存**（亮度4个字节,两个色度各1个字节）。**

下面八个像素为：**[Y0 U0 V0] [Y1 U1 V1]**[Y2 U2 V2] [Y3 U3 V3]

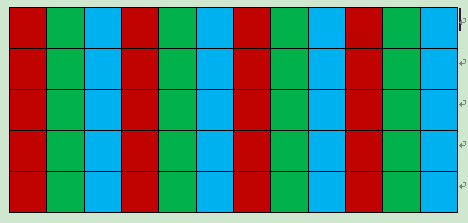
**[Y5 U5 V5] [Y6 U6 V6]** [Y7U7 V7] [Y8 U8 V8]

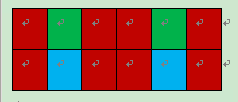
存放的码流为：**Y0 U0 Y1** Y2 U2 Y3

**Y5 V5 Y6**Y7 V7 Y8

映射出的像素点为：[Y0 U0 V5] [Y1 U0 V5] [Y2 U2 V7] [Y3 U2 V7]

[Y5 U0 V5] [Y6 U0 V5] [Y7U2 V7] [Y8 U2 V7]





**Y'UV420p (and Y'V12 or YV12)**

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 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;

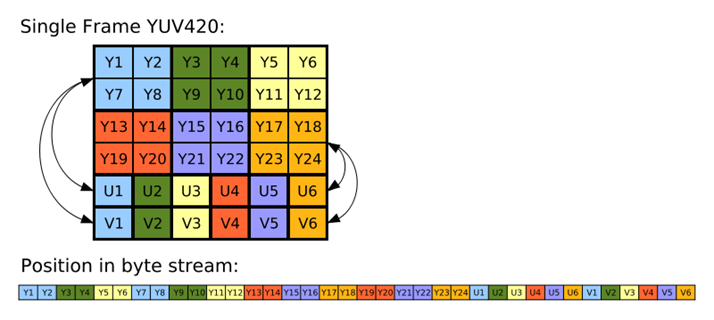
y = yuv[position.y \* size.width + position.x];

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 "/" is Div not division.

[](http://en.wikipedia.org/wiki/File:Yuv420.svg)

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).