FORMATION OF YUV (YCbCr) IMAGE FROM JPEG DATA

A thesis submitted in partial fulfillment of the requirements for the award of the degree of

MASTER OF ENGINEERING

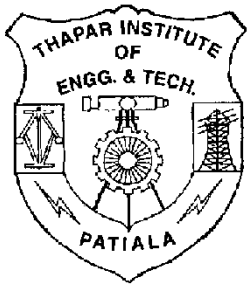
IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted By:
ANSHUMAN GOYAL
Roll No-8044106

Under the guidance of:

Mr. Kulbir Singh (TIET Patiala)
Mr. Yogender Gupta (HCL Technologies, Noida)



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY (DEEMED UNIVERSITY), PATIALA – 147004.

Year - 2006

CERTIFICATE

I, Anshuman Goyal hereby certify that the work which is being presented in this thesis entitled **"Formation of YUV (YCbCr) Image From JPEG Data"** (Work Carried out at **HCL Technologies, Sector – 24, Noida**) by me in partial fulfillment of requirements for the award of degree of Master of Engineering in Electronics and Communication from Thapar Institute of Engineering and Technology (Deemed University), Patiala, is an authentic record of my own work carried under the supervision of Mr. Yogender Gupta (Project Manager, HCL Technologies) and Mr. Kulbir Singh, TIET Patiala.

The matter presented in this thesis has not been submitted in any other University or Institute for the award of Master of Engineering.

(**Anshuman Goyal**)

This is certified that the above statement made by the candidate is correct to the best of my knowledge

(Yogender Gupta)
HCL Technologies, Noida.

(Kulbir Singh)
Senior Lecturer, TIET Patiala

Counter Signed:

Head of Department, ECED
T.I.E.T, Patiala.

Dean of Academic Affairs,
T.I.E.T, Patiala.

ACKNOWLEDGEMENT

Words are often to less to revels one's deep regards. An understanding of the work like this is never the outcome of the efforts of a single person. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis.

First, I would like to thank the Supreme Power, the GOD, one who has always guided me to work on the right path of the life. Without his grace, this would never come to be today's realty.

This work would not have been possible without the encouragement and able guidance of my **Supervisor 'Mr. Kulbir Singh'**, and **Project Manager** at HCL Technologies, **'Mr. Yogender Gupta'.** Their enthusiasm and optimism made this experience both rewarding and enjoyable. Most of the novel ideas and solutions found in this thesis are the result of our numerous stimulating discussions. His feedback and editorial comments were also invaluable for the writing of this thesis. I am grateful to **'Dr. R. S. Kaler' Prof. & Head, ECED** for providing the facilities for the completion of thesis. I am also grateful to **HCL Technologies, Sector – 24 Noida**, for providing me the facilities for the completion of thesis.

I take pride of my self being son of ideal great parents whose everlasting desire, sacrifice, affectionate blessing and help without which it would have not been possible for me to complete my studies.

At last, I would like to thank to all the members and employees of Electronics and Communication Department, TIET Patiala and HCL Technologies, Sector – 24, Noida whose love and affection made my stay at both the places a memorable.

Place: HCL Technologies, Noida (Anshuman Goyal)

Abstract

JPEG file format is supported by a wide variety of applications on a variety of platforms and is the most common file format used. JPEG images are constructed from RGB data. Compression allows more images to be saved in the same amount of disk space while preserving the ICC Color profile and caption information. Note, however, that JPEG compression results in loss of image information that can not afterwards be restored. EXIF is an extension of the JPEG format that allows thumbnail data and information about the image to be included in a JPEG file.

Images are saved in uncompressed YCbCr format, where the picture data is represented in one luminance (Y) and 2 color channels (Cb and Cr) instead of the normal 3 color channels (RGB). The two color channels occupy the same space as the luminance channel and the file size is therefore two thirds of the size of a RGB file. YCbCr is a more efficient mode of the image representation than RGB and has the same quality but occupies less space. We use YCbCr if we want the best combination of high quality and the lowest file size in an uncompressed finished file. YUV models human perception of color more closely than the standard RGB model used in computer graphics hardware. The primary advantages of luminance/chrominance systems such as YUV are that they remain compatible with black and white analog television. 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 YUV 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. By understanding this human shortcoming, standards such as NTSC are used to reduce the amount of data consumed by the chrominance channels and considerably leaving the eye to extrapolate more of the color. NTSC saves only 11% of the original blue and 30% of the red. The green information is usually preserved in the Y channel. Therefore, the resulting U and V signals can be substantially compressed.

LIST OF ABBREVIATIONS

YUV	YUV Image Format
BER	Bit Error Rate
CD	Compact Disk
CDMA	Code Division Multiple Access
CR	Compression Ratio
DCT	Discrete Cosine Transform
DVD	Digital Versatile Disk
DWT	Discrete Wavelet Transform
FFT	Fast Fourier Transform
FT	Fourier Transform
HDTV	High Definition Television
HVS	Human Visual System
LSB	Least Significant Bit
MSE	Mean Square Error
PSNR	Peak Signal to Noise Ratio
RMSE	Root Mean Square Error

Table of Contents

Chapter 1 - Introduction

Modern hardware and database technology has made it possible to store gigabytes of information in databases. However, it can be difficult to utilize fully the information in a large database with complicated structures. Many methods, such as data mining, have been proposed and studied to help users better understand and analyze the information. Database visualization is one of the effective solutions to this problem. Database visualization becomes more appealing when handling large data sets with complex relationships because information presented in the form of images is more direct and easily understood by humans

When processing images for a human observer, it is important to consider how images are converted into information by the viewer. Understanding visual perception helps during algorithm development.

Image data represents physical quantities such as chromaticity and luminance. Chromaticity is the color quality of light defined by its wavelength. Luminance is the amount of light. To the viewer, these physical quantities may be perceived by such attributes as color and brightness.

How we perceive color image information is classified into three perceptual variables: hue, saturation and lightness. When we use the word color, typically we are referring to hue. Hue distinguishes among colors such as green and yellow. Hues are the color sensations reported by an observer exposed to various wavelengths. It has been shown that the predominant sensation of wavelengths between 430 and 480 nanometers is blue. Green characterizes a broad range of wavelengths from 500 to 550 nanometers. Yellow covers the range from 570 to 600 nanometers and wavelengths over 610 nanometers are categorized as red. Black, gray, and white may be considered colors but not hues.

Saturation is the degree to which a color is undiluted with white light. Saturation decreases as the amount of a neutral color added to a pure hue increases. Saturation is often thought of as how pure a color is. Unsaturated colors appear washed-out or faded, saturated colors are bold and vibrant. Red is highly saturated; pink is unsaturated. A pure

color is 100 percent saturated and contains no white light. A mixture of white light and a pure color has saturation between 0 and 100 percent.

Lightness is the perceived intensity of a reflecting object. It refers to the gamut of colors from white through gray to black; a range often referred to as gray level. A similar term, brightness, refers to the perceived intensity of a self-luminous object such as a CRT. The relationship between brightness, a perceived quantity, and luminous intensity, a measurable quantity, is approximately logarithmic.

Contrast is the range from the darkest regions of the image to the lightest regions. The mathematical representation is:

$$Contrast = \frac{I_{max} - I_{min}}{I_{max} + I_{min}}$$

Where I_{max} and I_{min} are the maximum and minimum intensities of a region or image. High-contrast images have large regions of dark and light. Images with good contrast have a good representation of all luminance intensities.

As the contrast of an image increases, the viewer perceives an increase in detail. This is purely a perception as the amount of information in the image does not increase. Our perception is sensitive to luminance contrast rather than absolute luminance intensities.

1.1 - Objective

The RGB and YUV color spaces are linear transformations of each other. RGB has channels for red-green-blue components, and YUV has channels for color (UV) and illumination (Y). Because color and illumination are separate, YUV has the ability to be more illumination-independent which means shadows and illumination changes don't affect it as much. In this thesis the work has been carried out to process a JPEG image and convert that image into YUV (YCbCr) image format. The advantage in using this process is that this will make the image size very small, since in YUV image the Luminance component (Y) has got all the information about the image and we can with the help of coding and decoding of (Cb and Cr) components reduce the size of image up to very small bits as compared to JPEG Image.

The work presented in this thesis has the main objective of converting the JPEG images into YCbCr (YUV) images so as to compress the image data so that it can be easily transferred to the networks having very less bandwidth.

In the following thesis first of all the various image formats have been explained such as GIF, JPEG, PNG and YUV. Since the YUV format is the most important one so stress has been laid on the Then the YUV formats have been explained in detail. The advantages of YUV images and their compression property have also been explained after that.

1.2 - Summary of Work Done

In the following thesis we have converted a JPEG image into YUV (YCbCr) image format by taking the DCT and Quantizing the image. There are several ways to convert to/from YCbCr. This is the CCIR (International Radio Consultant Committee) recommendation 601-1 and is the typical method used in JPEG compression.

A Decoder has been designed for the same, which converts the JPEG image into YCbCr format. The conversion process has been achieved with the use of C language. The final step which is the compression of the Chrominance component of the YUV image has been achieved by the use of Lossless coding techniques, so as to make the image size smaller enough so that it can be transferred easily on low bandwidth networks.

In the second stage of this project, the JPEG image is reconstructed from the YUV Image. This conversion is the part of Encoder which converts the YUV image into JPEG format. The quality of the image thus received is checked from the PSNR Ratio.

The size of the file has been compared in the Table – 5.3.3 which concludes that the size of the image can be reduced to very large extent if YUV images are used.

Chapter 2: The Various Image Formats

2.1 - The RGB Image Format

If we take a look at our computer monitor's display with a magnifying lens we would see that it consists of a very large number of triplets of red, green and blue dots [1 - 5]. Depending on the monitor manufacturer they may be round dots or small squares or other shapes but there always will be triplets of red, green and blue elements. In most monitors the dots are quite large and easy to see even with a weak lens.

Images on the computer display are formed when the monitor precisely varies the brightness of the red, green and blue elements in each triplet [3, 4, 5, 6]. Because the dots are close together the human eye will fuse the three red, green and blue dots of varying brightness into a single dot that appears to be the color combination of the three levels of red, green and blue color [2]. For those of us who missed art class in school, all colors perceived by humans can be formed by the right brightness combination of red, green and blue color. Amazing, but true!

From here on in we'll refer to the Red, Green and Blue elements of images as "R", "G" and "B".

Images on computers are usually nothing more than a series of number triplets where each triplet of three numbers is intended to control a single triplet of R, G and B dots on the monitor. Each triplet of numbers is a single pixel. A number of triplets such as 67, 228, 180 which means to turn the R element up to 67 brightness, the G element to 228 brightness and the B element to 180 brightness. The result will be seen by humans as a pretty shade of green-blue color. The RGB numbers are called color values.

Image files are arranged so that there is a series of pixels in rows that correspond to the rows of tiny triplet dots on the monitor. When an image is displayed on the monitor so that each pixel triplet in the image file controls one color triplet on the monitor we say that the image is "natural size" or "100% zoom".

In most modern software packages we take for granted the ability to zoom in or zoom out of an image to see it smaller or larger than natural size. If we zoom in, our graphics software will take the RGB values in a pixel that were originally intended to

control just one color triplet and will use that same value across and down as many triplets as is necessary to make the image bigger on the monitor. If we zoom out so that the image is much smaller than intended, then there will be more RGB values than there are color triplets to control. In that case our graphics software will average out the RGB pixel values to figure out what average value should be used to drive the color triplet for that spot in the image. Manifold System performs these functions automatically.

2.2 - The GIF Image Format

GIF stands for Graphics Interchange Format. It is probably the most common image format used on the Web. GIF's have the advantage of usually being very small in size, which makes them fast-loading [12, 13, 28]. Unlike JPEG's, GIF's use lossless compression, which means they make the file size small without losing or blurring any of the image itself.

Another good thing that GIF's can do is animation. We can make an animated GIF by drawing each frame of the animation in a graphics package that supports the animated GIF format, then export the animation to a single GIF file.

The major disadvantage of GIF's is that they only support up to 256 colors (this is known as 8-bit color and is a type of indexed color image). This means they're not good for photographs, or any other image application that contains lots of different colors.

2.3 - The PNG Image Format

PNG is a relatively new invention. It stands for Portable Network Graphics. It was designed to be an alternative to the GIF file format, but without the licensing issues that are involved in the GIF compression method.

There are two types of PNG: PNG-8 format, which holds 8 bits of color information (comparable to GIF), and PNG-24 format, which holds 24 bits of color (comparable to JPEG).

PNG-8 often compresses images even better than GIF, resulting in smaller file sizes [29, 12, 13]. On the other hand, PNG-24 is often less effective than JPEG's at compressing true-color images such as photos, resulting in larger file sizes than the

equivalent quality JPEG's. However, unlike JPEG, PNG-24 is lossless, meaning that all of the original image's information is preserved.

PNG also supports transparency like GIF, but can have varying degrees of transparency for each pixel, whereas GIF's can only have transparency turned on or off for each pixel. This means that whereas transparent GIF's often have jagged edges when placed on complex or ill-matching backgrounds, transparent PNG's will have nice smooth edges.

Note that unlike GIF, PNG-8 does not support animation. Since this is the new format, therefore some old image processing applications don't understand this format.

2.4 - The JPEG Image Format

JPEG stands for Joint Photographic Experts Group, a bunch of boffins who invented this format to display full-color photographic images in a portable format with a small file size. Like GIF images, they are also very common on the Web. Their main advantage over GIF's is that they can display true-color images [29, 22, 23] (up to 16 million colors), which makes them much better for images such as photographs and illustrations with large numbers of colors.

The main disadvantage of the JPEG format is that it is lossy. This means that we lose some of the detail of our image when we convert it to JPEG format. Boundaries between blocks of color may appear more blurry, and areas with lots of detail will lose their sharpness. On the other hand, JPEG's do preserve all of the color information in the image, which of course is great for high-color images such as photographs. JPEG's also can't do transparency or animation - in these cases; we have to use the GIF format (or PNG format for transparency).

2.5 - The YUV (YCbCr) Image Format

The YUV model defines a color space in terms of one luminance and two chrominance components. YUV is used in the PAL and NTSC systems of television broadcasting, which are the standards in much of the world.

YUV models human perception of color more closely than the standard RGB model used in computer graphics hardware, but not as closely as the HSL color space and HSV color space.

Y stands for the luminance component (the brightness) and U and V are the chrominance (color) components. The YCbCr or YPbPr color space, used in component video, is derived from it (Cb/Pb and Cr/Pr are simply scaled versions of U and V), and is sometimes inaccurately called "YUV". The YIQ color space used in the NTSC television broadcasting system is related to it, although in a more complex way.

YUV signals are 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; and V by subtracting the Y from the red, and then scaling by a different factor. This can be accomplished easily with analog circuitry.

The following equations can be used to derive Y, U and V from R, G and B:

$$Y = 0.299R + 0.587G + 0.114B$$
$$U = 0.492(B - Y) = -0.147R - 0.289G + 0.436B$$
$$V = 0.877(R - Y) = 0.615R - 0.515G - 0.100B$$

2.6 - Color Spaces

A color space is a mathematical representation of a set of colors. The three most popular color models are RGB (used in computer graphics); YIQ, YUV, or YCbCr (used in video systems); and CMYK (used in color printing) [25 -28]. However, none of these color spaces are directly related to the intuitive notions of hue, saturation, and brightness. This resulted in the temporary pursuit of other models, such as HSI and HSV, to simplify programming, processing, and end user manipulation. All of the color spaces can be derived from the RGB information supplied by devices such as cameras and scanners [31].

2.6.1 - RGB Color Space

The red, green, and blue (RGB) color space is widely used throughout computer graphics. Red, green, and blue are three primary additive colors (individual components are added together to form a desired color) and are represented by a three-dimensional, Cartesian coordinate system (Figure 2.6.1). The indicated diagonal of the cube, with equal amounts of each primary component, represents various gray levels. Table 2.6.1 contains the RGB values for 100% amplitude, 100% saturated color bars, a common video test signal.

	Normal Range	White	Yellow	Cyan	Green	Magenta	Red	Blue	Black
R	0 to 255	255	255	0	0	255	255	0	0
G	0 to 255	255	255	255	255	0	0	0	0
B	0 to 255	255	0	255	0	255	0	255	0

Table2.6.1 → *100% RGB Color Bars*

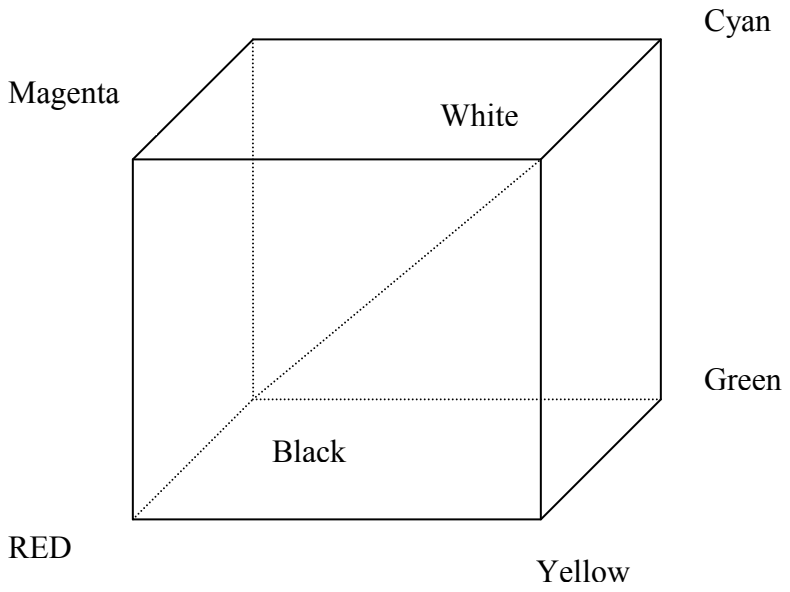


Figure 2.6.1 → *The RGB Color Cube*

The RGB color space is the most prevalent choice for computer graphics because color displays use red, green, and blue to create the desired color [15 -19]. Therefore, the choice of the RGB color space simplifies the architecture and design of the system. Also, a system that is designed using the RGB color space can take advantage of a large number of existing software routines, since this color space has been around for a number of years. However, RGB is not very efficient when dealing with "real-world" images. All three RGB components need to be of equal bandwidth to generate any color within the RGB color cube. The result of this is a frame buffer that has the same pixel depth and display resolution for each RGB component. Also, processing an image in the RGB color space is usually not the most efficient method. For example, to modify the intensity or color of a given pixel, the three RGB values must be read from the frame buffer, the intensity or color calculated, the desired modifications performed, and the new RGB values calculated and written back to the frame buffer. If the system had access to an image stored directly in the intensity and color format, some processing steps would be faster. For these and other reasons, many video standards use luma and two color difference signals. The most common are the YUV, YIQ, and YCbCr color spaces. Although all are related, there are some differences.

2.6.2 - YUV Color Space

The YUV model defines a color space in terms of one luminance and two chrominance components [19]. YUV is used in the PAL and NTSC systems of television broadcasting, which is the standard in much of the world.

YUV models human perception of color more closely than the standard RGB model used in computer graphics hardware, but not as closely as the HSV color space.

Y stands for the luminance component (the brightness) and U and V are the chrominance (color) components. The YCbCr or YPbPr color space, used in component video, is derived from it (Cb/Pb and Cr/Pr are simply scaled versions of U and V), and is sometimes inaccurately called "YUV". The YIQ color space used in the NTSC television broadcasting system is related to it, although in a more complex way.

YUV signals are 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; and V by subtracting the Y from the red, and then scaling by a different factor. This can be accomplished easily with analog circuitry.

The YUV color space is used by the PAL (Phase Alternation Line), NTSC (National Television System Committee), and SECAM (Sequentiel Couleur Avec Mémoire or Sequential Color with Memory) composite color video standards [24 -27]. The black-and-white system used only luma (Y) information; color information (U and V) was added in such a way that a black-and-white receiver would still display a normal black-and-white picture. Color receivers decoded the additional color information to display a color picture.

The basic equations to convert between gamma-corrected RGB (notated as R′G′B′ and discussed later in this chapter) and YUV are:

$$Y = 0.299R' + 0.587G' + 0.114B' \qquad Eq - 2.6.2.1$$

$$U = -0.147R' - 0.289G' + 0.436B' \qquad Eq - 2.6.2.2$$

$$= 0.492\,(B' - Y)$$

$$V = 0.615R' - 0.515G' - 0.100B' \qquad Eq - 2.6.2.3$$

$$= 0.877(R' - Y)$$

$$R' = Y + 1.140V \qquad Eq - 2.6.2.4$$

$$G' = Y - 0.395U - 0.581V \qquad Eq - 2.6.2.5$$

$$B' = Y + 2.032U \qquad Eq - 2.6.2.6$$

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.578 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad Eq - 2.6.2.7$$

For digital R′G′B′ values with a range of 0 - 255, Y has a range of 0–255, U a range of 0 to ±112, and V a range of 0 to ±157. These equations are usually scaled to

simplify the implementation in an actual NTSC or PAL digital encoder or decoder. Note that for digital data, 8-bit YUV and R′G′B′ data should be saturated at the 0 and 255 levels to avoid underflow and overflow wrap-around problems.

If the full range of (B′ – Y) and (R′ – Y) had been used, the composite NTSC and PAL levels would have exceeded what the (then current) black-and-white television transmitters and receivers were capable of supporting. Experimentation determined that modulated sub-carrier excursions of 20% of the luma (Y) signal excursion could be permitted above white and below black. The scaling factors were then selected so that the maximum level of 75% amplitude, 100% saturation yellow and cyan color bars would be at the white level (100 IRE) [27].

2.6.3 - YUV Formats

YUV formats fall into two distinct groups, the packed formats where Y, U (Cb) and V (Cr) samples are packed together into macro pixels which are stored in a single array, and the planar formats where each component is stored as a separate array, the final image being a fusing of the three separate planes [23].

In the diagrams below, the numerical suffix attached to each Y, U or V sample indicates the sampling position across the image line, so, for example, V_0 indicates the leftmost V sample and Y_n indicates the Y sample at the $(n+1)^{th}$ pixel from the left.

Sub sampling intervals in the horizontal and vertical directions may merit some explanation. The horizontal sub sampling interval describes how frequently across a line a sample of that component is taken while the vertical interval describes on which lines samples are taken. For example, UYVY format has a horizontal sub sampling period of 2 for both the U and V components indicating that U and V samples are taken for every second pixel across a line. Their vertical sub sampling period is 1 indicating that U and V samples are taken on each line of the image [4 -6].

For YVU9, though, the vertical sub sampling interval is 4. This indicates that U and V samples are only taken on every fourth line of the original image. Since the horizontal sampling period is also 4, a single U and a single V sample are taken for each square block of 16 image pixels.

2.6.4 - Sampling systems and ratios

The sub sampling in a video system is usually expressed as a three part ratio. The three terms of the ratio are: the number of brightness ("luminance" "luma" or Y) samples, followed by the number of samples of the two color ("chroma") components: U/Cb then V/Cr, for each complete sample area. For quality comparison, only the ratio between those values is important, so 4:4:4 could easily be called 1:1:1; however, traditionally the value for brightness is always 4, with the rest of the values scaled accordingly.

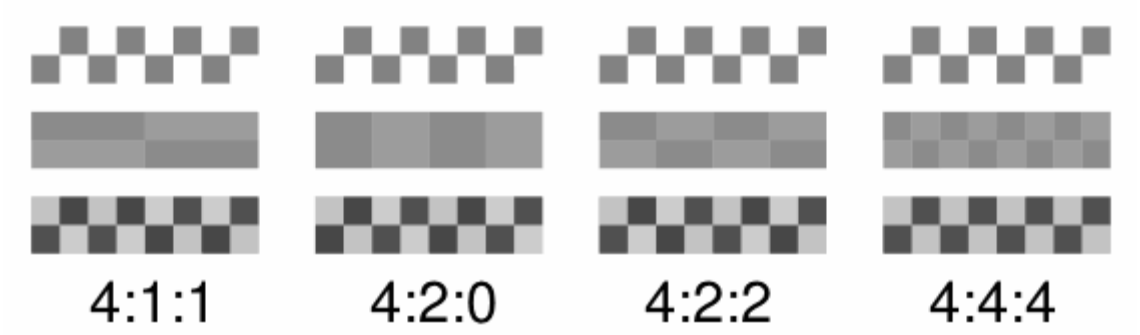


Figure 2.6.4.1 → *YCbCr Image Layout*

Sometimes, four part relations are written, like 4:2:2:4. In these cases, the fourth number means the sampling frequency ratio of a key channel. In virtually all cases, that number will be 4, since high quality is very desirable in keying applications.

The mapping examples given are only theoretical and for illustration. The bit streams of real-life implementations will most likely differ.

2.6.4.1 - 4:4:4

Each of the three channels has the same sample rate, so each single pixel in the resulting image gets three full words (usually 8 or 10 bits long) of information, resulting in 3 bytes per pixel for 8-bit quantization when not using compression.

Mapping:

The bit stream [Y_0 U_0 V_0 Y_1 U_1 V_1 Y_2 U_2 V_2 Y_3 U_3 V_3] will map to the following four pixels: [Y_0 U_0 V_0] [Y_1 U_1 V_1] [Y_2 U_2 V_2] [Y_3 U_3 V_3]

This is the best color sampling ratio (it yields a perfect representation of each pixel's color), and is used as an intermediate format in high-end film scanners and cinematic postproduction. Note that 4:4:4 may (and often indeed does, as in some modes of HDCAM SR) also mean that the three values are all color values in the RGB color space, which must always be sampled at the same frequency.

2.6.4.2 - 4:2:2

Each of the two color-difference channels has half the sample rate of the brightness channel, so horizontal color resolution is only half that of 4:4:4. For uncompressed video and 8-bit quantization, each macro pixel of two neighboring pixels uses 4 bytes of memory.

Mapping:

The bit stream [Y_0 U_0 Y_1 V_1 Y_2 U_2 Y_3 V_3] will map to the following four pixels:

[Y_0 U_0 V_1] [Y_1 U_0 V_1] [Y_2 U_2 V_3] [Y_3 U_2 V_3]

This is still a very good quality, and most higher-end digital video formats use this ratio:

- Digital Beta cam.
- DVCPRO50.
- D-9.
- CCIR 601 / Serial Digital Interface / D1.

2.6.4.3 - 4:2:0

4:2:0 does not mean that there is no V or Cr information stored at all, it means that in each line, only one color difference channel is stored with half the horizontal resolution [25, 27, 28]. The channel which is stored flips each line, so the ratio is 4:2:0 for one line, 4:0:2 in the next, then 4:2:0 again, and so on. This leads to half the horizontal as well as half the vertical resolution, giving a quarter of the color resolution overall. The PAL and SECAM color systems are especially well-suited to this kind of data reduction. Uncompressed video in this format with 8-bit quantization uses 6 bytes for every macro pixel (2×2 pixels).

Mapping:

The bit stream $[Y_{o0}\ U_{o0}\ Y_{o1}\ Y_{o2}\ U_{o2}\ Y_{o3}]\ [Y_{e0}\ V_{e0}\ Y_{e1}\ Y_{e2}\ V_{e2}\ Y_{e3}]$ will map to the following two lines of four pixels each:

$[Y_{o0}\ U_{o0}\ V_{e0}]\ [Y_{o1}\ U_{o0}\ V_{e0}]\ [Y_{o2}\ U_{o2}\ V_{e2}]\ [Y_{o3}\ U_{o2}\ V_{e2}]$

$[Y_{e0}\ U_{o0}\ V_{e0}]\ [Y_{e1}\ U_{o0}\ V_{e0}]\ [Y_{e2}\ U_{o2}\ V_{e2}]\ [Y_{e3}\ U_{o2}\ V_{e2}]$

The quality of this method is very close to 4:1:1, and it is used in the following formats:

- DVD and other Main Profile MPEG-2 implementations.
- PAL DV and DVCAM.
- HDV.
- Most common JPEG and MJPEG implementations.

2.6.4.4 - 4:0:0

This ratio is possible (indeed, some codec do support it), but not widely used, since its color fidelity is even below that of VHS [25, 27, 28]. It means half the vertical and quarters the horizontal color resolutions, with only one eighth of the bandwidth of the maximum color resolutions used. Uncompressed video in this format with 8-bit quantization uses 10 bytes for every macro pixel (4 x 2 pixels).

Mapping:

The bit stream $[Y_{o0}\ U_{o0}\ Y_{o1}\ Y_{o2}\ Y_{o3}]\ [Y_{e0}\ V_{e0}\ Y_{e1}\ Y_{e2}\ Y_{e3}]$ will map to the following two lines of four pixels each:

$[Y_{o0}\ U_{o0}\ V_{e0}]\ [Y_{o1}\ U_{o0}\ V_{e0}]\ [Y_{o2}\ U_{o0}\ V_{e0}]\ [Y_{o3}\ U_{o0}\ V_{e0}]$

$[Y_{e0}\ U_{o0}\ V_{e0}]\ [Y_{e1}\ U_{o0}\ V_{e0}]\ [Y_{e2}\ U_{o0}\ V_{e0}]\ [Y_{e3}\ U_{o0}\ V_{e0}]$

Some video codec may operate at 4:0:0.5 or 4:0:0.25 as an option, so as to allow higher than VHS quality without having to take too large of a hit on bandwidth

2.6.5 - YIQ Color Space

The YIQ color space is derived from the YUV color space and is optionally used by the NTSC composite color video standard [25, 27, 28]. (The "I" stands for "in phase" and the "Q" for "quadrature," which is the modulation method used to transmit the color information.) The basic equations to convert between R′G′B′ and YIQ are:

$$Y = 0.299R' + 0.587G' + 0.114B' \qquad Eq - 2.6.5.1$$

$$I = 0.596R' - 0.275G' - 0.321B' \qquad Eq - 2.6.5.2$$

$$= V\cos 33° - U\sin 33°$$

$$= 0.736(R' - Y) - 0.268(B' - Y)$$

$$Q = 0.212R' - 0.523G' + 0.311B' \qquad Eq - 2.6.5.3$$

$$= V\sin 33° + U\cos 33°$$

$$= 0.478(R' - Y) + 0.413(B' - Y)$$

$$\begin{bmatrix} I \\ Q \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \cos(33) & \sin(33) \\ -\sin(33) & \cos(33) \end{bmatrix} \begin{bmatrix} U \\ V \end{bmatrix} \qquad Eq - 2.6.5.4$$

$$R' = Y + 0.956I + 0.621Q \qquad Eq - 2.6.5.5$$

$$G' = Y - 0.272I - 0.647Q \qquad Eq - 2.6.5.6$$

$$B' = Y - 1.107I + 1.704Q \qquad Eq - 2.6.5.7$$

For digital R′G′B′ values with a range of 0-255, Y has a range of 0–255, I have a range of 0 to ±152, and Q has a range of 0 to ±134. I and Q are obtained by rotating the U and V axes 33° [25, 27, 28]. These equations are usually scaled to simplify the implementation in an actual NTSC digital encoder or decoder.

Note that for digital data, 8-bit YIQ and R′G′B′ data should be saturated at the 0 and 255 levels to avoid underflow and overflow wrap-around problems.

2.6.6 - YCbCr Color Space

The YCbCr color space was developed as part of ITU-R BT.601 during the development of a world-wide digital component video standard. YCbCr is a scaled and offset version of the YUV color space [26, 27, 29]. Y is defined to have a nominal 8-bit range of 16 - 235; C_b and C_r are defined to have a nominal range of 16–240. There are several YCbCr sampling formats, such as 4:4:4, 4:2:2, 4:1:1 and 4:2:0 that are also described.

2.6.7 - RGB - YCbCr Equations: SDTV

The basic equations to convert between 8-bit digital R′G′B′ data with a 16–235 nominal range and YCbCr are:

$$Y_{601} = 0.299R' + 0.587G' + 0.114B' \qquad Eq - 2.6.7.1$$

$$Cb = -0.172R' - 0.339G' + 0.511B' + 128 \qquad Eq - 2.6.7.2$$

$$Cr = 0.511R' - 0.428G' - 0.083B' + 128 \qquad Eq - 2.6.7.3$$

$$R' = Y_{601} + 1.371(Cr - 128) \qquad Eq - 2.6.7.4$$

$$G' = Y_{601} - 0.698(Cr - 128) - 0.336\,(Cb - 128) \qquad Eq - 2.6.7.5$$

$$B' = Y601 + 1.732\,(Cb - 128) \qquad Eq - 2.6.7.6$$

When performing YCbCr to R′G′B′ conversion, the resulting R′G′B′ values have a nominal range of 16–235, with possible occasional excursions into the 0–15 and 236–255 values [25, 27, 28]. This is due to Y and CbCr occasionally going outside the 16–235 and 16–240 ranges, respectively, due to video processing and noise. Note that 8-bit YCbCr and R′G′B′ data should be saturated at the 0 and 255 levels to avoid underflow and overflow wrap-around problems. Table 2.6.8.1 lists the YCbCr values for 75% amplitude, 100% saturated color bars, a common video test signal.

2.6.8 - Computer Systems Considerations

If the R′G′B′ data has a range of 0–255, as is commonly found in computer systems, the following equations may be more convenient to use:

$$Y_{601} = 0.257R' + 0.504G' + 0.098B' + 16 \qquad Eq - 2.6.8.1$$

$$Cb = -0.148R' - 0.291G' + 0.439B' + 128 \qquad Eq - 2.6.8.2$$

$$Cr = 0.439R' - 0.368G' - 0.071B' + 128 \qquad Eq - 2.6.8.3$$

$$R' = 1.164(Y601 - 16) + 1.596(Cr - 128) \qquad Eq - 2.6.8.4$$

$$G' = 1.164(Y601 - 16) - 0.813(Cr - 128) - 0.391(Cb - 128) \qquad Eq - 2.6.8.5$$

$$B' = 1.164(Y601 - 16) + 2.018(Cb - 128) \qquad Eq - 2.6.8.6$$

Note that 8-bit YCbCr and R′G′B′ data should be saturated at the 0 and 255 levels to avoid underflow and overflow wrap-around problems.

	Nominal Range	White	Yellow	Cyan	Green	Magenta	Red	Blue	Black
				SDTV					
Y	16 To 235	180	162	131	112	84	65	35	16
Cb	16 To 240	128	44	156	72	184	100	212	128
Cr	16 To 240	128	142	44	58	198	212	114	128
				HDTV					
Y	16 To 235	180	168	145	133	69	51	28	16
Cb	16 To 240	128	44	147	63	193	109	212	128
Cr	16 To 240	128	136	44	52	204	212	120	128

Table 2.6.8.1 → *75% YCbCr Color Bars*

2.6.9 - RGB - YCbCr Equations: HDTV

The basic equations to convert between 8-bit digital R′G′B′ data with a 16–235 nominal range and YCbCr are:

$$Y709 = 0.213R' + 0.715G' + 0.072B' \qquad Eq - 2.6.9.1$$

$$Cb = -0.117R' - 0.394G' + 0.511B' + 128 \qquad Eq - 2.6.9.2$$

$$Cr = 0.511R' - 0.464G' - 0.047B' + 128 \qquad Eq - 2.6.9.3$$

$$R' = Y709 + 1.540(Cr - 128) \qquad Eq - 2.6.9.4$$

$$G' = Y709 - 0.459(Cr - 128) - 0.183(Cb - 128) \qquad Eq - 2.6.9.5$$

$$B' = Y709 + 1.816(Cb - 128) \qquad Eq - 2.6.9.6$$

When performing YCbCr to R′G′B′ conversion, the resulting R′G′B′ values have a nominal range of 16–235, with possible occasional excursions into the 0–15 and 236 - 255 values [25, 27, 28]. This is due to Y and CbCr occasionally going outside the 16–235 and 16–240 ranges, respectively, due to video processing and noise. Note that 8-bit YCbCr and R′G′B′ data should be saturated at the 0 and 255 levels to avoid underflow and overflow wrap-around problems.

Table 2.6.8.1 lists the YCbCr values for 75% amplitude, 100% saturated color bars, a common video test signal.

2.6.10 - Computer Systems Considerations

If the R′G′B′ data has a range of 0–255, as is commonly found in computer systems, the following equations may be more convenient to use:

$$Y709 = 0.183R' + 0.614G' + 0.062B' + 16 \qquad Eq - 2.6.10.1$$

$$Cb = -0.101R' - 0.338G' + 0.439B' + 128 \qquad Eq - 2.6.10.2$$

$$Cr = 0.439R' - 0.399G' - 0.040B' + 128 \qquad Eq - 2.6.10.3$$

$$R' = 1.164(Y709 - 16) + 1.793(Cr - 128) \qquad Eq - 2.6.10.4$$

$$G' = 1.164(Y709 - 16) - 0.534(Cr - 128) - 0.213(Cb - 128) \qquad Eq - 2.6.10.5$$

$$B' = 1.164(Y709 - 16) + 2.115(Cb - 128) \qquad Eq - 2.6.10.6$$

Note that 8-bit YCbCr and R′G′B′ data should be saturated at the 0 and 255 levels to avoid underflow and overflow wrap-around problems.

2.6.11 - 4:4:4 YCbCr Format

Figure 2.6.11.1 illustrates the positioning of YCbCr samples for the 4:4:4 formats. Each sample has a Y, a Cb, and a Cr value [25, 27, 28]. Each sample is typically 8 bits (consumer applications) or 10 bits (pro-video applications) per component. Each sample therefore requires 24 bits (or 30 bits for pro-video applications).

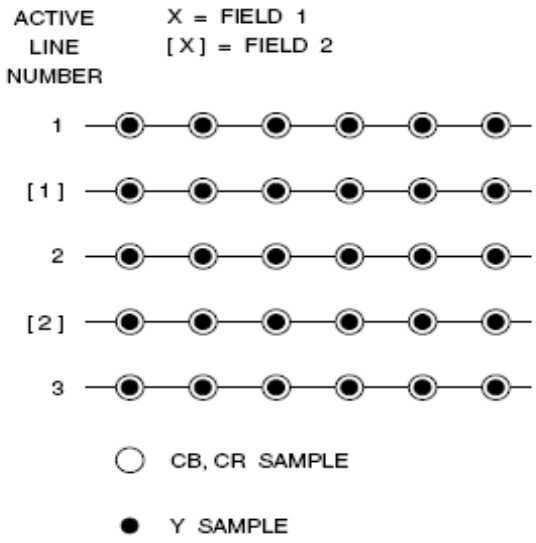


Figure 2.6.11.1 → *4:4:4 Co-Sited Sampling.*

The sampling positions on the active scan lines of an interlaced picture.

2.6.12 - 4:2:2 YCbCr Format

Figure 2.6.12.1 illustrates the positioning of YCbCr samples for the 4:2:2 formats. For every two horizontal Y samples, there is one Cb and Cr sample. Each sample is typically 8 bits (consumer applications) or 10 bits (pro-video applications) per component. Each sample therefore requires 16 bits (or 20 bits for pro-video applications), usually formatted as shown in Figure 2.6.12.1

To display 4:2:2 YCbCr data, it is first converted to 4:4:4 YCbCr data, using interpolation to generate the missing Cb and Cr samples [25, 27, 28].

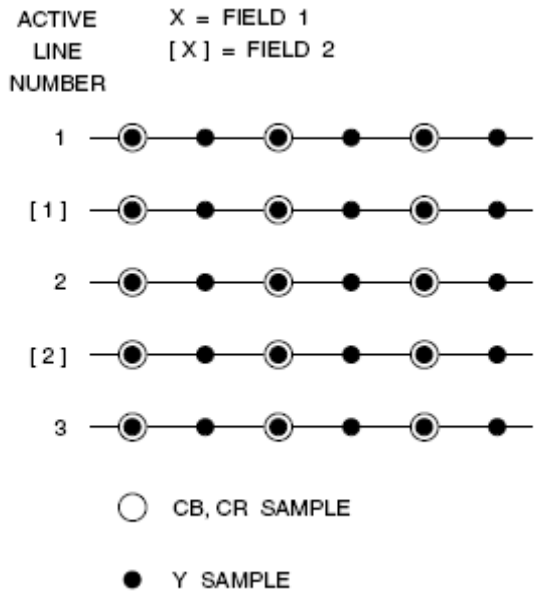


Figure 2.6.12.1 → *4:2:2 Co-Sited Sampling.*

The sampling positions on the active scan lines of an interlaced picture.

2.6.13 - 4:1:1 YCbCr Format

Figure 2.6.13.1 illustrates the positioning of YCbCr samples for the 4:1:1 format (also known as YUV12), used in some consumer video and DV video compression applications [25, 27, 28]. For every four horizontal Y samples, there is one Cb and Cr value. Each component is typically 8 bits. Each sample therefore requires 12 bits, usually formatted as shown in Figure 2.6.13.1.

To display 4:1:1 YCbCr data, it is first converted to 4:4:4 YCbCr data, using interpolation to generate the missing Cb and Cr samples.

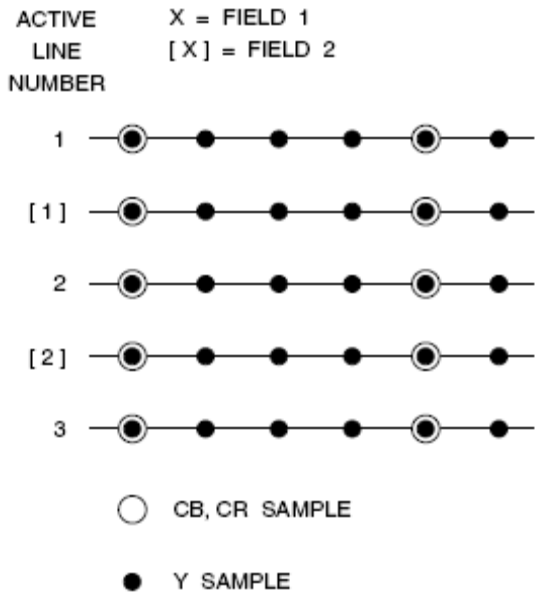


Figure 2.6.13.1 → *4:1:1 Co-Sited Sampling.*

The sampling positions on the active scan lines of an interlaced picture.

2.6.14 - 4:2:0 YCbCr Format

Rather than the horizontal-only 2:1 reduction of Cb and Cr used by 4:2:2, 4:2:0 YCbCr implements a 2:1 reduction of Cb and Cr in both the vertical and horizontal directions [25, 26, 27, 28]. It is commonly used for video compression. As shown in Figure 2.6.14.1 through 2.6.14.5, there are several 4:2:0 sampling formats. Table 2.6.14.1 lists the YCbCr formats for various DV applications. To display 4:2:0 YCbCr data, it is first converted to 4:4:4 YCbCr data, using interpolation to generate the new Cb and Cr samples.

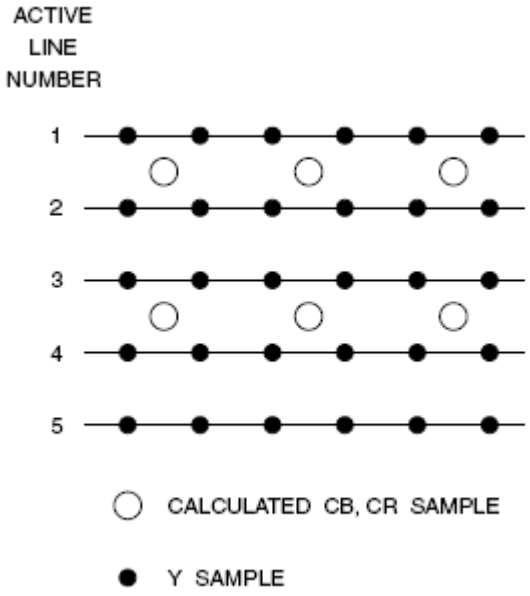


Figure 2.6.14.1 → *4:2:0 Sampling for H.261, H.263, and MPEG 1.*

The sampling positions on the active scan lines of a progressive or non-interlaced picture.

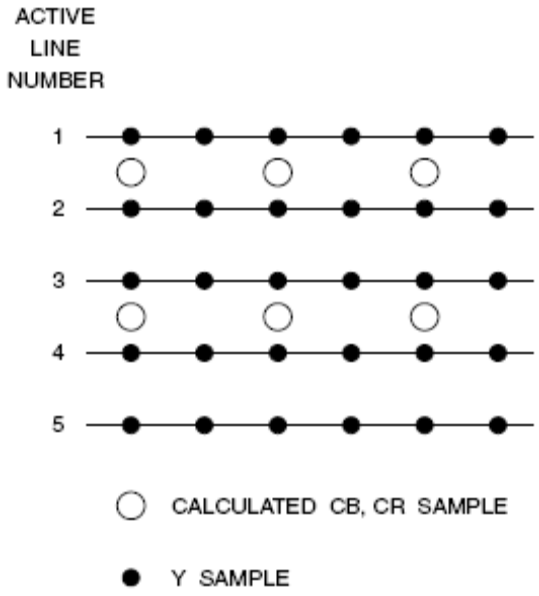


Figure 2.6.14.2 → *4:2:0 Sampling for MPEG 2.*

The sampling positions on the active scan lines of a progressive or non-interlaced picture.

	480-Line DV	576-Line DV	480-Line DVCAM	576-Line DVCAM	DVCPRO	DVCPRO 50	Digital Beta-cam	Digital S	MPEG 1, 2	H.261, H.263
4:2:2 Co-Sited						X	X	X		
4:1:1 Co-Sited	X		X		X					
4:2:0									X	X
4:2:0 Co-Sited		X		X						

Table 2.6.14.1 → *YCbCr Formats for Various DV Applications.*

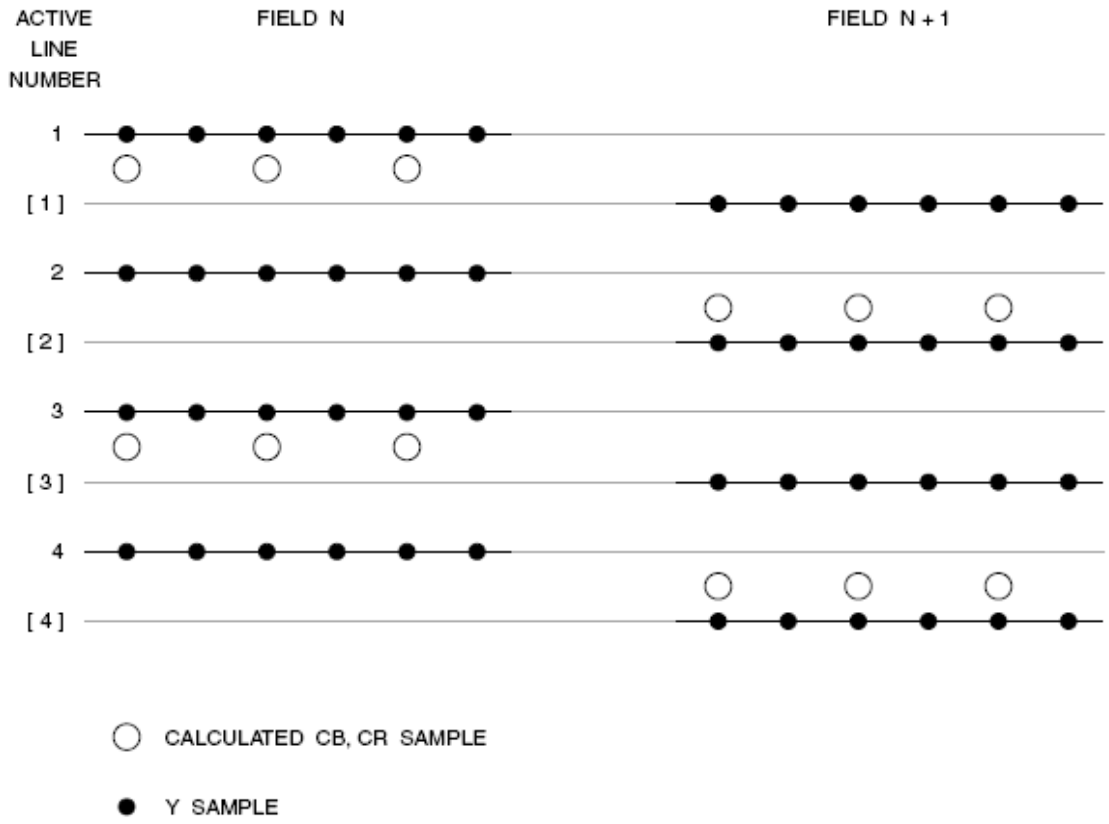


Figure 2.6.14.3 → *4:2:0 Sampling for MPEG 2.*

The sampling positions on the active scan lines of an interlaced picture

(top field first = 1).

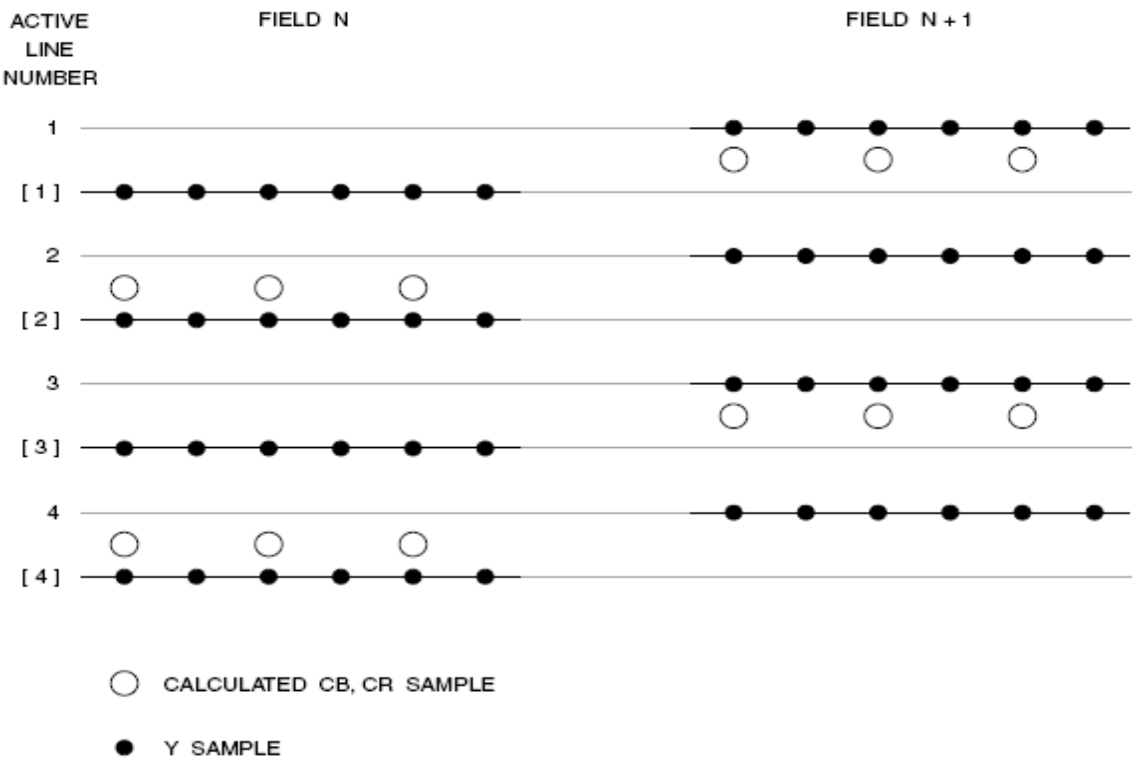


Figure 2.6.14.4 → *4:2:0 Sampling for MPEG 2.*

The sampling positions on the active scan lines of an interlaced picture

(top field first = 0).

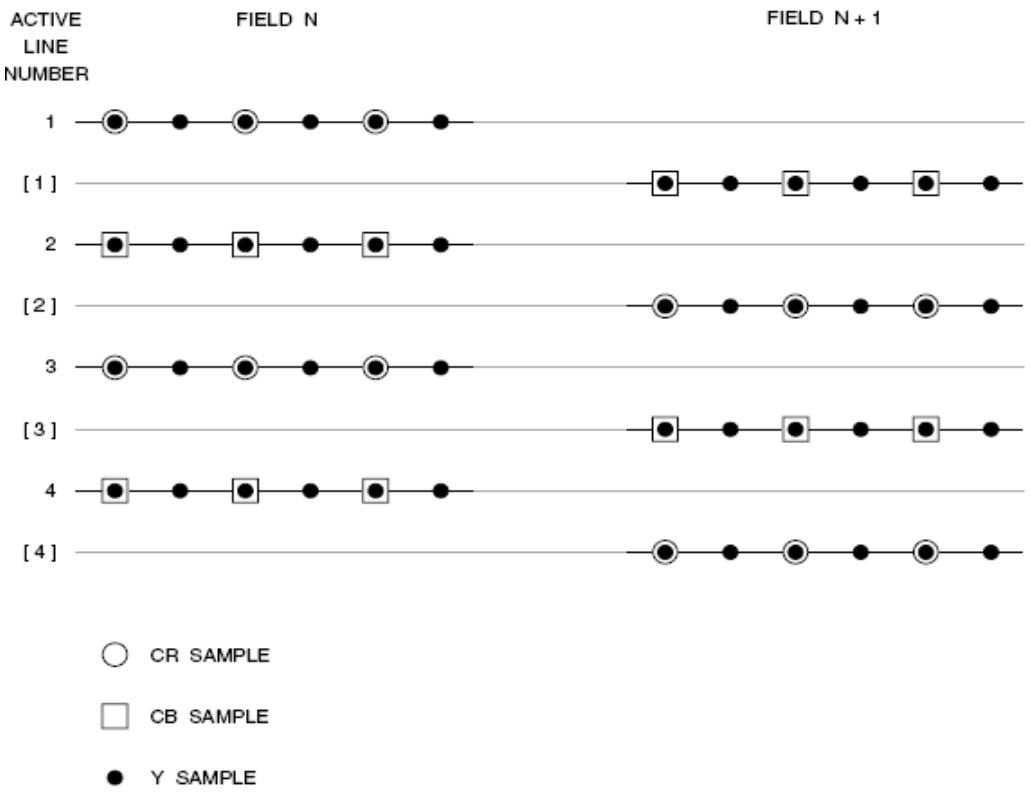


Figure 2.6.14.5 → *4:2:0 Co-Sited Sampling for 576-Line DV and DVCAM.*

The sampling positions on the active scan lines of an interlaced picture.

2.6.15 - Progressive JPEG

Unlike normal JPEG images, which are displayed one line at a time from top to bottom, progressive JPEGs are displayed in alternating lines, and then filled in on a second pass [29]. Depending on which graphics viewer or Web browser is being used, progressive JPEGs may produce a "Venetian blind" effect or simply a blurry or blocky image that gradually sharpens. Pages using progressive JPEGs let people see at least the outline of an image sooner, and often appear to load faster than pages using normal JPEGs.

2.6.16 - Summary of Image Formats

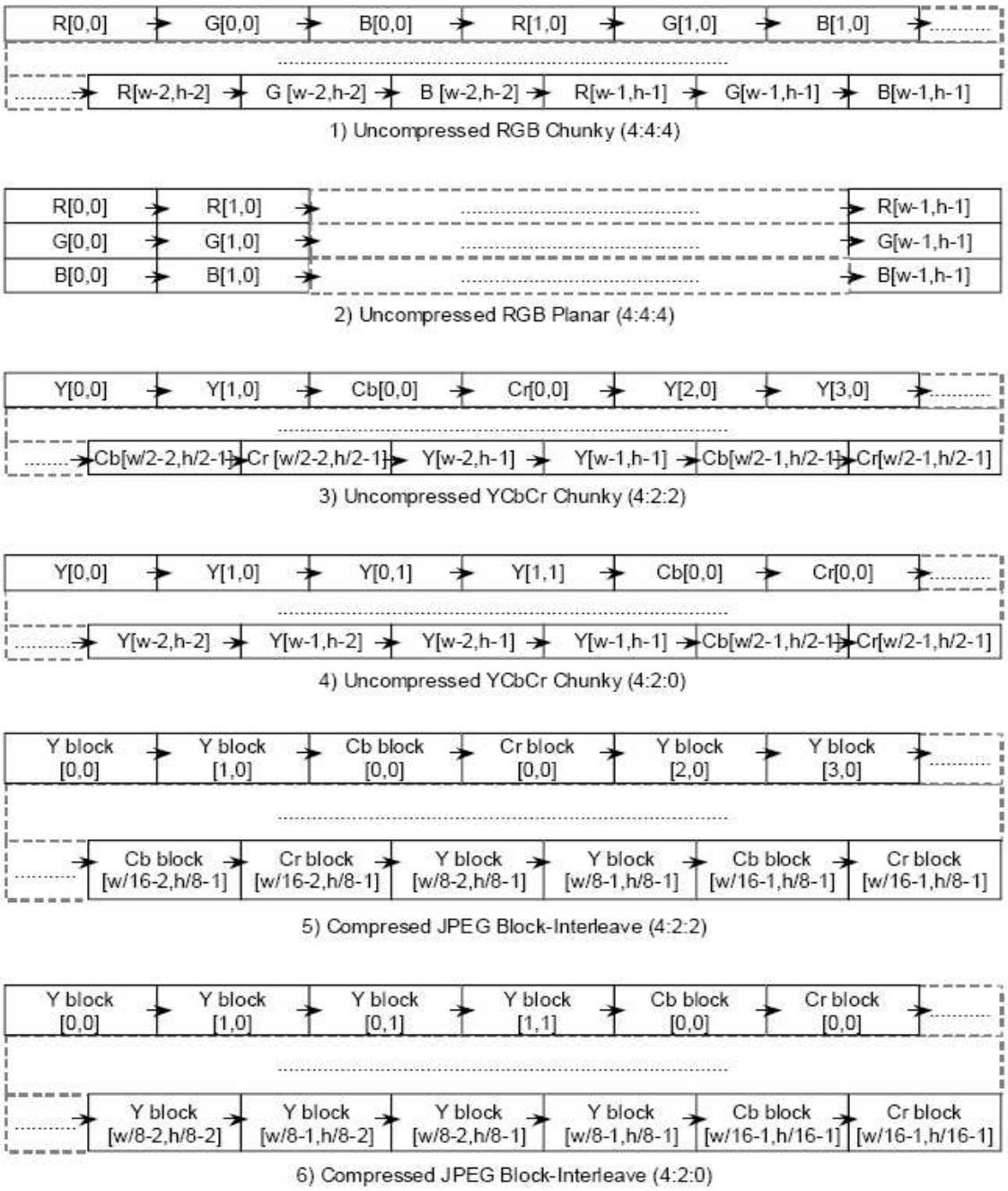


Figure 2.6.16.1 → *The RGB, YCbCr and JPEG File Layout [30, 12]*

This table summarizes the key differences between the image formats.

GIF	JPEG	PNG – 8	PNG – 24
Better for clipart and drawn graphics with few color, or large blocks of color	Better for photographs with lots of color or fine color detail	Better for clipart and drawn graphics with few color, or large blocks of color	Better for photographs with lots of color or fine color detail
Can only have up to 256 color	Can have up to 16 million color	Can only have up to 256 color	Can have up to 16 million color
Images are "lossless" - they contain the same amount of information as the original (but with only 256 color)	Images are "lossy" - they contain less information than the original	Images are "lossless" - they contain the same amount of information as the original (but with only 256 color)	Images are "lossless" - they contain the same amount of information as the original
Can be animated	Cannot be animated	Cannot be animated	Cannot be animated
Can have transparent areas	Cannot have transparent areas	Can have transparent areas	Can have transparent areas

Table 2.6.16.2 → *Key difference between the image formats*

On the whole, GIF's are better for the Web, as Web graphics tend to be small, have a small amount of colors, and sometimes need fancy stuff like animation or transparency. However, JPEGs are really handy when we have a very complex or large image (such as a photo), and/or we want a full color image. PNG-8 can often produce smaller file sizes than GIF, and both types of PNG are lossless.

We will be working more on JPEG's as the work carried out is mainly for photographs and image data.

Chapter 3 - Compression Models & JPEG

JPEG works on either full-color or gray-scale images; it does not handle bilevel (black and white) images, at least not well [1, 5, 12, 24]. It doesn't handle color mapped images either; and we have to pre-expand those into an unmapped full-color representation. JPEG works best on "continuous tone" images. Images with many sudden jumps in color values will not compress well.

There are a lot of parameters to the JPEG compression process. By adjusting the parameters, we can trade off compressed image size against reconstructed image quality over a 'very' wide range. We can get image quality ranging from op-art (at 100x smaller than the original 24-bit image) to quite indistinguishable from the source (at about 3x smaller). Usually the threshold of visible difference from the source image is somewhere around 10x to 20x smaller than the original, i.e., 1 to 2 bits per pixel for color images.

Grayscale images do not compress as much. In fact, for comparable visual quality, a grayscale image needs perhaps 25% less space than a color image; certainly not the 66% less that we might naively expect.

JPEG defines a "baseline" lossy algorithm, plus optional extensions for progressive and hierarchical coding discussed later in the text. There is also a separate lossless compression mode; this typically gives about 2:1 compression, i.e. about 12 bits per color pixel. Most currently available JPEG hardware and software handles only the baseline mode.

3.1 - The outline of the baseline compression algorithm

3.1.1 - Transform the image into a suitable color space.

There is a no option for grayscale, but for color images we generally want to transform RGB into a luminance/chrominance color space (YCbCr, YUV, etc) [29]. The luminance component is grayscale and the other two axes are color information. The reason for doing this is that we can afford to lose a lot more information in the chrominance components than we can in the luminance component: the human eye is not as sensitive to high-frequency chroma info as it is to high-frequency luminance. We don't have to change the color space if we don't want to, since the remainder of the algorithm works on each color

component independently, and doesn't care just what the data is. However, compression will be less since we will have to code all the components at luminance quality. Note that color space transformation is slightly lossy due to round off error, but the amount of error is much smaller than what we typically introduce later on.

3.1.2 - Down sample each by averaging together groups of pixels.

The luminance component is left at full resolution, while the chroma components are often reduced 2:1 horizontally and either 2:1 or 1:1 (no change) vertically. In JPEG-speak these alternatives are usually called 2h2v and 2h1v sampling, but we may also see the terms "411" and "422" sampling. This step immediately reduces the data volume by one-half or one-third. In numerical terms it is highly lossy, but for most images it has almost no impact on perceived quality, because of the eye's poorer resolution for chroma info. Note that down sampling is not applicable to grayscale data; this is one reason color images are more compressible than grayscale.

3.1.3 - Group the pixel values for each component into 8x8 blocks.

Transform each 8x8 block through a discrete cosine transform (DCT). The DCT is a relative of the Fourier transform and likewise gives a frequency map, with 8x8 components. Thus we now have numbers representing the average value in each block and successively higher-frequency changes within the block. The motivation for doing this is that we can now throw away high-frequency information without affecting low-frequency information. (The DCT transform itself is reversible except for round off error.)

- In each block, divide each of the 64 frequency components by a separate "quantization coefficient", and round the results to integers. This is the fundamental information-losing step. The larger the quantization coefficient, the more data is discarded. Note that even the minimum possible quantization coefficient, 1, loses some info, because the exact DCT outputs are typically not integers. Higher frequencies are always quantized less accurately (given larger coefficients) than lower, since they are less visible to the eye. Also, the luminance data is typically quantized more accurately than the chroma data, by using separate 64-element

quantization tables. Tuning the quantization tables for best results is something of a black art, and is an active research area. Most existing encoders use simple linear scaling of the example tables given in the JPEG standard, using a single user-specified "quality" setting to determine the scaling multiplier. This works fairly well for midrange qualities (not too far from the sample tables themselves) but is quite non optimal at very high or low quality settings.

- Encode the reduced coefficients using either Huffman or arithmetic coding. (Strictly speaking, baseline JPEG only allows Huffman coding; arithmetic coding is an optional extension.) [2 -5] Notice that this step is lossless, so it doesn't affect image quality. The arithmetic coding option uses Q-coding; it is identical to the coder used in JBIG. Be aware that Q-coding is patented. Most existing implementations support only the Huffman mode, so as to avoid license fees. The arithmetic mode offers may be 5 or 10% better compression, which isn't enough to justify paying fees.
- Tack on appropriate headers, etc, and output the result. In a normal "interchange" JPEG file, all of the compression parameters are included in the headers so that the de-compressor can reverse the process. These parameters include the quantization tables and the Huffman coding tables. For specialized applications, the spec permits those tables to be omitted from the file; this saves several hundred bytes of overhead, but it means that the de-compressor must know a-priori what tables the compressor used. Omitting the tables is safe only in closed systems. The decompression algorithm reverses this process. The de-compressor multiplies the reduced coefficients by the quantization table entries to produce approximate DCT coefficients. Since these are only approximate, the reconstructed pixel values are also approximate, but if the design has done what it's supposed to do, the errors won't be highly visible. A high-quality de-compressor will typically add some smoothing steps to reduce pixel-to-pixel discontinuities. The JPEG standard does not specify the exact behavior of compressors and de-compressors, so there's some room for creative implementation. In particular, implementations can trade off speed against image quality by choosing more accurate or faster-but-less-accurate approximations to the DCT. Similar tradeoffs exist for the down-sampling / up-sampling and color space conversion steps.

3.2 - Extensions

The progressive mode is intended to support real-time transmission of images. It allows the DCT coefficients to be sent piecemeal in multiple "scans" of the image. With each scan, the decoder can produce a higher-quality rendition of the image. Thus a low-quality preview can be sent very quickly, and then refined as time allows [31]. The total space needed is roughly the same as for a baseline JPEG image of the same final quality. (In fact, it can be somewhat *less* if a custom Huffman table is used for each scan, because the Huffman codes can be optimized over a smaller, more uniform population of data than appears in a baseline image's single scan.) The decoder must do essentially a full JPEG decode cycle for each scan: inverse DCT, up-sample, and color conversion must all be done again, not to mention any color quantization for 8-bit displays. So this scheme is useful only with fast decoders or slow transmission lines. Up until 1995, progressive JPEG was a rare bird, but its use is now spreading as software decoders have become fast enough to make it useful with modem-speed data transmission.

The hierarchical mode represents an image at multiple resolutions. For example, one could provide 512x512, 1024x1024, and 2048x2048 versions of the image. The higher-resolution images are coded as differences from the next smaller image, and thus require many fewer bits than they would if stored independently. (However, the total number of bits will be greater than that needed to store just the highest-resolution frame in baseline form.) The individual frames in a hierarchical sequence can be coded progressively if desired. Hierarchical mode is not widely supported at present.

Part 3 of the JPEG standard, approved at the end of 1995, introduce several new extensions. The one most likely to become popular is variable quantization, which allows the quantization table to be scaled to different levels in different parts of the image. In this way the "more critical" parts of the image can be coded at higher quality than the "less critical" parts. A signaling code can be inserted at any DCT block boundary to set a new scaling factor.

The third major extension added by Part 3 is a "tiling" concept that allows an image to be built up as a composite of JPEG frames, which may have different sizes, resolutions, quality settings, even color spaces. (For example, a color image that occupies a small part of a mostly-grayscale page could be represented as a separate frame, without

having to store the whole page in color.) Again, there's some overlap in functionality with variable quantization and selective refinement. The general case of arbitrary tiles is rather complex and is unlikely to be widely implemented. In the simplest case all the tiles are the same size and use similar quality settings. This case may become popular even if the general tiling mechanism doesn't, because it surmounts the 64K pixel on a side image size limitation that was (not very foresightedly) built into the basic JPEG standard. The individual frames are still restricted to 64K for compatibility reasons, but the total size of a tiled JPEG image can be up to 2^{32} pixels on a side.

3.3 - Lossless JPEG

The separate lossless mode does not use DCT, since round off errors prevent a DCT calculation from being lossless. For the same reason, one would not normally use color space conversion or down sampling, although these are permitted by the standard. The lossless mode simply codes the difference between each pixel and the "predicted" value for the pixel. The predicted value is a simple function of the already-transmitted pixels just above and to the left of the current one (for example, their average; 8 different predictor functions are permitted). The sequence of differences is encoded using the same back end (Huffman or arithmetic) used in the lossy mode.

Lossless JPEG with the Huffman back end is certainly not a state-of-the-art lossless compression method, and wasn't even when it was introduced. The arithmetic-coding back end may make it competitive, but we are probably best off looking at other methods if we need only lossless compression.

The main reason for providing a lossless option is that it makes a good adjunct to the hierarchical mode: the final scan in a hierarchical sequence can be a lossless coding of the remaining differences, to achieve overall losslessness. This isn't quite as useful as it may at first appear, because exact losslessness is not guaranteed unless the encoder and decoder have identical IDCT implementations (i.e., identical round off errors). And we can't use down sampling or color space conversion either if we want true losslessness. But in some applications the combination is useful.

3.3.1 – JPEG Image Formation & Block Diagram

As we have seen, the YCbCr color space seems to be very appropriate for image compression [26 – 30]. Indeed, we can discard most of the information in both chrominance components without impairing the quality of the image. Jpeg and Jpeg2000 both use this color space in their compression scheme and exploit this property in different ways.

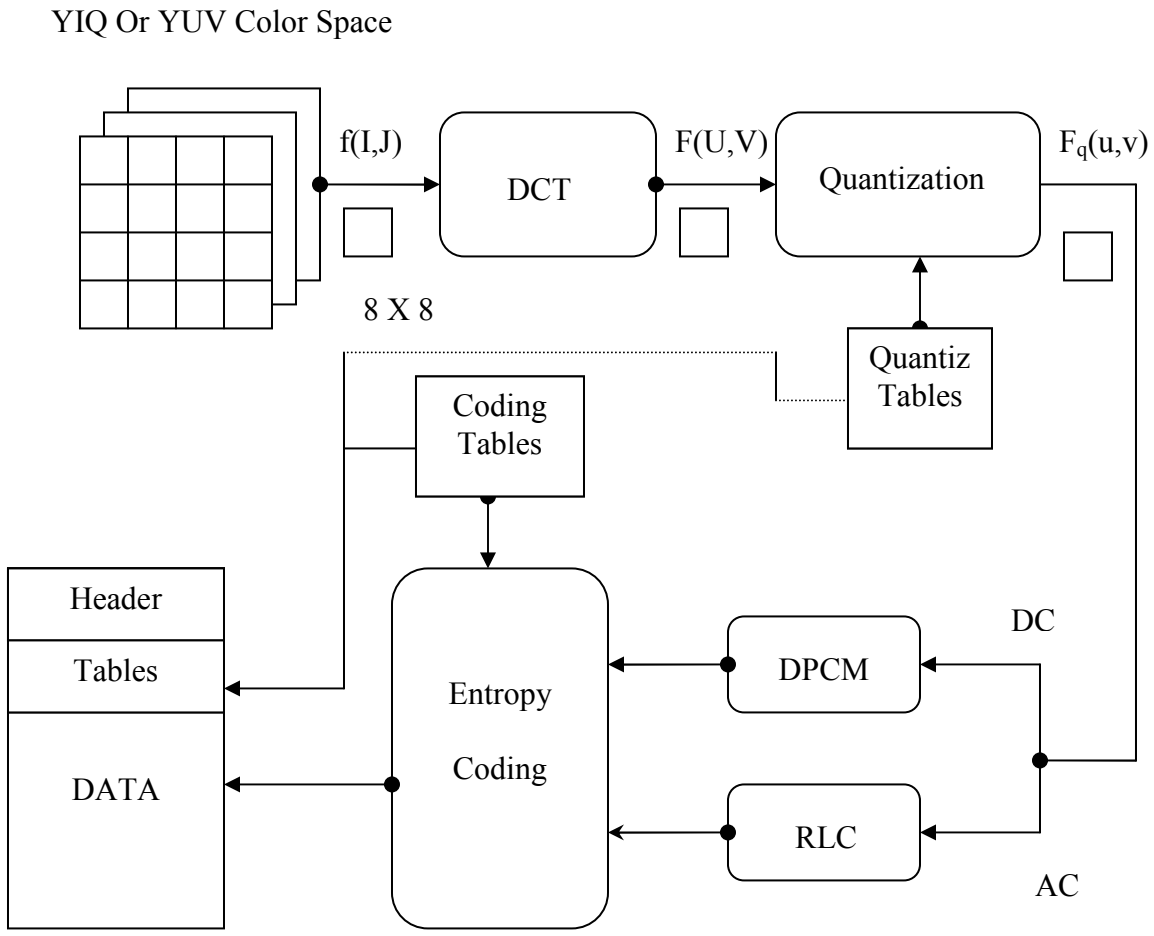


Figure 3.3.1.1 → *JPEG Image Formation Block Diagram.*

3.3.2 - The JPEG Formation Process

The following is a general overview of the JPEG process. Later, detailed method of JPEG's will be explained so that a more comprehensive understanding of the process may be acquired [30].

1. The image is broken into 8x8 blocks of pixels.
2. Working from left to right, top to bottom, the DCT is applied to each block.
3. Each block is compressed through quantization.
4. The array of compressed blocks that constitute the image is stored in a drastically reduced amount of space.
5. When desired, the image is reconstructed through decompression, a process that uses the Inverse Discrete Cosine Transform (IDCT).

3.3.3 - The DCT Equation

$$D(i,j)=\frac{1}{\sqrt{2N}}C(i)C(j)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}p(x,y)\cos\left[\frac{(2x+1)i\Pi}{2N}\right]\cos\left[\frac{(2y+1)i\Pi}{2N}\right] \qquad Eq-3.2.3.1$$

$$C(u)=\begin{Bmatrix}\frac{1}{\sqrt{2}} & if\ u=0\\ 1 & if\ u>0\end{Bmatrix} \qquad Eq-3.2.3.2$$

P(x,y) is the x, y[th] element of the image represented by the matrix p. N is the size of the block that the DCT is done on [29]. The equation calculates one entry (i.j[th]) of the transformed image from the pixel values of the original image matrix. For the standard 8x8 block that JPEG compression uses, N equals 8 and x and y range from 0 to 7. Therefore D (i, j) would be as in Equation 3.3.3.3.

$$D(i,j)=\frac{1}{4}C(i)C(j)\sum_{x=0}^{7}\sum_{y=0}^{7}p(x,y)\cos\left[\frac{(2x+1)i\Pi}{16}\right]\cos\left[\frac{(2y+1)i\Pi}{16}\right] \qquad Eq-3.2.3.3$$

Because the DCT uses cosine functions, the resulting matrix depends on the horizontal, diagonal, and vertical frequencies [28]. Therefore an image block with a lot of

change in frequency has a very random looking resulting matrix, while an image matrix of just one color, has a resulting matrix of a large value for the first element and zeroes for the other elements.

3.3.4 - The DCT Matrix

To get the matrix form of Equation 3.3.4.1, we will use the following equation:

$$T(i,j) = \begin{Bmatrix} \frac{1}{\sqrt{N}} & if\ i = 0 \\ \sqrt{\frac{2}{N}}\ Cos\left[\frac{(2j+1)i\Pi}{2N}\right] & if\ i > 0 \end{Bmatrix} \qquad Eq - 3.3.4.1$$

For an 8x8 block it results in the following matrix:

$$T = \begin{bmatrix} .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 & .3536 \\ .4904 & .4157 & .2778 & .0975 & -.0975 & -.2778 & -.4157 & -.4904 \\ .4619 & .1913 & -.1913 & -.4619 & -.4619 & -.1913 & .1913 & .4619 \\ .4157 & -.0975 & -.4904 & -.2778 & .2778 & .4904 & .0975 & -.4157 \\ .3536 & -.3536 & .-3536 & .3536 & .3536 & -.3536 & -.3536 & .3536 \\ .2778 & -.4904 & .0975 & .4157 & -.4157 & -.0975 & .4904 & -.2778 \\ .1913 & -.4619 & .4619 & -.1913 & -.1913 & .4619 & -.4619 & .1913 \\ .0975 & -.2778 & .4157 & -.4904 & .4904 & -.4157 & .2778 & -.0975 \end{bmatrix} \qquad Eq - 3.3.4.2$$

The first row (i=1) of the matrix has all the entries equal to 1/18 as expected from Equation 3.3.4.1.

The columns of T form an ortho-normal set, so T is an orthogonal matrix. When doing the inverse DCT the orthogonality of T is important, as the inverse of T is T' which is easy to calculate [28].

3.3.5 - Doing the DCT on an 8x8 Block

The pixel values of a black-and-white image range from 0 to 255 in steps of 1, where pure black is represented by 0 and pure white by 255. Thus it can be seen how a photo, illustration, etc. can be accurately represented by these 256 shades of gray [30].

Since an image comprises hundreds or even thousands of 8x8 blocks of pixels, the following description of what happens to one 8x8 block is a microcosm of the JPEG process; what is done to one block of image pixels is done to all of them, in the order earlier specified [32].

Now, let's start with a block of image-pixel values. This particular block was chosen from the very upper- left-hand corner of an image.

$$Original = \begin{bmatrix} 154 & 123 & 123 & 123 & 123 & 123 & 123 & 136 \\ 192 & 180 & 136 & 154 & 154 & 154 & 136 & 110 \\ 254 & 198 & 154 & 154 & 180 & 154 & 123 & 123 \\ 239 & 180 & 136 & 180 & 180 & 166 & 123 & 123 \\ 180 & 154 & 136 & 167 & 166 & 149 & 136 & 136 \\ 128 & 136 & 123 & 136 & 154 & 180 & 198 & 154 \\ 123 & 105 & 110 & 149 & 136 & 136 & 180 & 166 \\ 110 & 136 & 123 & 123 & 123 & 163 & 154 & 136 \end{bmatrix} \quad Eq - 3.3.5.1$$

Because the DCT is designed to work on pixel values ranging from -128 to 127, the original block is "leveled off' by subtracting 128 from each entry. This results in the following matrix.

$$M = \begin{bmatrix} 26 & -5 & -5 & -5 & -5 & -5 & -5 & 8 \\ 64 & 52 & 8 & 26 & 26 & 26 & 8 & -18 \\ 126 & 70 & 26 & 26 & 52 & 26 & -5 & -5 \\ 111 & 52 & 8 & 52 & 52 & 38 & -5 & -5 \\ 52 & 26 & 8 & 39 & 38 & 21 & 8 & 8 \\ 0 & 8 & -5 & 8 & 26 & 52 & 70 & 26 \\ -5 & -23 & -18 & 21 & 8 & 8 & 52 & 38 \\ -18 & 8 & -5 & -5 & -5 & 8 & 26 & 8 \end{bmatrix} \quad Eq - 3.3.5.2$$

We are now ready to perform the Discrete Cosine Transform, which is accomplished by matrix multiplication.

$$D = TMT' \qquad Eq\text{-}\ 3.3.5.3$$

In Equation 3.3.5.3 matrix M is first multiplied on the left by the DCT matrix T from the previous section; this transforms the rows. The columns are then transformed by multiplying on the right by the transpose of the DCT matrix. This yields the following matrix.

$$D = \begin{bmatrix} 162.3 & 40.6 & 20.0 & 72.3 & 30.3 & 12.5 & -19.7 & -11.5 \\ 30.5 & 108.4 & 10.5 & 32.3 & 27.7 & -15.5 & 18.4 & -2.0 \\ -94.1 & -60.1 & 12.3 & -43.4 & -31.3 & 6.1 & -3.3 & 7.1 \\ -38.6 & -83.4 & -5.4 & -22.2 & -13.5 & 15.5 & -1.3 & 3.5 \\ -31.3 & 17.9 & -5.5 & -12.4 & 14.3 & .6.0 & 11.5 & -6.0 \\ -0.9 & -11.8 & 12.8 & 0.2 & 28.1 & 12.6 & 8.4 & 2.9 \\ 4.6 & -2.4 & 12.2 & 6.6 & -18.7 & -12.8 & 7.7 & 12.0 \\ -10.0 & 11.2 & 7.8 & -16.3 & 21.5 & 0.0 & 5.9 & 10.7 \end{bmatrix} \quad Eq - 3.3.5.4$$

This block matrix 3.3.5.4 now consists of 64 DCT coefficients, C_{ij}, where i and j range from 0 to 7. The top-left coefficient, C_{00}, correlates to the low frequencies of the original image block. As we move away from C_{00} in all directions, the DCT coefficients correlate to higher and higher frequencies of the image block, where C_{77} corresponds to the highest frequency. It is important to note that the human eye is most sensitive to low frequencies, and results from the quantization step will reflect this fact.

3.3.6 – Quantization

Our 8x8 block of DCT coefficients is now ready for compression by quantization. A remarkable and highly useful feature of the JPEG process is that in this step, varying levels of image compression and quality are obtainable through selection of specific quantization matrices [31, 32]. This enables the user to decide on quality levels ranging from 1 to 100, where 1 gives the poorest image quality and highest compression, while 100 gives the best quality and lowest compression. As a result, the quality/compression ratio can be tailored to suit different needs.

Subjective experiments involving the human visual system have resulted in the JPEG standard quantization matrix. With a quality level of 50, this matrix renders both high compression and excellent decompressed image quality.

$$Q_{50} = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \qquad Eq - 3.3.6.1$$

If, however, another level of quality and compression is desired, scalar multiples of the JPEG standard quantization matrix may be used. For a quality level greater than 50 (less compression, higher image quality), the standard quantization matrix is multiplied by (100-quality level)/50 [26, 29, 30]. For a quality level less than 50 (more compression, lower image quality), the standard quantization matrix is multiplied by 50/quality level. The scaled quantization matrix is then rounded and clipped to have positive integer values ranging from 1 to 255. For example, the following quantization matrices yield quality levels of 10 and 90.

$$Q_{10} = \begin{bmatrix} 80 & 60 & 50 & 80 & 120 & 200 & 255 & 255 \\ 55 & 60 & 70 & 95 & 130 & 255 & 255 & 255 \\ 70 & 65 & 80 & 120 & 200 & 255 & 255 & 255 \\ 70 & 85 & 110 & 145 & 255 & 255 & 255 & 255 \\ 90 & 110 & 185 & 255 & 255 & 255 & 255 & 255 \\ 120 & 175 & 255 & 255 & 255 & 255 & 255 & 255 \\ 245 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 & 255 & 255 & 255 & 255 \end{bmatrix} \qquad Eq - 3.3.6.2$$

$$Q_{90} = \begin{bmatrix} 3 & 2 & 2 & 3 & 5 & 8 & 10 & 12 \\ 2 & 2 & 3 & 4 & 5 & 12 & 12 & 11 \\ 3 & 3 & 3 & 5 & 8 & 11 & 14 & 11 \\ 3 & 3 & 4 & 6 & 10 & 17 & 16 & 12 \\ 4 & 4 & 7 & 11 & 14 & 22 & 21 & 15 \\ 5 & 7 & 11 & 13 & 16 & 12 & 23 & 18 \\ 10 & 13 & 16 & 17 & 21 & 24 & 24 & 21 \\ 14 & 18 & 19 & 20 & 22 & 20 & 20 & 20 \end{bmatrix} \qquad Eq - 3.3.6.3$$

Quantization is achieved by dividing each element in the transformed image matrix D by the corresponding element in the quantization matrix, and then rounding to the nearest integer value. For the following step, quantization matrix Q_{50} is used.

$$C_{ij} = round\left(\frac{D_{ij}}{Q_{ij}}\right) \qquad Eq - 3.3.6.4$$

$$C = \begin{bmatrix} 10 & 4 & 2 & 5 & 1 & 0 & 0 & 0 \\ 3 & 9 & 1 & 2 & 1 & 0 & 0 & 0 \\ -7 & -5 & 1 & -2 & -1 & 0 & 0 & 0 \\ -3 & -5 & 0 & -1 & 0 & 0 & 0 & 0 \\ -2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad Eq - 3.3.6.5$$

Recall that the coefficients situated near the upper-left comer correspond to the lower frequencies to which the human eye is most sensitive of the image block. In addition, the zeros represent the less important, higher frequencies that have been discarded, giving rise to the lossy part of compression [24, 25]. As mentioned earlier, only the remaining nonzero coefficients will be used to reconstruct the image. It is also interesting to note the effect of different quantization matrices; use of Q_{10} would give C significantly more zeros, while Q_{90} would result in very few zeros.

3.3.7 – Coding

The quantized matrix C is now ready for the final step of compression. Before storage, all coefficients of C are converted by an encoder to a stream of binary data (01101011...). In-depth coverage of the coding process is beyond the scope of this article. However, we can point out one key aspect that the reader is sure to appreciate. After quantization, it is quite common for most of the coefficients to equal zero. JPEG takes advantage of this by encoding quantized coefficients in the zigzag sequence shown in Figure 1 [27, 28]. The advantage lies in the consolidation of relatively large runs of zeros,

which compress very well. The sequence in Figure 3.3.7.1 (4x4) continues for the entire 8x8 block.

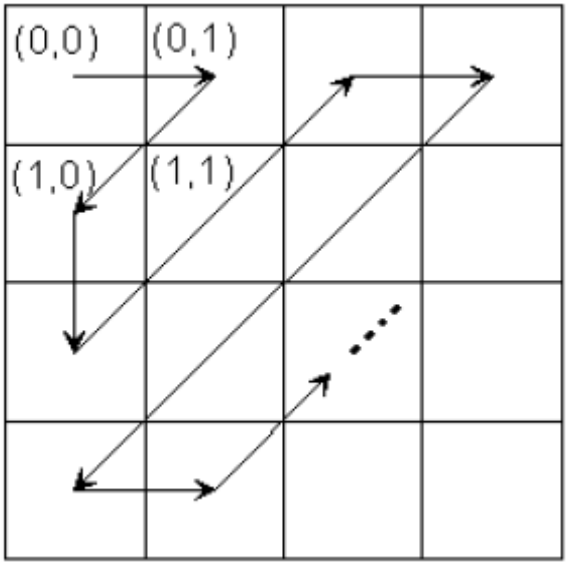


Figure 3.3.7.1 → *Encoding quantized coefficients in the zigzag sequence*

3.3.8 – Decompression

Reconstruction of our image begins by decoding the bit stream representing the quantized matrix C. Each element of C is then multiplied by the corresponding element of the quantization matrix originally used.

$$R_{ij} = Q_{ij} \; x \; C_{ij} \qquad Eq - 3.3.8.1$$

$$R = \begin{bmatrix} 160 & 44 & 20 & 80 & 24 & 0 & 0 & 0 \\ 36 & 108 & 14 & 38 & 26 & 0 & 0 & 0 \\ -98 & -65 & 16 & -48 & -40 & 0 & 0 & 0 \\ -42 & -85 & 0 & -29 & 0 & 0 & 0 & 0 \\ -36 & 22 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad Eq - 3.3.8.2$$

The IDCT is next applied to matrix R, which is rounded to the nearest integer. Finally, 128 is added to each element of that result, giving us the decompressed JPEG version N of our original 8x8 image block M [28 – 30].

$$N = round(T' \; R \; T) + 128$$

3.3.9 - Comparison of Matrices

Let us now see how the JPEG version of our original pixel block compares.

$$Original = \begin{bmatrix} 154 & 123 & 123 & 123 & 123 & 123 & 123 & 136 \\ 192 & 180 & 136 & 154 & 154 & 154 & 136 & 110 \\ 254 & 198 & 154 & 154 & 180 & 154 & 123 & 123 \\ 239 & 180 & 136 & 180 & 180 & 166 & 123 & 123 \\ 180 & 154 & 136 & 167 & 166 & 149 & 136 & 136 \\ 128 & 136 & 123 & 136 & 154 & 180 & 198 & 154 \\ 123 & 105 & 110 & 149 & 136 & 136 & 180 & 166 \\ 110 & 136 & 123 & 123 & 123 & 163 & 154 & 136 \end{bmatrix} \quad Eq - 3.3.9.1$$

$$Decompressed = \begin{bmatrix} 149 & 134 & 119 & 116 & 121 & 126 & 127 & 128 \\ 204 & 168 & 140 & 144 & 155 & 150 & 135 & 125 \\ 253 & 195 & 155 & 166 & 183 & 165 & 131 & 111 \\ 245 & 185 & 148 & 166 & 184 & 160 & 124 & 107 \\ 188 & 149 & 132 & 155 & 172 & 159 & 147 & 136 \\ 132 & 123 & 125 & 143 & 160 & 166 & 168 & 171 \\ 109 & 119 & 126 & 128 & 139 & 158 & 168 & 166 \\ 111 & 127 & 127 & 114 & 118 & 141 & 147 & 135 \end{bmatrix} \quad Eq - 3.3.9.2$$

This is a remarkable result, considering that nearly 70% of the DCT coefficients were discarded prior to image block decompression/reconstruction. Given that similar results will occur with the rest of the blocks that constitute the entire image, it should be no surprise that the JPEG image will be scarcely distinguishable from the original. Remember, there are 256 possible shades of gray in a black-and-white picture, and a difference of, say, 10, is barely noticeable to the human eye.

Chapter 4: Formation of YUV (YCbCr) & JPEG Image

4.1 - Lossy Encoding

The lossy compression scheme is where all the action happens for JPEGs. This scheme makes use of the discrete cosine transform to convert images and compresses the resulting DCT coefficients [31, 32]. The lossy compression scheme can vary the level of compression ratio used, giving control over the final image quality (and file size, roughly). The amount of compression JPEG can supply depends almost entirely on the source image. Best results come from using images with few high frequency details.

The DCT is applied to 8x8 pixel blocks, this size being selected as a trade-off between computational complexity, compression speed and quality. Two methods are used in compression; the coefficients are quantized, and are then Huffman or arithmetically compressed. The quantizing of the coefficients is where the lossy part of the sequence, where high frequency information is discarded.

4.2 - Lossy Compression Steps

The steps in DCT encoding an image can be loosely broken up into 9 steps.

I. Convert non-grayscale images into YCbCr components.
II. Down sample CbCr components.
III. Group pixels into 8x8 blocks for processing.
IV. DCT each pixel block.
V. Un-wrap the coefficients.
VI. Scale each coefficient by a 'quantization' factors.
VII. Eliminate near-zero coefficients.
VIII. Huffman encoding of data.
IX. Add header info and quantization factors.

4.2.1 - I. Convert color images into YCbCr color space

Each component of color images is compressed independently, as with the lossless compression scheme. RGB color space is not the most efficient way to JPEG compress images, as it is particularly susceptible to color changes due to quantization.

Color images are converted from RGB components into YCbCr color space, which consists of the luminance (grayscale) and two chroma (color) components. Note that although some literature (including some previously produced by the Hyper Media Unit) states that images are converted into YUV components - or is just very hazy about what format the image is converted to - it is not YUV exactly. However, the YCbCr color space is simply a lightened, gamma adjusted version of YUV color space [32]. These equations are (to 1 decimal place):

$$Y = 0.3R + 0.6G + 0.1B \qquad Eq-4.3.1.1$$

$$U = B - Y \qquad Cb = \frac{U}{2} + 0.5 \qquad Eq-4.3.1.2$$

$$V = R - Y \qquad Cr = \frac{V}{2} + 0.5 \qquad Eq-4.3.1.3$$

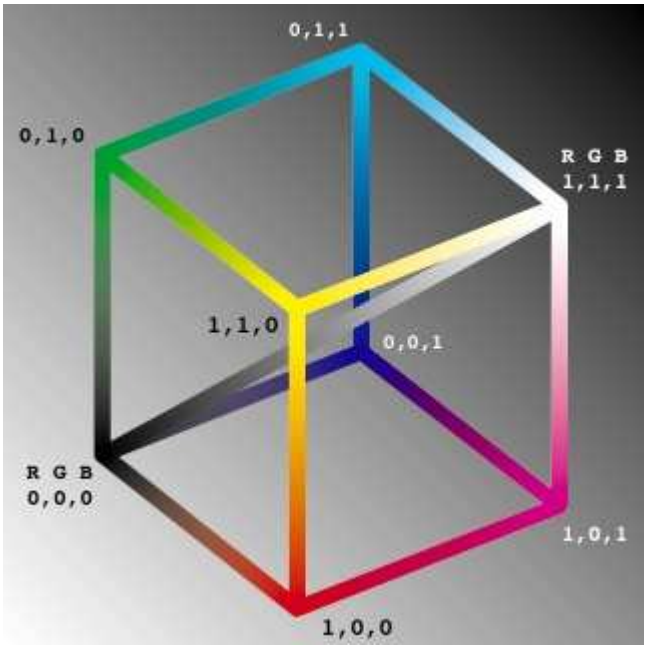


Figure 4.2.1.1 → *The RGB Color Space Cube*

The Y component contains the luminance information of the images and is in effect a grayscale version of the image. On an RGB color cube, this would correspond to

a line running from (0,0,0) to (1,1,1). Cb and Cr are perpendicular color planes, approximately green minus magenta and blue minus yellow. Note that the weightings in the YUV equation above gives most detail to the green component and least to the blue. This reflects the fact that the human eye is more susceptible to variations in the color green and less to blue.

4.2.2 - II. Down sample CbCr components

Once the image has been converted into YCbCr color space (if required), the Cb and Cr components are down sampled by a factor of 2 [22, 23, 27]. We can do this because the human eye gets more detail from the luminance information, than the chrominance [26]. This is how the Cr and Cb components have much less contrast (and thus less information) than the luminance component.

This down sampling gives an immediate 50% reduction in the size of the file - which explains why color images always seem smaller than grayscale images when JPEG compressed. Down sampling ratios are expressed in the usual manner. i.e. 4:1:1 means that the U and V components have been sub sampled 4 times (into 4x4 pixel blocks).

The human eye is more susceptible to change in luminance (grey levels) than change in chrominance (colors). As most of the details the human eye picks up are stored in the Y component, more error can be tolerated in the CbCr color components.

4.2.3 - III & IV. Group Pixels into 8x8 Blocks and DCT encode

This step seems fairly obvious. Each of the three YCbCr color planes are encoded separately but using the same scheme. Eight by eight pixel blocks (64 pixels in total) were chosen as the size for computational simplicity. Other block sizes have been used for other encoding schemes [26, 31]. For example, the Cinepak codec used by Apple's QuickTime software uses 4x4 blocks to give very quick (but often poorer quality) DCT encoding and decoding. Older professional video capture cards made by Radius use 16x16 block because they can supply the extra processing power required. If the size of the image is not a factor of 8, it is padded out to the required size, with extra space being added on the left and bottom of the image. When the image is decoded, these pad pixels are chopped off.

The 64 data points are DCT encoded using the equation:

$$F(u,v) = \frac{1}{4} C(u)\ C(v) \sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) Cos\left(\frac{(2x+1)u\pi}{16}\right)\ Cos\left(\frac{(2y+1)v\pi}{16}\right) \quad \text{Eq – 4.2.3.1}$$

Where

$$C(u),\ C(v) = \frac{1}{\sqrt{N}} \quad For\ u,v = 0$$
$$C(u),\ C(v) = 1 \quad Otherwise \qquad \text{Eq – 4.2.3.3}$$

4.2.4 - V & VI. Unwrap and Scale the Coefficients

The coefficients are unwrapped in the infamous 'zigzag' pattern which just converts the 8x8 array into a single 64 element array with the coefficients of most significance occurring first.

These coefficients are then scaled - except for the DC component which is the scaled average intensity value of the entire image (or YUV plane, in the case of a color image) [24, 25]. The quantization factors used in scaling the coefficients is not actually dictated in the JPEG specification. The encoding algorithm can actually use whatever scale factors it likes, as these are included with the final data. The JPEG standard has a generic set of quantization factors which are often used.

Quantization achieves two goals:

1. It allows more detail to be kept in the (more important) low frequency data.
2. It averages to zero, coefficient values which are close to zero

4.2.5 - VII. Quantize and Eliminate Near Zero Coefficients

Next, coefficients which have been scaled to near zero values are actually given zero values, in the hope that there will be many zero coefficients and therefore compress well [30]. This is the second lossy part of the compression process, where we deliberately and irretrievably loose information.

The major lossy part of the JPEG process is quantization, where each coefficient is divided by its own scale factor. The larger this number is, the more compression will be applied to the image data. This step is where the intended quality of the final image

can be specified. Although it is possible to calculation and encode custom quantization tables, in practice the default ISO JPEG tables are used most of the time.

4.2.6 - VIII & IX. Huffman Encode Data and add Header Info

The resulting quantized integer coefficients are then Huffman encoded to squeeze that extra bit of compression out of the image. Huffman compression is lossless of course. Header information is added tot he data, including the quantization factor, the scale factors and the Huffman tables. Do a bit of formatting and out squirts a JPEG file!

4.3 - Decompressing JPEG images

The JPEG compression scheme can be described as asymmetrical, in that the method used for compression, when reversed, is the method for decompression.

1. Remove header info and quantization factors.
2. Extract data from Huffman encode bit stream.
3. Scale each coefficient by the inverse 'quantization' factors.
4. Prepare the coefficients for IDCT in 8x8 blocks.
5. IDCT each coefficient block.
6. Put the 8x8 pixel blocks into the image buffer.
7. Scale up the CbCr components.
8. Convert the YCbCr components into an RGB image.

The main difference in decoding is the use of the Inverse Discrete Cosine Transform, which is:

$$F(u,v) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} F(x,y)\,C(u)\,C(v)\,Cos\left(\frac{(2x+1)u\pi}{16}\right)\,Cos\left(\frac{(2y+1)v\pi}{16}\right) \quad \text{Eq} - 4.3.1$$

Where

$$C(u),\ C(v) = \frac{1}{\sqrt{N}} \quad For\, u, v = 0$$
$$C(u),\ C(v) = 1 \quad Otherwise \qquad \text{Eq} - 4.3.3$$

4.4 Image Coding

Storage, transmission and retrieval of image data involve massive amounts of data. A single black and white video image contains 768 x 575 pixels. When we use 1 byte for each pixel this means 0.431 MByte for each image [29 – 31]. The question arises whether we could assign the bits more intelligently to our sampled data so that we could use less space. In this chapter we will discuss a number of techniques to do so [27]. We will discuss coding and decoding techniques that preserve the full data (lossless techniques), as well as methods that introduce distortions in the data (lossy techniques) and have a trade-off between the number of bits needed and the accuracy of the coding. When the information is program data, for instance, no distortion can be tolerated. In image data some loss can be present without noticing it in the displayed images [30].

4.4.1 – Entropy Coding

Let us take as an example that we want to encode the characters in English texts. Not all characters in the English occur equally frequently. The e and the space are much more frequent than the z or the x. In straightforward encoding, the same number of bits (fixed length code) would encode each character. By assigning to the e and space short codes and to the z and x longer codes, we can reduce the number of bits needed to encode a text fragment [24, 25, 26]. A well-known coding method using this principle is the Huffman code. The method is based upon dividing the set of symbols to be encoded each time in two equally probable subsets, and coding these two subsets with a 1 and a 0. If one of the subsets contains only one symbol, we are done with that subset [24 -28].

We will illustrate the procedure by a simple example. Assume that we have to transmit the output of a sensor, whose output is quantized into 8 possible values 0, 1, 2, 3, 4, 5, 6, 7. This sensor monitors a process that in its normal state has value 3 or 4. Only in exceptional conditions higher or lower values will occur. The probability that a certain value occurs is given in the second column of Table 4.4.1.1 [27]. The procedure starts by taking together the two values with lowest probability and summing their probabilities to get the probability of the combined values. In this case the values 0 and 7 are combined with probabilities 0.02 and 0.03. The combination has a probability of 0.05. This

procedure is illustrated in Figure 4.4.1.1. Next 1 and 6 (0.04 and 0.05) are combined with a resulting probability of 0.09. Now the least probable are the combinations (0, 7) and (1, 6), these are taken together with a combined probability of 0.14. Then 2 and 5 are taken together and so on. This procedure of repeated combination creates the graph shown in figure 4-2. Next the codes are assigned by starting at the root and assigning a 0 to a left branch and a 1 to a right branch [23 - 27]. The 0's and 1's we come across traveling from the root to the value we want to encode, constitute the Huffman code. The resulting code table is given in the third column of Table 4.4.1.1. The average number of bits required by this coding procedure for a given sensor value is obtained by multiplying the probability of a code (2nd column) with the number of bits (4th column) and summing these values. The average number of bits is 2.53, and we obtain a reduction of 16% compared to the straightforward fixed length encoding with 3 bits.

Value	Probability	Code	No. Of Bits	$-\log P_i$	$-P_i \log P_i$
0	0.02	0 0 0	4	5.64	0.11
1	0.04	0 0 1 0	4	4.64	0.18
2	0.12	0 1 0	3	3.06	0.37
3	0.30	1 0	2	1.74	0.52
4	0.31	1 1	2	1.69	0.52
5	0.13	0 1 1	3	2.94	0.38
6	0.05	0 0 1 1	4	4.32	0.22
7	0.03	0 0 0 1	4	5.06	0.14

Table 4.4.1.1 → *Symbols, their probability and the Huffman codes. The average bit length is 2.53, the entropy is 2.45. Straightforward the coding results in a fixed 3 bit length.*

In the graph, we assigned a 0 to a left branch and a 1 to a right branch. This assignation is completely arbitrary. We could have assigned the values also the opposite way or even arbitrarily. In any case the result would be valid Huffman code.

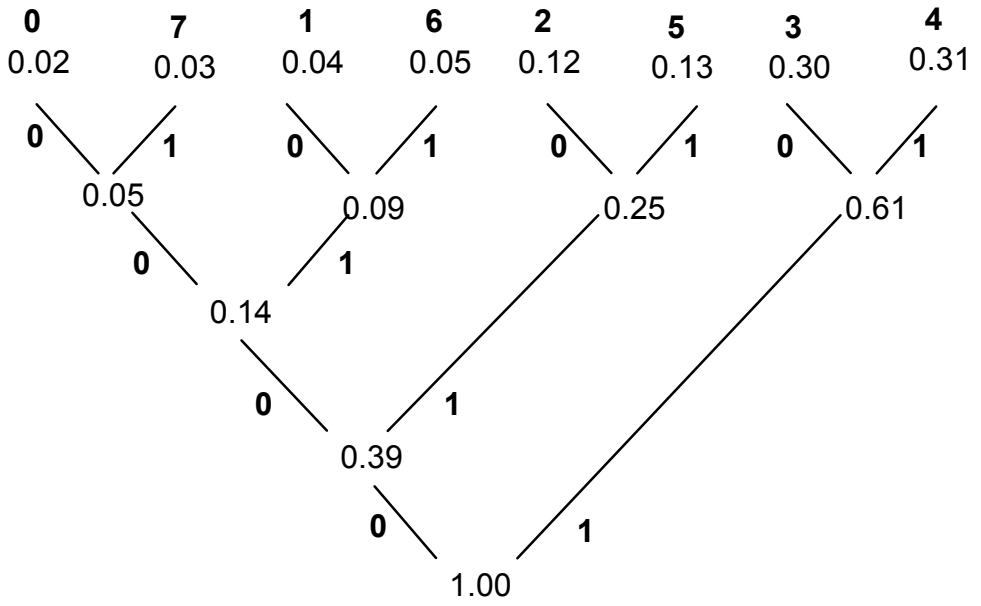


Figure 4.4.1.2 → *Construction of the Huffman code for sensor data with 8 values given in the top row. The probabilities are given in small letters.*

The lowest bit rate which we can attain with these types of coding can be obtained from the basics of information theory. When we send a message to somebody, who almost surely knows beforehand what the message will be, he receives only very little information [23 -28]. When we send somebody a message, which he does not expect at all, he receives a lot of information. This basic aspect of the information content of a message can also be presented more formally. If we know beforehand that a message almost surely will be a, then 'a' is very probable, in other words, its probability P (a) ≈1. On the other hand, the information content H of the message will be very small:

$$P(a) \approx 1 \quad \Rightarrow \quad H(a) \approx 0 \qquad \text{Eq – 4.4.1.1}$$

Conversely, when we receive a message b with is very unlikely, $p(b) \approx 0$, and its information content is almost infinite. Or

$$P(b) \approx 0 \quad \Rightarrow \quad H(b) \to \infty \qquad \text{Eq – 4.4.1.2}$$

When there are only two equally probable messages a and b, with $P(a) = P(b) = 0.5$, the answer to whether we received a or b provides us with 1 bit of information:

$$P(a) = P(b) = 0.5 \quad \Rightarrow \quad H(a) = H(b) = 1 \text{ bit} \qquad \text{Eq – 4.4.1.3}$$

Generalizing this concept, the information H(*a)* obtained when we receive *a* is defined as:

$$H(a) = -2 * \log P(a) \qquad \text{Eq – 4.4.1.4}$$

If P(*a*) is the a priori probability of message *a*. Because of the minus sign, H is a positive quantity, as we would expect. This gives us a new look on the bits we are daily dealing with! The average information $\bar{H}$ we obtain when we receive a message can be calculated as a sum over all possible messages (in a given context) of the information associated with each message, weighted by its a priori probability:

$$\bar{H} = \Sigma -P_i \log P_i \qquad \text{Eq – 4.4.1.5}$$

The average information (expressed in bits) is also called the entropy. As the entropy is the average information we obtain from a message with the given probabilities, it is the lower bound on the average number of bits we need to encode these messages, when the encoding would be optimal. In the Huffman code we have to divide the set of outcomes in equally probable subsets. This is only possible in approximation, and the entropy is a lower bound for the bit rate we can obtain with the Huffman code. In the example the average bit rate is 2.53, the entropy is 2.45, which we should compare to the 3 bits needed when we would have encoded the messages straightforwardly.

When successive sensor values are correlated, a further reduction in bit rate can be obtained. An example is the maximum daily temperature. The maximum of the temperature tomorrow can be expected to be close to today's temperature. The difference will most probably be only a few degrees [25 – 28]. In that case it is more efficient to code the temperature difference that the temperature itself. This is called de-correlation.

Take as an example sensor data that may have 256 different, but equally probable values. When the sensor data reading at time i is k, the following probabilities are given:

$$P(s_{i+1} = k - 1 \mid s_i = k) = 0.25 \qquad \text{Eq – 4.4.1.6}$$

$$P(s_{i+1} = k \mid s_i = k) = 0.5 \qquad \text{Eq – 4.4.1.7}$$

$$P(s_{i+1} = k + 1 \mid s_i = k) = 0.25 \qquad \text{Eq – 4.4.1.8}$$

$$P(s_{i+1} = m \mid s_i = k) = 0 \text{ for } |m - k| > 1. \quad \text{Eq – 4.4.1.9}$$

This means that a certain amount of correlation exists between subsequent data values. The entropy of the first value of our data sequence would be 8 bits (all 256 values being equally probable). However, the entropy of each next symbol is only 1.5 bits. So by encoding the difference, we shrink from 8 bits to 1.5 bits.

Besides the Huffman coding, other entropy-based encoding schemes exist, one of which is the *arithmetic* coding.

An important aspect is that we have to know beforehand the probabilities of the possible messages [24 – 28]. When these probabilities in reality deviate strongly from the guessed or theoretical expected probabilities used to construct the code table, the encoding is still correct, though it will be far from optimal: we can even obtain an increase in bit-rate instead of a reduction.

Another aspect of the Huffman code is that is very sensitive to errors. If there is an error in the transmission of the data, all codes following the error will be incorrect because the synchronization between encoding and decoding is lost. The starting point of the next code will be affected. To prevent the propagation of error, the Huffman codes can be sent in blocks of n codes, so that only one block is affected, and the next blocks are correct again.

4.4.2 – LZW Coding

A coding method for which it is not needed to know a priori probabilities is the Lempel-Ziv-Welch or briefly LZW coding. In the LZW technique, fixed-length blocks of data (say of 8 Kbytes) are encoded and decoded, and the coding table is built simultaneously on both sides. It is not necessary to transmit a coding table separately. The LZW technique exploits the fact that certain combinations of symbols occur more frequently than others do. It discovers repetitions in the data and uses these in the encoding.

Loosely speaking, the algorithm starts by transmitting the individual values, recording all pairs of values it encountered and assigning additional codes to them. As soon as it discovers a pair that it has come across before, the code of this pair is transmitted and the algorithm starts now recording triplets beginning with this specific pair. The procedure goes on in this way, and when longer repeating patterns are present they are successively encoded. On the decoder side the same process takes place. Based

upon the received sequence, the decoder records the pairs it encounters and assigns (the same) additional codes to them. So when a pair occurs a second time, the system knows the code and the underlying coded values.

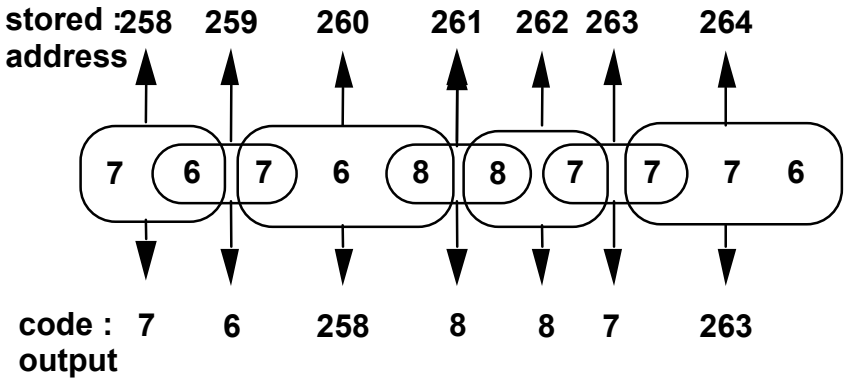
We will illustrate the LZW technique with the case that we want to transmit sensor data with 256 different values. Straightforwardly encoded that would cost 8 bits for each value. Initially the coding tables are filled on the coding and decoding side from entry 0 to entry 255 with the code values 0 to 255. The entries 256 and 257 are reserved for a Clear Table code and for a special End of Information code. The first free position in the table is entry 258. As a result we start with transmitting 9 bits of information. As soon as entry 510 is occupied, we switch over to 10-bits codes. Similarly a switch is made to 11 bit codes after 1022, and so on. When we reach the end of the table (e.g. 4095) a Clear Table code is transmitted, and the whole process starts again with 9 bits-codes.

The procedure is realized by concatenation of the incoming values into strings, as long as the concatenated strings are in the code table and so a code value exists. If the string is not in the table it should be assigned a new code value, and so it is stored in the next free address in the table. The string without the last concatenation is the largest string we can encode and so its table addresses (code value) is send to the output. We start the procedure of the string concatenation again with the last input value.

This procedure is illustrated in Figure 7.2.1 for a data block starting with the characters 7, 6, 7, 6, 8, 8, 7, 7, 7, 6,..... When we take the first character 7 which is in the initial table at entry 7, we do nothing but initializing the string to 7. The next character is 6, and as 7,6 is not present in the table we store 7,6 at position 258 and output 7. The new string is set to the last character: 6. Input of 7 results in 6,7 which is also not in the table, therefore 6,7 is stored at position 259, we output a 6, and initialize the string to 7. When again a 6 is received, 7,6 is in the table so we concatenate the next value which is 8. As 7,6,8 is not in the table we store that string at position 260, output 258 (code of 7,6) and initialize the string to 8. This process is exactly the same at the decoder side, except that we now have to look up 258 to find the values 7,6 and to use only the first value in the concatenation procedure. It can be seen in Figure 7.2.1 illustrating the encoding: the overlap between two successively encoded parts is only 1 value.

A special situation occurs when we encode the next part of our data block 7,7,7,6. The pair 7,7 is not in the table so this string is stored at position 262. Then we encounter with the next 7 again 7,7 which is now present in the table so there is no output, until we have the next 6 and we output 262. On the decoder side this looks like a problem, because a code is received that was not constructed in the decoder. As this situation can only occur when the first character of the previous string is repeated, the last value of the unknown code should be the same as the first value of the previous code and can be constructed.

LWZ Encoder



LWZ Decoder

Received Code	Table Value	Table Address	Decoded Output
7	----	----	7
6	7, 6	258	6
258	6, 7	259	7, 6
8	7, 6, 8	260	8
8	8, 8	261	8
7	8, 7	262	7
263	7, 7	263	7, 7

Figure 4.4.2.1 *→ LWZ encoding and decoding of the sequence (7, 6, 7, 6, 8, 8, 7, 7, 7, 6)*

4.4.3 - Run encoding

A coding method specially suited for binary images, like fax images, is run length encoding. The idea is that pixels do not flip their value or color very often if one walks along a line of the image [28]. It is thus wise to code the flips of the pixels instead of the pixel-values themselves.

Let us see how Run-Length coding works run. Take a (horizontal) line from the image. Say the first pixel is black. Now we count the number of black pixels on the line until we encounter a white pixel. This is called a run, and the number of pixels counted is the length of this run. The same can be done for the next run of white pixels and then again for the black pixels etc., until the whole line has been coded. The resulting code for a line is a list of {pixel value, run length} pairs which describe the line exactly. If we agree that a line always starts with a white pixel we can eliminate the pixel values in the list so that the list reduces to a list of only run-lengths. Note that this coding method becomes inefficient when the image contains frequently alternating pixel values.

Besides the run length coding, some other forms of run coding are used as illustrated in Figure 4.4.3.1. In the run position coding the begin position of each run is coded.

It is again assumed that the first run is white, and when this is not the case an additional first code 0 is added. In the run position/length encoding the beginning length of one run type is encoded. These last two run coding schemes are particularly useful when the coding is combined with processing of run encoded binary images.

The transmission of a fax goes two steps further. First, the histogram of the run lengths of a 'typical' letter is shown in Figure 4.4.3.1. Peaks in the histogram correspond to the width of the paper, the inter-word distance and the inter-character distance. Since the probability of the run lengths is not equal for all runs we can code the run lengths with a Huffman coding. But with Huffman coding we have to transmit a 'codebook' in which a translation is given which code resolves to which run length. The second trick is to assume that fax messages all have a histogram [25]. Assuming this, the codebook of the Huffman coding is equal for all fax messages, so there is no need to transmit the codebook anymore, since it is known both at the transmitter side and at the receiver side.

In practice the CCITT norm, an international standard, has different codebooks for the background and for the foreground colors, since they have different run-length probabilities.

Run-Length coding

Coded parameter is de runlength. It is assume that the first run is
When the first run is black → first run-length

runposition coding

Coded parameter is the begin position of a run . First run black results
first position = 0

position and length coding

Coded parameters are the begin and the length of **one** run type**:** the black (or white) runs.

Figure 4.4.3.1 → *Different types of run coding.*

Chapter 5: Results & Discussions

5.1 Quantization of Images

Various test images have been quantized to different levels and the output has been displayed. It may be noticed that the images which have higher quantization levels have good amount of information associated with them but the compression may be very less. So there needs a trade-off in between the quantization levels and the quality of the image. Here three standard test images have been taken (lena.tif, cameraman.tif and rice.tif). All of these images have been quantized to three different levels Q10 Q50 Q90 and the resultant images have been shown.

The DCT (Discrete Cosine Transform) of the image is also shown. It may however be noticed from the image that after taking the DCT of an image the higher frequency components and the lower frequency components of the image are separated out and we can thus discard the lower frequency components so as to compress the image as in JPEG image formation.

It may be noticed from the results in tables 5.1.1 – 5.1.3 that the images which have been quantized to higher levels have some information while the images which have been quantized to the lower levels have the redundant or very less information. Thus there needs to be a trade off in between the quantization levels and the quality of image.

After the image has been quantized the tables 5.2.1 – 5.2.3 shows the image restored after the applying the inverse quantization or de-quantization. It may be clearly seen that the images which were quantized to lower levels are not restored up to the mark and there is some redundancy in the final image obtained. While the images restored after de-quantization of higher lever quantized images, there is very less redundant information in them.

In the similar way we quantize the Cb and Cr components up-to very low level of degree (High lossless compression) as both of these components don't add to the quality of image. The Cb and Cr component are less visible to the image and thus adds only to the color of image so they can be suppressed to the higher level of degree.

Table 5.1.1 – The DCT & Quantized Image (Lena.tif)

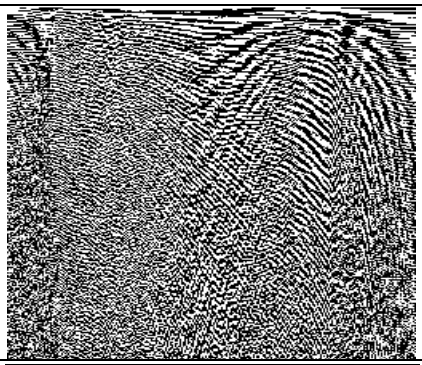
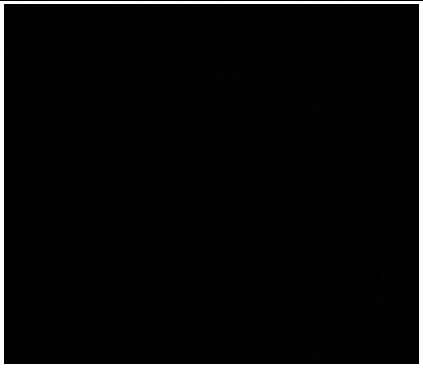
Original Image	Transformed Image	
 Original Image	DCT Image	
	Quantized Image Q_{10}	
	Quantized Image Q_{50}	
	Quantized Image Q_{90}	

Table 5.1.2 – The DCT & Quantized Image (Cameraman.tif)

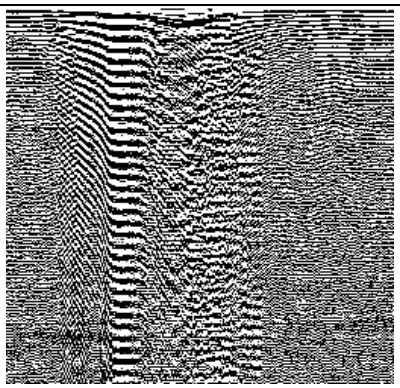
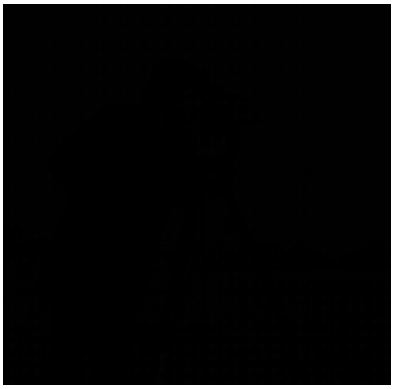
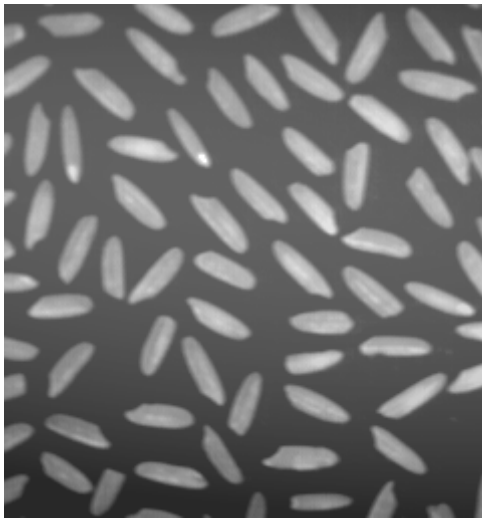
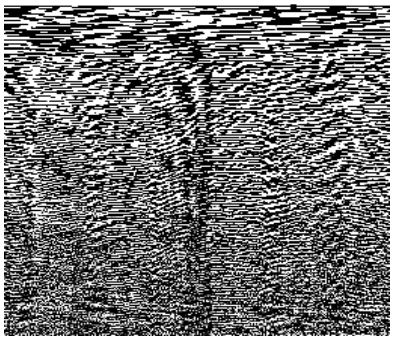
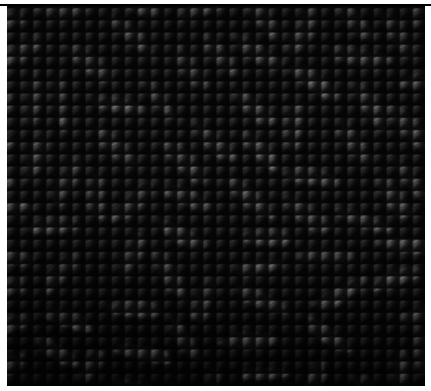
Original Image	Transformed Image	
 Original Image	DCT Image	
	Quantized Image Q_{10}	
	Quantized Image Q_{50}	
	Quantized Image Q_{90}	

Table 5.1.3 – The DCT & Quantized Image (Rice.tif)

Original Image	Transformed image	
	DCT Image	
	Quantized Image Q_{10}	
	Quantized Image Q_{50}	
	Quantized Image Q_{90}	

5.2 - Recovered Image

The image recovered after the de-quantization of the images is shown here under. The PSNR (Peak Signal to Noise Ratio) is also calculated. Comparing the original image with the restored one.

5.2.1 - PSNR Calculation (Quantized Images)

$$PSNR = 10 * \log\left(\frac{Max(Orignial\ \ image)^2}{MSE}\right)$$

$$MSE = \frac{\sum_{x=1}^{M}\sum_{y=1}^{N}\left[I(x,y) - \hat{I}(x,y)\right]^2}{NM} \qquad \text{Eq. – 5.2.1.1}$$

Where:

$$I(x,y) = Original\ \text{Im}age$$
$$\hat{I}(x,y) = Transformed\ \text{Im}age$$
$$M, N = DimensionsOf\ \text{Im}age$$

Table 5.2.1 – De-quantized and IDCT Image (Lena.tif)

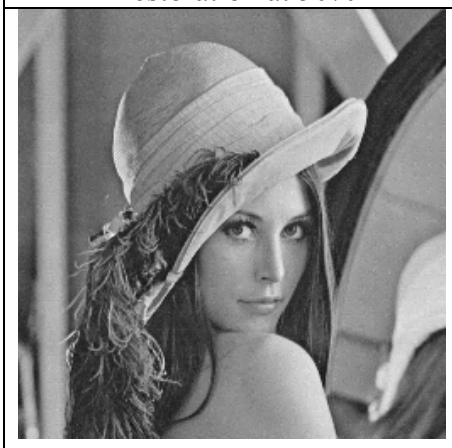
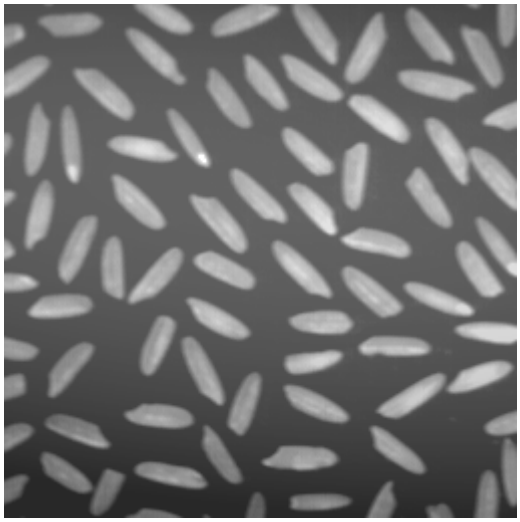
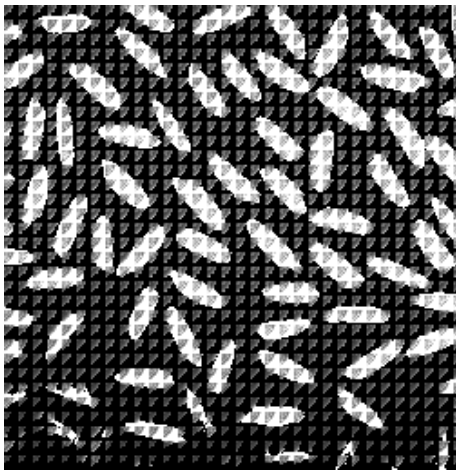
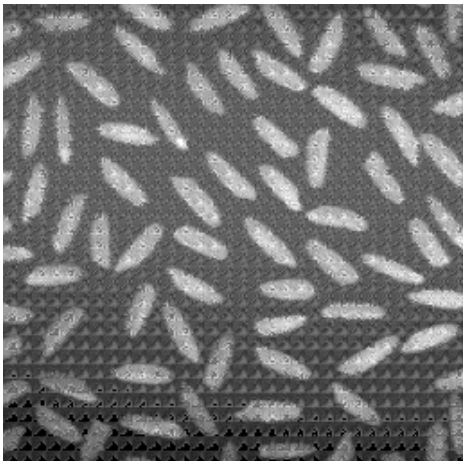
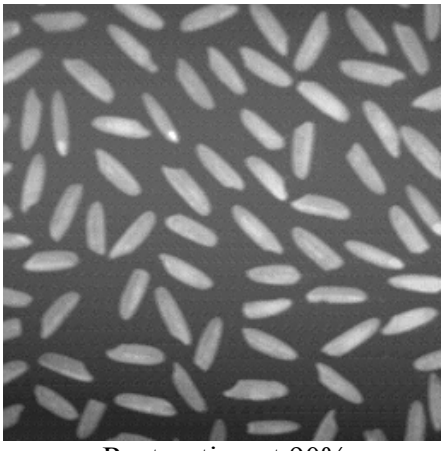
Original Image	Recovered Image	PSNR
	 Restoration at 10%	3.7508
	 Restoration at 50%	5.8102
	 Restoration at 90%	6.9812

Table 5.2.2– De-quantized and IDCT Image (Cameraman.tif)

Original Image	Recovered Image	PSNR
	 Restoration at 10%	3.3853
	 Restoration at 50%	5.7028
	 Restoration at 90%	6.9127

Table 5.2.3– De-quantized and IDCT Image (Rice.tif)

Original Image	Recovered Image	PSNR
	 Restoration at 10%	5.8093
	 Restoration at 50%	6.8308
	 Restoration at 90%	8.1267

5.3 - Decoder Output (YUV Image)

400 Format (Y – Component Only)

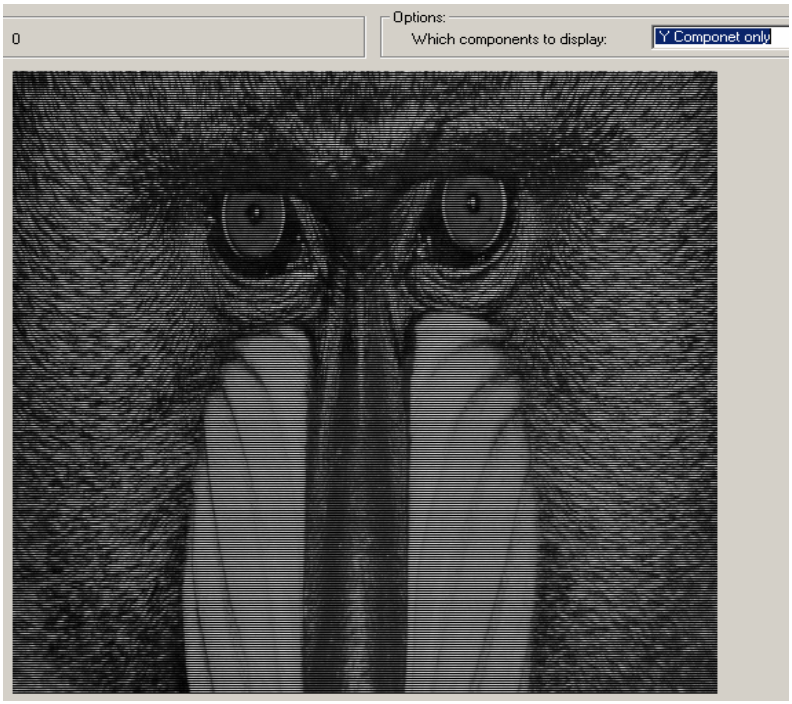


Figure 5.3.1 – 400 Format (Y – Component Only)

420 Format (Y- Component)

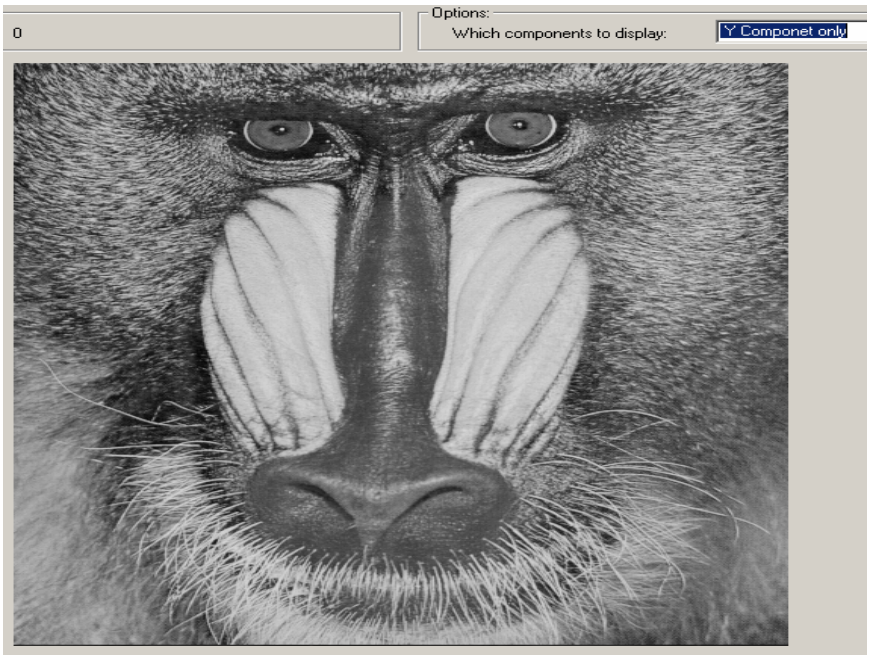


Figure 5.3.3 – 420 Format (Y – Component)

420 Format (U - Component)

Figure 5.3.3 – 420 Format (U – Component)

420 Format (V - Component)

Figure 5.3.3 – 420 Format (V – Component)

422 Format (YCbCr Image)

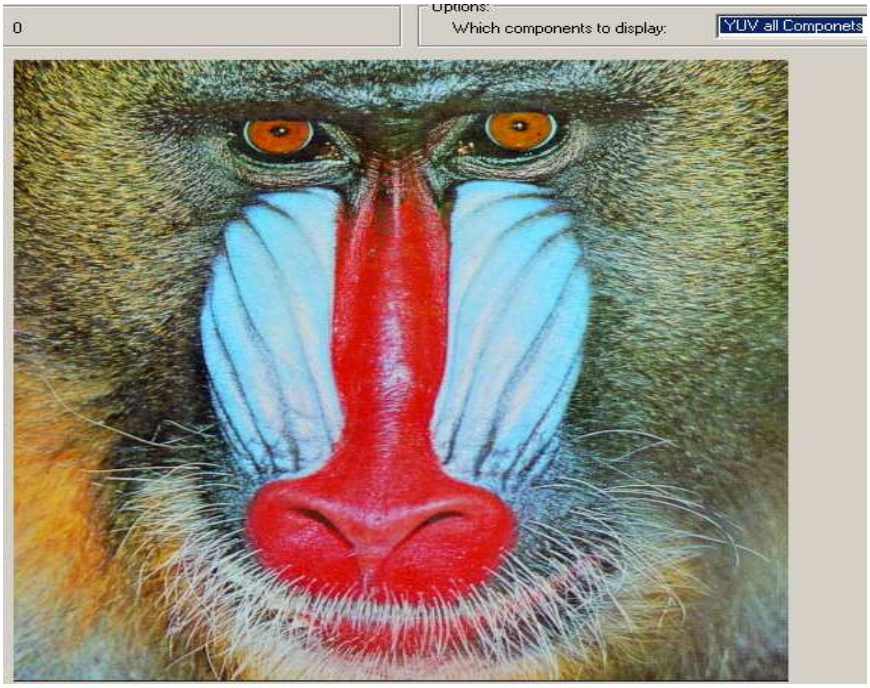


Figure 5.3.5 – 422 Format (YCbCr Image)

---

422 Format (Y Component)

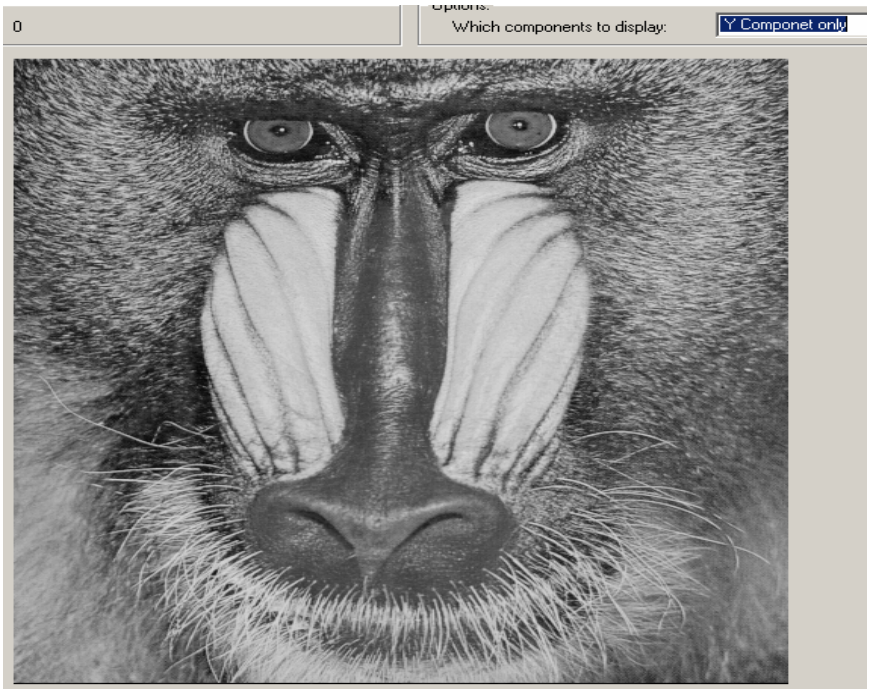


Figure 5.3.6 – 422 Format (Y – Component)

---

422 Format (U Component)

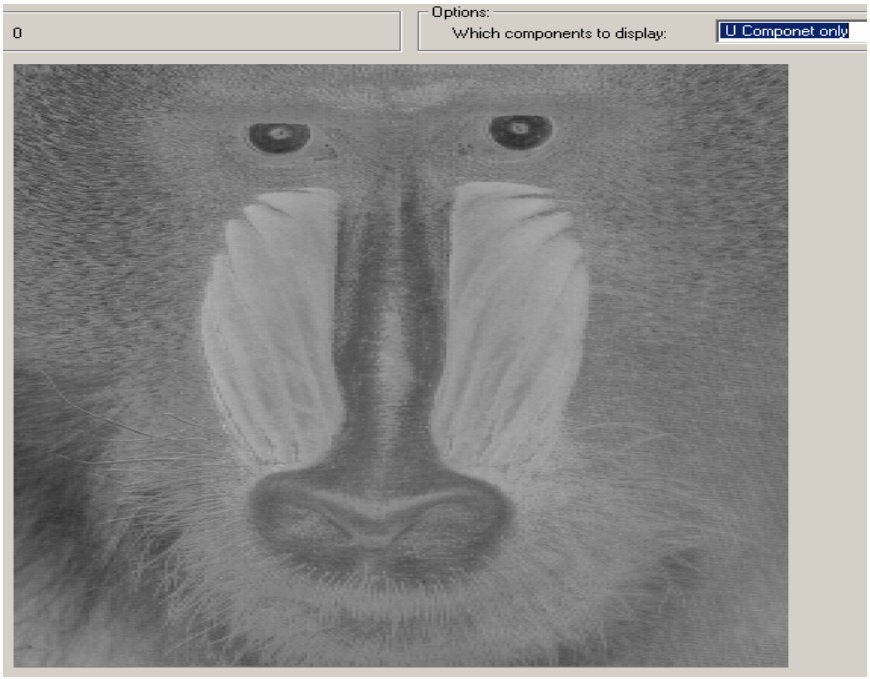


Figure 5.3.7 – 422 Format (U – Component)

422 Format (V Component)

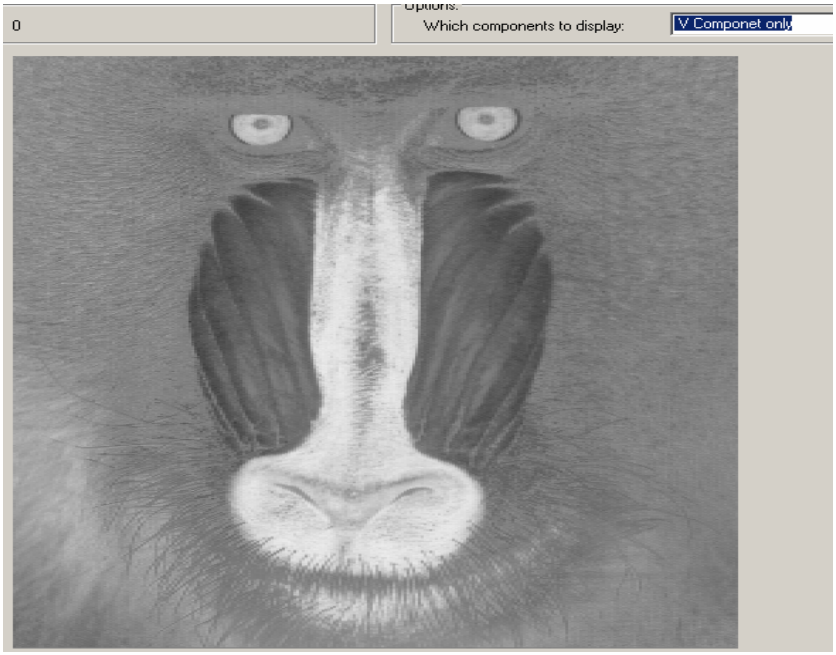


Figure 5.3.8 – 422 Format (V – Component)

444 Format (YCbCr Image)

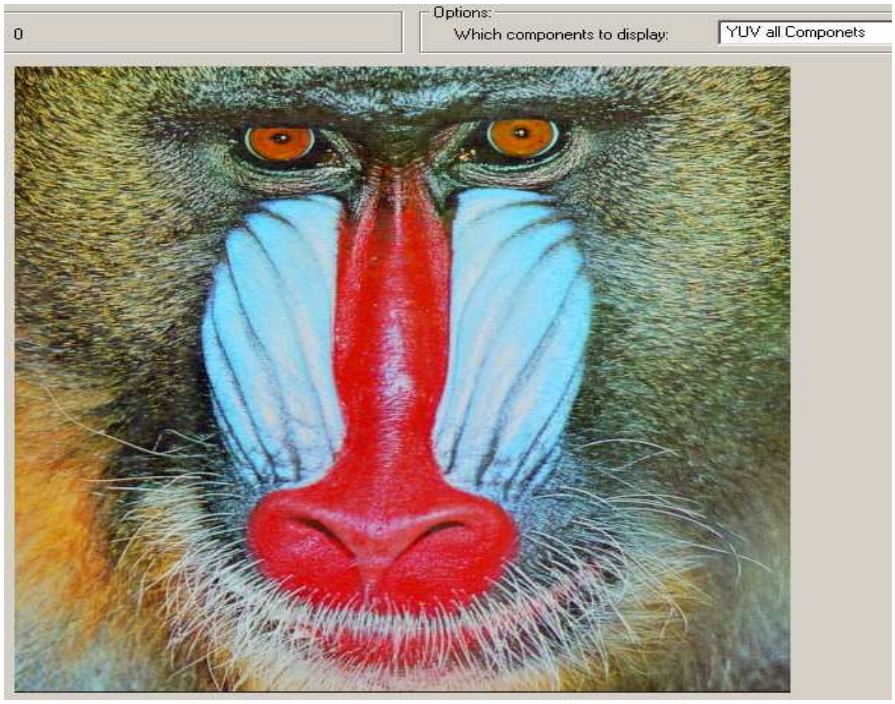


Figure 5.3.9 – 444 Format (YCbCr Image)

444 Format (Y Component)

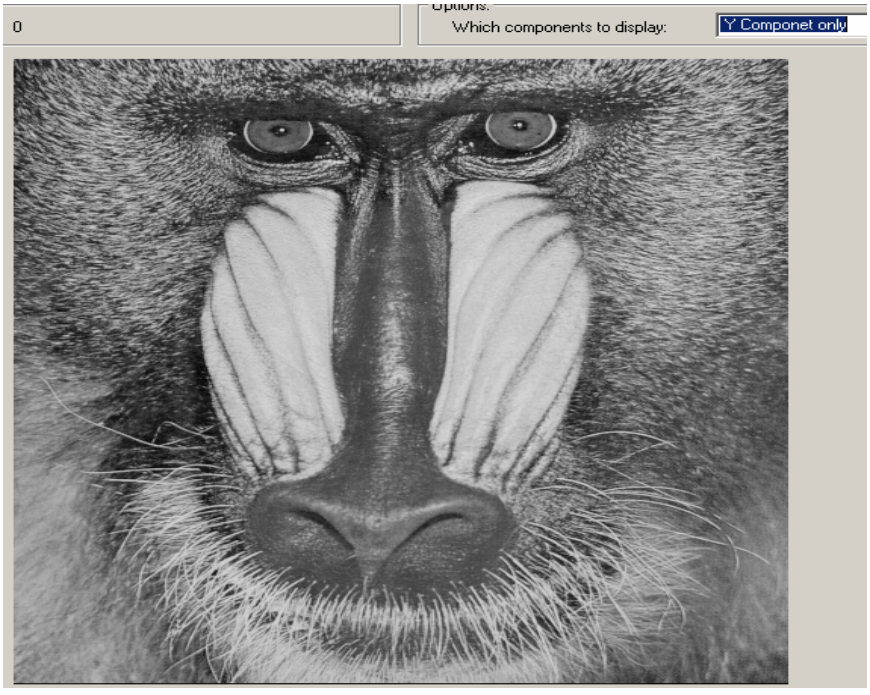


Figure 5.3.10 – 444 Format (Y - Component)

444 Format (U Component)

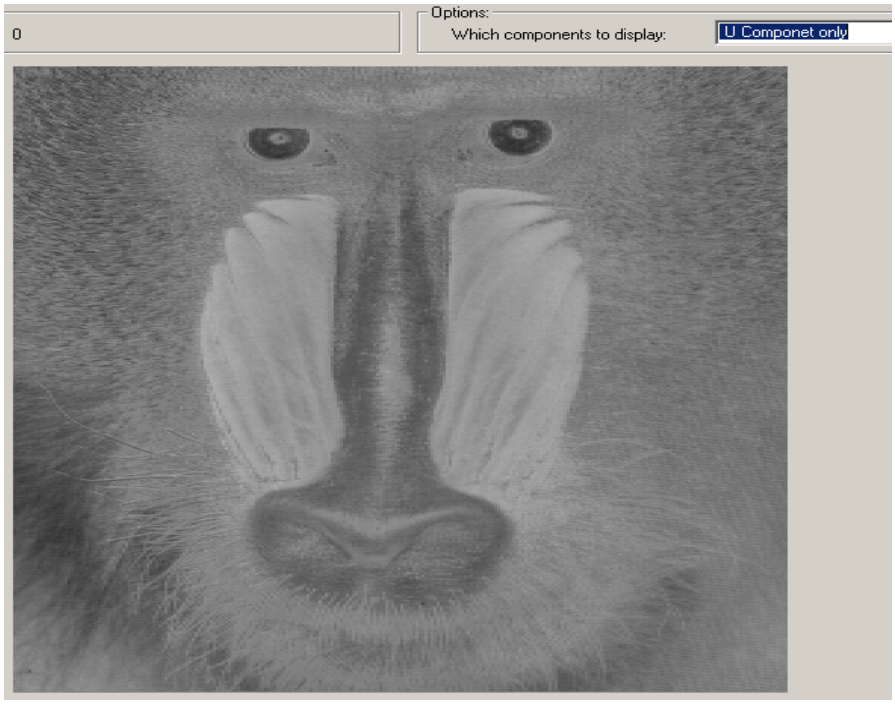


Figure 5.3.11 – 444 Format (U - Component)

444 Format (V Component)

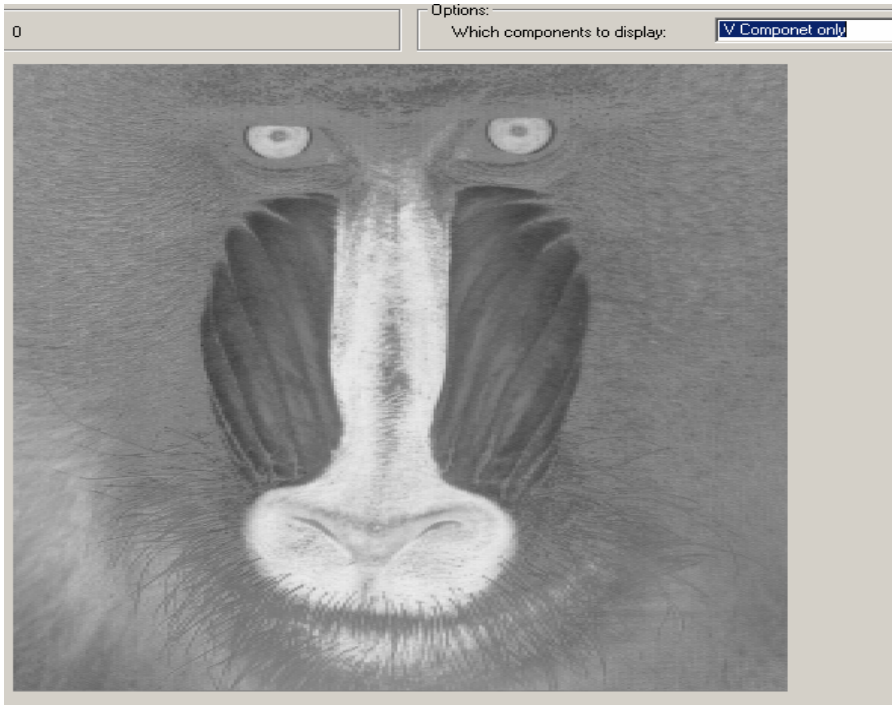


Figure 5.3.12 – 444 Format (V - Component)

RGB16 Format (Original Image)

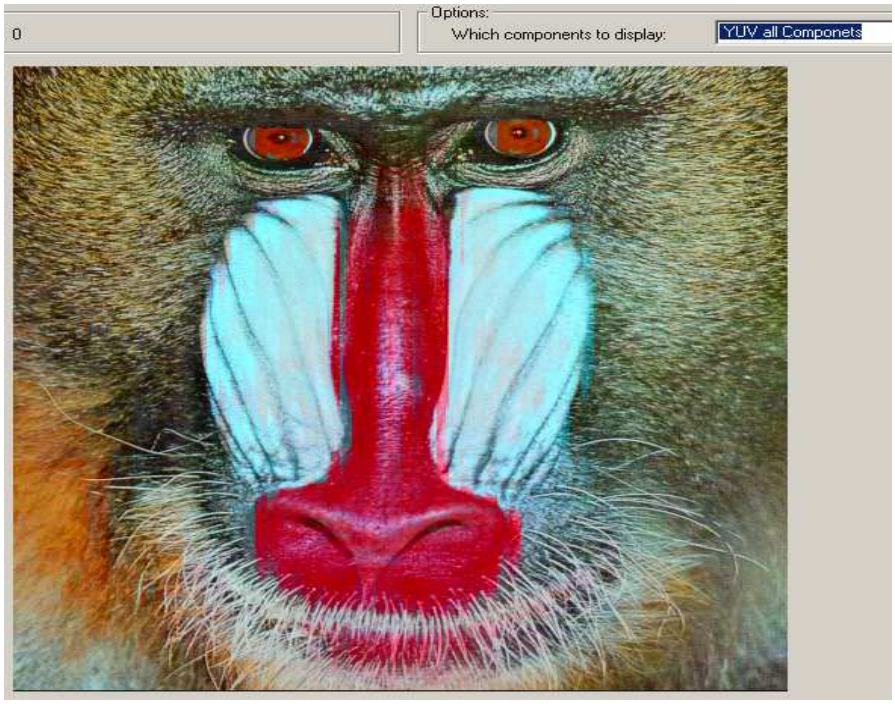


Figure 5.3.13 – RGB16 Format (Original Image)

RGB16 Format (Y Component)

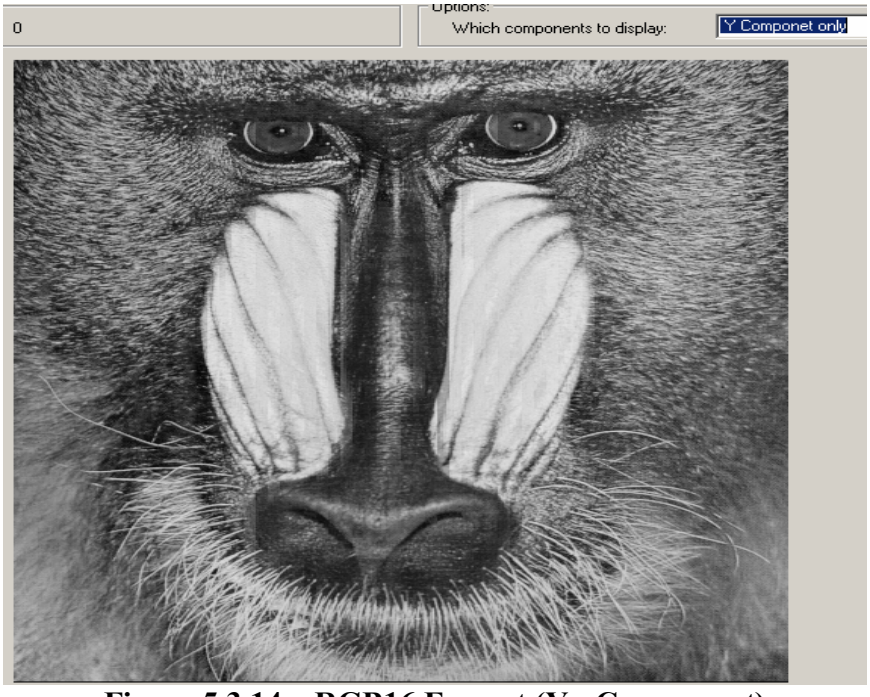


Figure 5.3.14 – RGB16 Format (Y - Component)

RGB16 Format (U Component)

Figure 5.3.15 – RGB16 Format (U - Component)

RGB16 (V Component)

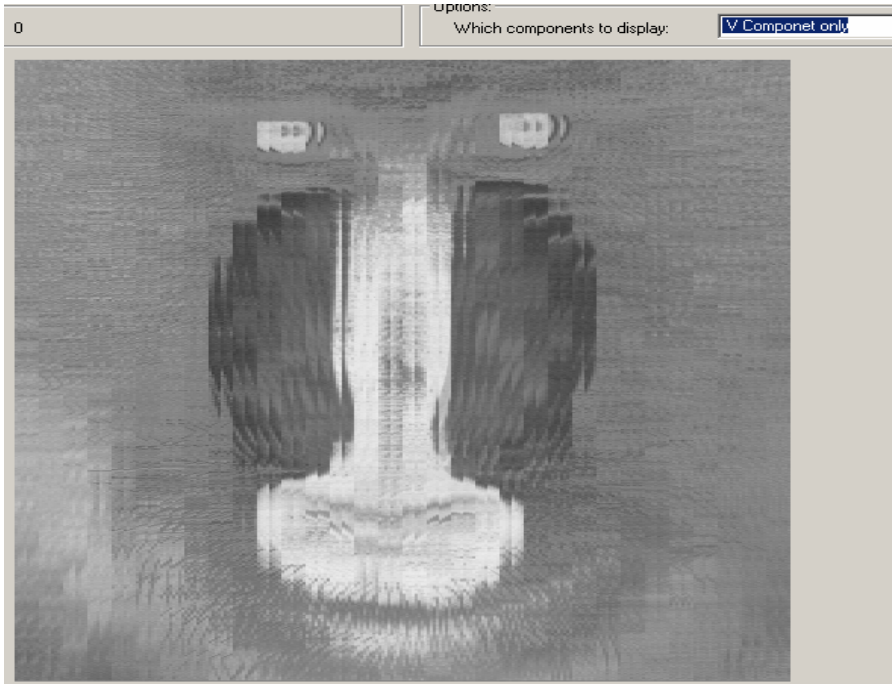


Figure 5.3.16 – RGB16 Format (V – Component)

5.3.1 - PSNR & Its Calculation (YUV Images)

To calculate the Peak Signal to Noise Ratio (PSNR) we use the formula:

$$PSNR = 10 * \log\left(\frac{Max(Orignial\ \ image)^2}{MSE}\right)$$

$$MSE = \frac{\sum_{x=1}^{M}\sum_{y=1}^{N}\left[I(x,y) - \hat{I}(x,y)\right]^2}{NM}$$

Where:

$$I(x,y) = Original\ \text{Im}age$$
$$\hat{I}(x,y) = Transformed\ \text{Im}age$$
$$M, N = DimensionsOf\ \text{Im}age$$

Here we took two standard test images (Lena & Airplane) in RGB and JPEG format. First of all we convert the RGB image into YUV Format with the help of Decoder and then we convert the JPEG file of the same image and convert it into YUV format. Then we calculate the PSNR from both of these files. We also calculate the no. of cycles (CPU Clock cycles) required to decode an image. This is determined by a tool named as "ARM" Processor which tells us about the no. of cycles required for the whole process. The Results thus received are as shown in Table 5.3.1 – 5.3.3.

Table 5.3.1 Filename: airplane (512 x 512, ARM Processor)

Input Image Format	Encode Format	Quality Factor	Cycles Used	PSNR	
400	400	5	32953396	49.897975	
400	400	100	32953718	49.897975	
400	400	1000	32953877	49.897975	
420	400	5	33085261	49.897975	
420	400	100	33085583	49.897975	
420	400	1000	33085742	49.897975	
420	420	5	47436657	Y	49.897975
				Cb	50.036000
				Cr	50.023174
420	420	100	47436979	Y	49.897975
				Cb	50.036000
				Cr	50.023174
420	420	1000	47437138	Y	49.897975
				Cb	50.036000
				Cr	50.023174
422	400	5	33221200	49.897975	
422	400	100	33221522	49.897975	
422	400	1000	33221681	49.897975	
422	420	5	46896407	Y	49.897975
				Cb	41.261183
				Cr	40.715854
422	420	100	46896729	Y	49.897975
				Cb	41.261183
				Cr	40.715854
422	420	1000	46896888	Y	49.897975
				Cb	41.261183
				Cr	40.715854
422	422	5	61024768	Y	49.897975
				Cb	50.031224
				Cr	50.008190
422	422	100	61025090	Y	49.897975
				Cb	50.031224
				Cr	50.008190
422	422	1000	61025249	Y	49.897975
				Cb	50.031224
				Cr	50.008190

Table 5.3.3 Filename: Baboon (512 x 512, ARM Processor)

Image Format	Encode Format	Quality Factor	Cycles Used	PSNR	
400	400	5	34155923	49.863668	
400	400	100	34156241	49.863668	
400	400	1000	34156403	49.863668	
420	400	5	34287788	49.863668	
420	400	100	34288106	49.863668	
420	400	1000	34288268	49.863668	
420	420	5	49195710	Y	49.863668
				Cb	49.943581
				Cr	49.885723
420	420	100	49196028	Y	49.863668
				Cb	49.943581
				Cr	49.885723
420	420	1000	49196190	Y	49.863668
				Cb	49.943581
				Cr	49.885723
422	400	5	34423727	49.863668	
422	400	100	34424045	49.863668	
422	400	1000	34424207	49.863668	
422	420	5	48825577	Y	49.863668
				Cb	42.696632
				Cr	42.307690
422	420	100	48825895	Y	49.863668
				Cb	42.696632
				Cr	42.307690
422	420	1000	48826057	Y	49.863668
				Cb	42.696632
				Cr	42.307690
422	422	5	63775168	Y	49.863668
				Cb	49.926413
				Cr	49.902282
422	422	100	63775486	Y	49.863668
				Cb	49.926413
				Cr	49.902282
422	422	1000	63775648	Y	49.863668
				Cb	49.926413
				Cr	49.902282

Table 5.3.3 File Size after the formation of YUV images

Image Name (Width x Height)	Original (JPEG) Size	Quality Factor	Image Format	Transformed (YUV) Size	% Size Reduced
Lena.jpg (512 x 512)	655 KB	5	400	232 KB	64.58
Lena.jpg (512 x 512)	655 KB	10	400	232 KB	64.58
Lena.jpg (512 x 512)	655 KB	50	400	232 KB	64.58
Lena.jpg (512 x 512)	655 KB	90	400	232 KB	64.58
Lena.jpg (512 x 512)	655 KB	5	420	356 KB	45.65
Lena.jpg (512 x 512)	655 KB	10	420	359 KB	45.19
Lena.jpg (512 x 512)	655 KB	50	420	364 KB	44.43
Lena.jpg (512 x 512)	655 KB	90	420	368 KB	43.82
Lena.jpg (512 x 512)	655 KB	5	422	415 KB	36.64
Lena.jpg (512 x 512)	655 KB	10	422	418 KB	36.18
Lena.jpg (512 x 512)	655 KB	50	422	422 KB	35.57
Lena.jpg (512 x 512)	655 KB	90	422	425 KB	35.11
Lena.jpg (512 x 512)	655 KB	5	444	555 KB	15.27
Lena.jpg (512 x 512)	655 KB	10	444	559 KB	14.66
Lena.jpg (512 x 512)	655 KB	50	444	562 KB	14.20
Lena.jpg (512 x 512)	655 KB	90	444	582 KB	11.15
Baboon.jpg (512 x 512)	734 KB	5	400	285 KB	61.17
Baboon.jpg (512 x 512)	734 KB	10	400	285 KB	61.17
Baboon.jpg (512 x 512)	734 KB	50	400	285 KB	61.17
Baboon.jpg (512 x 512)	734 KB	90	400	285 KB	61.17
Baboon.jpg (512 x 512)	734 KB	5	420	396 KB	46.05
Baboon.jpg (512 x 512)	734 KB	10	420	402 KB	45.23
Baboon.jpg (512 x 512)	734 KB	50	420	410 KB	44.14
Baboon.jpg (512 x 512)	734 KB	90	420	414 KB	43.60
Baboon.jpg (512 x 512)	734 KB	5	422	501 KB	31.74
Baboon.jpg (512 x 512)	734 KB	10	422	506 KB	31.06
Baboon.jpg (512 x 512)	734 KB	50	422	510 KB	30.52
Baboon.jpg (512 x 512)	734 KB	90	422	515 KB	29.84
Baboon.jpg (512 x 512)	734 KB	5	444	594 KB	19.07
Baboon.jpg (512 x 512)	734 KB	10	444	600 KB	18.26
Baboon.jpg (512 x 512)	734 KB	50	444	609 KB	17.03
Baboon.jpg (512 x 512)	734 KB	90	444	617 KB	15.94

The above table shows the original size of the JPEG image and the size of the YUV image formed after the decoding process. It can thus be clearly seen that the YUV format reduces the size of the image. With the different quality factor the size of the image varies.

5.4 - Discussion(s)

From the above tables and results it is verified that the YUV image so formed is having a very less size than the original JPEG image. But there needs to be trade off between the image quality and the size of the image. It has been noticed that the 400 format have very small image size which is due to the reason that there is no color component (Chrominance) component in it. This is the reason that there is no change in the image size due to the change in quality factor for 400 Images.

It can also be seen from tables 5.3.1 – 5.3.3 that the number of cycles required to decode the 400 image is less than the number of cycles to decode the 420 and 422 images. This is due to the reason that more processing is required while we are decoding the chroma components.

The images so received should have a good trade off with the image perception and the quality factor. The quality factor needs to be maintained in such a fashion so that the quality of image is good in respect to the Human vision perception.

So, YUV formats are definitely going to revolutionize the way images are transmitted over the low bandwidth channels.

Conclusion(s)

From this piece of work we can conclude that transformation of JPEG image into its equivalent YCbCr image, not only requires less amount of data to be stored but it can also provide us very high compression rates. The Chroma components are very less sensitive to the eyes and hence applying more lossless coding on the chroma components we can compress images up to very low size. This compressed image when checked with the human Perception of image vision will come nearly equal to the standard JPEG image.

The applications of the YUV images are mainly to the web applications, as now days more and more people are using the internet with multimedia and images with smaller size will lead to less bandwidth requirement as well as less storage.

In the end we can conclude that the YCbCr image format is going to be the revolutionary image format in the coming time. Not only the images are being decoded in this format the video files specially MPEG – 4 Part -2 is being converted in YUV Formats in order to compete with the bandwidth requirements of the present industries.

References

[1] Digital Compression and Coding of Continuous-Tone Still Images, Part 1, Requirements and Guidelines. ISO/IEC JTC1 Committee Draft 10918-1, Feb. 1991.

[2] Digital Compression and Coding of Continuous-Tone Still images, Part 2, Compliance Testing. ISO/IEC JTCI Committee Draft 10918-2. l~ be published Summer 1991.

[3] http://exchange.manifold.net/manifold/manuals/manifold/images/rgb_images_and_channels.htm.

[4] Office Document Architecture (ODA) and Interchange Format, Part 7: Raster Graphics Content Architectures. ISO/IEC JTC1 International Standard 8613-7 1994.

[5] Pennebaker, W.B., JPEG Tech. Specification, Revision 8. Informal working paper JPEG-8-R8, 1990.

[6] Pennebaker, W.B., Mitchell, J.L., et. At. Arithmetic coding articles. *IBM J. Res. Dev. 32, 6 Special Issues pp 717-774, 1988.

[7] K. R. Rao, P. Yip, Discrete cosine transform: algorithms, advantages, applications, Academic Press Professional, Inc., San Diego, CA, 1990

[8] Wallace, G.K. Overview of the JPEG (ISO/CCITT) still image compression standard. Image Processing Algorithms and Techniques. In Proceedings of the SPIE, VOL. 1244, pp. 220-233, 1990.

[9] Wallace, G., Vivian, R., and Poulsen, H. Subjective testing results for still picture compression algorithms for international standardization. In Proceedings of the IEEE Global Telecommunications Conference, IEEE Communications Society, pp 1022-1027, 1988.

[10] D. Salmon, “Data Compression, the Complete Reference,” Springer-Verlag, NY: New York, pp. 101-162, 1998.

[11] P. G. Howard and J. S. Vitter, “Design and Analysis of Fast Text Compression Based on Quasi-Arithmetic Coding,” Proceedings of the IEEE Data Compression Conference (DCC '93), Snowbird, UT, 1993.

[12] www.eee.manchester.ac.uk/intranet/ug/coursematerial/2nd%20Year/EE2034-Programming%20II/Lecture01.pdf

[13] Benson, K. Blair, Television Engineering Handbook. McGraw-Hill, Inc., 1986.

[14] Devereux, V. G., 1987, Limiting of YUV digital video signals, BBC Research Department Report BBC RD pp 22, 1987.

[15] EIA Standard EIA-189-A, Encoded Color bar Signal, 1976.

[16] Faroudja, Yves Charles, NTSC and Beyond. IEEE Transactions on Consumer Electronics, VOL. 34, No. 1, 1988.

[17] ITU-R BT.601–5, 1995, Studio Encoding Parameters of Digital Television for Standard 4:3 and Widescreen 16:9 Aspect Ratios 1996.

[18] ITU-R BT.709–4, Parameter Values for the HDTV Standards for Production and International Programmer Exchange, 2000.

[19] Photo CD Information Bulletin, Fully Utilizing Photo CD Images–PhotoYCC Color Encoding and Compression Schemes, Eastman Kodak Company, 1994.

[20] Wang, Q. and Ward, R.K., "A New Orientation-Adaptive Interpolation Method for JPEG Image Formation," IEEE Trans. On Image Processing, accepted**, 2005.**

[21] Pourazad, M.T., Nasiopoulos, P., Ward, R.K., "An H.264-Based Video Encoding Scheme for 3D TV," 14th European Signal Processing Conference Florence, Italy, accepted, 2002.

[22] Video Demystified A Handbook for the Digital Engineer Third Edition by Keith Jack by Keith Jack, LLH Technology Publishing and Eagle Rock VA 2001.

[23] http://www.cnet.com/Resources/Info/Glossary/Terms/progressivejpeg.html

[24] Brian A. Wandell, Foundation of Vision, Stanford University, pp 199-206, 1995.

[25] Allen B. Poirson & Brian A. Wandell, Pattern-color separable pathways predict sensitivity to simpled colored patterns, Vision Research, 1995.

[26] Marcus J. Nadenau & Julien Reichel, Opponent Color, Human Vision and Wavelets for Image Compression.
http://dewww.epfl.ch/~reichel/Publication/CIC99.pdf

[27] David S. Taubman & Michael W. Marcellin, JPEG 2000 Image Compression Fundamentals Standards and Practices, Kluwer Academic Publishers, 2002.

[28] William B. Pennebaker & Joann L. Mitchell, Jpeg Image Data Compression Standard, Van Nostrand Reinhold, 1993.

[29] JPEG Still Image Compression Standard. William B. Pennebaker, Joan L. Mitchaell, Van Nostrand Reinhold, 1993.

[30] The JPEG still picture compression standard. Communications of the ACM.G. K. Wallace, 1991.

[31] Aldus Developers Desk, TIFF Tag Image File Format Revision 6.0. Aldus Corporation, Seattle WA, 1992.

[32] Wallace G.K., The JPEG still picture compression standard, Communications of the ACM, , VOL 34, no.4, pp 31- 44, 1991.