An Introduction to Video Compression in C/C++

Fore June

Chapter 1

Imaging Basics

3.1 Sampling and Quantization

$Sampling$ is the process of examining the value of a continuous function at regular intervals. We might measure the voltage of an analog waveform every millisecond, or measure the brightness of a photograph every millimeter, horizontally and vertically. Sampling rate is the rate at which we make the measurements and can be defined as

$$\textit{Sampling rate} = \frac{1}{\textit{Sampling interval}}\ Hz$$

If sampling is performed in the time domain, Hz is $cycles/sec$.

In the case of image processing, we can treat an $image$ as a two-dimensional light-intensity function $f(x, y)$ of spatial coordinates (x, y). Since light is a form of energy, $f(x, y)$ must be nonnegative. In order that we can store an $image$ in a computer, which processes data in discrete form, the image function $f(x, y)$ must be digitized both spatially and in amplitude. Digitization of the spatial coordinates (x, y) is referred to as $image\ sampling$ or $spatial\ sampling$, and digitization of the amplitude f is referred to as $quantization$. Moreover, for moving video images, we have to digitize the time component and this is called $temporal\ sampling$. Figure 3-1 shows the concept of spatial sampling.

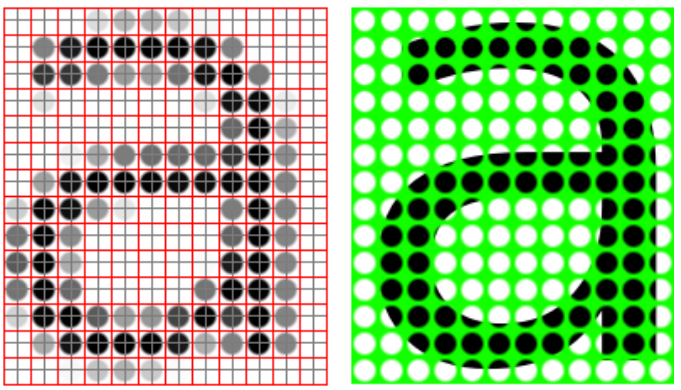

Figure 3-1 Spatial Sampling

Digital video is a representation of a real-world scene, sampled spatially and temporarily and with the light intensity value quantized at each spatial point. A scene is sampled at an instance of time

to produce a $frame$, which consists of the complete visual scene at that instance, or a $field$, which consists of odd- or even-numbered lines of spatial samples. Figure 3-2 shows spatial and temporal sampling of videos.

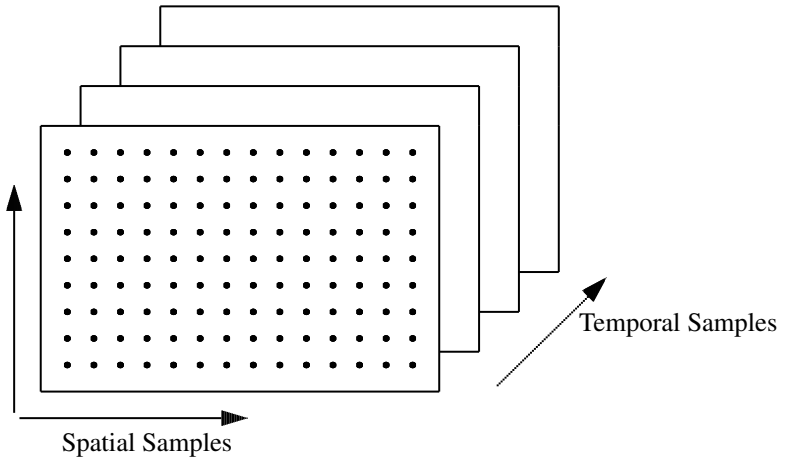

Figure 3-2 Temporal Sampling and Spatial Sampling

3.1.1 Spatial Sampling

Very often a two-dimensional (2D) sampled image is obtained by projecting a video scene onto a 2D sensor, such as an array of Charge Coupled Devices (CCD array) . For colour images, each colour component is filtered and projected onto an independent 2D CCD array. The CCD array outputs analogue signals representing the intensity levels of the colour component. Sampling the signal at an instance in time produces a sampled image or frame that has specified values at a set of spatial sampling points in the form of an $N \times M$ array as shown in the following equation.

$$f(x,y) \approx \begin{pmatrix} f(0,0) & f(0,1) & ... & f(0,M-1) \\ f(1,0) & f(1,1) & ... & f(1,M-1) \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & ... & \cdot \\ f(N-1,0) & f(N-1,1) & ... & f(N-1,M-1) \end{pmatrix} \tag{3.1}$$

The right image of Figure 3-1 shows that a rectangular grid is overlaid on a 2D image to obtain sampled values $f(x,y)$ at the intersection points of the grid. We may approximately reconstruct the sampled image by representing each sample as a square picture element (pixel) as shown on the left image of Figure 3-1. The visual quality of the reconstructed image is affected by the choice of the sampling points. The more sampling points we choose, the higher resolution the resulted sampled image will be. Of course, choosing more sampling points requires more computing power and storage.

3.1.2 Temporal Sampling

$Temporal\ sampling$ of video images refers to the process of taking a rectangular 'snapshot' of the image signal at regular time intervals. The rate at which we take the the snapshots is the $sampling\ rate$ and is defined as the $frame\ rate$ or $field\ rate$. When we play back a sequence of frames obtained in this way at the same rate, an illusion of motion may be created. A higher frame rate produces apparently smoother motion but requires more computing resources to process the larger number of samples. Early silent films used anything between 16 and 24 frames per second (fps). Current television standards use sampling rate of 25 or 30 frames per second.

There are two commonly used temporal sampling techniques, $progressive$ sampling and $interlaced$ sampling. Progressive sampling is a frame-based sampling technique where a video signal is sampled as a series of complete frames. Film is a progressive sampling source for video. Interlaced sampling is a field-based sampling technique where the video is sampled periodically at two sample fields; half of the data in a frame (one field) are scanned at one time. To reconstruct the frame, a pair of sample fields are superimposed on each other (interlaced). In general, a field consists of either the odd-numbered or even-numbered scan lines within a frame as shown in Figure 3-3.

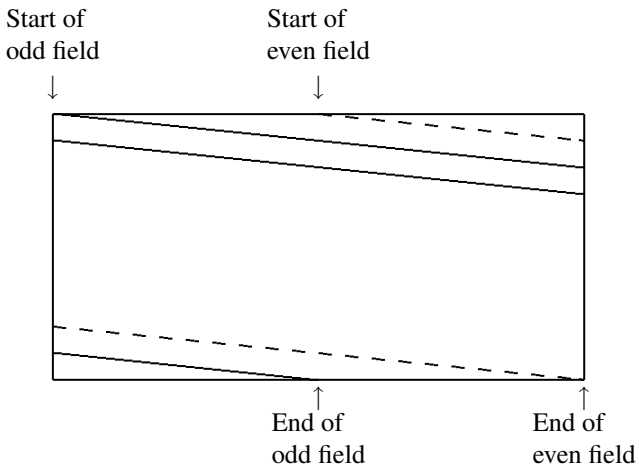

Figure 3-3 Interlaced Scanning

An interlaced video sequence contains a sequence of fields, each of which consists of half the data of a complete frame. The interlaced sampling technique can give the appearance of smoother motion as compared to the progressive sampling method when the data are sampled at the same rate. This is due to the "motion blur" effect of human eyes; the persistence of vision can cause images shown rapidly in sequence to appear as one. When we rapidly switch between two low quality fields, they appear like a single high quality image. Because of this advantage, most current video image formats, including several high-definition video standards, use interlaced techniques rather than progressive methods.

3.1.3 Quantization

$Quantization$ is the procedure of constraining the value of a function at a sampling point to a predetermined finite set of discrete values. Note that the original function can be either continuous or discrete. For example, if we want to specify the temperature of Los Angels, ranging from 0^oC to 50^oC, up to a a precision of 0.1^oC, we must be able to represent 1001 possible values, which require 10 bits to represent one sample. On the other hand, if we only need a precision of 1^oC,

we only have 51 possible values requiring 6 bits for the representation. For image processing, higher precision gives higher image quality but requires more bits in the representation of the samples. We will come back to this topic and discuss how to use quantization to achieve lossy image compression.

3.2 Color Spaces

To describe an image, we need a way to represent the color information. A gray-level image only requires one number to indicate the brightness or luminance of each spatial sample. In reality, our perception of light depends on the light frequency and other properties. When we view a source of light, our eyes respond to three main sensations. The first one is the **color**, which is the main frequencies of the light. The second one is the **intensity** (or brightness), which corresponds to the total energy and can be quantified as the luminance of the the light. The third one is the **purity** (or saturation) of the light, which describes how close a light appears to be a pure spectral color, such as green. Pale colors have low purity and they appear to be almost white. We use the term **chromaticity** to collectively refer to the two characteristics of light, color purity and dominant frequency (hue).

Very often, we employ a **color model** to precisely describe the color components or intensities. In general, a color model is any method for explaining the properties or behavior of color within some particular context. No single model can explain all aspects of color, so people make use of different models to help describe different color characteristics. Here, we consider a **color model** as an abstract mathematical model that describes how colors are presented as tuples of numbers, typically as three or four values or color components; the resulting set of colors that define how the components are to be interpreted is called a **color space**. The commonly used **RGB** color model naturally fits the representation of colors by computers. However, it is not a good model for studying the characteristics of an image.

3.3 RGB Color Model

X-ray, light, infrared radiation, microwave and radio waves are all electromagnetic (EM) waves with different wavelengths. Light waves lie in the visible spectrum with a narrow wavelength band from about 350 to 780 nm. The retina of a human eye can detect only EM waves lying within this visible spectrum but not anything outside. The eye contains two kinds of light-sensitive receptor cells, **cones** and **rods** that can detect light.

The **cones** are sensitive to colors and there are three types of cones, each responding to one of the three primary colors, red, green and blue. Scientists found that our perception of color is a result of our cones' relative response to the red, green and blue colors. Any color can be considered as a combination of these three colors with certain intensity values. The human eye can distinguish about 200 intensities of each of the red, green and blue colors. Therefore, it is natural that we represent each of these colors by a byte which can hold 256 values. In other words, 24 bits are enough to represent the 'true' color. More bits will not increase the quality of an image as human eyes cannot resolve the extra colors. Each eye has 6 to 7 million cones located near the center of the eye, allowing us to see the tiny details of an object.

On the other hand, the **rods** cannot distinguish colors but are sensitive to dim light. Each eye has 75 million to 150 millions rods located near its corner, allowing us to detect peripheral objects in an environment of near darkness.

We can characterize a visible color by a function $C(\lambda)$ where λ is the wavelength of the color in the visible spectrum. The value for a given wavelength λ gives the relative intensity of that wavelength in the color. This description is accurate when we measure the color with certain

physical instrument. However, the human visual system (HVS) does not perceive color in this way. Our brains do not receive the entire distribution $C(\lambda)$ of the visible spectrum but rather three values – the **tristimulus values** – that are the responses of the three types (red, green and blue) of cones to a color. This human characteristics leads to the formulation of the trichromatic theory: *If two colors produce the same tristimulus values, they are visually indistinguishable.* A consequence of of this theory is that it is possible to match all of the colors in the visible spectrum by appropriate mixing of three primary colors. In other words, any color can be created by combining red, green, and blue in varying proportions. This leads to the development of the **RGB color model**.

The RGB (short for red, green, blue) color model decomposes a color into three components, Red (R), Green (G), and Blue (B); we can represent any color by three components R, G, B just like the case that a spatial vector is specified by three components x, y, z. If the color components R, G and B are confined to values between 0 and 1, all definable colors lie in a unit cube as shown in Figure 3-4. This color space is most natural for representing computer images, in which a color specification such as (0.1, 0.8, 0.23) can be directly translated into three positive integer values, each of which is represented by one byte.

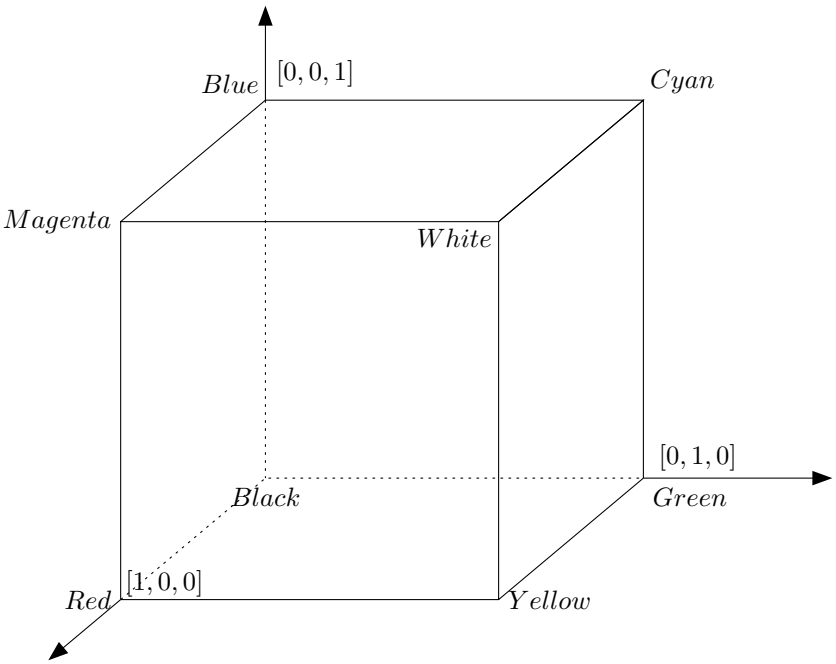

Figure 3-4 RGB Color Cube

In this model, we express a color C in the vector form,

$$C = \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad 0 \leq R, G, B \leq 1 \tag{3.2}$$

In some other notations, the authors like to consider R, G, and B as three unit vectors like the three spatial unit vectors **i**, **j**, and **k**. Just as a spatial vector $\mathbf{V}$ can be expressed as $\mathbf{v} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$, any color is expressed as $C = (rR + gG + bB)$, and the red, green, blue intensities are specified by the values of r, g, and b respectively. In our notation here, R, G, and B may represent the

intensity values of the color components. The next few sections discuss in more detail the color representation of various standards.

Suppose we have two colors C_1 and C_2 given by

$$C_1 = \begin{pmatrix} R_1 \\ G_1 \\ B_1 \end{pmatrix}, \qquad C_2 = \begin{pmatrix} R_2 \\ G_2 \\ B_2 \end{pmatrix}$$

Does it make sense to add these two colors to produce a new color C? For instance, consider

$$C = C_1 + C_2 = \begin{pmatrix} R_1 + R_2 \\ G_1 + G_2 \\ B_1 + B_2 \end{pmatrix}$$

You may immediately notice that the sum of two components may give a value larger than 1, which lies outside the color cube and thus does not represent any color. Just like adding two points in space is illegitimate, we cannot arbitrarily combine two colors. A linear combination of colors makes sense only if the sum of the coefficients is equal to 1. Therefore, we can have

$$C = \alpha_1 C_1 + \alpha_2 C_2 \tag{3.3}$$

when

$$0 \leq \alpha_1, \alpha_2 \quad \textit{and} \quad \alpha_1 + \alpha_2 = 1$$

In this way, we can guarantee that the resulted components will always lie within the color cube as each value will never exceed one. For example,

$$R = \alpha_1 R_1 + \alpha_2 R_2 \leq \alpha_1 \times 1 + \alpha_2 \times 1 = 1$$

which implies

$$R \leq 1$$

The linear combination of colors described by Equation (3.3) is called *color blending*.

3.4 Color Systems

In the RGB model described above, a given color is a point in a color cube as shown in Figure 3-4, and can be expressed as

$$C = \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad 0 \leq R, G, B \leq 1$$

However, RGB systems do not produce identical perceptions and they vary significantly from one to another. For example, suppose we have a yellow color described by the triplet (0.9, 0.8, 0.0). If we feed these values to a CRT and a film image recorder, we shall see different colors, even though in both cases the red is 90 percent of the maximum, the green is 80 percent of the maximum, and there is no blue. The reason is that the CRT phosphors and the film dyes have different color distribution responses. Consequently, the range of displayable colors (or the color **gamut**) is different in each case.

Different organizations have different interests and emphasis on color models. For example, the graphics community is interested in device-independent graphics, and the real differences among display properties are not addressed by most graphics software APIs. Fortunately, this bas been addressed in colorimetery literature, and standards exist for many common color systems. For example, the National Television System Committee (NTSC) defines an RGB system which forms the basis for many CRT systems. We can view the differences in color systems as the differences between various coordinate system for representing the tristimulus values. If

$$C_1 = \begin{pmatrix} R_1 \\ G_1 \\ B_1 \end{pmatrix}, \quad \text{and} \quad C_2 = \begin{pmatrix} R_2 \\ G_2 \\ B_2 \end{pmatrix} \tag{3.4}$$

are the representations of the same color in two different systems, we can find a 3×3 color conversion matrix M such that

$$C_2 = MC_1 \tag{3.5}$$

Regardless of the way we find this matrix, it allows us to produce similar displays on different color systems.

However, this is not a good approach because the color gamuts of two systems may not be the same; even after the conversion of the color components from one system to another, the color may not be producible on the second system. Also, the printing and graphic arts industries use a four color subtractive system (CMYK) that includes black (K) as the fourth primary. Moreover, the linear color theory is only an approximation to human perception of colors. The distance between two points in the color cube does not necessarily measure how far apart the colors are perceptually. For example, humans are particularly sensitive to color shifts in blue.

The International Commission on Illumination, referred to as the CIE (Commission Internationale de l'Eclarage) defined in 1931 three standard primaries, which are actually imaginary colors. They are defined mathematically with positive color-matching functions shown in Figure 3-5. If the spectral power distribution (SPD) for a colored object is weighted by the curves of Figure 3-5, the CIE chromaticity coordinates can be calculated. This provides an international standard definition of all colors, and the CIE primaries eliminate negative-value color-matching and other problems related to the selection of a set of real primaries.

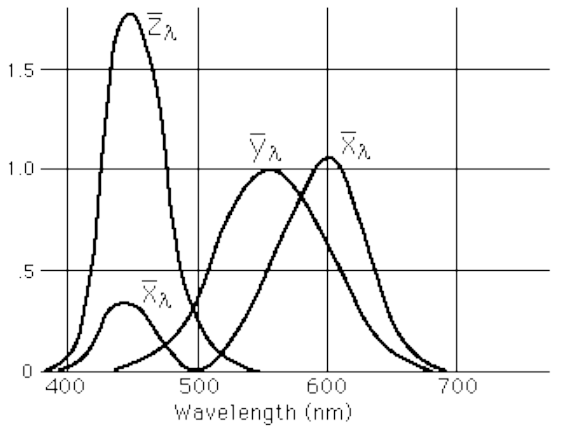

Figure 3-5 Matching functions of the three CIE primaries

3.4.1 The XYZ Color Model

The set of CIE primaries defines a color model that is in general referred to as the **XYZ color model**, where parameters X, Y, and Z represent the tristimulus values, the amount of each CIE primary required to produce a given color. The tristimulus values do not correspond to real colors, but they do have the property that any real color can be represented as a positive combination of them. Therefore, an RGB model describes a color in the same way as the XYZ model does. Actually, most color standards are based on this theoretical XYZ model. In this model, the Y primary is the luminance of the color and all colors can be represented by positive tristimulus values.

Due to the nature of the distribution of cones in the eye, the tristimulus values depend on the observer's field of view. To eliminate this variable, the CIE defined the standard (colorimetric) observer, which is characterized by three color matching functions. The color matching functions are the numerical description of the chromatic response of the observer. The three color-matching functions are referred to as $\overline{X}(\lambda), \overline{Y}(\lambda)$, and $\overline{Z}(\lambda)$, which can be thought of as the spectral sensitivity curves of three linear light detectors that yield the CIE XYZ tristimulus values X, Y, and Z. The tabulated numerical values of these functions are known collectively as the CIE standard observer.

The tristimulus values for a color with a spectral power distribution $I(\lambda)$ are given in terms of the standard observer by:

$$\begin{aligned} X &= \int_0^\infty \overline{X}_\lambda I(\lambda) d\lambda \\ Y &= \int_0^\infty \overline{Y}_\lambda I(\lambda) d\lambda \\ Z &= \int_0^\infty \overline{Z}_\lambda I(\lambda) d\lambda \end{aligned} \tag{3.6}$$

where λ is the wavelength of the equivalent monochromatic light.

A color can be specified by the tristimulus values, X, Y, and Z:

$$C = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{3.7}$$

We may also represent a color in the XYZ color space as an additive combination of the primaries using unit vectors $\mathbf{X}$, $\mathbf{Y}$, and $\mathbf{Z}$. Therefore, we can express Equation (3.7) as

$$C = X\mathbf{X} + Y\mathbf{Y} + Z\mathbf{Z} \tag{3.8}$$

We can use 3×3 matrices to convert from XYZ color representation to representations in other standard systems. Also, it is convenient to normalize the X, Y, and Z values against the sum $X + Y + Z$, which is the total light energy. The normalized values are usually referred to as the **chromaticity coordinates**:

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z}, \quad z = \frac{Z}{X+Y+Z} \tag{3.9}$$

As $x + y + z = 1$, any color can be represented with just the x and y coordinates if the total energy is known. The parameters x and y depend only on hue and purity of the color and are called **chromaticity values**. Instead of using the total energy, people typically use the luminance Y and the chromaticity values x, and y to specify a color. The other two CIE values can be calculated as

$$X = \frac{x}{y}Y, \quad Z = \frac{z}{y}Y \tag{3.10}$$

where $z = 1 - x - y$. Using chromaticity coordinates (x, y), we can represent all colors on a two-dimensional diagram as shown in Figure 3-6.

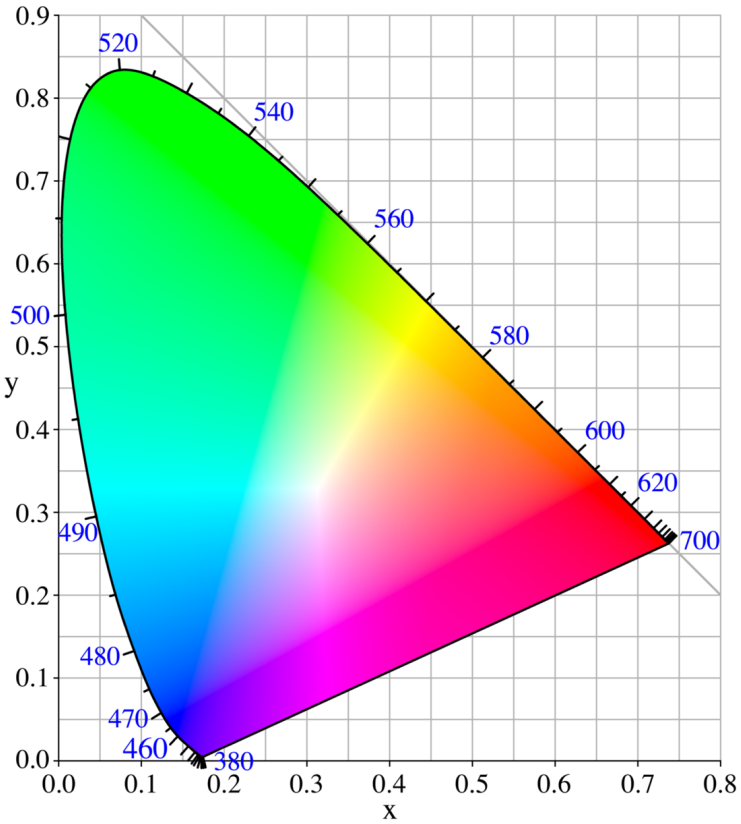

Figure 3-6 CIE Chromaticity Diagram

In Figure 3-6, we plot the normalized coordinates x and y for colors in the visible spectrum and obtain the horseshoe-shaped curve. Ticks along the curve are the spectral colors (pure colors). A tick number indicates the wavelength of the color in nm. For example, the wavelength of the red color is 700 nm and that of yellow is about 570 nm. The line connecting the endpoints of the horseshoe is known as the non-spectral *line of purples*, which is not part of the spectrum.

Lumninance values are not included in the chromaticity diagram; colors with different luminance but with the same chromaticity map to the same point in the diagram. The chromaticity diagram is useful for

- comparing color gamuts for different set of primaries,
- identifying complementary colors, and
- determining purity and dominant wavelength of a given color.

Color Gamuts

The gamut is the set of possible colors within a color system. No one system can reproduce all possible colors in the visible spectrum. It is not possible for a designer to create every color in the spectrum with either additive or subtractive colors. Both systems can reproduce a subset of all

visible colors, and while those subsets generally overlap, there are colors which can be reproduced with additive color and not with subtractive color and vice versa.

We identify color gamuts on the chromaticity diagram as straight line segments or polygon regions. Figure 3-7 shows schematically the gamut of reproducible colors for the RGB primaries of a typical color CRT monitor, CMYK color printing, and for the NTSC television.

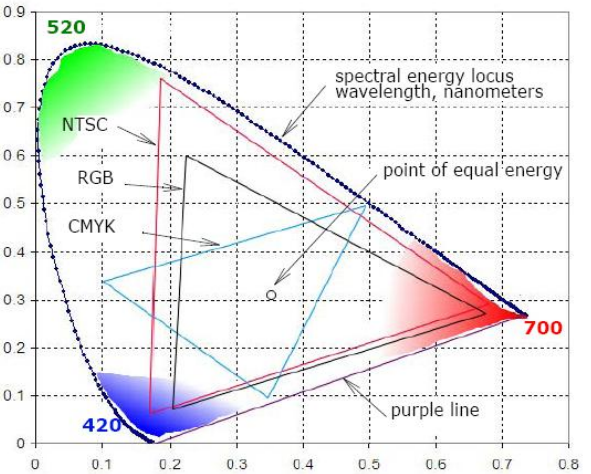

Figure 3-7 Chromaticity Diagram and Color Gamut

3.4.2 YUV Color Model

While the RGB color model is well-suited for displaying color images on a computer screen, it is not an effective model for image processing or video compression. This is because the human visual system (HVS) is more sensitive to luminance (brightness) than to colors. Therefore, it is more effective to represent a color image by separating the luminance from the color information and representing luma with a higher resolution than color.

The YUV color model, defined in the TV standards, is an efficient way of representing color images by separating brightness from color values. Historically, YUV color space was developed to provide compatibility between color and black /white analog television systems; it is not defined precisely in the technical and scientific literature. In this model, Y is the luminance (luma) component, which is the same as the Y component in the CIE XYZ color space, and U and V are the color differences known as chrominance or chroma, which is defined as the difference between a color and a reference white at the same luminance. The conversion from RGB to YUV is given by the following formulas:

$$\begin{aligned} Y &= k_r R + k_g G + k_b B \\ U &= B - Y \\ V &= R - Y \end{aligned} \tag{3.11}$$

with

$$\begin{aligned} &0 \le k_r, k_b, k_g \\ &k_r + k_b + k_g = 1 \end{aligned} \tag{3.12}$$

Note that equations (3.11) and (3.12) imply that $0 \le Y \le 1$ if the R, G, B components lie within the unit color cube. However, U and V can be negative. Typically,

$$k_r = 0.299, k_g = 0.587, k_b = 0.114 \tag{3.13}$$

which are values used in some TV standards. For convenience, in the forthcoming discussions, we always assume that $0 \leq R, G, B \leq 1$ unless otherwise stated.

The complete description of an image is specified by Y (the luminance component) and the two color differences (chrominance) U and V. If the image is black-and-white, $U = V = 0$. Note that we do not need another difference ($G - Y$) for the green component because that would be redundant. We can consider (3.11) as three equations with three unknowns, R, G, and B, and thus we can always solve for the three unknowns and recover R, G, and B. A fourth equation is not necessary.

It seems that there is no advantage of using YUV over RGB to represent an image as both system requires three components to specify an image sample. However, as we mentioned earlier, human eyes are less sensitive to color than to luminance. Therefore, we can represent the U and V components with a lower resolution than Y and the reduction of the amount of data to represent chrominance components will not have an obvious effect on visual quality. Representing chroma with less number of bits than luma is a simple but effective way of compressing an image.

The conversion from RGB space to YUV space can be also expressed in matrix form:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.299 & -0.587 & 0.886 \\ 0.701 & -0.587 & -0.114 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{3.14}$$

The conversion from YUV space to RGB space using matrix is accomplished with the inverse transformation of (3.14):

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & -0.509 & -0.194 \end{pmatrix} \begin{pmatrix} Y \\ U \\ V \end{pmatrix} \tag{3.15}$$

3.4.3 YCbCr Color Model

The YCbCr color model defined in the standards of ITU (International Telecommunication Union) is closely related to YUV but with the chrominace components scaled and shifted to ensure that they lie within the range 0 and 1. It is sometimes abbreviated to YCC. It is also used in the JPEG and MPEG standards. In this model, an image sample is specified by a luminance (Y) component and two chrominance components (Cb, and Cr). The following equations convert an RGB image to one in YCbCr space.

$$\begin{aligned} Y &= k_r R + k_g G + k_b B \\ C_b &= \frac{B - Y}{2(1 - k_b)} + 0.5 \\ C_r &= \frac{(R - Y)}{2(1 - k_r)} + 0.5 \\ k_r &+ k_b + k_g = 1 \end{aligned} \tag{3.16}$$

An image may be captured in the RGB format and then converted to YCbCr to reduce storage or transmission requirements. Before displaying the image, it is usually necessary to convert the image back to RGB. The conversion from YCbCr to RGB can be done by solving for R, G, B in the equations of (3.16). The equations for converting from YCbCr to RGB are shown below:

$$
\begin{aligned}
R &= Y + (2C_r - 1)(1 - k_r) \\
B &= Y + (2C_b - 1)(1 - k_b) \\
G &= \frac{Y - k_r R - k_b B}{k_g} \\
&= Y - \frac{k_r(2C_r - 1)(1 - k_r) + k_b(2C_b - 1)(1 - k_b)}{k_g}
\end{aligned}
\tag{3.17}
$$

If we use the ITU standard values $k_b = 0.114, k_r = 0.299, k_g = 1 - k_b - k_r = 0.587$ for (3.16) and (3.17), we will obtain the following commonly used conversion equations.

$$
\begin{aligned}
Y &= 0.299R + 0.587G + 0.114B \\
C_b &= 0.564(B - Y) + 0.5 \\
C_r &= 0.713(R - Y) + 0.5 \\
\\
R &= Y + 1.402C_r - 0.701 \\
G &= Y - 0.714C_r - 0.344C_b + 0.529 \\
B &= Y + 1.772C_b - 0.886
\end{aligned}
\tag{3.18}
$$

In equations (3.16), it is obvious that $0 \leq Y \leq 1$ as $0 \leq R, G, B \leq 1$. It turns out that the chrominance components C_b and C_r defined in (3.16) also always lie within the range [0, 1]. We prove this for the case of C_b. From (3.16), we have

$$
\begin{aligned}
C_b &= \frac{B - Y}{2(1 - k_b)} + \frac{1}{2} \\
&= \frac{B - k_r R - k_g G - k_b B + 1 - k_b}{2(1 - k_b)} \\
&= \frac{B}{2} + \frac{-k_r R - k_g G + 1 - k_b}{2(1 - k_b)} \\
&\geq \frac{B}{2} + \frac{-k_r \times 1 - k_g \times 1 + 1 - k_b}{2(1 - k_b)} \\
&= \frac{B}{2} \\
&\geq 0
\end{aligned}
$$

Thus

$$
C_b \geq 0 \tag{3.19}
$$

Also,

$$\begin{aligned} C_b &= \frac{B-Y}{2(1-k_b)} + \frac{1}{2} \\ &= \frac{B-k_rR-k_gG-k_bB}{2(1-k_b)} + \frac{1}{2} \\ &\leq \frac{B-k_bB}{2(1-k_b)} + \frac{1}{2} \\ &= \frac{B}{2} + \frac{1}{2} \\ &\leq \frac{1}{2} + \frac{1}{2} \\ &= 1 \end{aligned}$$

Thus

$$C_b \leq 1 \tag{3.20}$$

Combining (3.19) and (3.20), we have

$$0 \leq C_b \leq 1 \tag{3.21}$$

Similarly

$$0 \leq C_r \leq 1 \tag{3.22}$$

In summary, we have the following situation.

$$\begin{aligned} &\text{If} \quad 0 \leq R, G, B \leq 1 \\ &\text{then} \quad 0 \leq Y, C_b, C_r \leq 1 \end{aligned} \tag{3.23}$$

Note that the converse is not true. That is, if $0 \leq Y, C_b, C_r \leq 1$, it does **not** imply $0 \leq R, G, B \leq 1$. A knowledge of this helps us in the implementations of the conversion from RGB to YCbCr and vice versa. We mentioned earlier that the eye can only resolve about 200 different intensity levels of each of the RGB components. Therefore, we can quantize all the RGB components in the interval [0,1] to 256 values, from 0 to 255, which can be represented by one byte of storage without any loss of visual quality. In other words, one byte (or an 8-bit unsigned integer) is enough to represent all the values of each RGB component. When we convert from RGB to YCbCr, it only requires one 8-bit unsigned integer to represent each YCbCr component. This implicitly implies that all conversions can be done efficiently in integer arithmetic that we shall discuss below.

3.3 Conversions between RGB and YCbCr

It is straightforward to write a C/C++ program to convert RGB to YCbCr or from YCbCr to RGB. We discussed in the previous section that the implementation can be effectively done in integer arithmetic. However, for clarity of presentation, we shall first discuss a floating point implementation. The C/C++ programs presented in this book are mainly for illustration of concepts. In most cases, error checking is omitted and some variable values are hard-coded.

3.3.1 Floating Point Implementation

The program listed below shows the conversion between RGB and YCbCr using ITU standard coefficients. It is a direct implementation of equations (3.18). The R, G, and B values, which must lie between [0,1] are hard-coded and converted to Y, Cb, and Cr, which are then converted back to R, G, and B.

Listing 3-1 (rgbyccf.cpp) Conversions Between RGB and YCbCr

```
int main ()
{
  //0 <= R, G, B <= 1, sample values
  double R = 0.3, G = 0.7, B = 0.2, Y, Cb, Cr;

  printf("\nOriginal R, G, B:\t%f, %f, %f", R, G, B );

  Y   = 0.299 * R + 0.587 * G + 0.114 * B;
  Cb = 0.564 * (B - Y) + 0.5;
  Cr = 0.713 * (R - Y) + 0.5;

  printf("\nConverted Y, Cb, Cr:\t%f, %f, %f", Y, Cb, Cr );

  //recovering R, G, B
  R = Y + 1.402 * Cr - 0.701;
  G = Y - 0.714 * Cr - 0.344 * Cb  + 0.529;
  B = Y + 1.772 * Cb - 0.886;

  printf("\nRecovered R, G, B:\t%f, %f, %f\n\n", R, G, B );
  return 0;
}
```

The program generates the following outputs:

```
Original R, G, B:       0.300000, 0.700000, 0.200000
Converted Y, Cb, Cr:    0.523400, 0.317602, 0.340716
Recovered R, G, B:      0.300084, 0.699874, 0.200191
```

The recovered R, G, and B values differ slightly from the original ones due to rounding errors in computing and the representation of numbers in binary form.

3.3.2 Integer Implementation

The above program illustrates the conversion between RGB and YCbCr using floating-point calculations. However, such an implementation is not practical. Not only that rounding errors are introduced in the computations, floating-point arithmetic is very slow. When compressing an image, we need to apply the conversion to every pixel. Switching to integer-arithmetic in calculations can easily shorten the computing time by a factor of two to three. In RGB-YCbCr conversion, using integer-arithmetic is quite simple because we can always approximate a real number as a fraction between two integers. For example, the coefficients for calculating Y from RGB can be expressed as:

$$\begin{aligned} 0.299 &= 19595/2^{16} \\ 0.587 &= 38470/2^{16} \\ 0.114 &= 7471/2^{16} \end{aligned} \tag{3.24}$$

The integer-arithmetic expression for Y can be obtained by multiplying the equation

$$Y = 0.299R + 0.587G + 0.114B$$

by 2^{16}, which becomes

$$2^{16}Y = 19595R + 38470G + 7471B \tag{3.25}$$

At the same time, we quantize the R, G, and B values from [0, 1] to $0, 1, ..., 255$ which can be done by multiplying the floating-point values by 255. We also need to quantize the shifting constants 0.5, 0.701, 0.529, and 0.886 of (3.18) using the same rule by multiplying them by 255, which will become

$$\begin{aligned} 0.5 \times 255 &= 128 \\ 0.701 \times 255 &= 179 \\ 0.529 \times 255 &= 135 \\ 0.886 \times 255 &= 226 \end{aligned} \tag{3.26}$$

Actually, representing a component of RGB with integer values 0 to 255 is the natural way that a modern computer handles color data. In practice, each pixel has three components (R, G, and B) and each component value is saved as an 8-bit unsigned number.

As shown in (3.18), in floating-point representation, the C_b component is given by

$$C_b = 0.564(B - Y) + 0.5$$

After quantization, it becomes

$$C_b = 0.564(B - Y) + 128 \tag{3.27}$$

Multiplying (3.27) by 2^{16}, we obtain

$$2^{16}C_b = 36962(B - Y) + 128 \times 2^{16} \tag{3.28}$$

The corresponding equation for C_r is:

$$2^{16}C_r = 46727(R - Y) + 128 \times 2^{16} \tag{3.29}$$

As R, G, and B have become integers, we can carry out the calculations using integer multiplications and then divide the results by 2^{16}. In binary calculations, dividing a value by 2^{16} is the same as shifting the value right by 16. Therefore, from (3.27), (3.28) and (3.29), the calculations of Y and C_b using integer-arithmetic can be carried out using the following code.

$$\begin{aligned} Y &= (19595 * R + 38470 * G + 7471 * B) >> 16; \\ Cb &= (36962 * (B - Y) >> 16) + 128; \\ Cr &= (46727 * (R - Y) >> 16) + 128; \end{aligned} \tag{3.30}$$

One should note that the sum of the coefficients in calculating Y is 2^{16} (i.e. $19595 + 38470 + 7471 = 65536 = 2^{16}$), corresponding to the requirement $k_r + k_g + k_b = 1$ in the floating-point representation.

The constraints of (3.23) and the requirement of $0 \leq R, G, B \leq 255$ imply that in our integer representation,

$$\begin{aligned} 0 \leq Y \leq 255 \\ 0 \leq Cb \leq 255 \\ 0 \leq Cr \leq 255 \end{aligned} \tag{3.31}$$

In (3.9) the R component is obtained from Y and C_r:

$$R = Y + 1.402C_r - 0.701$$

In integer-arithmetic, this becomes

$$2^{16}R = 2^{16}Y + 91881C_r - 2^{16} \times 179 \tag{3.32}$$

The value of R is obtained by dividing (3.23) by 2^{16} as shown below in C/C++ code:

$$R = (Y + 91881 * Cr >> 16) - 179; \tag{3.33}$$

We can obtain similar equations for G and B. Combining all these, we can express equations of (3.18) in integer-arithmetic using C/C++ code like the following form:

$$\begin{aligned} Y &= (19595 * R + 38470 * G + 7471 * B) >> 16; \\ Cb &= (36962 * (B - Y) >> 16) + 128; \\ Cr &= (46727 * (R - Y) >> 16) + 128; \end{aligned} \tag{3.34}$$

$$\begin{aligned} R &= Y + (91881 * Cr >> 16) - 179; \\ G &= Y - ((46793 * Cr + 22544 * Cb) >> 16) + 135; \\ B &= Y + (116129 * Cb >> 16) - 226; \end{aligned}$$

Listing 3-2 (rgbycci.cpp) RGB-YCbCr Conversions Using Integer Arithmetic

```
----------------------------------------------------------------------------
/* Note:
 *  2^16 = 65536
 *  kr = 0.299 = 19595 / 2^16
 *  kg = 0.587 = 38470 / 2^16
 *  Kb = 0.114 =  7471 / 2^16
 *  0.5 = 128 / 255
 *  0.564 = 36962 / 2^16
 *  0.713 = 46727 / 2^16
 *  1.402 = 91881 / 2^16
 *  0.701 = 135 / 255
 *  0.714 = 46793 / 2^16
 *  0.344 = 22544 / 2^16
 *  0.529 = 34668 / 2^16
 *  1.772 = 116129 / 2^16
 *  0.886 = 226 / 255
 */

int main ()
{
  unsigned char R, G, B; //RGB components
  unsigned char Y, Cb, Cr; //YCbCr components

  //some sample values for demo
   R = 252; G = 120; B = 3;

  //convert from RGB to YCbcr
  Y = (unsigned char)((19595 * R + 38470 * G + 7471 * B ) >> 16);
  Cb = (unsigned char)( (36962 * ( B - Y )  >> 16) + 128 );
```

```
  Cr = (unsigned char)( (46727 * ( R - Y )  >> 16) + 128 );

  printf("\nOriginal RGB and corresponding YCbCr values:");
  printf("\n\tR = %6d, G  = %6d, B  = %6d", R, G, B );
  printf("\n\tY = %6d, Cb = %6d, Cr = %6d", Y, Cb, Cr );

  //convert from YCbCr to RGB
  R = ( unsigned char ) (Y + (91881 * Cr  >> 16) - 179 );
  G = ( unsigned char ) (Y - ((22544 * Cb + 46793 * Cr) >> 16) + 135);
  B = ( unsigned char ) (Y  + (116129 * Cb  >> 16) - 226 );

  printf("\n\nRecovered RGB values:");
  printf("\n\tR = %6d, G  = %6d, B  = %6d\n\n", R, G, B );

  return 0;
}
-------------------------------------------------------------------------
```

In (3.34), it is obvious that a 32-bit integer is large enough to hold any intermediate calculations. Program Listing 3-2 above shows its implementation. The program generates the outputs shown below.

Outputs of Program Listing 3-2

```
Original RGB & corresponding YCbCr values:
        R =    252, G  =    120, B  =      3
        Y =    146, Cb =     47, Cr =    203

Recovered RGB values:
        R =    251, G  =    120, B  =      3
```

Again, some precision has been lost when we recover R, G, and B from the converted Y, Cb, and Cr values. This is due to the right shifts in the calculations, which are essentially truncate operations. Because of rounding or truncating errors, the recovered R, G, and B values may not lie within the range [0, 255]. To remedy this, we can have a function that check the recovered value; if the value is smaller than 0, we set it to 0 and if it is larger than 255, we set it to 255. The check should be done in the intermediate steps or we can represent R, G, and B with a signed-number having more than 8 bits. For example,

```
if ( R < 0 )
  R = 0;
else if ( R > 255 )
  R = 255;
```

However, this check is not necessary when we convert from RGB to YCbCr. This is because from (3.23), we know that we always have $0 \leq Y, C_b, C_r \leq 1$. For any positive real number, a and $0 \leq a \leq 1$ and any positive integer I,

$$0 \leq Round(aI) \leq Round(I) = I \quad \text{and similarly} \quad 0 \leq Truncate(aI) \leq I$$

This implies that after quantization and rounding, we always have $0 \leq Y, C_b, C_r \leq 255$.

3.4 YCbCr Sampling Formats

We mentioned earlier that we may represent the C_r and C_b components with less bits than Y without much effect on visual quality as our eyes are less sensitive to color than to luminance.

This is a simple way of compressing an image. In general, people consider four adjacent pixels of an image at a time and this leads to the standards 4:4:4, 4:2:2, and 4:2:0 sampling formats, which are supported by video standards MPEG-4 and H.264.

4:4:4 YCbCr Sampling Formats

4:4:4 YCbCr sampling means that for every four luma samples there are four C_b and four C_r samples and hence the three components, Y, C_b, and C_r have the same resolution. The numbers indicate the relative sampling rate of each component in the horizontal direction. So at every pixel position in the horizontal direction, a sample of each component of (Y, C_b, C_r) exists. The 4:4:4 YCbCr format requires as many bits as the RGB format and thus preserves the full fidelity of the chrominance components.

4:2:2 YCbCr Sampling Formats (High Quality Color Reproduction)

4:2:2 YCbCr sampling means that the chrominance components have the same vertical resolution as the luma but half the horizontal resolution. Therefore, for every four luma samples there are two C_b and two C_r samples. Sometimes this format is referred to as YUY2.

4:2:0 YCbCr Sampling Formats (Digital Television and DVD Storage)

4:2:0 YCbCr sampling means that each of the chrominance components has half the horizontal and vertical resolution of the luma component. That is, for every four luma samples (Y) there are one C_b and one C_r samples. It is sometimes known as YV12 and is widely used in video conferencing, digital television and digital versatile disk (DVD) storage. The term “4:2:0” is rather confusing as the numbers do not reflect relative resolutions between the components and apparently have been chosen due to historical reasons to distinguish it from the 4:4:4 and 4:2:2 formats.

Figure 3-8 shows the sampling format of 4:2:0; progressive sampling is used.

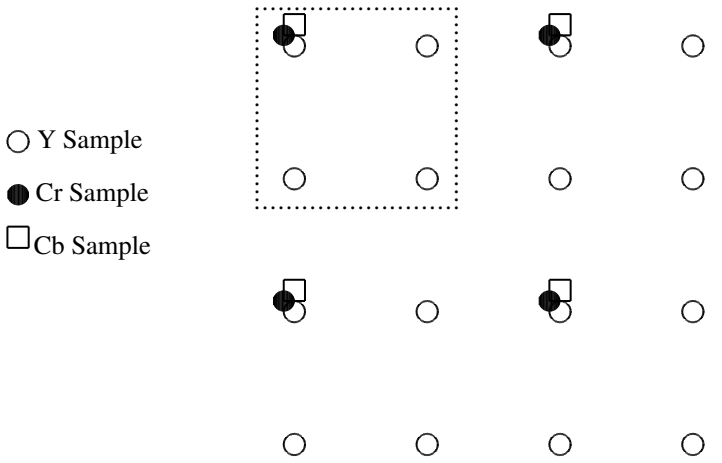

Figure 3-8 4:2:0 Sampling Patterns

Example 3-1

```
Image resolution: 1024 x 768 pixels

4:4:4 Y, Cb, Cr resolution: 1024 x 768 samples
Total number of bits: 1024x768x8x3 = 18874368 bits

4:2:0 Y resolution: 1024 x 768 samples
4:2:0 Cb, Cr resolution: 512x384 samples (8 bits for samples)
Total number of bits: (1024 x 768 x 8) + (512 x 384 x 8 x 2)
                      = 9437184 bits

The 4:2:0 format requires half as many bits as the 4:4:4
format and the RGB format.
```

3.5 Measuring Video Quality

It is important to have some agreed upon methods to measure the quality of video so that we can evaluate and compare various video images presented to the viewer. However, this is a difficult and often an imprecise process and inherently subjective as there are so many factors that can influence the measurement. In general, there are two classes of methods that people use to measure video quality: *subjective tests*, where human subjects are asked to assess or rank the images, and *objective tests*, which compute the distortions between the original and processed video sequences.

3.5.1 Subjective Quality Measurement

Subjective quality measurement asks human subjects to rank the quality of a video based on their own perception and understanding of quality. For example, a viewer can be asked to rate the quality on a 5-point scale, with quality ratings ranging from bad to excellent as shwon in Figure 3-9.

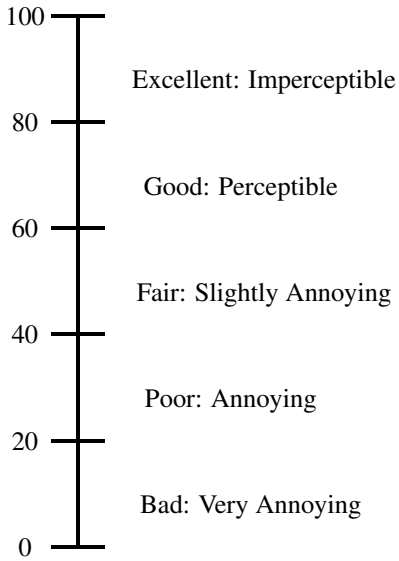

Figure 3-9 Example of video quality assessment scale used in subjective tests

Very often, a viewer's perception on a video is affected by many factors like the viewing environment, the lighting conditions, display size and resolution, the viewing distance, the state of mind of the viewer, whether the material is interesting to the viewer and how the viewer interacts

with the visual scene. It is not uncommon that the same viewer who observes the same video at different times under different environments may give significantly different evaluations on the quality of the video. For example, it has been shown that subjective quality ratings of the same video sequence are usually higher when accompanied by good quality sound, which may lower the evaluators' ability to detect impairments. Also, viewers tend to give higher ratings to images with higher contrast or more colorful scenes even though objective tests show that they have larger distortions in comparison to the originals.

Nevertheless, subjective quality assessment still remains the most reliable methods of measuring video quality. It is also the most efficient method to test the performance of components, like video codecs, human vision models and objective quality assessment metrics.

3.5.1.1 ITUR BT.500

The ITU-R Recommendation BT-500-11 formalizes video subjective tests by recommending various experiment parameters like viewing distance, room lighting, display features, selection of subjects and test material, assessment and data analysis methods. There are three most commonly used procedures from the standard: *Double Stimulus Continuous Quality Scale (DSCQS), Double Stimulus Impairment Scale (DSIS) and Single Stimulus Continuous Quality Evaluation (SSCQE) .*

Double Stimulus Continuous Quality Scale (DSCQS)

In the DSCQS method, a viewer is presented with a pair of images or short sequences X and Y, one after the other. The viewer is asked to rank X and Y by marking on a continuous line with five intervals ranging from 'Bad' to 'Excellent', which has an equivalent numerical scale from 0 to 100, like the one shown in Figure 3-9. The reference and test sequences are shown to the viewer twice in alternating fashion, the order chosen in random. The accessor does not know in advance which is the reference sequence and which is the test sequence. Figure 3-10 shows an experimental set-up that can be used for testing a video coder-decoder (CODEC); it is randomly assigned which sequence is X and which sequence is Y.

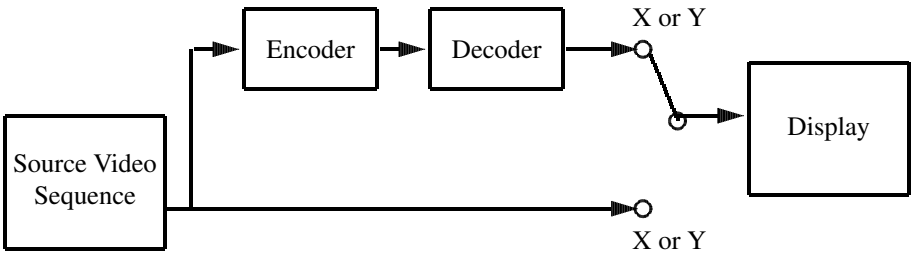

Figure 3-10 DSCQS Testing System

Double Stimulus Impairment Scale (DSIS)

In the DSIS method the reference sequence is always presented before the test sequence, and it is not necessary to show the pair twice. Viewers are asked to rate the sequences on a 5-point scale, ranging from "very annoying" to "imperceptible" like the one shown in Figure 3-9. This method is more effective for evaluating clearly visible impairments, such as noticeable artifacts caused by encoding or transmission.

Both the DSCQS and DSIS methods use short sequences (8 - 10 sec) in the test and this becomes a problem when we want to evaluate video sequences with long duration and quality varies significantly over time like those distributed via the Internet.

Single Stimulus Continuous Quality Evaluation (SSCQE)

SSCQE is designed to evaluate video sequences with significant temporal variations of quality. In this method, longer sequences (20 - 30 minutes) are presented to the viewers without any reference sequence. The accessors evaluate instantaneously the perceived quality by continuously adjusting a side slider on the DSCQS scale, ranging from "bad" to "excellent". The slider value is periodically sampled every 1 - 2 seconds. Using this method, differences between alternative transmission configurations can be analyzed in a more informative manner. However, as the accessor has to adjust the slider from time to time, she may be distracted and thus the rating may be compromised. Also, because of the 'recency or memory effect', it is quite difficult for the accessor to consistently detect momentary changes in quality, leading to stability and reliability problems of the results.

3.5.2 Objective Quality Measurement

Though subjective measurements are the most reliable method to evaluate video qualities, they are complex and expensive as human subjects are required to do the evaluation. It is a lot more convenient and cost-effective to automatically measure quality using an algorithm. Indeed, video processing system developers rely heavily on objective (algorithmic) measurement to access video qualities. The simplest and most widely used form of measuring the quality is Peak Signal to Noise Ratio (PSNR), which calculates the distortion at the pixel level. Peak Signal to Noise Ratio (PSNR) measures the mean squared error (MSE) between the reference and test sequences on a logarithmic scale, relative to the square of the highest possible signal value in the image, $(2^n - 1)^2$, where n is the number of bits per image sample. It is described by Equation (3.35):

$$PSNR_{db} = 10log_{10}\frac{(2^n - 1)^2}{MSE} \tag{3.35}$$

The mean squared error, MSE of two $M \times N$ images X and Y where one of the images is considered to be a noisy approximation of the other with sample values X_{ij} and Y_{ij} respectively can be calculated using the following equation:

$$MSE = \frac{1}{M \times N}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(X_{ij} - Y_{ij})^2 \tag{3.36}$$

Though PSNR is a straightforward metric to calculate, it cannot describe distortions perceived by a complex and multi-dimensional system like the human visual system (HVS), and thus fails to give good evaluations in many cases. For example, a viewer may be interested in an object of an image but not its background. If the background is largely distorted, the viewer would still rate that the image is of high quality; however, PSNR measure would indicate that the image is of poor quality. The limitations of this metric have led to recent research in image processing that has focused on developing metrics that resembles the response of real human viewers. Many approaches have been proposed but none of them can be accepted as a standard to be used as an alternative to subjective evaluation. The search of a good acceptable objective test for images will remain a research topic for some time.