Digital Image Processing

Color image processing

Erchan Aptoula

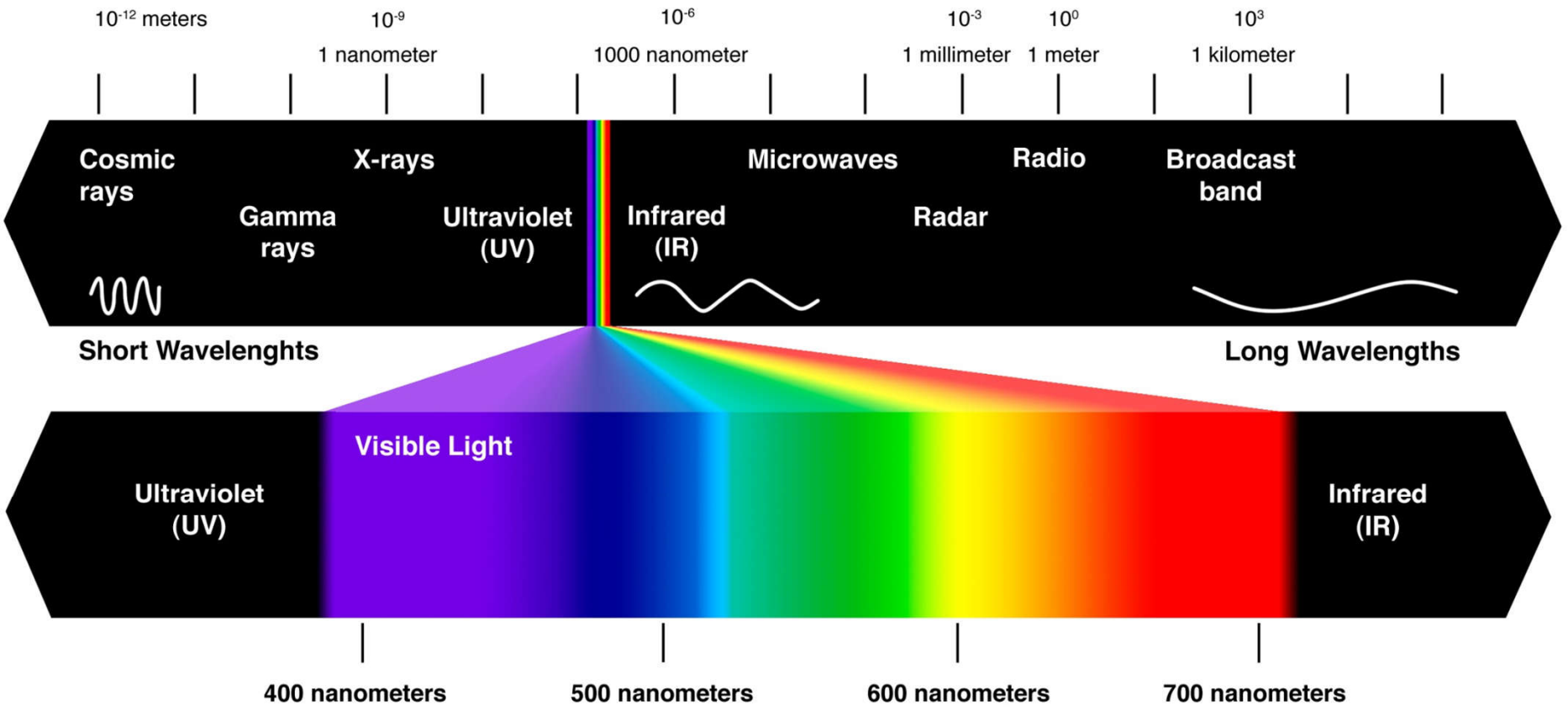
Spring 2016-2017

Contents

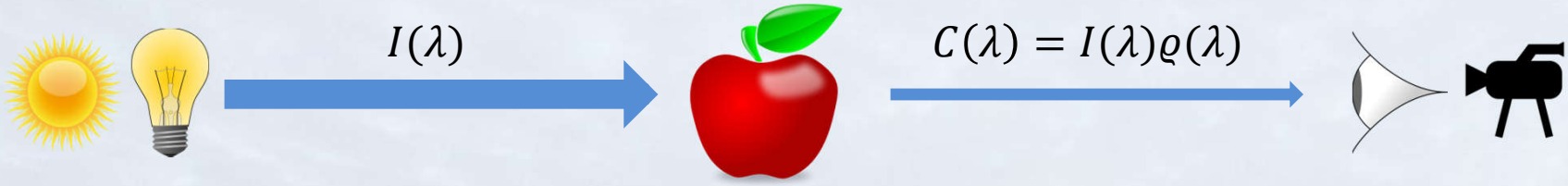
- Color perception
- Color models
- How to process color images
- ..and a few words about compression.

Following our introduction to binary and grayscale image processing, now let's advance to the wonderful world of color images!

The electromagnetic spectrum



Human sight is limited with wavelengths between approximately 390 to 700 nm; i.e. the visible spectrum.



illuminant

spectral reflectance
$\varrho(\lambda)$

Receiver's sensitivity
$S(\lambda) = C(\lambda)s(\lambda)$
$M(\lambda) = C(\lambda)m(\lambda)$
$L(\lambda) = C(\lambda)l(\lambda)$

Our eyes are **trichromatic**, i.e. we have three types of sensors, sensitive to **S**hort, **M**edium and **L**ong wavelengths. It is the total response of all three that creates the perception of color at a certain wavelength.

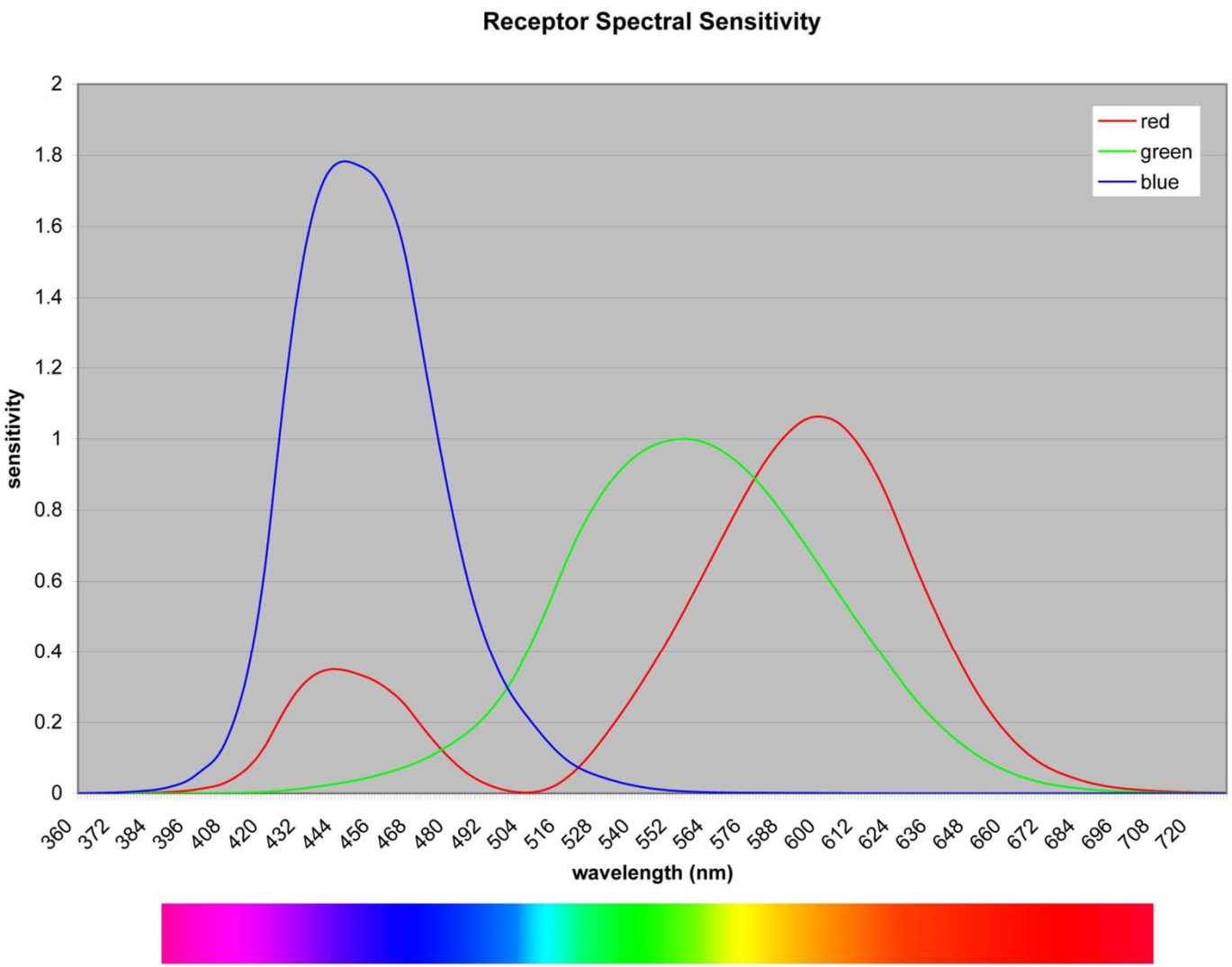
And the perceived sensations for each sensor type:

$\int_\lambda I(\lambda)\varrho(\lambda)s(\lambda)d\lambda$
Amount of red

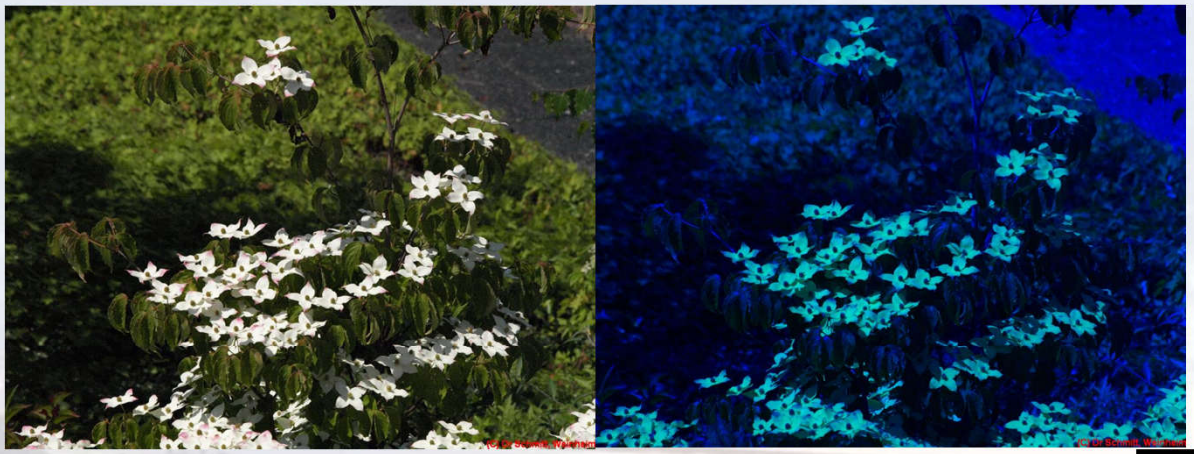
$\int_\lambda I(\lambda)\varrho(\lambda)m(\lambda)d\lambda$
Amount of green

$\int_\lambda I(\lambda)\varrho(\lambda)l(\lambda)d\lambda$
Amount of blue

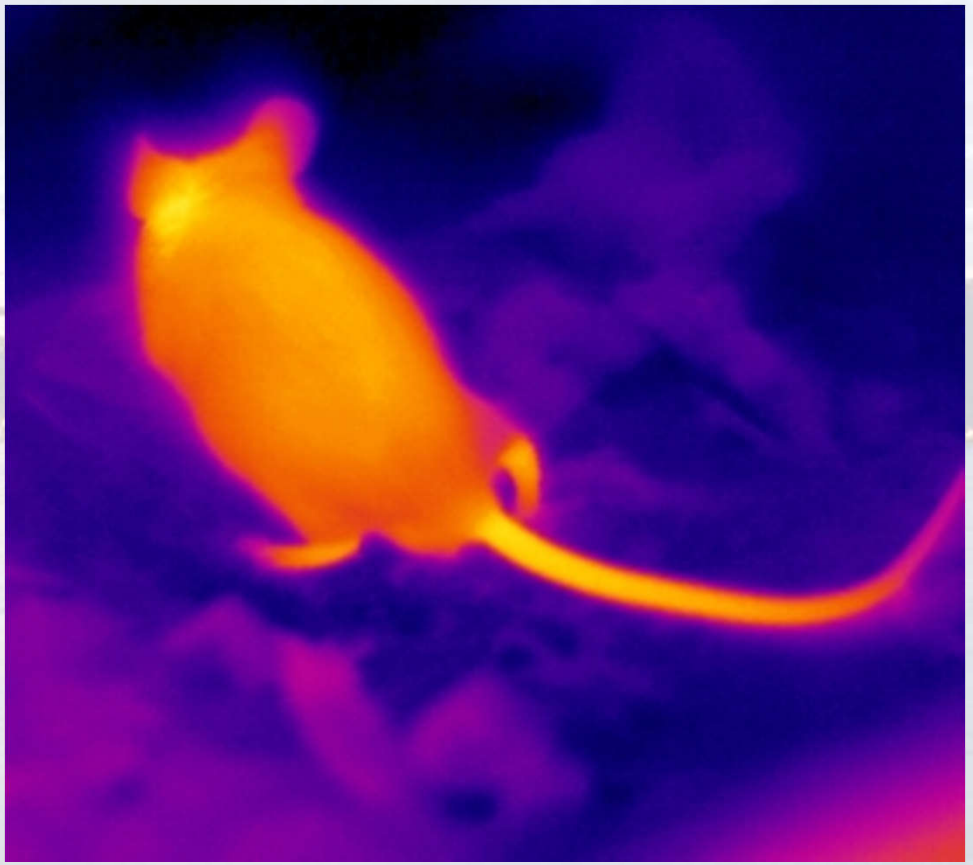
These are approximations of the normalized responses to the visible spectrum of the red (L), green(M), and blue (S) receptors of a typical human eye.



Other beings with trichromatic vision: **bee** UV+G+B; useful for pollination.



Reptiles have access to color, though at a more limited range than humans, but also to IR and UV.



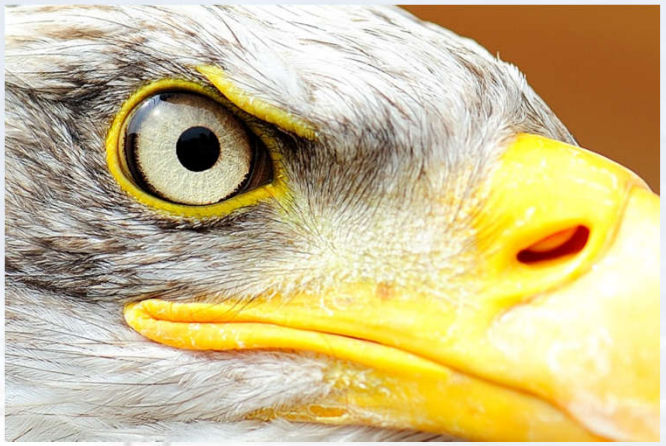
Cats: near-sighted, with limited color perception (dichromats) but with excellent night vision

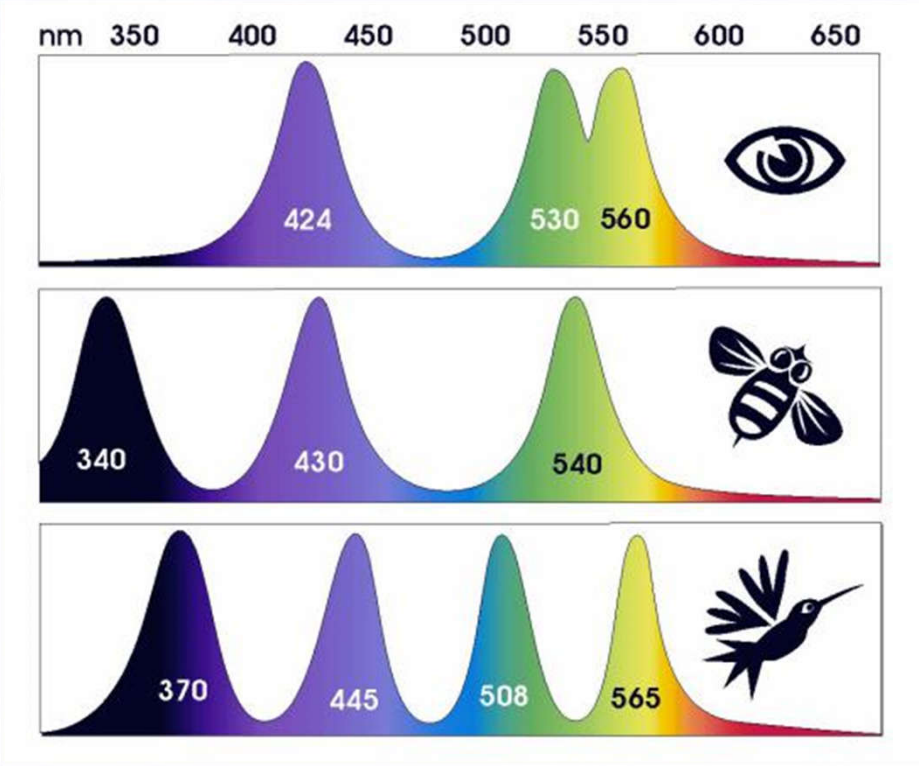
Aquatic mammals (whales, dolphins, seals, etc) and sharks (not mammals!) are believed to be monochromatic…makes sense, who needs colors deep in the water?

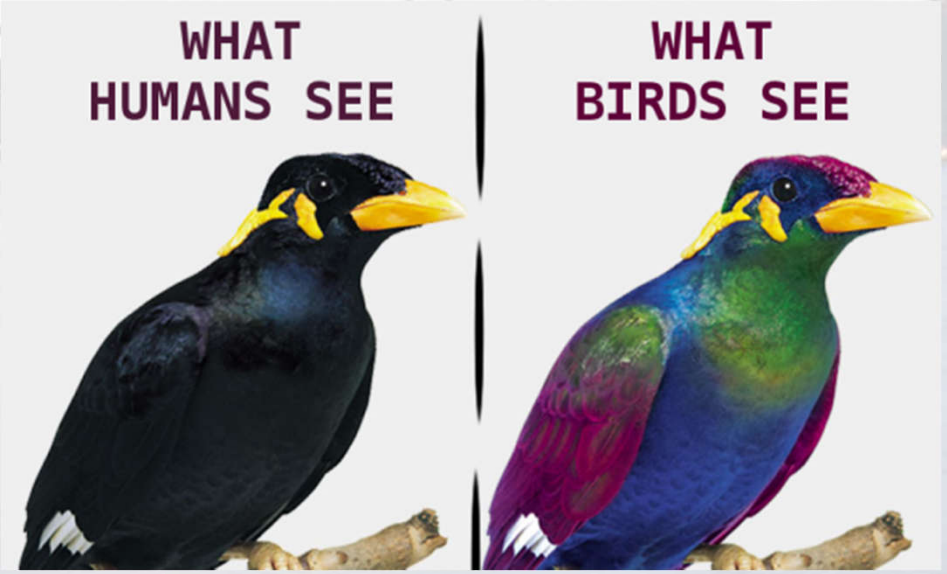
However sharks have an extremely well-developed capacity for picking up light; thus enabling them to see even in muddy waters.

Birds in general are tetra- or pentra-chromats; i.e. they have 4 or 5 types of photoreceptor cells.

A type of butterfly in Australia has at least 15.







The champion: the **mantis shrimp** has 16 photoreceptors.

Size: 10cm. Kills its prey by punching.

Its punch produces a force of 1500 Newtons, in less than 0.003 seconds.

Its punch is so powerful that the water around it **boils**, produces a **shockwave**, and even **light**!

The mantis shrimp can break aquarium glass.

Images from theoatmeal.com

What happens after the retina receives the light?

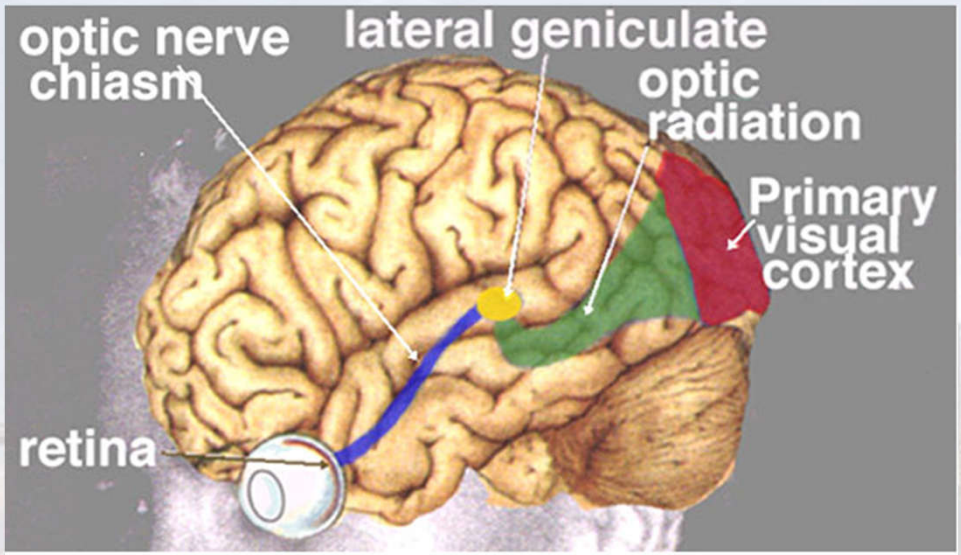
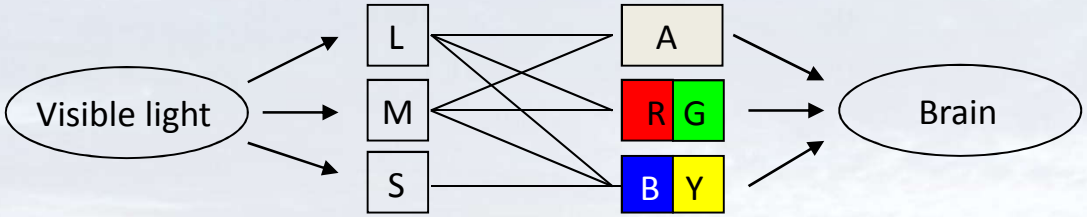


Fig. 3. The visual pathways from retina to visual cortex of the human brain.

We know a few things about the start of the path, but still have very little idea about how the visual cortex (occupies 20%-30% of the brain) operates.

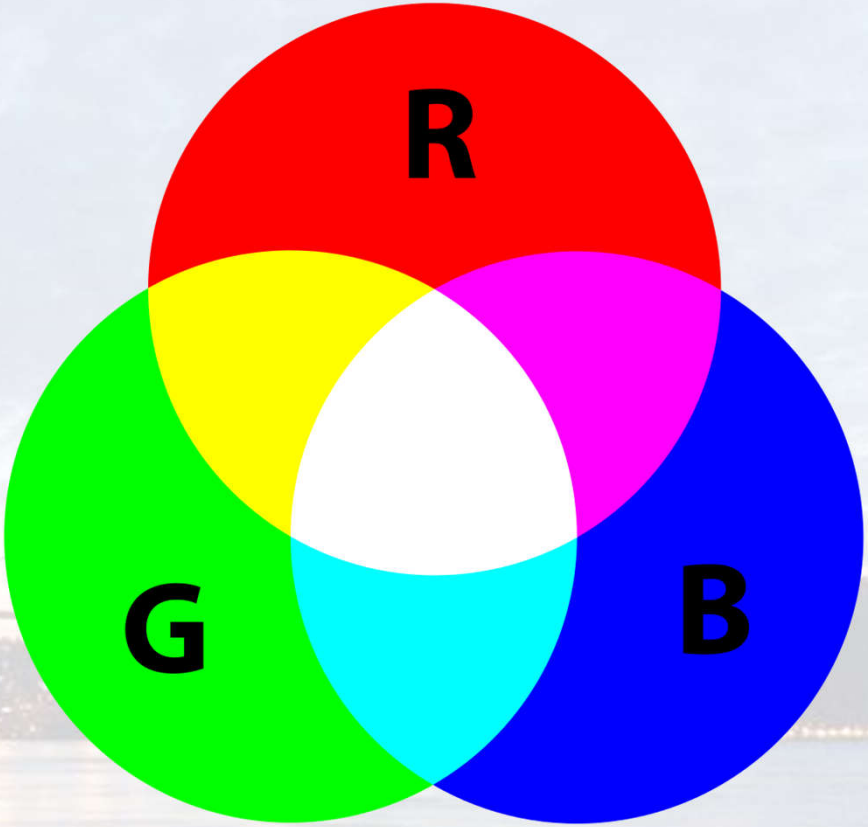
The modern color vision theory is a combination of trichromatic and opponent color theory. It is believed that the LMS responses that are collected from the retina, are "decorrelated" into one achromatic signal, and two opponent color signals.



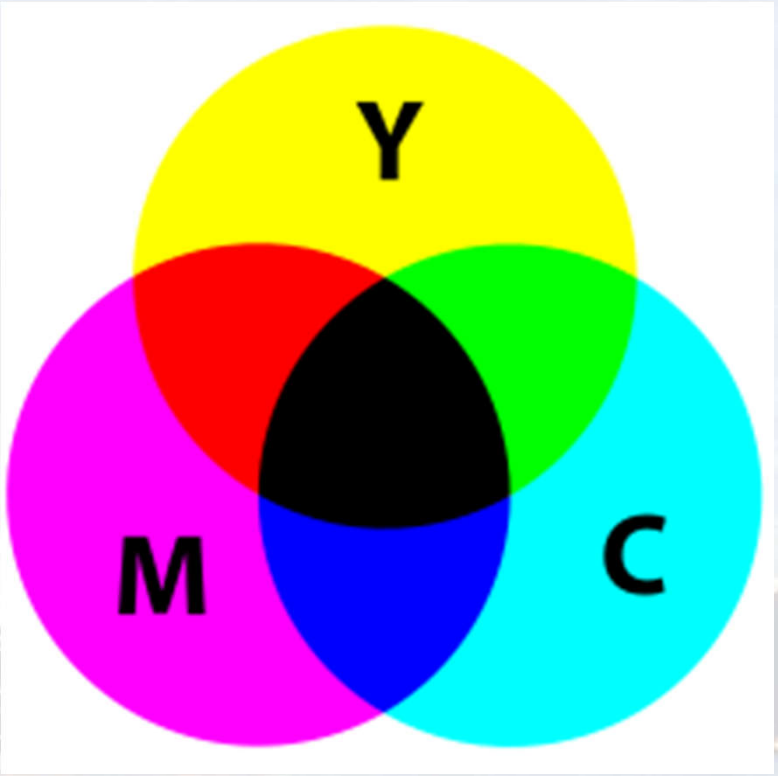
Researchers also *suspect* that further along the road to the "vision center" those signals are converted into **luminance** (intensity), **saturation** (colorfulness) and **hue** (dominant wavelength).

Color mixing models: **additive**

Used by monitors and projectors.



Color mixing models: **subtractive**



Used by printers, and with objects that reflect light.

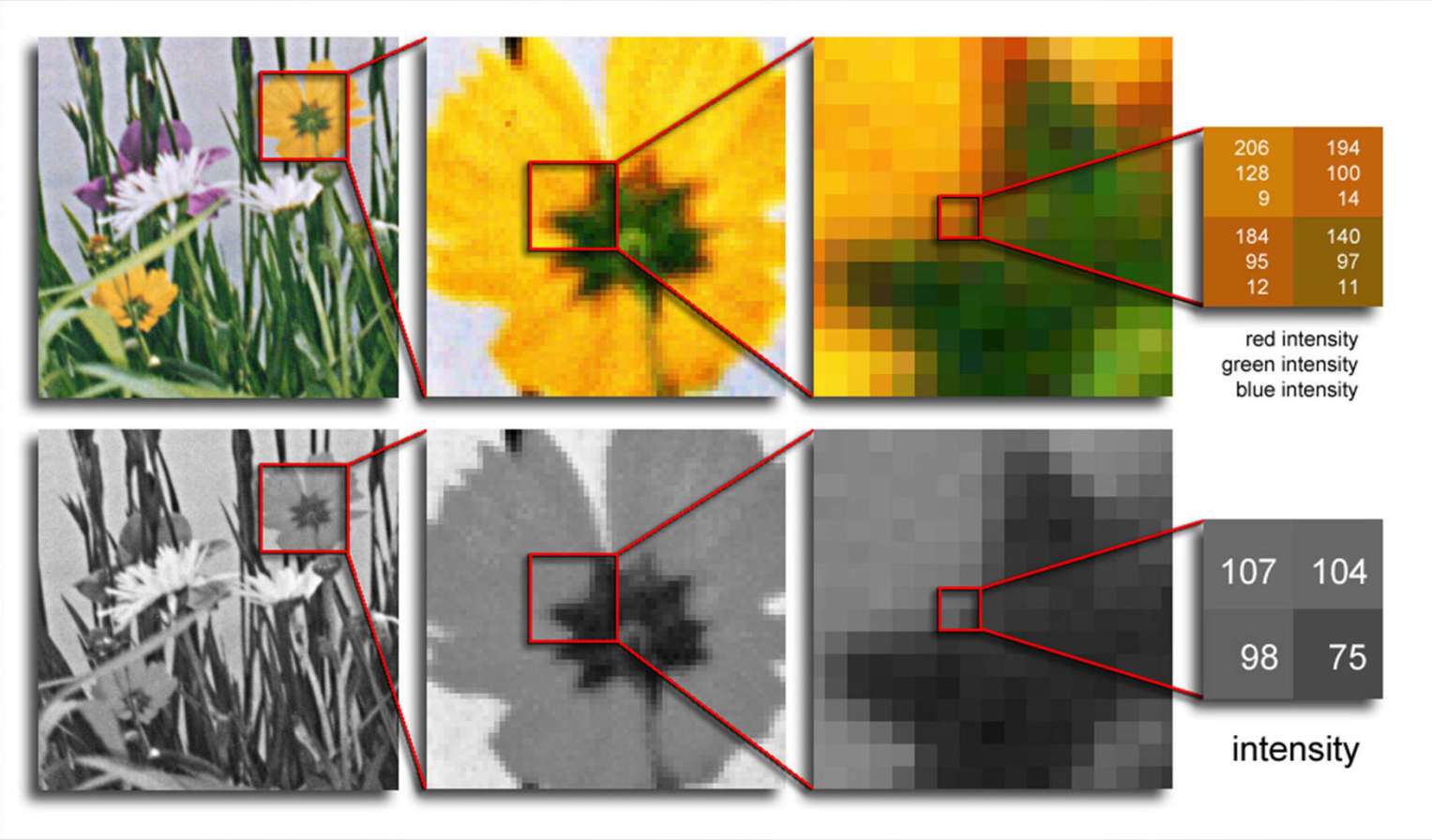
Why not use red ink? Because…

Red ink: absorbs blue and green.
Green ink: absorbs red and blue.
Red+Green: absorbs red, green and blue: black?!

Cyan: absorbs red, reflects blue and green
Yellow: absorbs blue, reflects green and red
Cyan+Yellow: absorbs red and blue, reflects green!

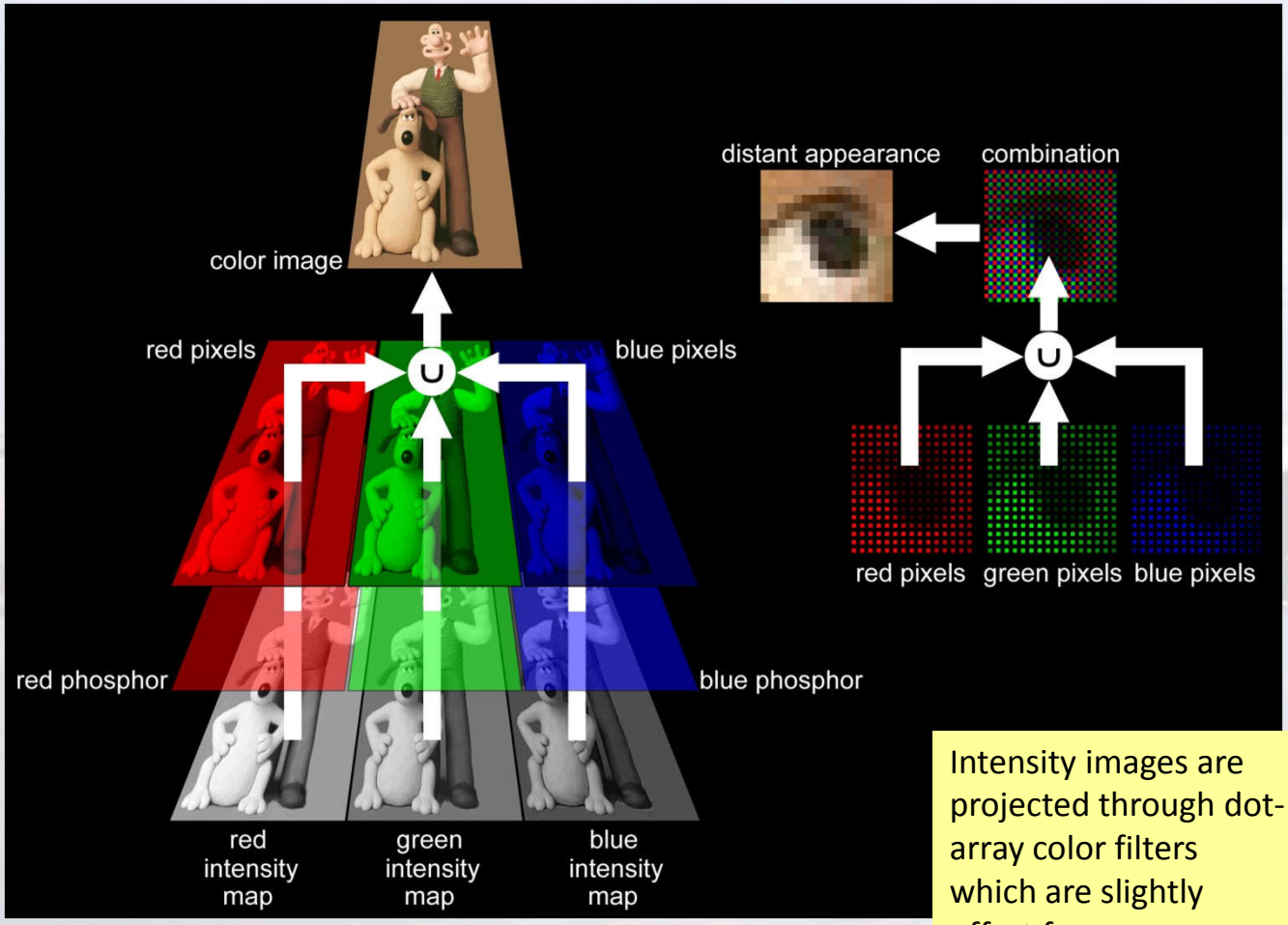
Color image processing

Digital color images are shown on monitors with additive mixing, but printed using subtractive mixing. They are made up of three matrices; one for each main color.



Each of the matrices is called a channel or band. Remote sensing images have hundreds of bands.

Digital color images shown on CRT, LCD monitors

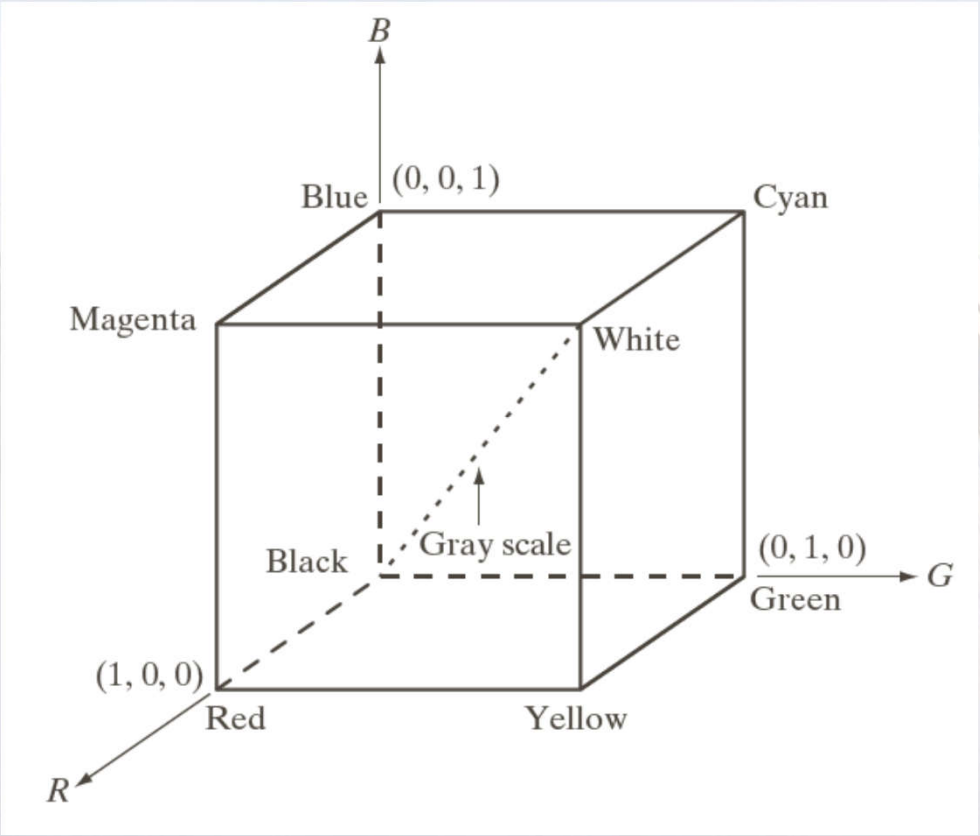


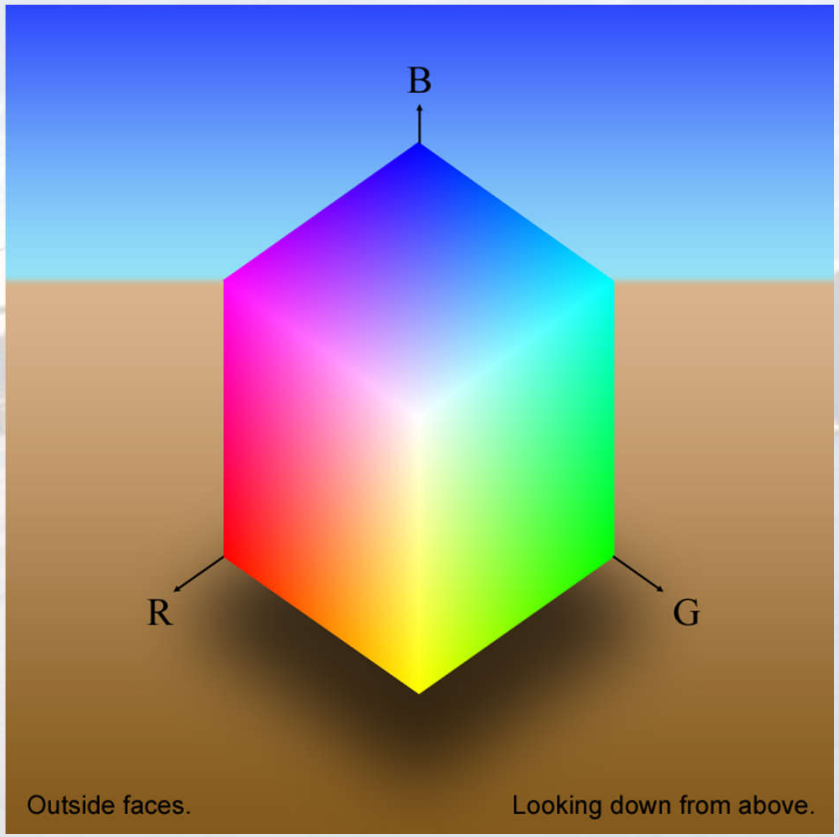
Intensity images are projected through dot-array color filters which are slightly offset from one another.

Color spaces: **RGB**

Fine for displaying color, unsuitable for image processing/analysis.

Pixel depth refers to the total number of bits used for representing a single color pixel; e.g. 24 bits is 8 bits for red+8 bits for green+8 bits for blue.





The **CMYK** color space (Cyan Magenta Yellow Black); suitable for printing, we won't see it again.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The **CIE 1931 color space: XYZ**

The International Illumination Commission published this color space in order to standardize the relationships between the physical aspect (the wavelengths) of color and its physiological perception.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} .490 & .310 & .200 \\ .177 & .813 & .011 \\ .000 & .010 & .990 \end{pmatrix} \begin{pmatrix} R_\lambda \\ G_\lambda \\ B_\lambda \end{pmatrix}$$

$$x = \frac{X}{X+Y+Z} \qquad y = \frac{Y}{X+Y+Z} \qquad z = \frac{Z}{X+Y+Z}$$

Since $x + y + z = 1$ you only need x, y in order to specify a color coordinate.

Color image processing

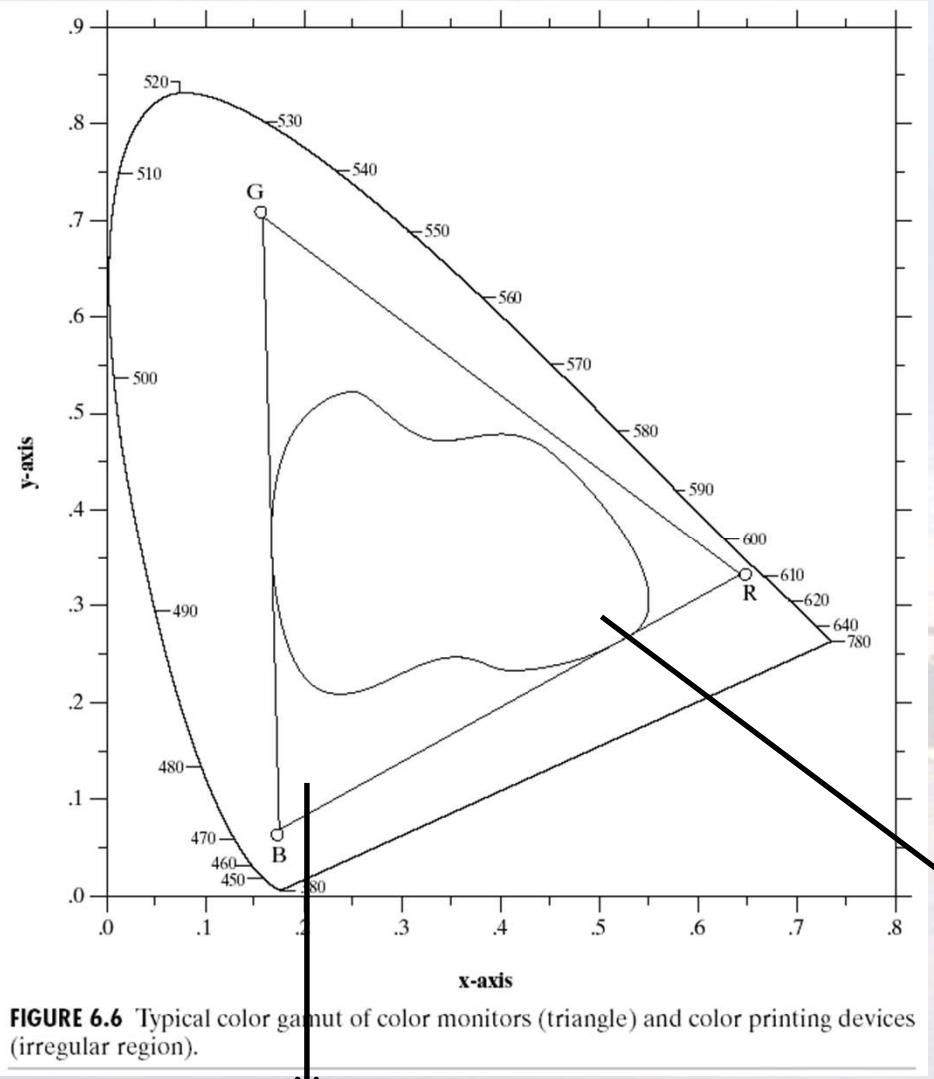
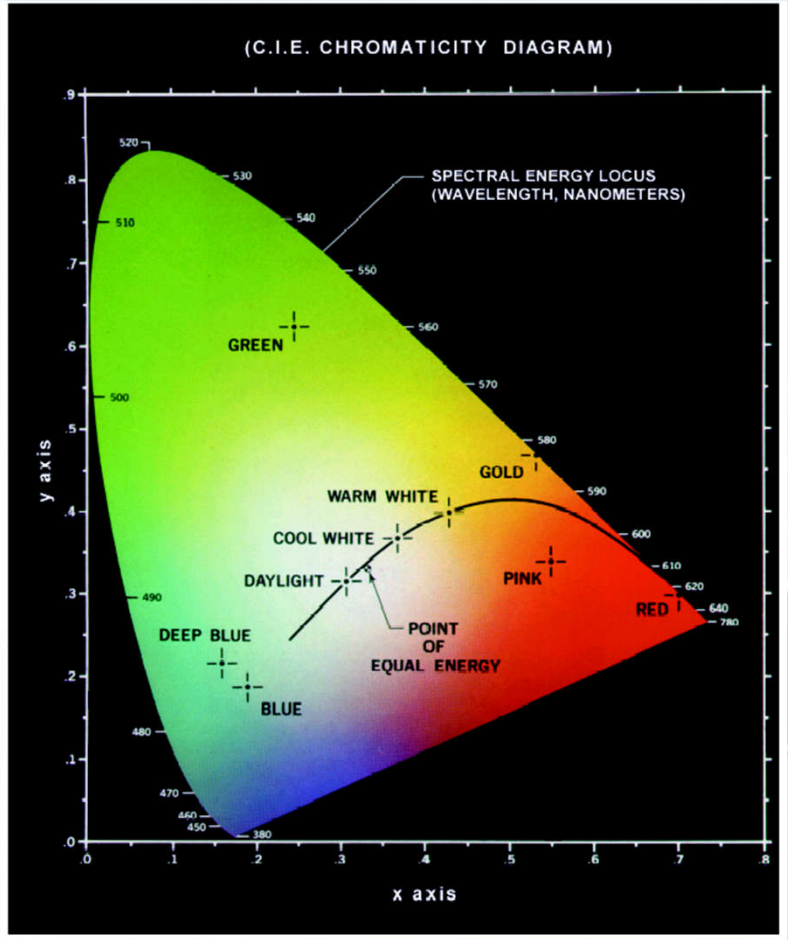


FIGURE 6.6 Typical color gamut of color monitors (triangle) and color printing devices (irregular region).



Color gamut for printers

Color gamut for monitors

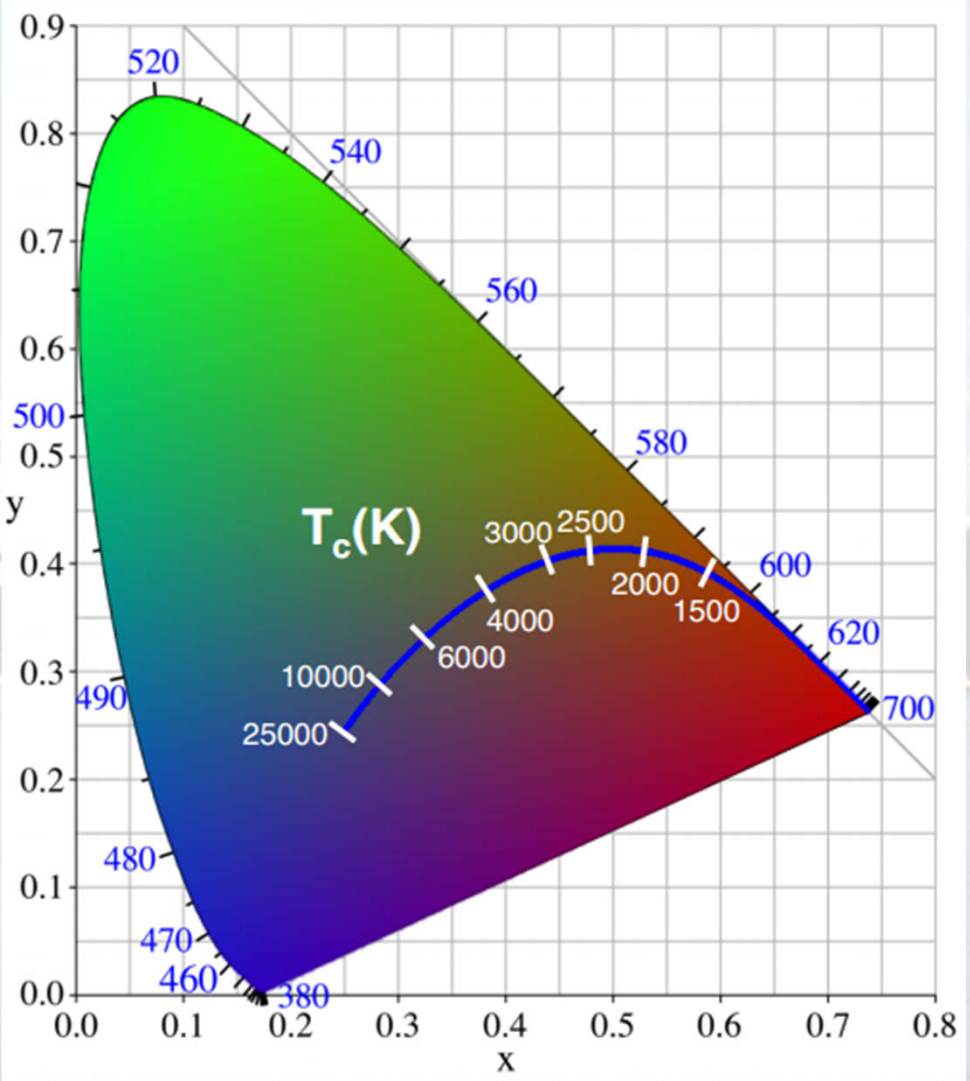
Pure colors are at the edges

White balance and color temperature

The “white” light coming from a light source is not perfectly white.
It has a color; yellowish, bluish, etc.

And its color affects the color of the objects that you perceive through it.

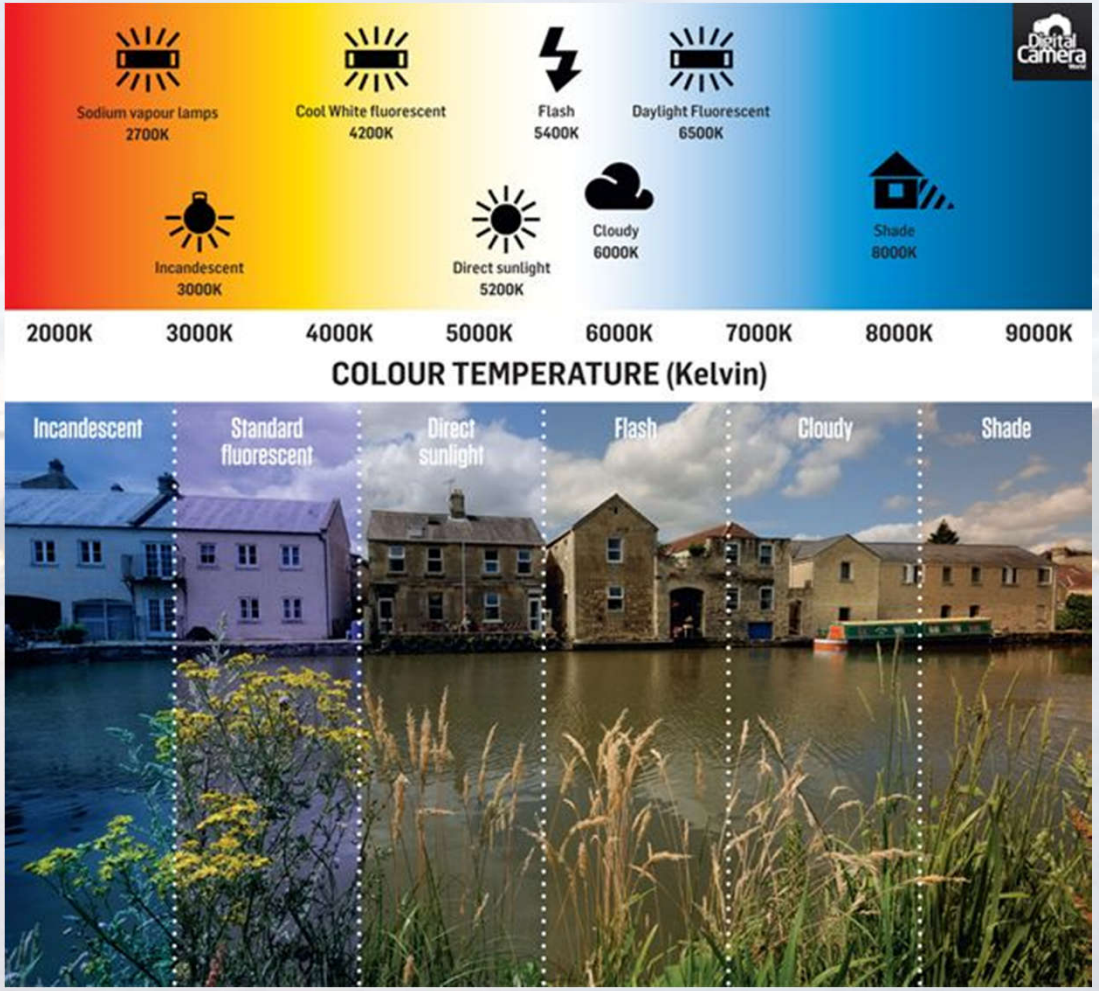
That’s why cameras have settings enabling you to change the reference white, so that they can compensate for that effect.



There are many “whites”: characterized by their temperature (in Kelvin), such as

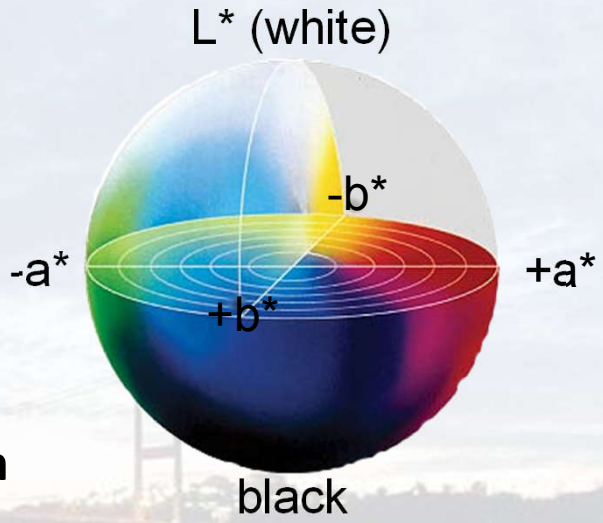
Daylight 6500K

Incandescent 2400K, etc



The CIELAB or L*a*b* color space

It was designed in order to address two issues:



1. **Device independence**: we want the colors that we record with our camera to appear correctly at the displays. The reference white information is needed.

2. **Perceptual uniformity**: we want relative **Euclidean** distances in CIELAB to correspond to perceptual color differences. It helps calculations.

You need to pass through XYZ to arrive to CIELAB.

Its conversion is non-linear and expensive.

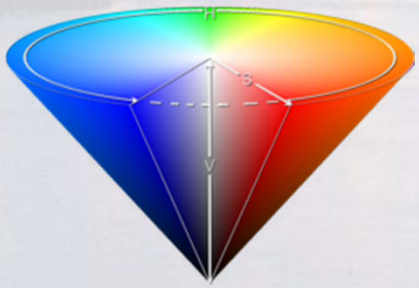
Phenomenal color spaces: HSV, HLS, HSI, HSY, etc.

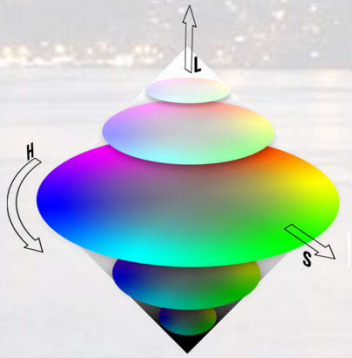
Despite different formulations these color spaces aim to represent color intuitively, in terms of:

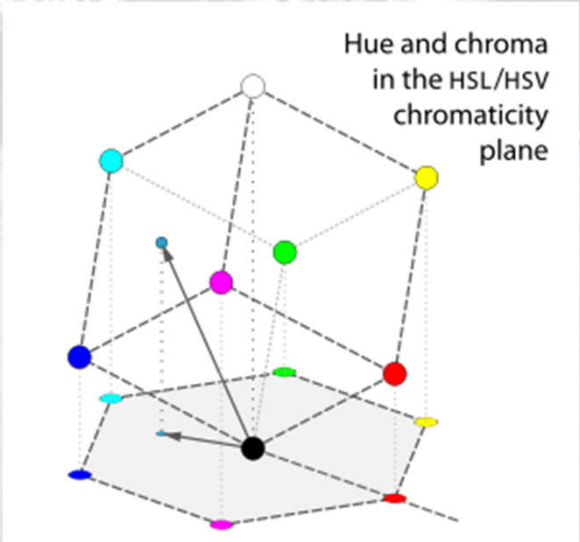
Luminance: the achromatic notion of intensity,

Saturation: relative purity of the color, determined by the amount of white

Hue: the dominant wavelength of a color, represented by an angle

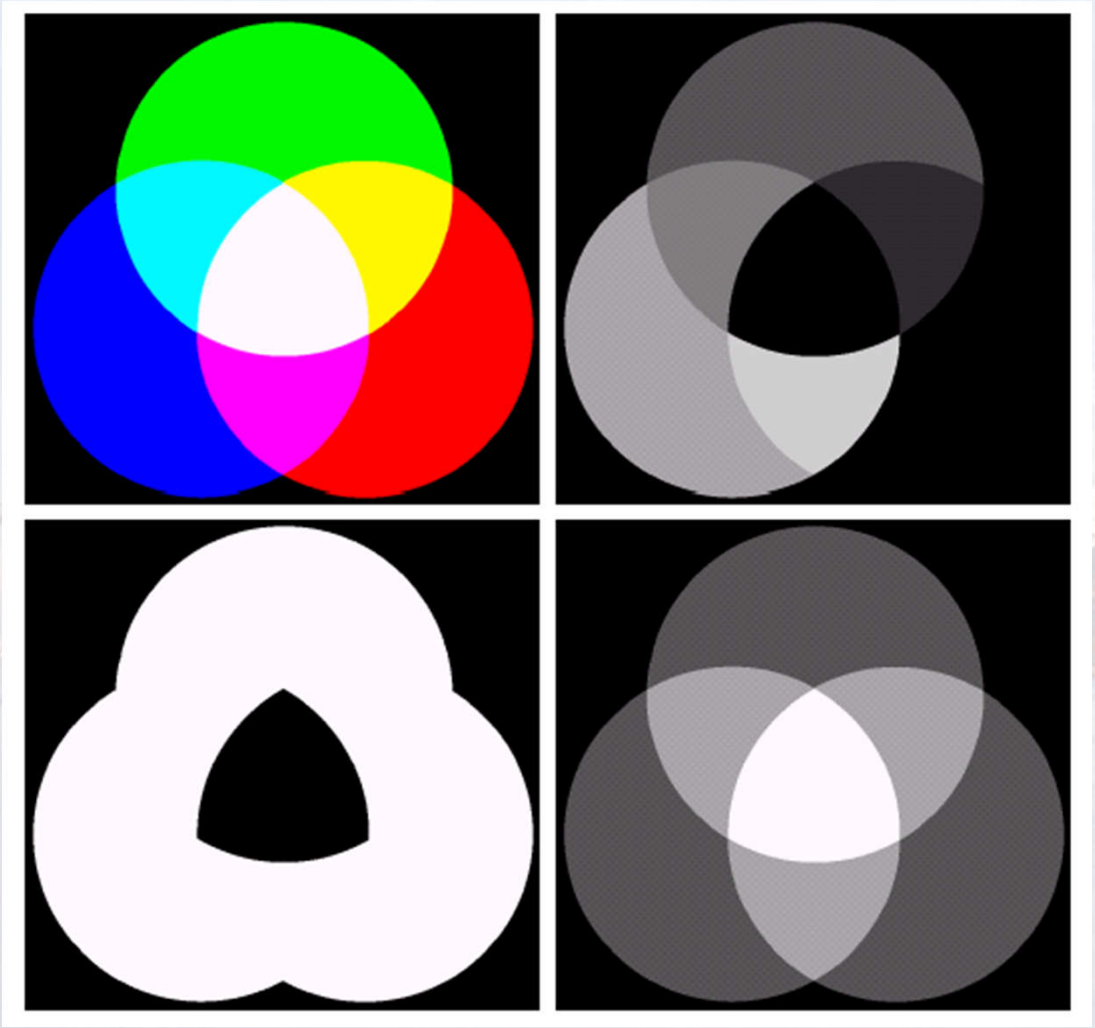






Color image processing

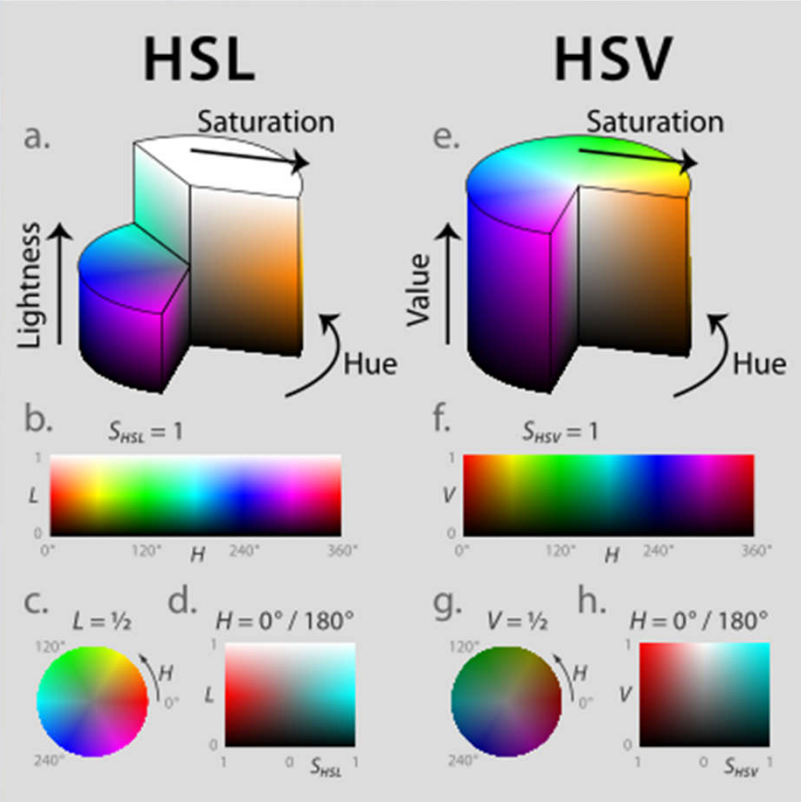
R,G,B

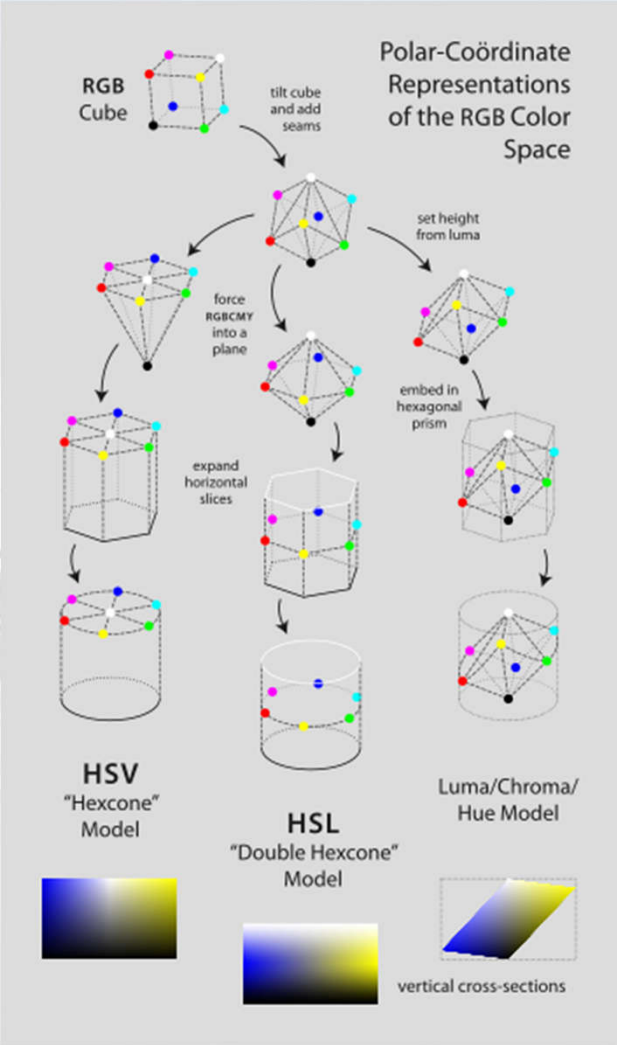


hue

saturation

intensity


HSL
HSV
a.
Saturation
Lightness
Hue
e.
Saturation
Value
Hue
b.
$S_{HSL} = 1$
L
H
f.
$S_{HSV} = 1$
V
H
c.
$L = ½$
H
d.
$H = 0° / 180°$
L
S_{HSL}
g.
$V = ½$
H
h.
$H = 0° / 180°$
V
S_{HSV}

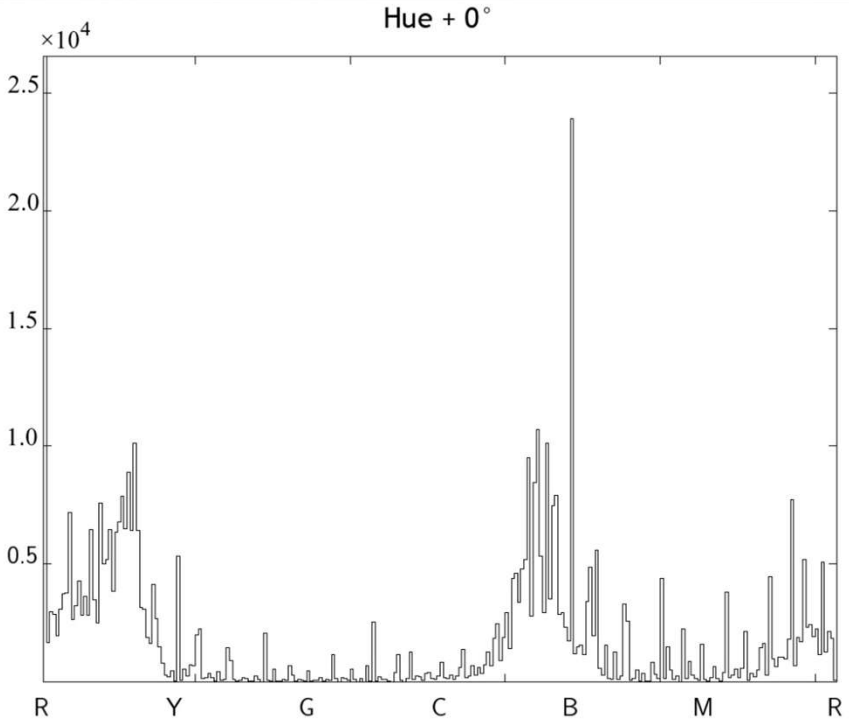

RGB Cube
tilt cube and add seams
Polar-Coördinate Representations of the RGB Color Space
set height from luma
force RGBCMY into a plane
embed in hexagonal prism
expand horizontal slices
HSV
"Hexcone" Model
HSL
"Double Hexcone" Model
Luma/Chroma/ Hue Model
vertical cross-sections

original

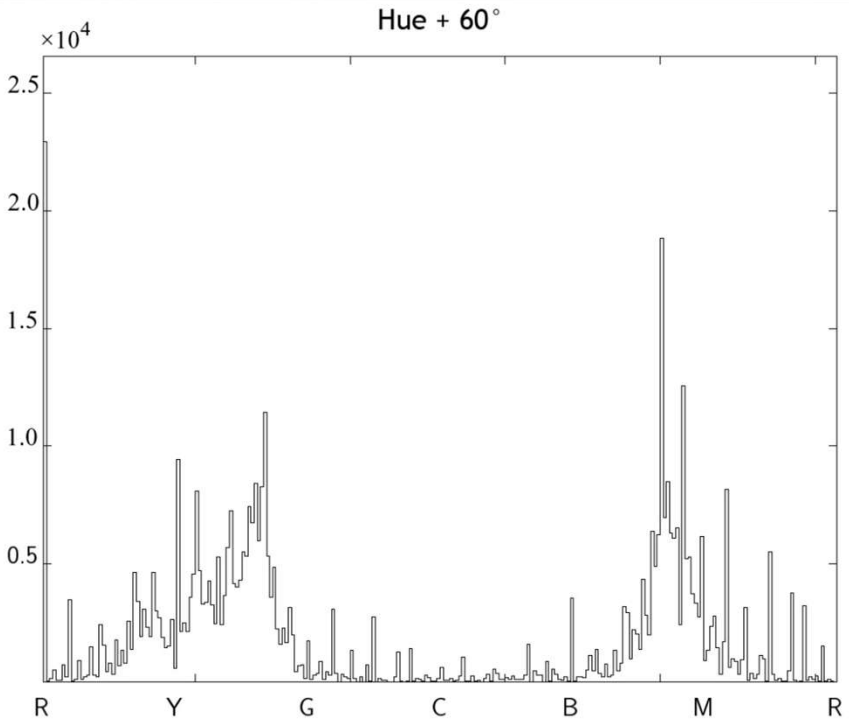
saturation + 50%

saturation - 50%

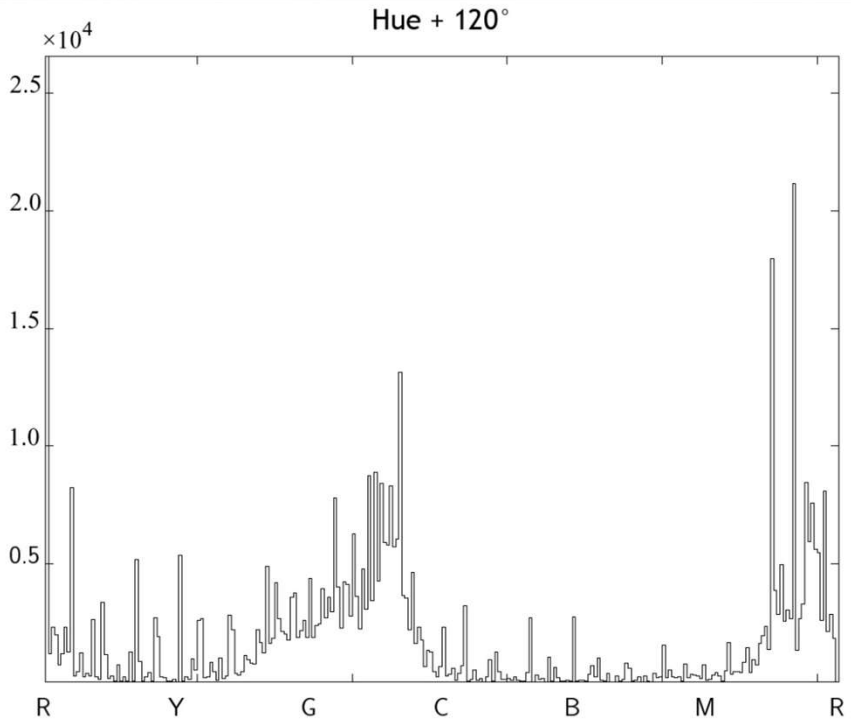
original



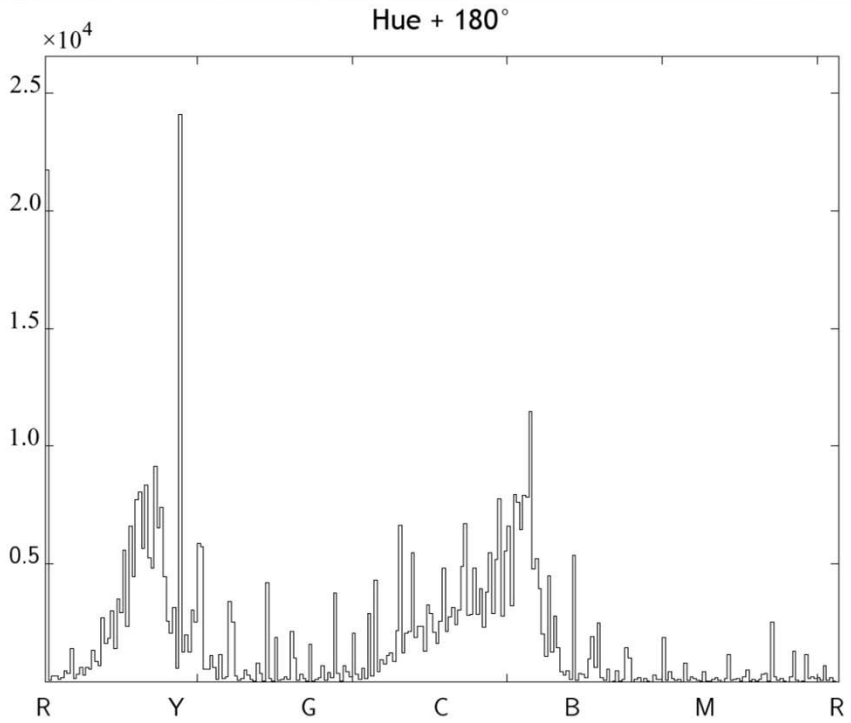
hue + 60°



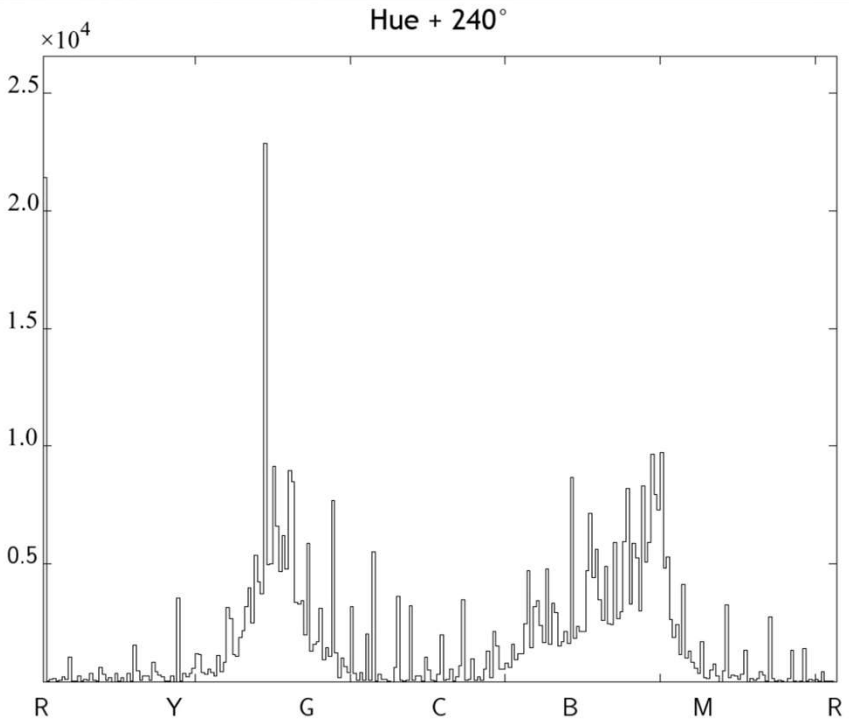
hue + 120°



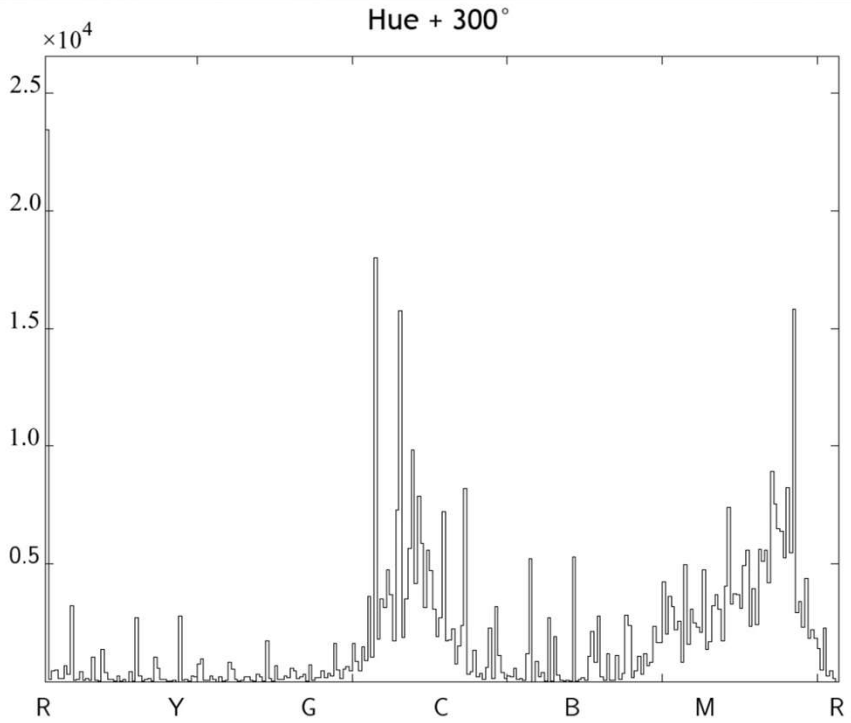
hue + 180°



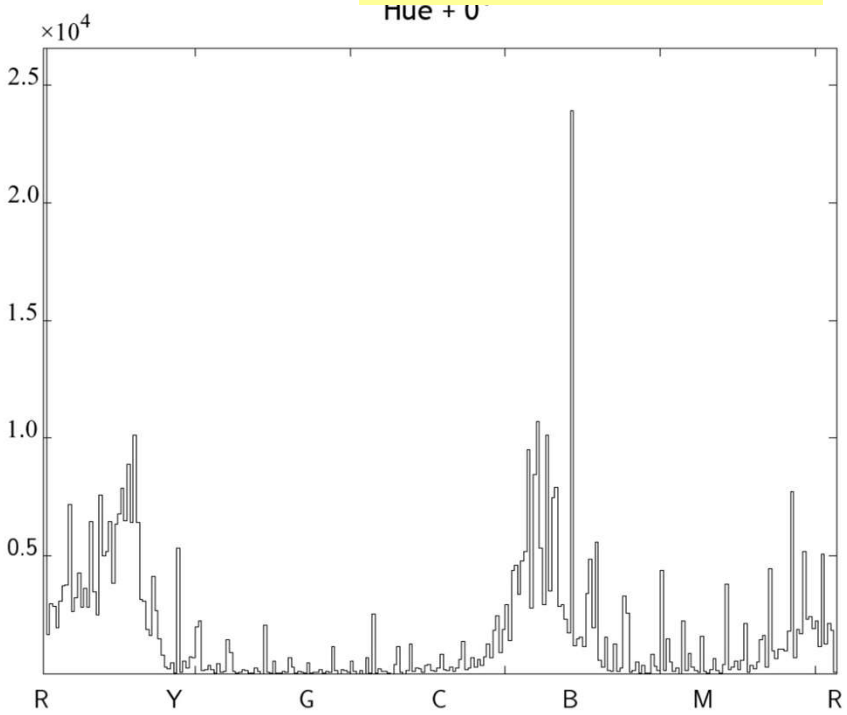
hue + 240°



hue + 300°



hue + 360° = original



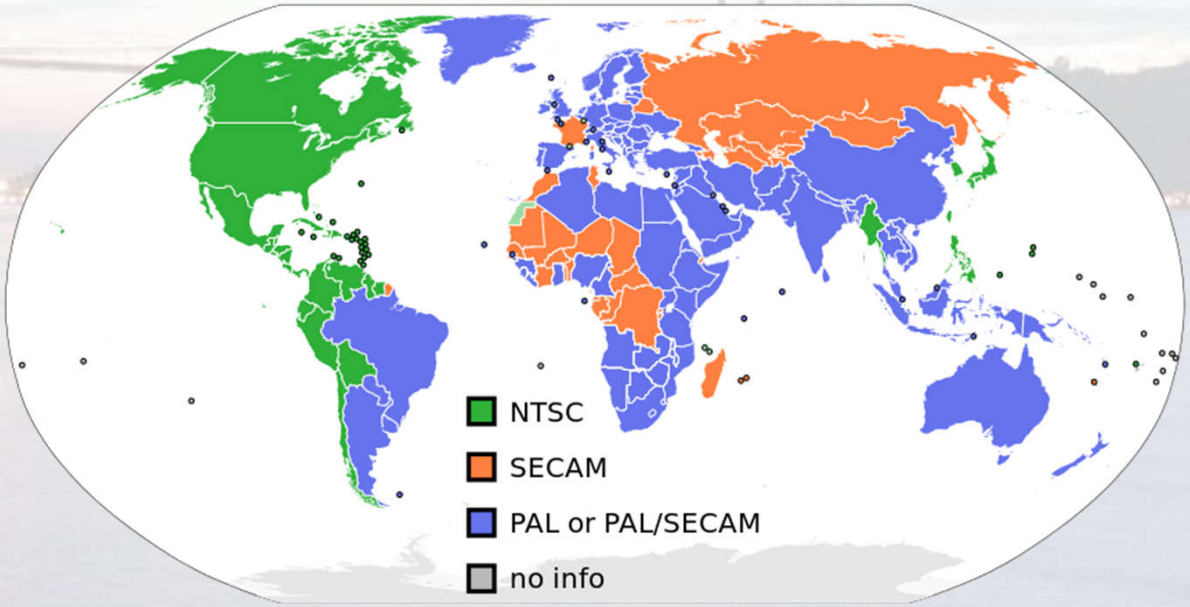
Other color spaces

YIQ: for NTSC color TV

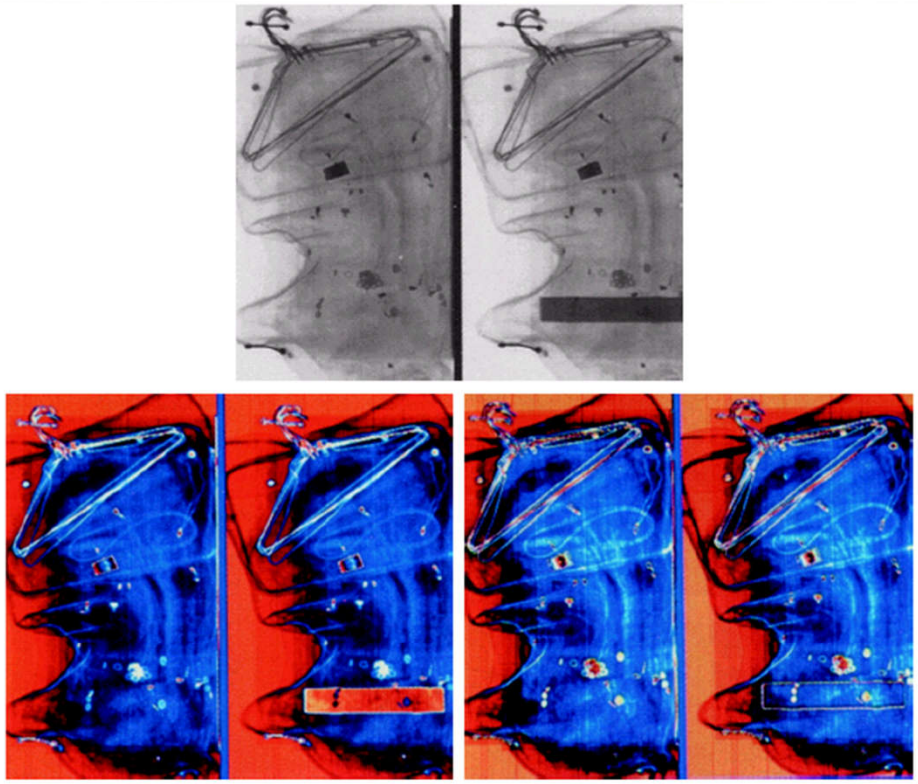
YUV: for PAL color TV, digital video encoding; Jpeg, Mpeg compression.

CIELUV: for computer graphics

etc.

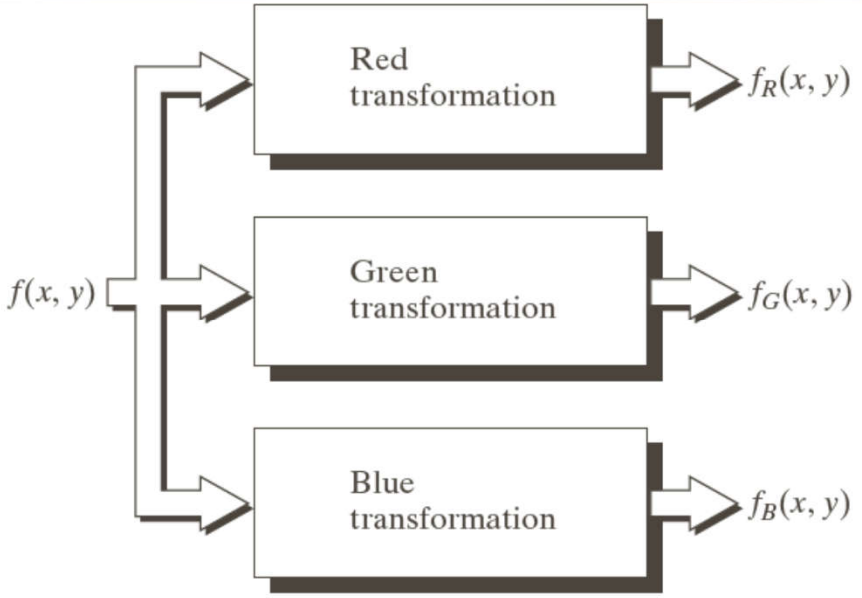


Pseudocolors: intensity to color transformation

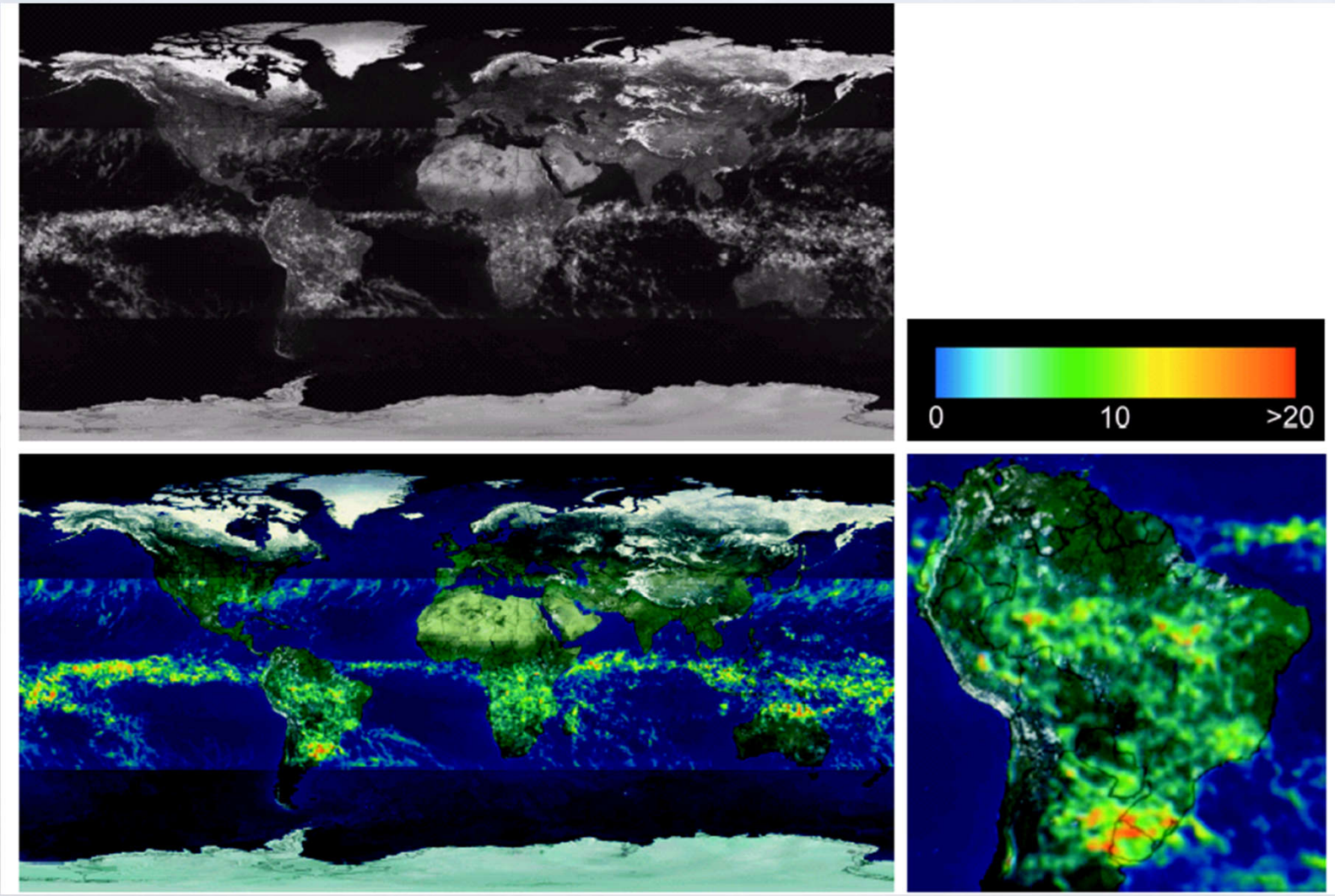


a
b c

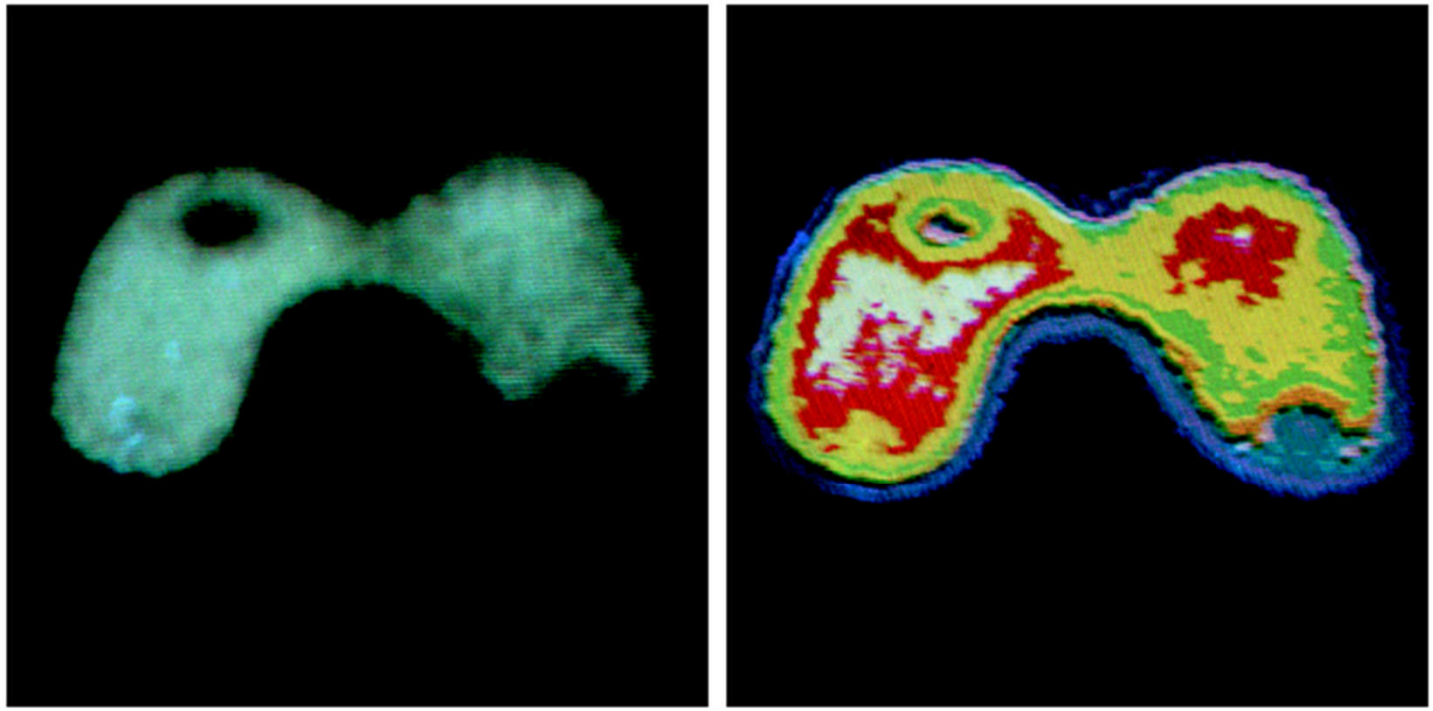
FIGURE 6.24 Pseudocolor enhancement by using the gray-level to color transformations in Fig. 6.25. (Original image courtesy of Dr. Mike Hurwitz, Westinghouse.)



Examples of pseudocolors



Example of pseudocolors



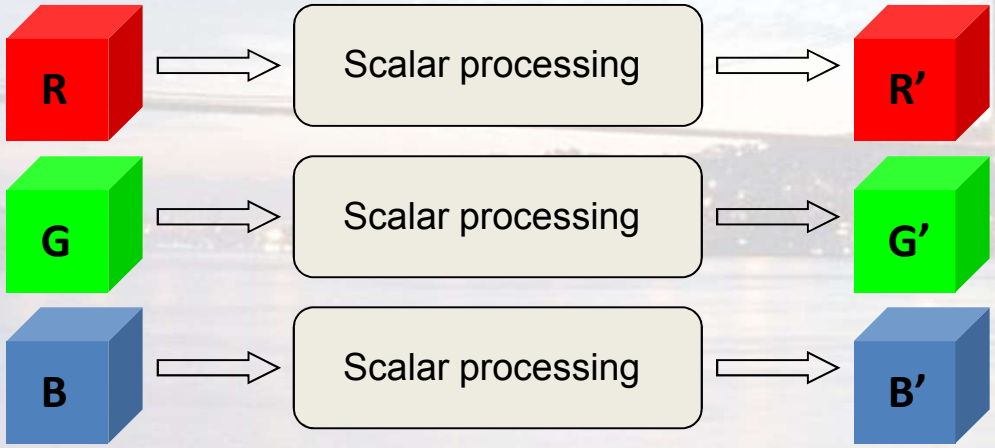
a b

FIGURE 6.20 (a) Monochrome image of the Picker Thyroid Phantom. (b) Result of density slicing into eight colors. (Courtesy of Dr. J. L. Blankenship, Instrumentation and Controls Division, Oak Ridge National Laboratory.)

How to process color images

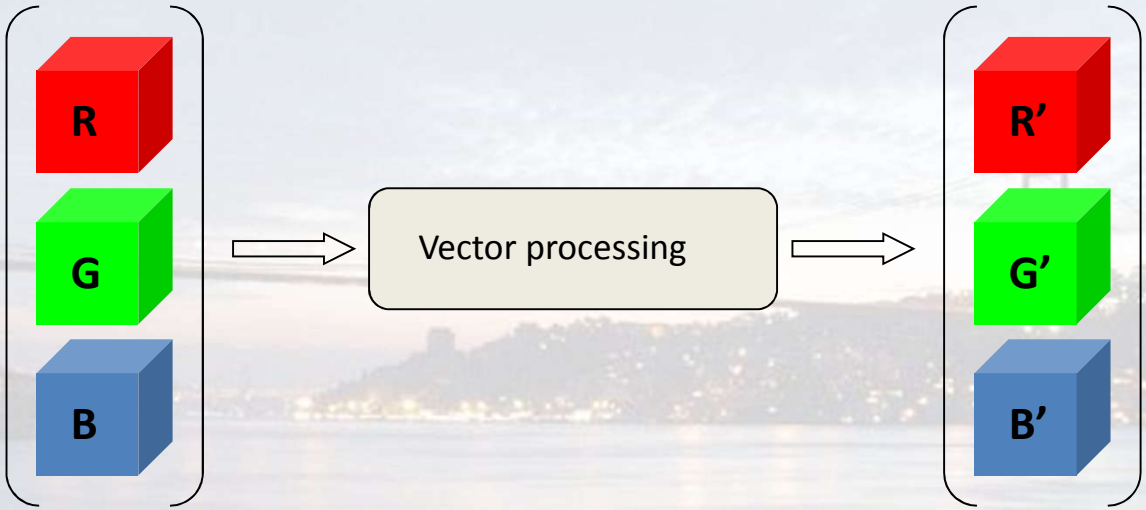
Basically there are 2 ways: scalar and vector processing

Scalar (marginal) processing: process each color independently and stack the results. Easy to implement, but ignores any correlation among channels.



Vector processing: process/manipulate every color pixel value as a 3D vector.

Can take into account any correlation available, but requires adapting existing filters.



Linear operators are often applied marginally.

Non-linear filters may go both ways.

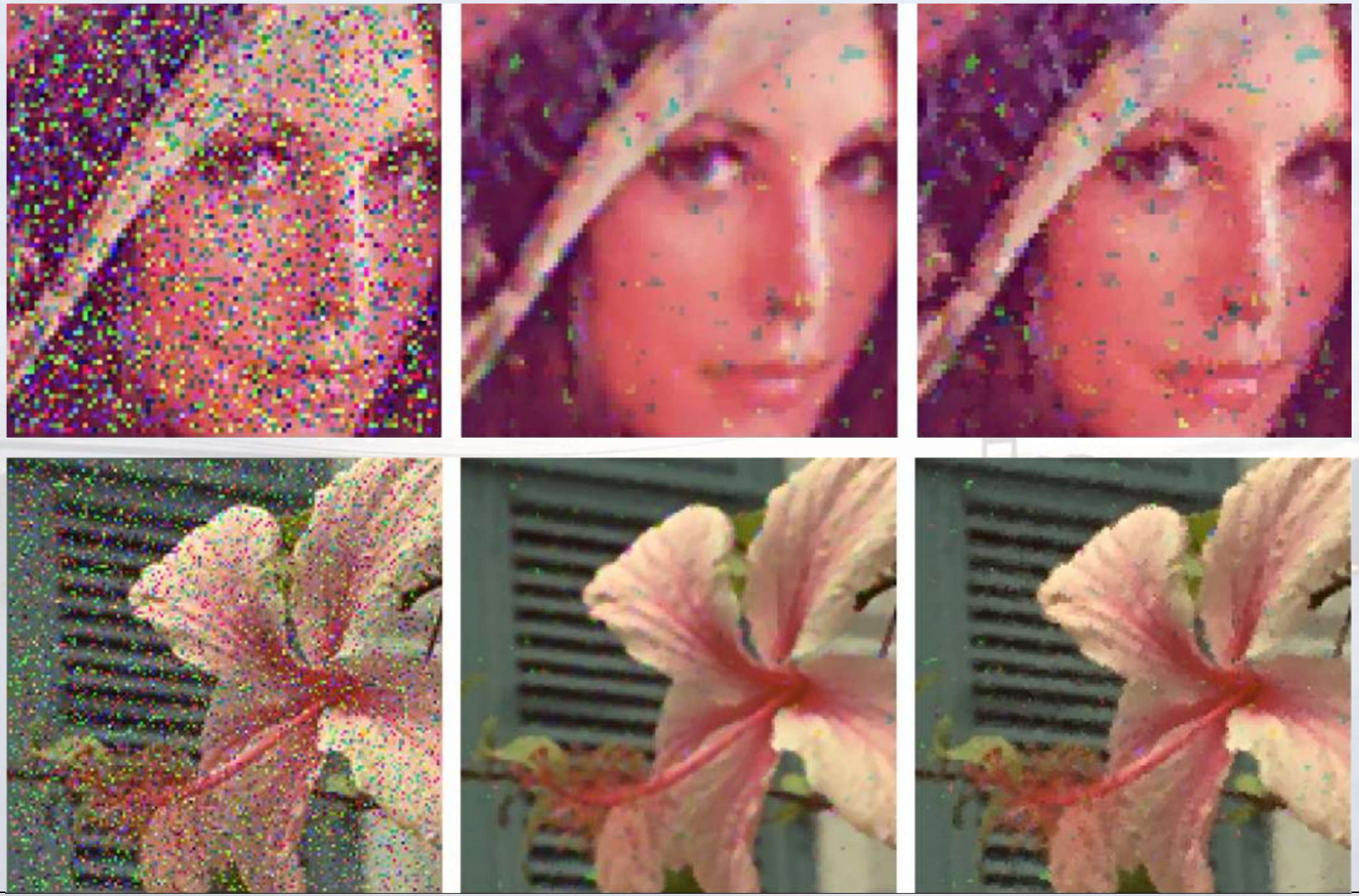
Example: the median filter

Given a color image its marginal implementation is straightforward. As to its vector implementation, it's just a question of being able to sort N 3D vectors; and not only sort, but do so in such way that extreme (impulse noise) pixel values will either be the least or greatest elements after sorting.

One way of achieving this is by associating every vector with its cumulative distance to the other N-1 vectors.

Original-Noisy Marginal Vector

Color image processing



Color mathematical morphology

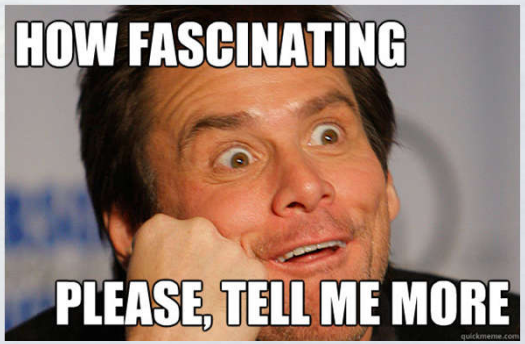
MM can be applied to any type of image data as long as its theoretical requirements are met: the imposition of a complete lattice structure on the range of pixel values.

Binary 2D discrete images: $f: \mathbb{Z}^2 \rightarrow \{0,1\}$ ordered through set inclusion $\subseteq$

Grayscale 2D discrete images: $f: \mathbb{Z}^2 \rightarrow \mathbb{Z}$ ordered through scalar ordering $\leq$

Color 2D discrete images: $f: \mathbb{Z}^2 \rightarrow \mathbb{Z}^3$ ordered through ????

So basically, if we can come up with a (at least partial) ordering for 3 dimensions, we can immediately start using the entire MM arsenal with a vector strategy.



There are tens of such ordering relations.

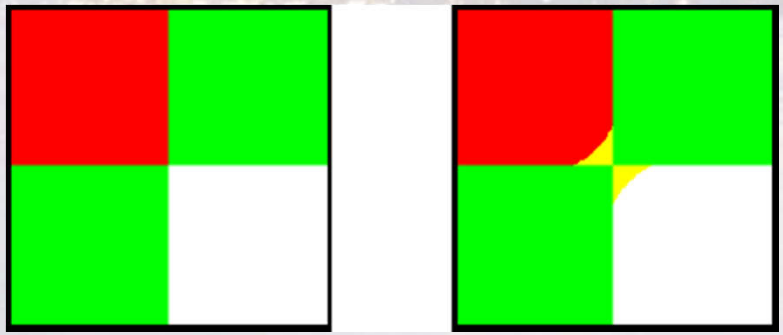
The **marginal ordering** relation compares every channel independently.

$$\forall \mathbf{v}, \mathbf{v}' \in \mathbb{R}^n, \quad \mathbf{v} \leqslant \mathbf{v}' \;\Leftrightarrow\; \forall i \in \{1, \ldots, n\}, \quad v_i \leqslant v'_i$$

It is a partial ordering, since there are incomparable vectors: $[1,2,3]^T$, $[3,5,1]^T$

This amounts to processing every channel independently.

It is fast, but it can introduce new colors into the output:



Marginal dilation

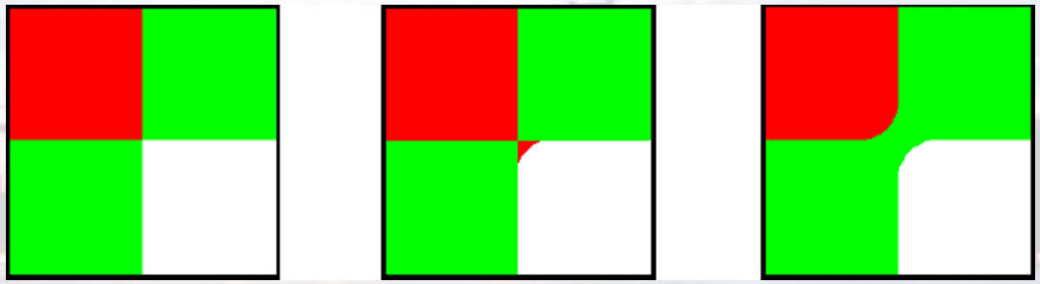
An alternative is the **lexicographical ordering**:

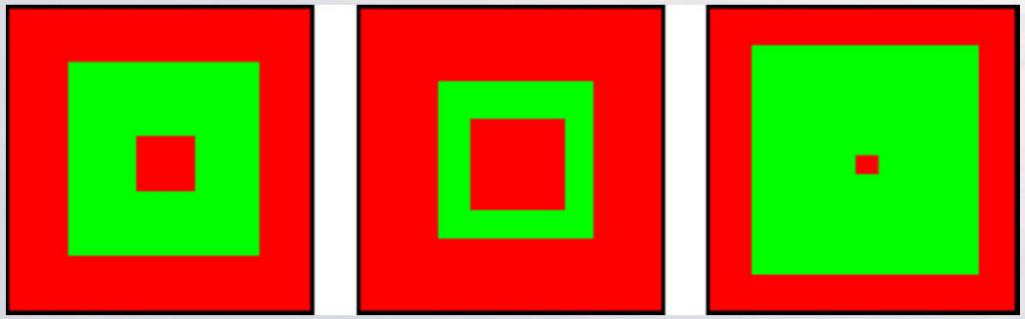
$$\forall \mathbf{v}, \mathbf{v}' \in \mathbb{R}^n, \quad \mathbf{v} \leqslant_L \mathbf{v}' \quad \Leftrightarrow \quad \exists i \in \{1, \ldots, n\},$$
$$(\forall j < i, \ v_j = v'_j) \wedge (v_i \leqslant v'_i).$$

It is a total ordering relation; i.e. any two vectors can be compared with it.

It preserves the colors of the input.

Vector dilation with lex (R,G,B) and lex (G,R,B)

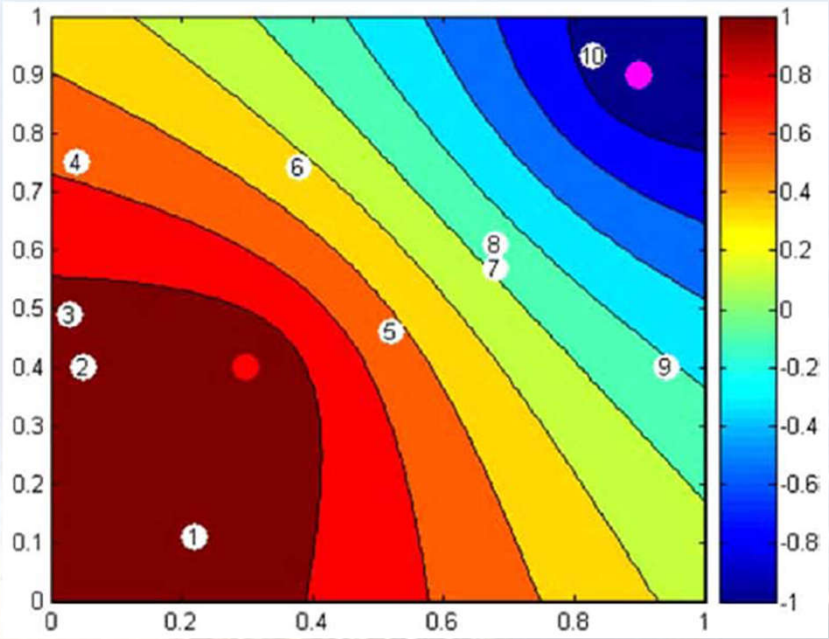




Extremely asymmetric

Most comparisons are decided at the first dimension, the rest go to waste.

There are also **supervised orderings**, where the expert selects the least and greatest colors and we use machine learning to determine the order in-between them.



Or you can select an application and a filter and try an intelligent search that optimizes the filter's performance with respect to the chosen application. It's still an open problem, new ideas are welcome.

Given a 24-bit color image, there are exactly $(2^{24})!$ different ordering relations.

And now that we have seen color images, let's take a look into **image compression**.

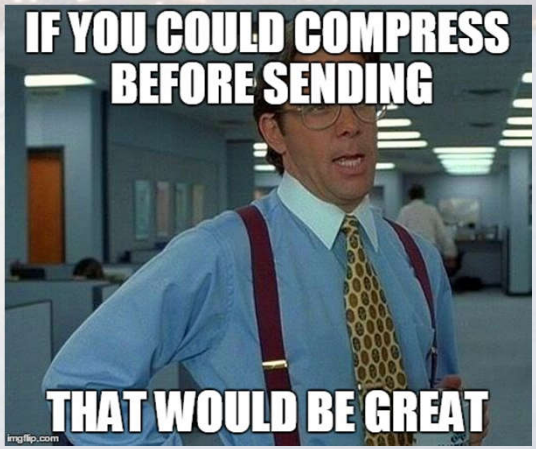
Visual data are voluminous.

An uncompressed 24-bit color photograph with a 12MPixel camera is

$$3 \times 12 \times 10^6 \approx 36\text{MB}$$

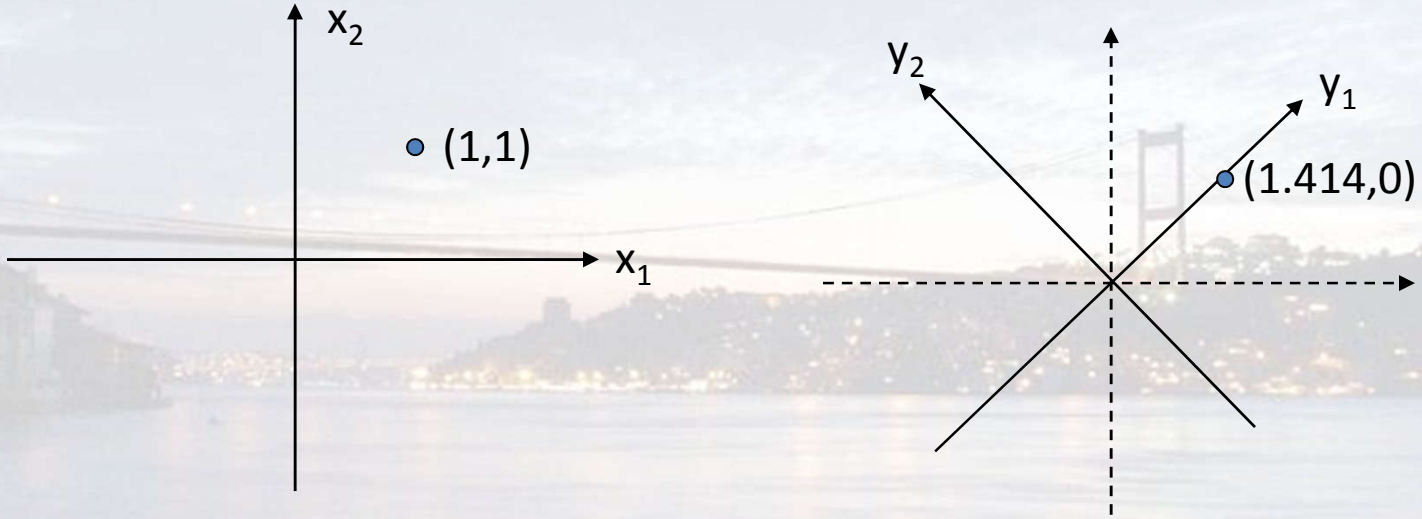
An uncompressed 1 minute HD (1920x1080) 24-bit color video at 24 frames per second would require $60 \times 24 \times 3 \times 1920 \times 1080 \approx 8.9\text{GB}$

Compression is mandatory.



In order to better understand the concepts that follow, we need to take a short step back and recall some notions:

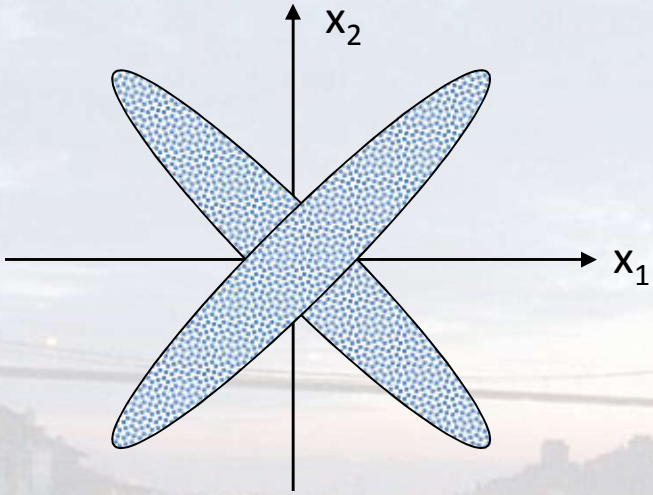
A transform is a change of coordinates. Example of a 1D Transform with **2 variables**:

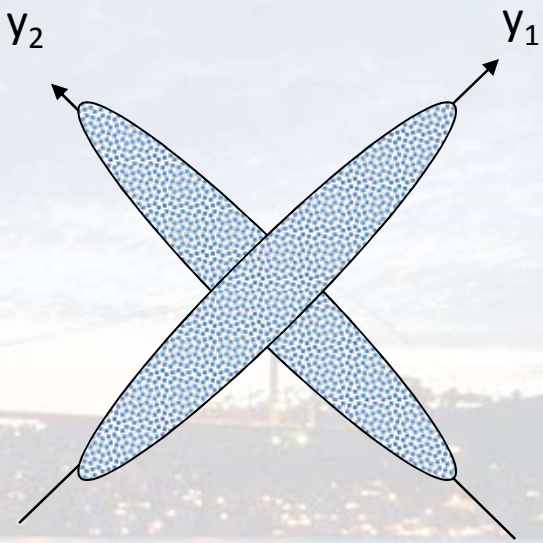


$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow \vec{y} = \mathbf{A}\vec{x}, \mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$$

Transform matrix

De-correlating property of a transform





x_1 and x_2 are **highly correlated**

$p(x_1x_2) \neq p(x_1)p(x_2)$

y_1 and y_2 are **less correlated**

$p(y_1y_2) \approx p(y_1)p(y_2)$

Generalization to **N variables**

forward transform $\vec{y}_{N\times 1} = \mathbf{A}_{N\times N}\vec{x}_{N\times 1}$

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & \cdots & a_{1N} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{N1} & \cdots & \cdots & a_{NN} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\vec{y} = \sum_{i=1}^{N} x_i \vec{b}_i, \vec{b}_i = [a_{i1}, \ldots, a_{iN}]^T$$

basis vectors (column vectors of transform matrix)

A unitary matrix and a unitary 1D transform

A matrix *A* is called **unitary** if $A^{-1} = A^{*T}$ (conjugate, transpose)

When the transform matrix A is unitary, the defined transform $\vec{y} = A\vec{x}$ is called a **unitary transform**

$$\mathbf{A} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \mathbf{A}^{-1} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & -1 \\ 1 & 1 \end{bmatrix} = \mathbf{A}^{T}$$

A real matrix *A* is unitary if $A^{-1} = A^{T}$

A matrix that is both **real** and **unitary** is called **orthogonal**

The inverse of a unitary transform

For a unitary transform, its inverse is defined by

$$\vec{x} = \mathbf{A}^{-1}\vec{y} = \mathbf{A}^{*T}\vec{y}$$

Inverse Transform

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} a_{11}^* & \cdots & \cdots & a_{N1}^* \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{1N}^* & \cdots & \cdots & a_{NN}^* \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}$$

$$\Updownarrow$$

$$\vec{x} = \sum_{i=1}^{N} y_i \vec{b}_i, \vec{b}_i = [a_{1i}^*, \ldots, a_{Ni}^*]^T$$

$\vec{b}_i$: basis vectors corresponding to inverse transform

Properties of a unitary transform

- Energy compaction: only a few coefficients have large magnitude (related to its decorrelating role)
- Energy preservation: unitary transforms preserve the 2-norm of input vectors; in other words, rotating coordinates does not affect Euclidean distances.

$$\mathbf{A} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \vec{x} = \begin{bmatrix} 100 \\ 98 \\ 98 \\ 100 \end{bmatrix}$$

Hadamard matrix

$$\vec{y} = \mathbf{A}\vec{x} = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} 100 \\ 98 \\ 98 \\ 100 \end{bmatrix} = \begin{bmatrix} 198 \\ 0 \\ 0 \\ 2 \end{bmatrix}$$

significant

insignificant

Energy preservation

$$\vec{y} = \mathbf{A}\vec{x} \quad \text{A is unitary} \Rightarrow \| \vec{y} \|^2 = \| \vec{x} \|^2$$

Proof

$$\| \vec{y} \|^2 = \sum_{i=1}^{N} | y_i |^2 = \vec{y}^{*T} \vec{y} = (\mathbf{A}\vec{x})^{*T} (\mathbf{A}\vec{x})$$

$$= \vec{x}^{*T} (\mathbf{A}^{*T} \mathbf{A}) \vec{x} = \vec{x}^{*T} \vec{x} = \sum_{i=1}^{N} | x_i |^2 = \| \vec{x} \|^2$$

In other words quantization can be performed directly on coefficients.

From 1D to 2D: do N 1D transformations

$$\mathbf{Y}_{N\times N} = \mathbf{A}_{N\times N}\mathbf{X}_{N\times N}$$

$$\begin{bmatrix} y_{11} & \cdots & \cdots & y_{1N} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ y_{N1} & \cdots & \cdots & y_{NN} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & \cdots & a_{1N} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{N1} & \cdots & \cdots & a_{NN} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & \cdots & x_{1N} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x_{N1} & \cdots & \cdots & x_{NN} \end{bmatrix}$$

$$[\vec{y}_1 | \ldots | \vec{y}_i | \ldots | \vec{y}_N] \qquad [\vec{x}_1 | \ldots | \vec{x}_i | \ldots | \vec{x}_N]$$

$$\vec{y}_i = \mathbf{A}\vec{x}_i, i = 1 - N$$

$$\vec{x}_i = [x_{i1}, \ldots, x_{iN}]^T, \vec{y}_i = [y_{i1}, \ldots, y_{iN}]^T$$

But our data varies in 2 dimensions
So we need to do this for both!

2D forward transform

$$\mathbf{Y}_{N\times N} = \mathbf{A}_{N\times N}\mathbf{X}_{N\times N}\mathbf{A}^{T}_{N\times N}$$

$$\begin{bmatrix} y_{11} & \cdots & \cdots & y_{1N} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ y_{N1} & \cdots & \cdots & y_{NN} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & \cdots & a_{1N} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{N1} & \cdots & \cdots & a_{NN} \end{bmatrix} \begin{bmatrix} x_{11} & \cdots & \cdots & x_{1N} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x_{N1} & \cdots & \cdots & x_{NN} \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & \cdots & a_{N1} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{1N} & \cdots & \cdots & a_{NN} \end{bmatrix}$$

1D column transform

1D row transform

- 1D transform matrix **A** consists of basis vectors (column vectors)

$$\vec{y} = \sum_{i=1}^{N} x_i \vec{b}_i, \vec{b}_i = [a_{i1}, \ldots, a_{iN}]^T$$

- 2D transform corresponds to a collection of N-by-N basis images

$$\mathbf{Y} = \sum_{i=1}^{N}\sum_{j=1}^{N} x_{ij}\mathbf{B}_{ij}, \mathbf{B}_{ij} = \vec{b}_i \vec{b}_j^{\,T}, \vec{b}_i = [a_{i1}, \ldots, a_{iN}]^T$$

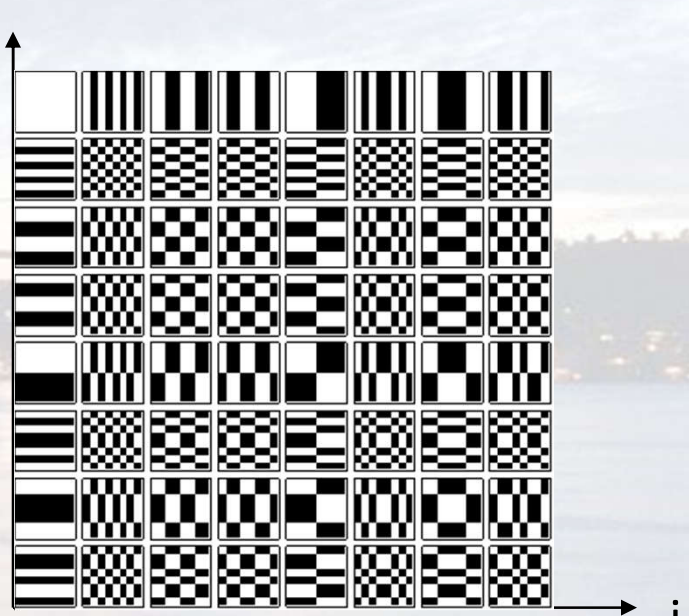
basis image

Examples of basis images

Hadamard matrix: $\mathbf{A}_1 = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \mathbf{A}_{2n} = \frac{1}{\sqrt{2}}\begin{bmatrix} \mathbf{A}_n & \mathbf{A}_n \\ \mathbf{A}_n & -\mathbf{A}_n \end{bmatrix}$

$$\mathbf{A} = \frac{1}{\sqrt{8}}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 & 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 & 1 & -1 & -1 & 1 \\ 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & -1 & 1 & -1 & -1 & 1 & -1 & 1 \\ 1 & 1 & -1 & -1 & -1 & -1 & 1 & 1 \\ 1 & -1 & -1 & 1 & -1 & 1 & 1 & -1 \end{bmatrix},$$

$\vec{b}_1$ $\vec{b}_8$



B_{ij}

There are two types of compression:

Lossy: e.g. jpg

- Some of the information cannot be recovered
- High compression ratios

Lossless: e.g. png

- Information is preserved
- Low compression ratios

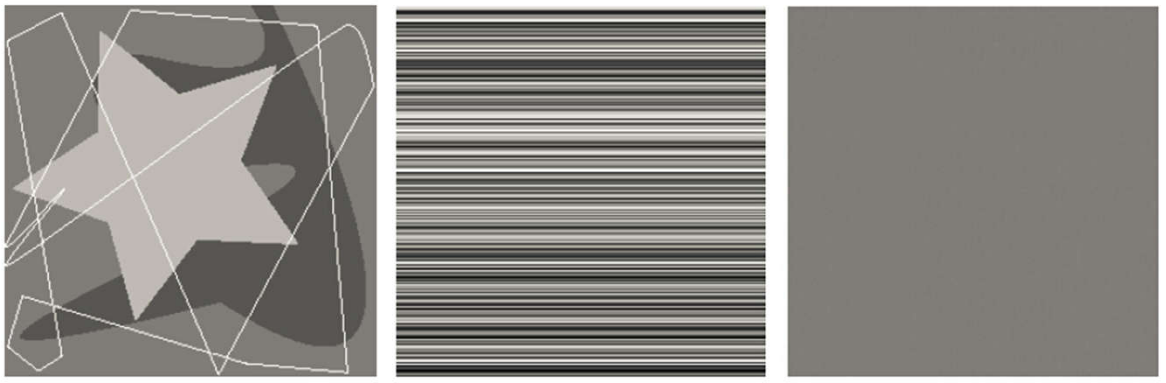
$$Compression\ ratio = \frac{Uncompressed\ size}{Compressed\ size}$$

Compression aims to remove redundancy; the types of redundancy:

Coding redundancy: are pixels encoded at the optimal number of bits?

Spatial and temporal redundancy: pixels can be spatially, and video can be temporally correlated

Irrelevant information: do we need every little detail in an image?



(a) (b) (c)

Fig. 1 Computed generated 256 by 256 by 8 bits images with (a) coding redundancy, (b) spatial and temporal redundancy, (c) irrelevant information.

Huffman coding targets coding redundancy (used in **jpg** and **png**).

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

FIGURE 8.7 Huffman source reductions.

Consider these to be the occurrence probabilities of 8-bit grayscale values in an image.

Huffman coding targets coding redundancy.

Original source			Source reduction							
Symbol	Probability	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

FIGURE 8.8 Huffman code assignment procedure.

The average length of every pixel after Huffman encoding would be:

$$L = 0.4 \times 1 + 0.3 \times 2 + 0.1 \times 3 + 0.1 \times 4 + 0.06 \times 5 + 0.04 \times 5 = 2.2 bits/pixel$$

Objective quality measures

- Mean Squared Error

original decoded

$$MSE = \frac{1}{HW}\sum_{i=1}^{H}\sum_{j=1}^{W}[X(i,j) - Y(i,j)]^2$$

- Peak Signal to Noise Ratio

$$PSNR = 10\log_{10}\frac{255^2}{MSE}(dB)$$

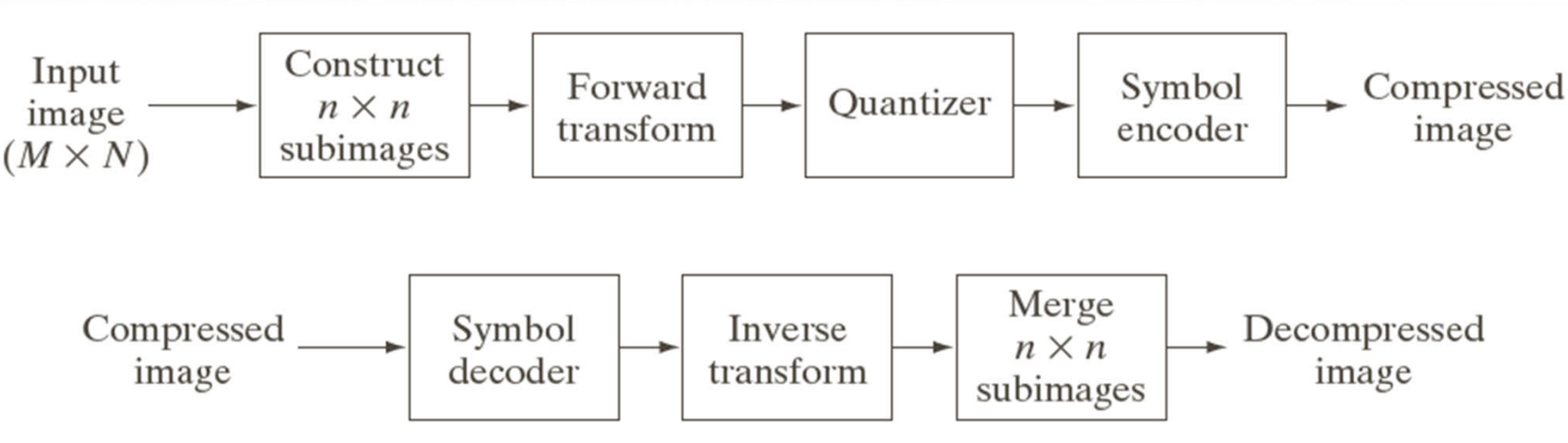
Run-Length encoding (RLE)

- RLE is a technique used to **reduce the size of a repeating string of characters**.
- This repeating string is called a *run*, typically RLE encodes a run of symbols into two bytes, a **count** and a **symbol**.
- RLE can compress any type of data
- RLE cannot achieve high compression ratios compared to other compression methods
- Easy to implement, quick to execute
- Encountered in BMP, TIFF and PCX formats.

WWWWWWWWWWWWBWWWWWWWWWWWWBBBWWWWWWWWWWWW
WWWWWWWWWWWWBWWWWWWWWWWWWWW

RLE coding: 12W1B12W3B24W1B14W

Block transform coding



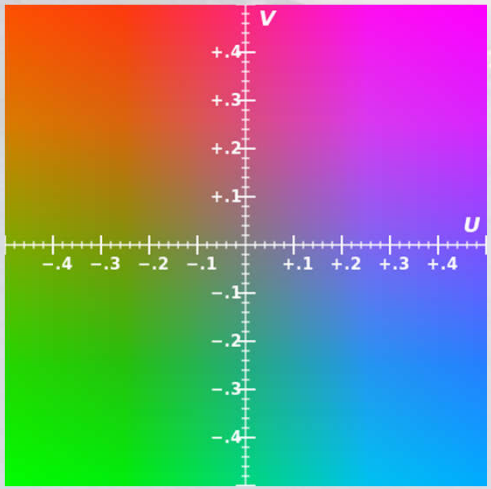
Lossy compression can lead to very high compression rates such 10:1 or even 50:1 while still providing images indistinguishable visually from the original. They target removing the image details that are hardly visible to the human eye. All the same, they commonly introduce artifacts into the images.

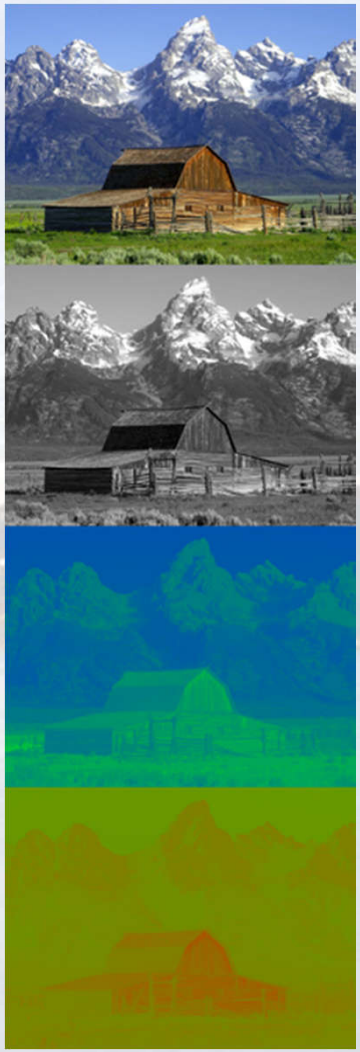
JPEG has variants (**JPEG2000**), works well on color photographs.

1. If the color is represented in RGB mode, translate it to YUV (optional).
2. Divide the file into 8 X 8 blocks.
3. Transform the pixel information of every block from the spatial domain to the frequency domain with the Discrete Cosine Transform (DCT).
4. Quantize the resulting values by dividing each coefficient by an integer value and rounding off to the nearest integer.
5. Look at the resulting coefficients in a zigzag order. Do a run-length encoding of the coefficients ordered in this manner. Follow by Huffman coding.

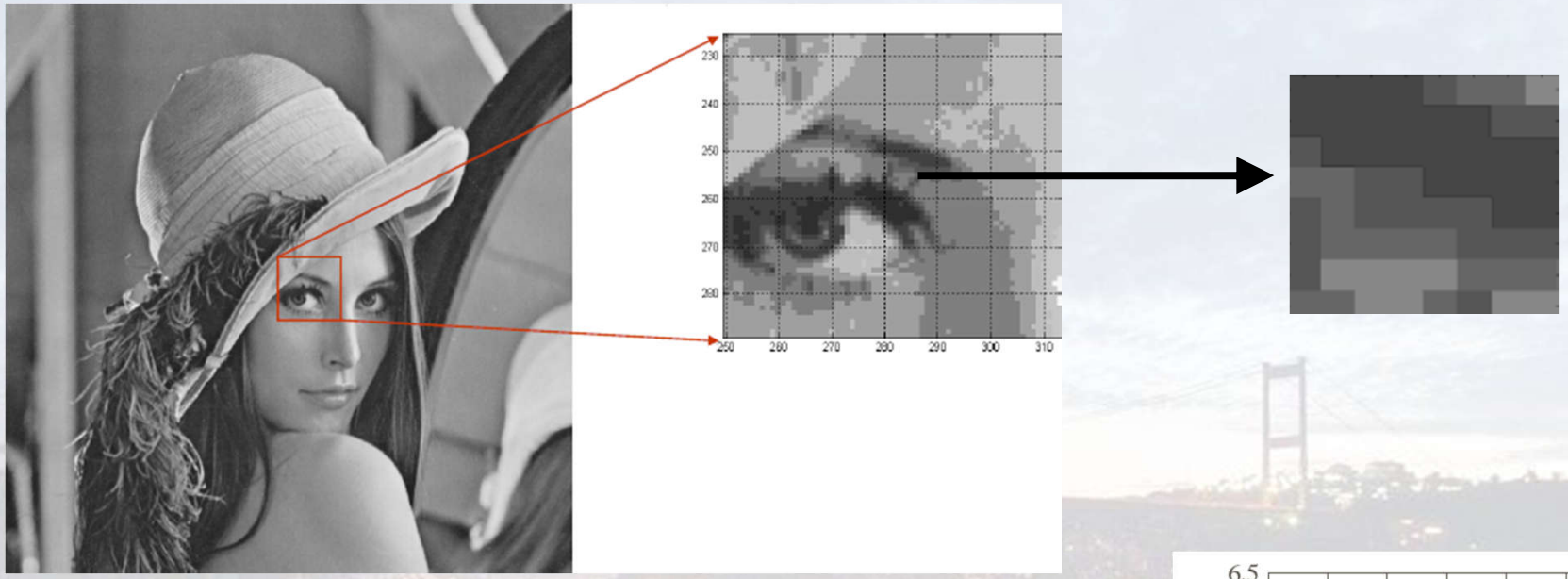
1. If the color is represented in RGB mode, translate it to YUV (optional).

YUV encodes color as Y: luminance and UV: chrominance.
The human eye is more sensitive to intensity variations than chrominance. Meaning that chrominance can be sub-quantized without significant loss of information.



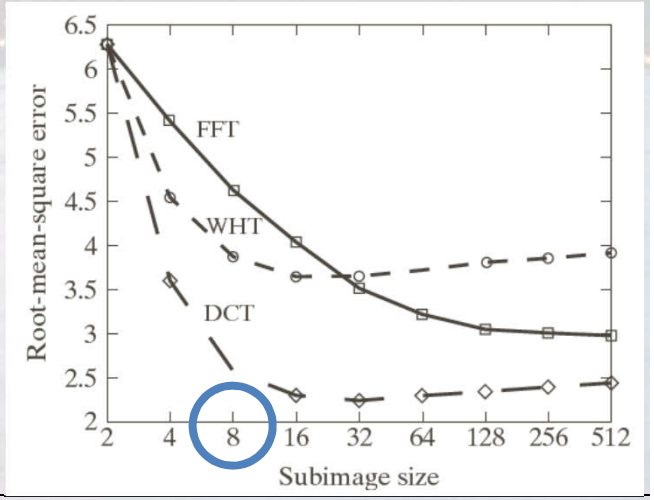


2. Divide the file into 8 X 8 blocks



Why not convert the whole image?

- The transform matrix will be too large and will require a lot of memory
- Image content can vary a lot, this way it's adaptive.
- Why 8x8? Because it's optimal for DCT

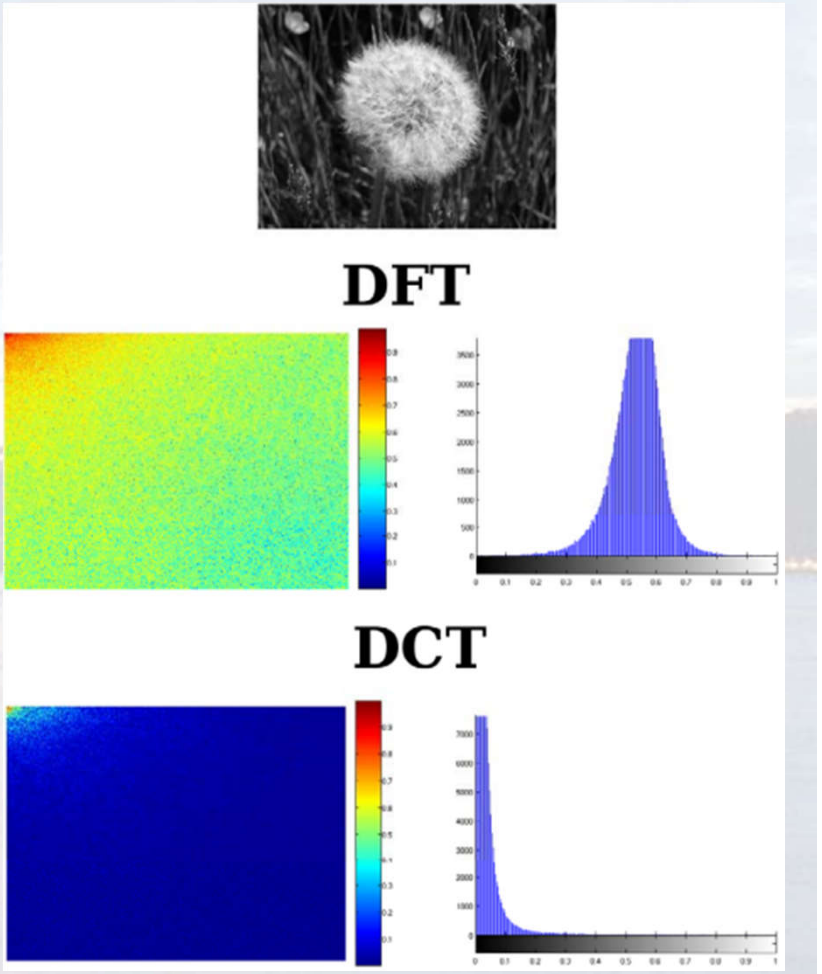


3. Transform the pixel information of every block from the spatial domain to the frequency domain with the Discrete Cosine Transform (DCT).

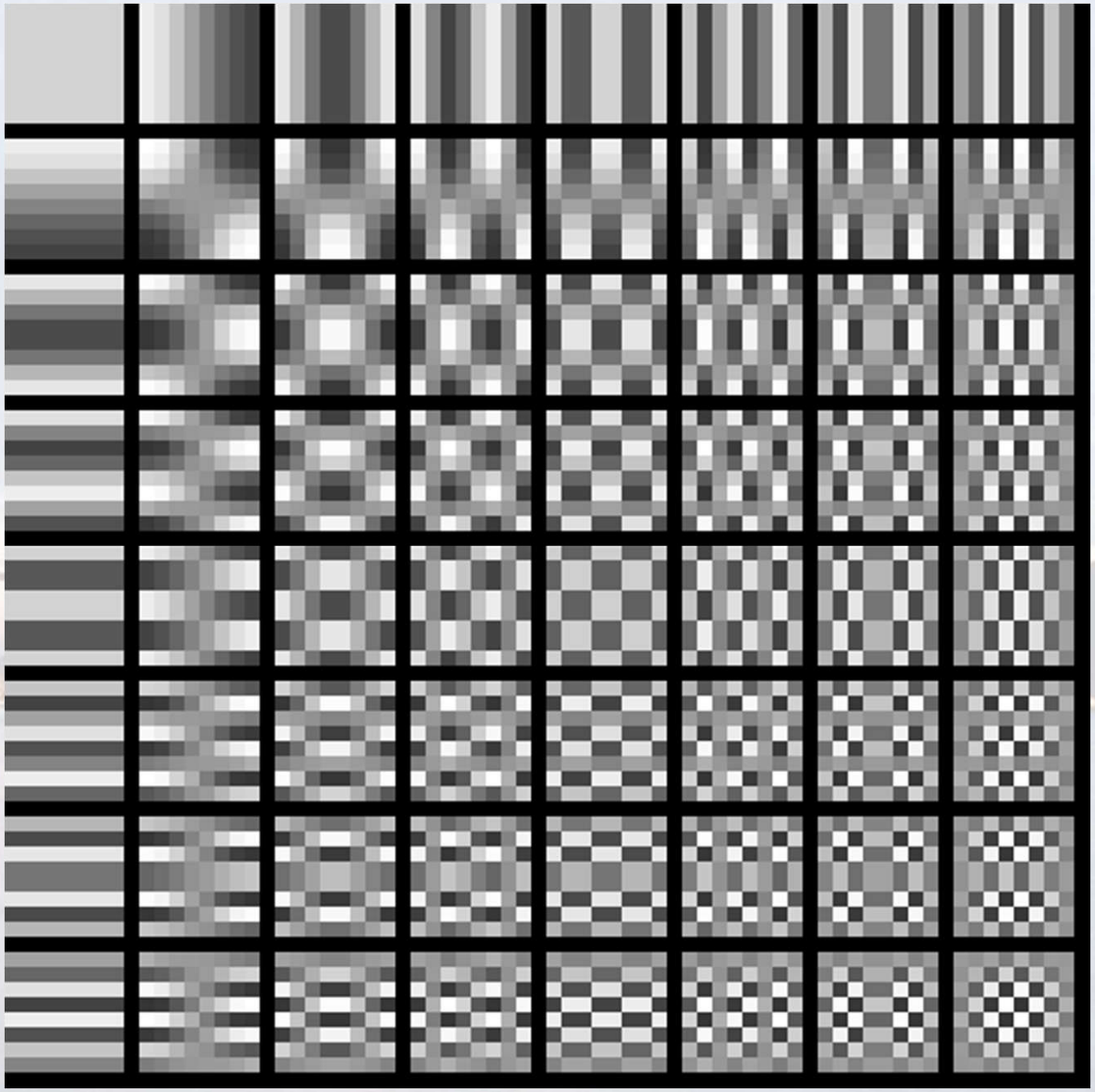
$$\mathbf{A}_{N\times N}=\begin{bmatrix} a_{11} & \dots & \dots & a_{1N} \\ \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots \\ a_{N1} & \dots & \dots & a_{NN} \end{bmatrix}=\mathbf{C}$$

$$a_{kl}=\begin{cases} 1/\sqrt{N}, & k=1, 1\le l\le N \\ \sqrt{2/N}\cos\dfrac{(2l-1)(k-1)\pi}{2N}, & 2\le k\le N, 1\le l\le N \end{cases}$$

Why DCT and not DFT? Because it has a higher energy compaction capacity.



DCT basis images (8x8)



4. Quantize the resulting values by dividing each coefficient by an integer value and rounding off to the nearest integer (throw away the high frequencies).

717.6	0.2	0.4	-19.8	-2.1	-6.2	-5.7	-7.6
-99.0	-35.8	27.4	19.4	-2.6	-3.8	9.0	2.7
51.8	-60.8	3.9	-11.8	1.9	4.1	1.0	6.4
30.0	-25.1	-6.7	6.2	-4.4	-10.7	-4.2	-8.0
22.6	2.7	4.9	3.4	-3.6	8.7	-2.7	0.9
15.6	4.9	-7.0	1.1	2.3	-2.2	6.6	-1.7
0.0	5.9	2.3	0.5	5.8	3.1	8.0	4.8
-0.7	-2.3	-5.2	-1.0	3.6	-0.5	5.1	-0.1

÷

Quantization Matrix

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

=

=

45	0	0	-1	0	0	0	0
-8	-3	2	1	0	0	0	0
4	-5	0	0	0	0	0	0
2	-1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Quantized 8x8 block

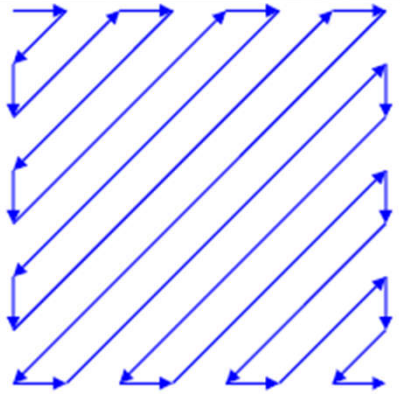
Quantization is limiting the number of possible values, in other words, loss of precision: F’(u)=round(F(u)/Q(u))

5. Look at the resulting coefficients in a zigzag order. Do a run-length encoding of the coefficients ordered in this manner. Follow by Huffman coding.

We go by zig-zag so that coefficients are in increasing frequency.

45	0	0	-1	0	0	0	0
-8	-3	2	1	0	0	0	0
4	-5	0	0	0	0	0	0
2	-1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Input



Zigzag scan procedure

`Result = 45,0,-8,4,-3,0,-1,2,-5,2,1,-1,0,1,0,0,0,0,0,0,1,EOB`

EOB symbol denotes the end-of-block condition

a b c
d e f

FIGURE 8.32 Two JPEG approximations of Fig. 8.9(a). Each row contains a result after compression and reconstruction, the scaled difference between the result and the original image, and a zoomed portion of the reconstructed image.

JPEG compression

10 (8k bytes)

50 (21k bytes)

90 (58k bytes)

0 worst quality, highest compression

best quality, 100 lowest compression

JPEG compression

raw image data

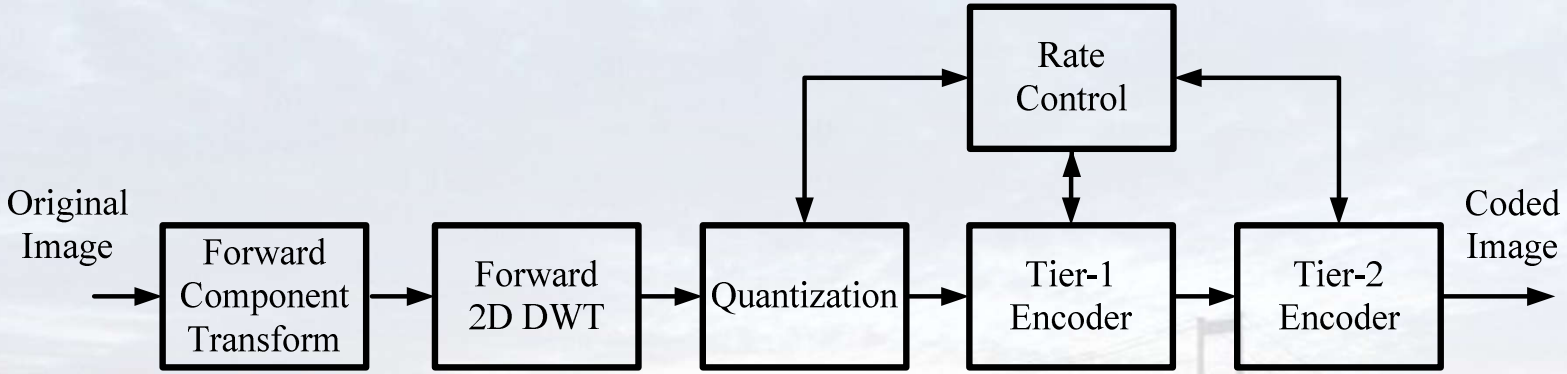
768×512 ×24 = 1.18MB

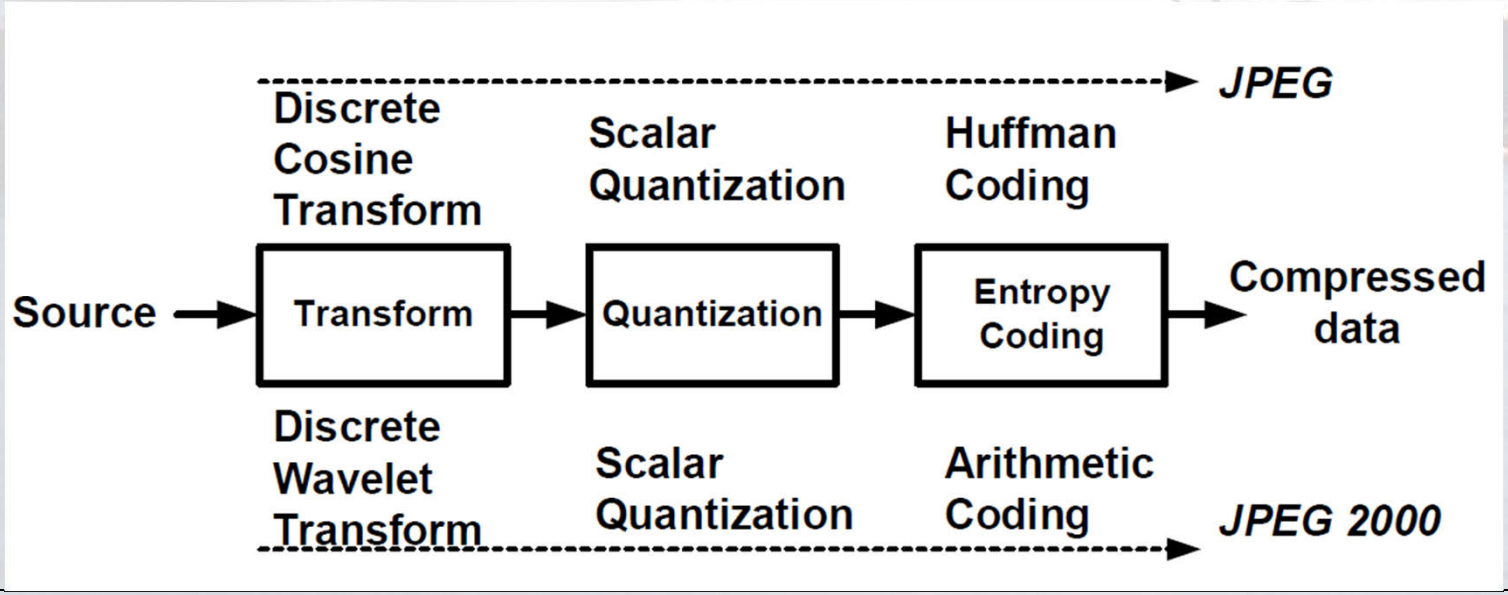
JPEG compressed data

26KB

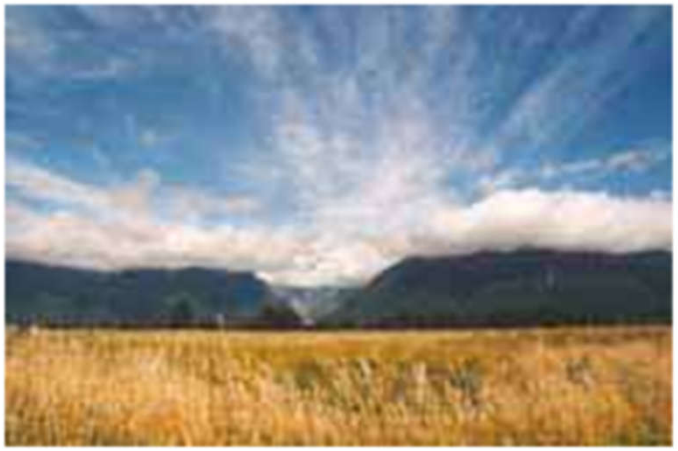
Compression ratio = 44.6

JPEG2000 (2000) is a newer standard that is more effective but more complex.





JPEG compression



JPEG

JPEG 2000

JPEG

JPEG 2000

Should you use lossless or lossy compression?

It depends on your application; if you are trying for instance to recognize fingerprints, then fine details are of crucial significance, and lossy compression could destroy your application.

If on the other hand you are handling for example scenery photographs it would make little sense to insist on lossless compression.