Computer Vision

ECE/CSE 576
Color and Texture

Linda Shapiro
Professor of Computer Science & Engineering
Professor of Electrical & Computer Engineering

We don’t really understand vision

- Visual cortex - highly studied part of the brain
- Only rough idea of what different components do
- New discoveries in vision all the time
  - Eye uses blinking to reset its rotational orientation
  - Visual cortex can make some “high-level” decisions

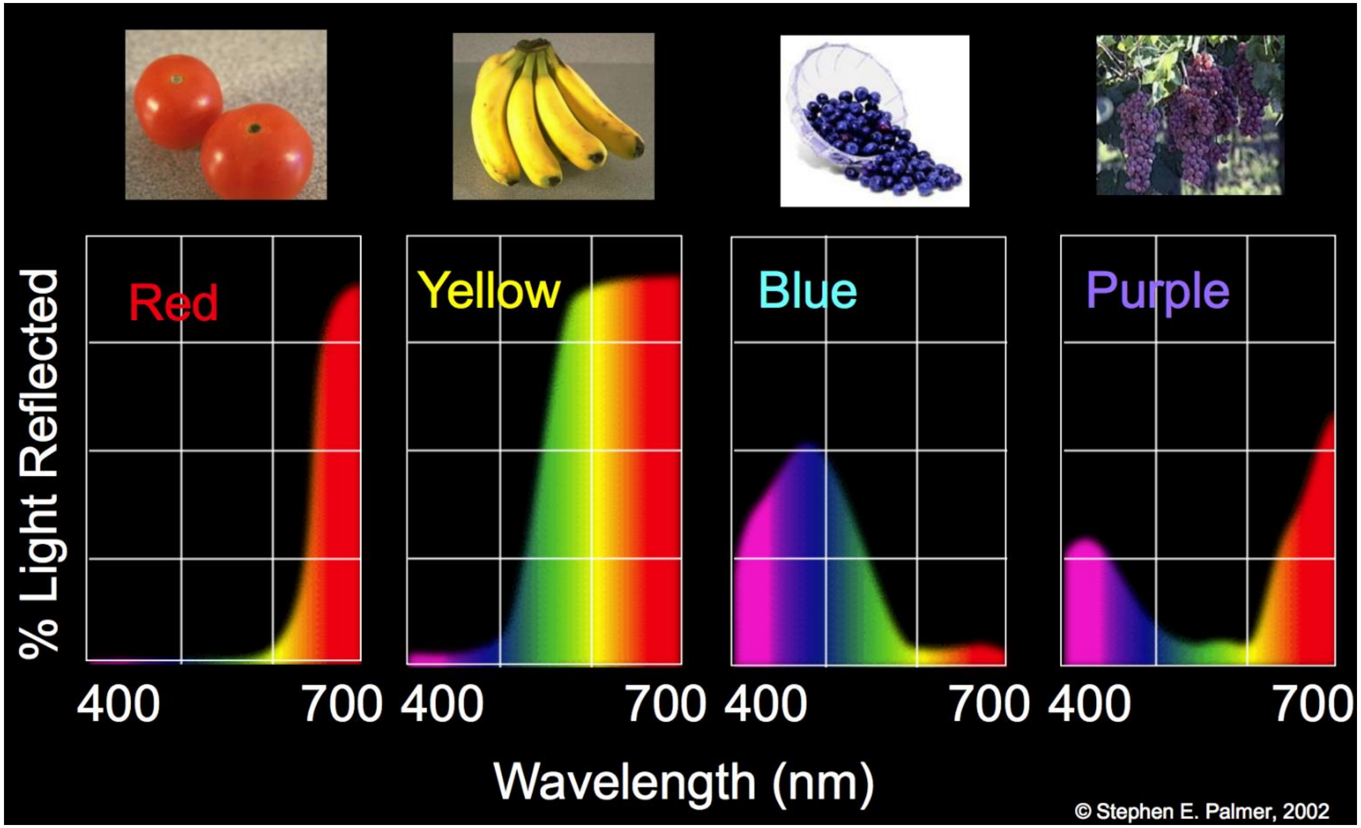
Is vision easy or hard for humans?

- What do you think?

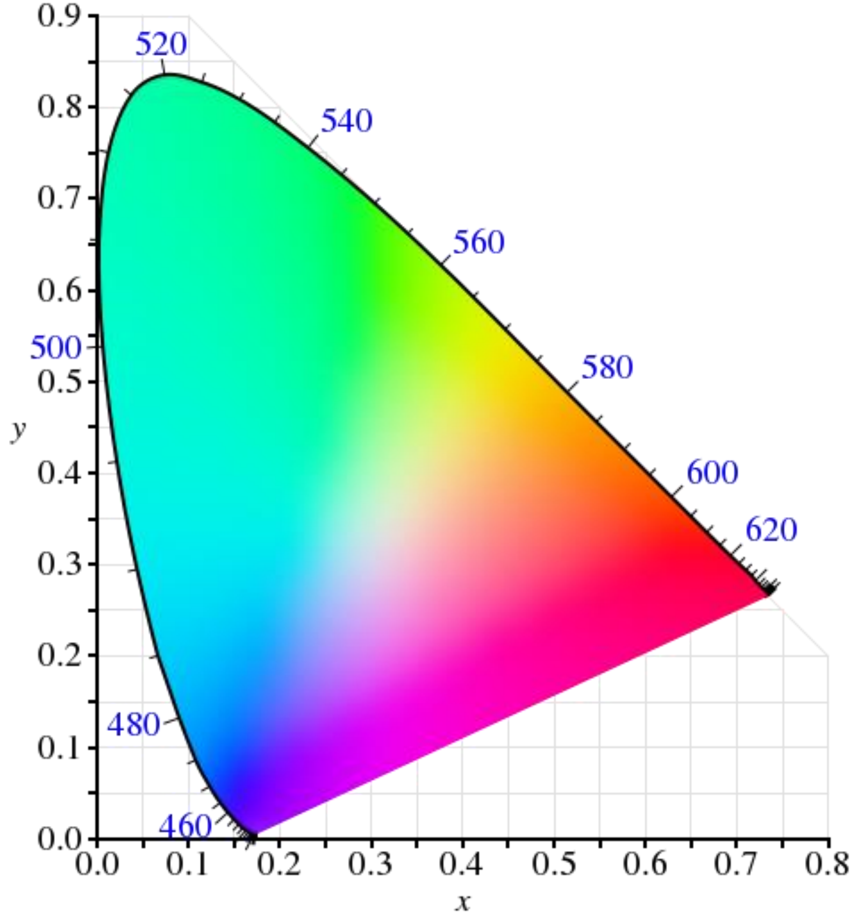
Is vision easy or hard for humans?

- What do you think now?

Objects reflect only some light



We can make a map of color

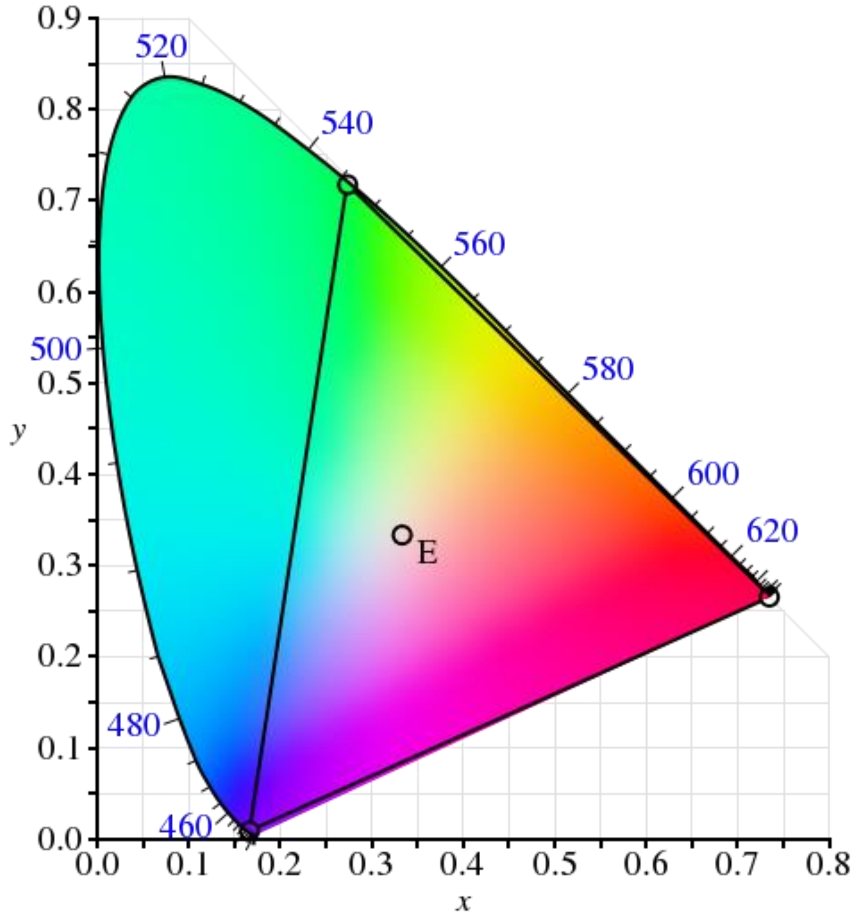


Linear colorspace

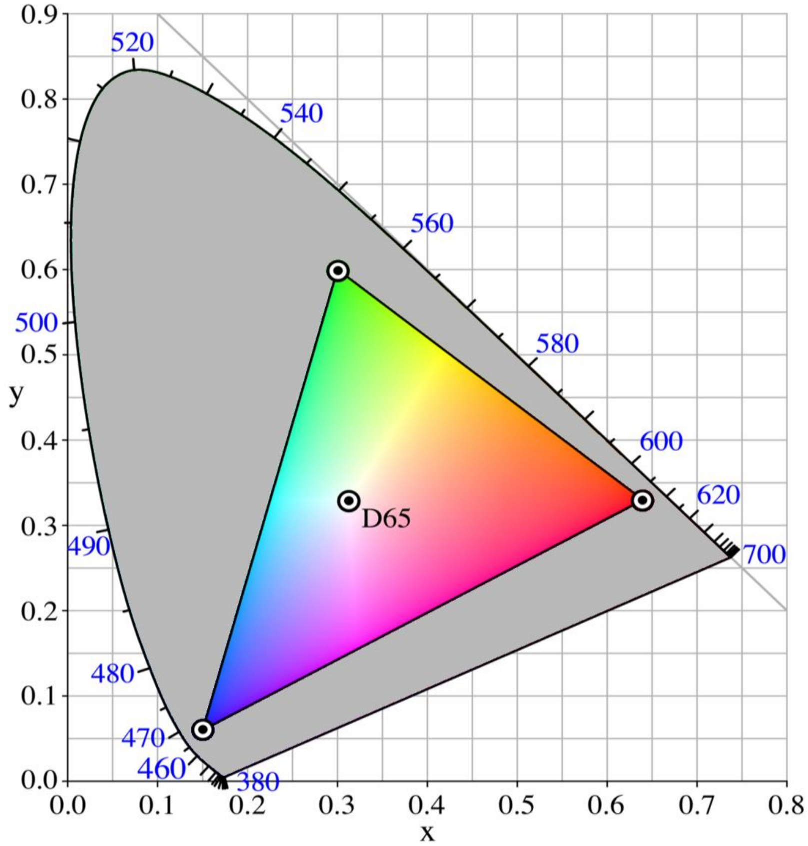
- Pick some primaries
- Can mix those primaries to match any color inside the triangle
- There is a commision that studies color!

- Commission internationale de l’éclairage **(CIE)** is a 100 year old organization that creates international standards related to light and color.

“Theoretical” CIE RGB primaries



Practical sRGB primaries, MSFT 1996



sRGB (standard Red Green Blue) is an RGB color space that HP and Microsoft created cooperatively in 1996 to use on monitors, printers, and the Internet. It was subsequently standardized by the IEC (International Electrotechnical Commission) as IEC 61966-2-1:1999.

What does this mean for computers?

- We represent images as grids of pixels
- Each pixel has a color, 3 components: RGB
- Not every color can be represented in RGB!
  - Have to go out in the real world sometimes
- RGB is not fully accurate
- We can represent a color with 3 numbers
  - #ff00ff; (1.0, 0.0, 1.0); 255,0,255
  - WHAT COLOR IS THAT?

https://homes.cs.washington.edu/~shapiro/schedule.html

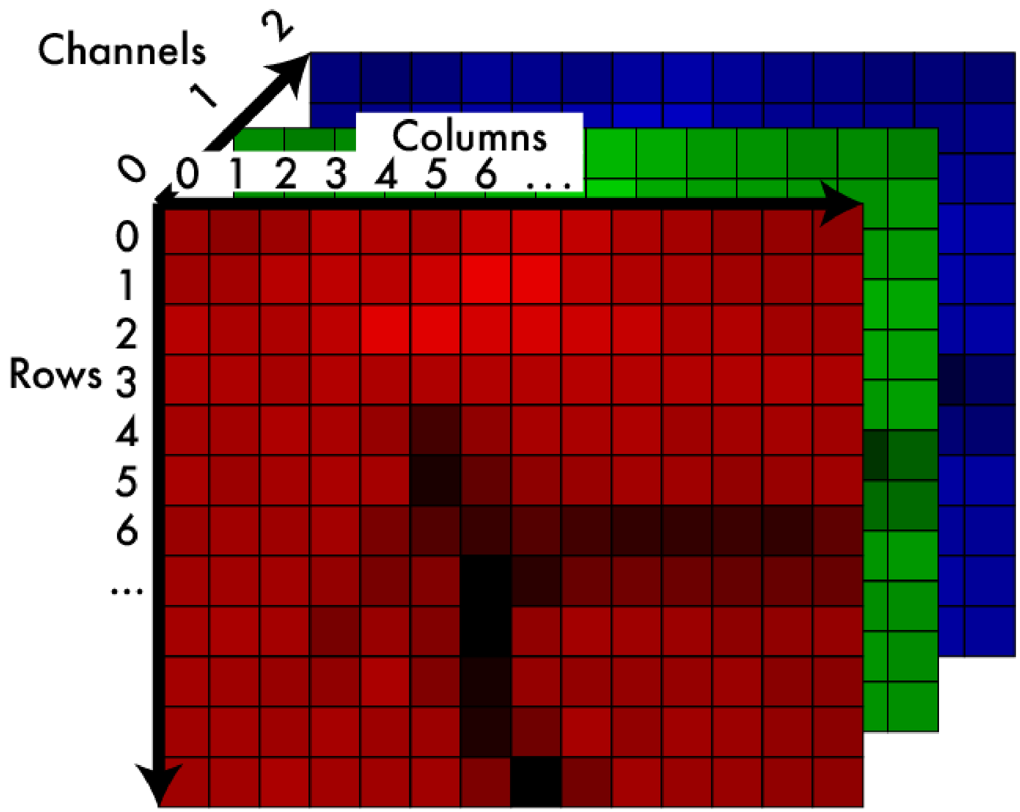
Linda Shapiro's Schedule: Spring 2020

FF00AA

Time	Monday	Tuesday	Wednesday	Thursday	Friday
9:30-10:30					
10:00-11:20		CSE 455 online		CSE 455 online	
11:00-12:00			Office Hour	ECE Faculty Meeting	
12:30-2:00	Lunch	Multimedia Meeting	Lunch	Lunch	Lunch
1:30-3:30				CSE Faculty Meeting	
3:30-4:30				CSE Colloquium	
4:30-6:00				FERG Meeting	

Image: 2d array of color

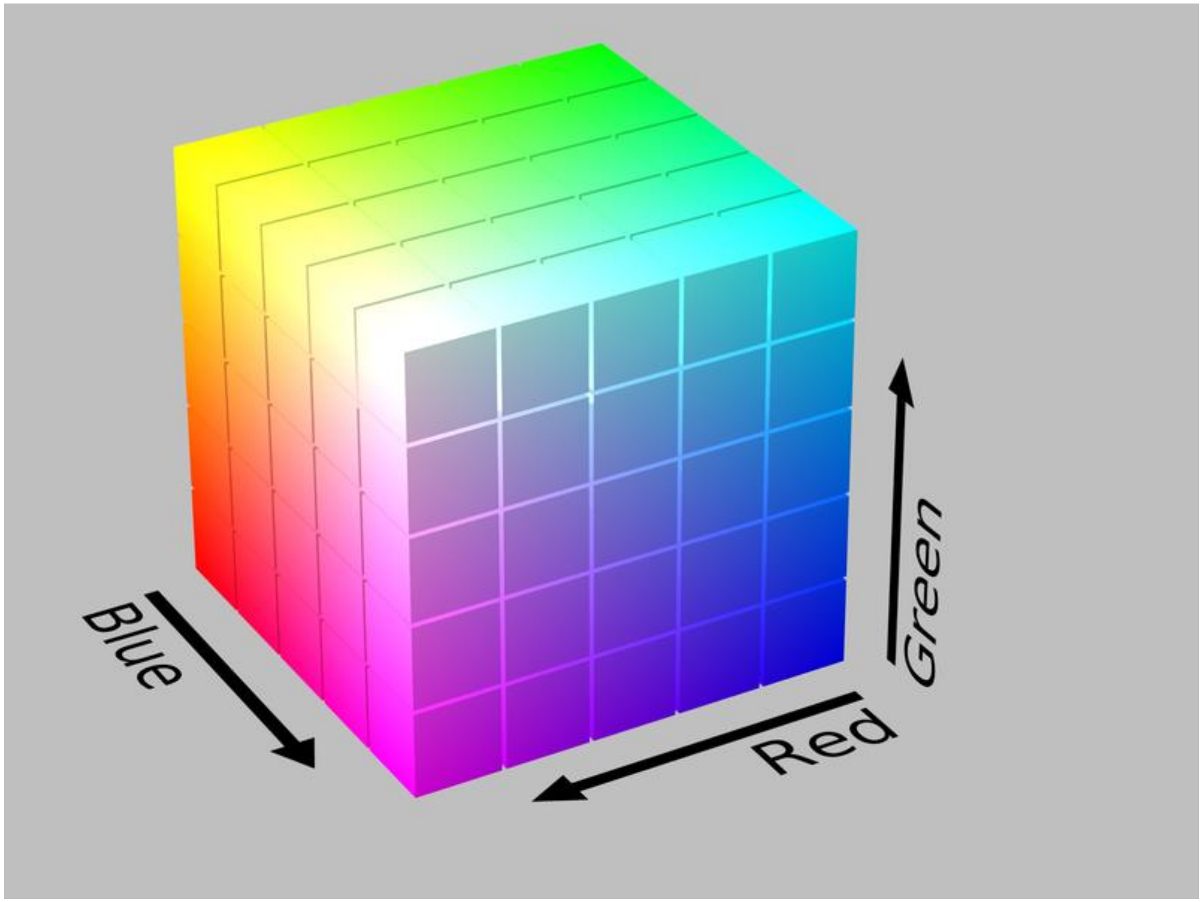
- Some range
  - [0,255]
  - 0.0 - 1.0
- We'll talk more about this later.



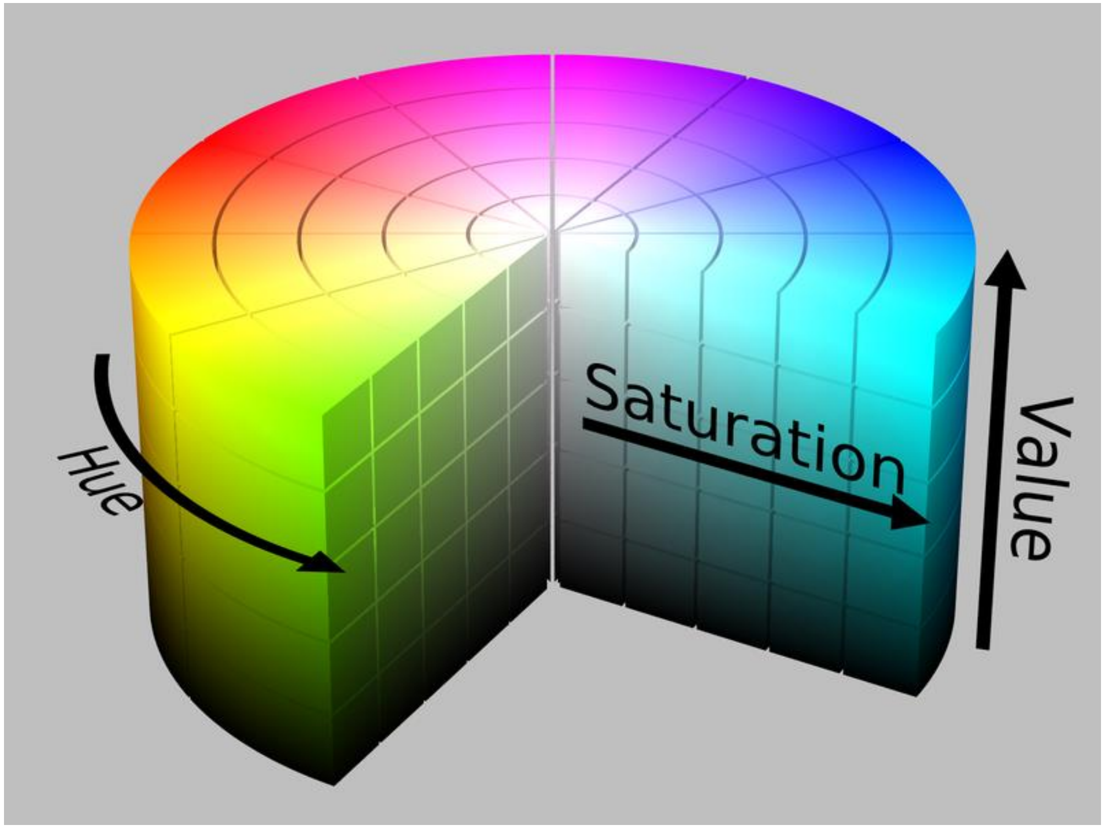
Grayscale - making color images not

- We can simulate monochromatic images from RGB
- Want a good approximation of how “bright” the image is without color information
- (R+B+G/3) - looks weird
- We *should*
  - Gamma decompress
  - Calculatight lightness
  - Gamma compress
- We can just operate on sRGB
  - Typically ~ .30R + .59G + .11B

RGB is a cube...

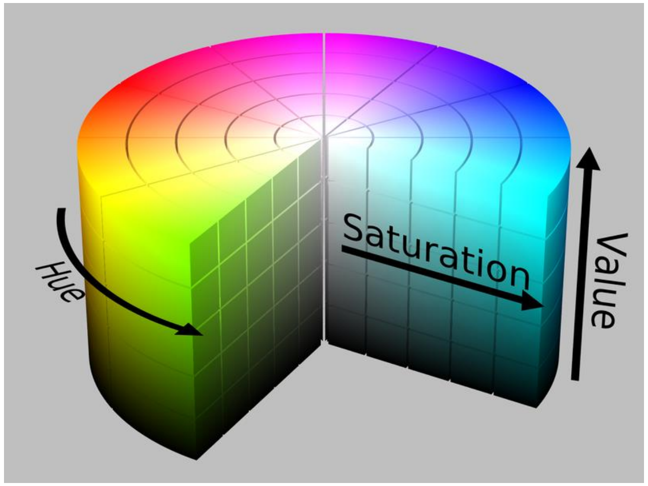


Hue, Saturation, Value: cylinder!

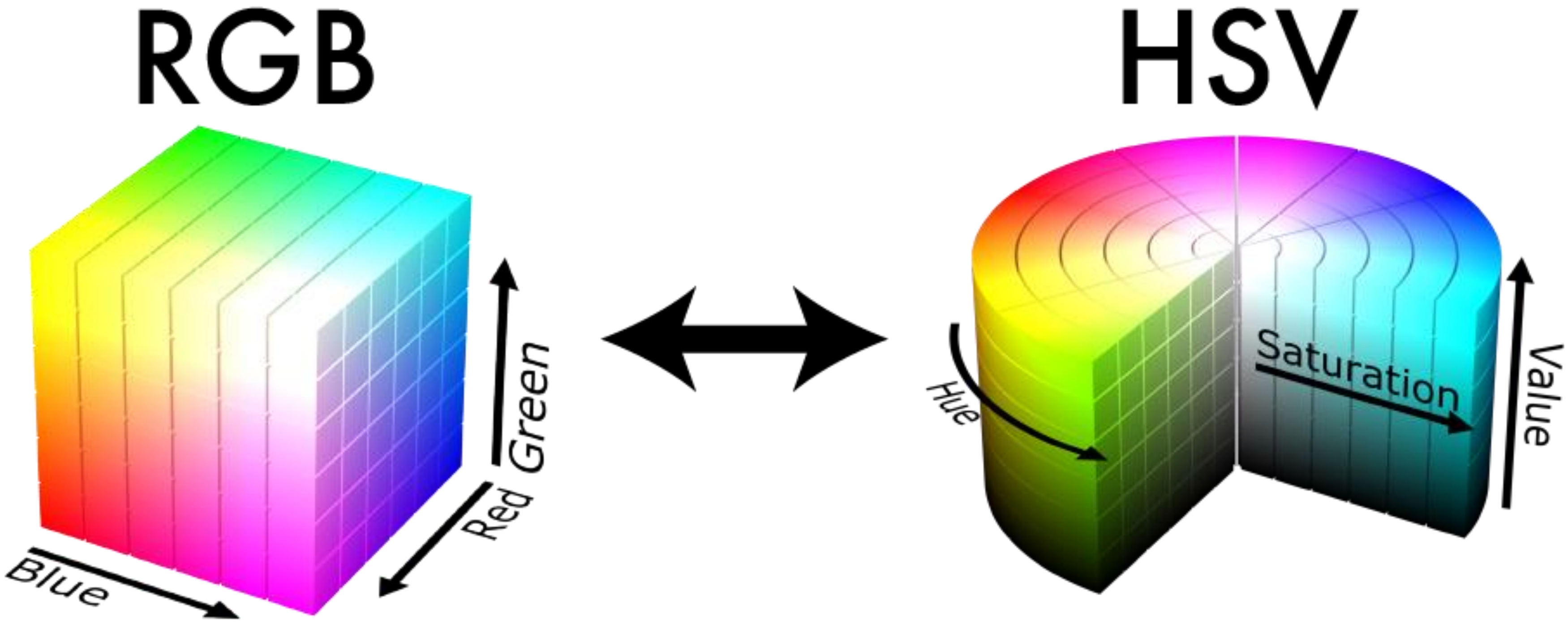


Hue, Saturation, Value

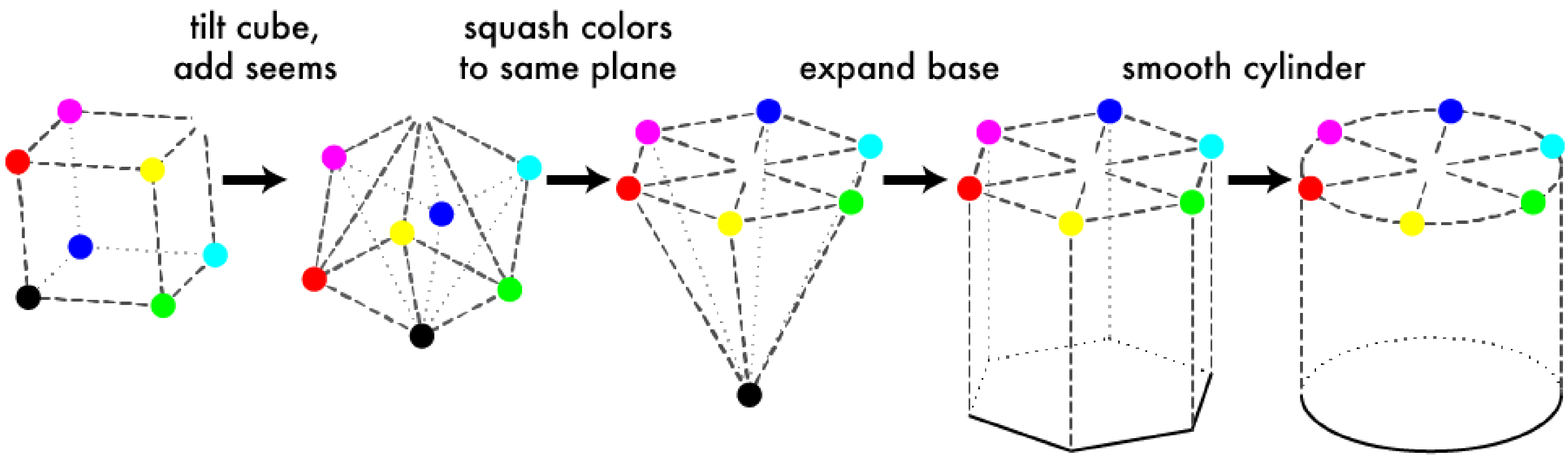
- Different model based on perception of light
- Hue: what color
- Saturation: how much color
- **Value**: how bright
- Allows easy image transforms
  - Shift the hue
  - Increase saturation



Other colorspaces are fun!

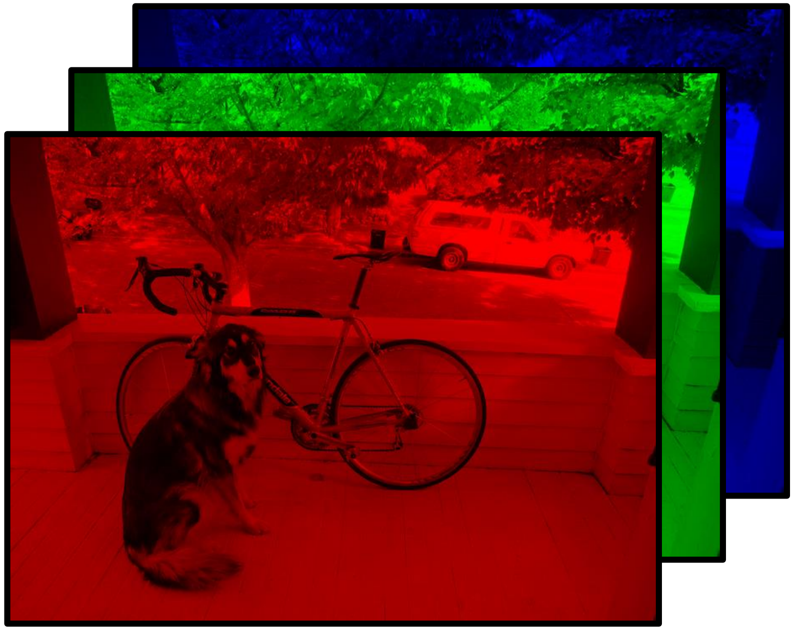


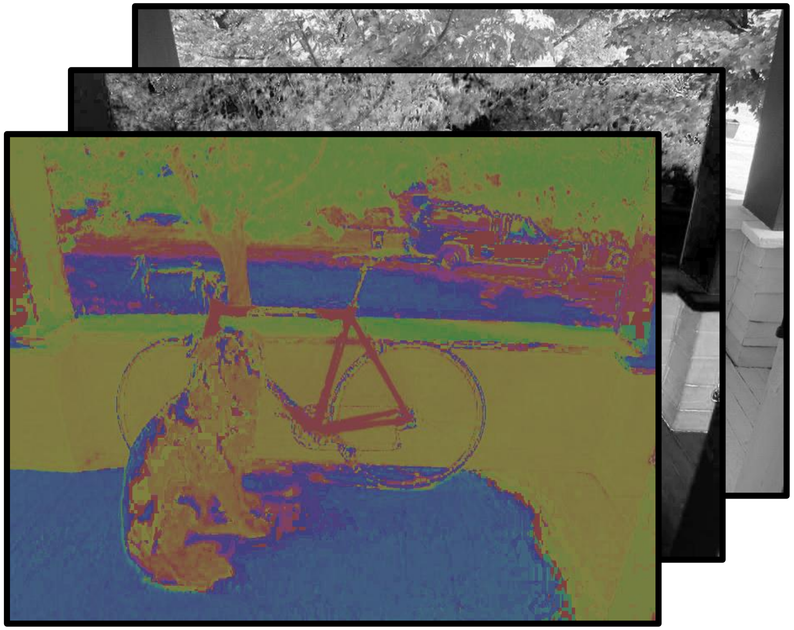
Geometric HSV to RGB:



An RGB Image

Still 3d tensor, different info

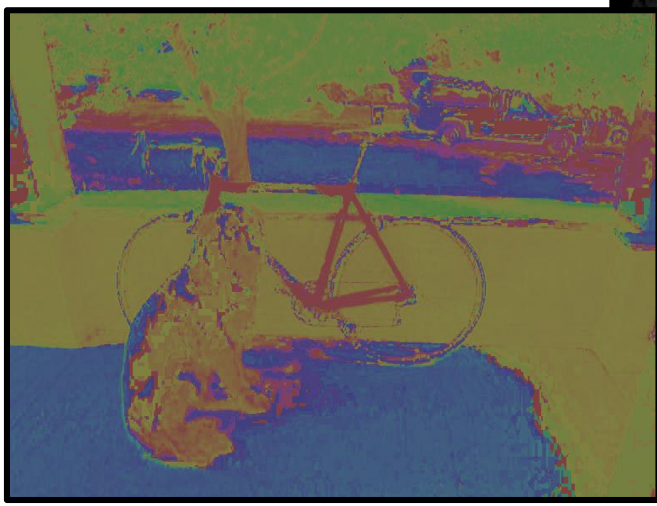




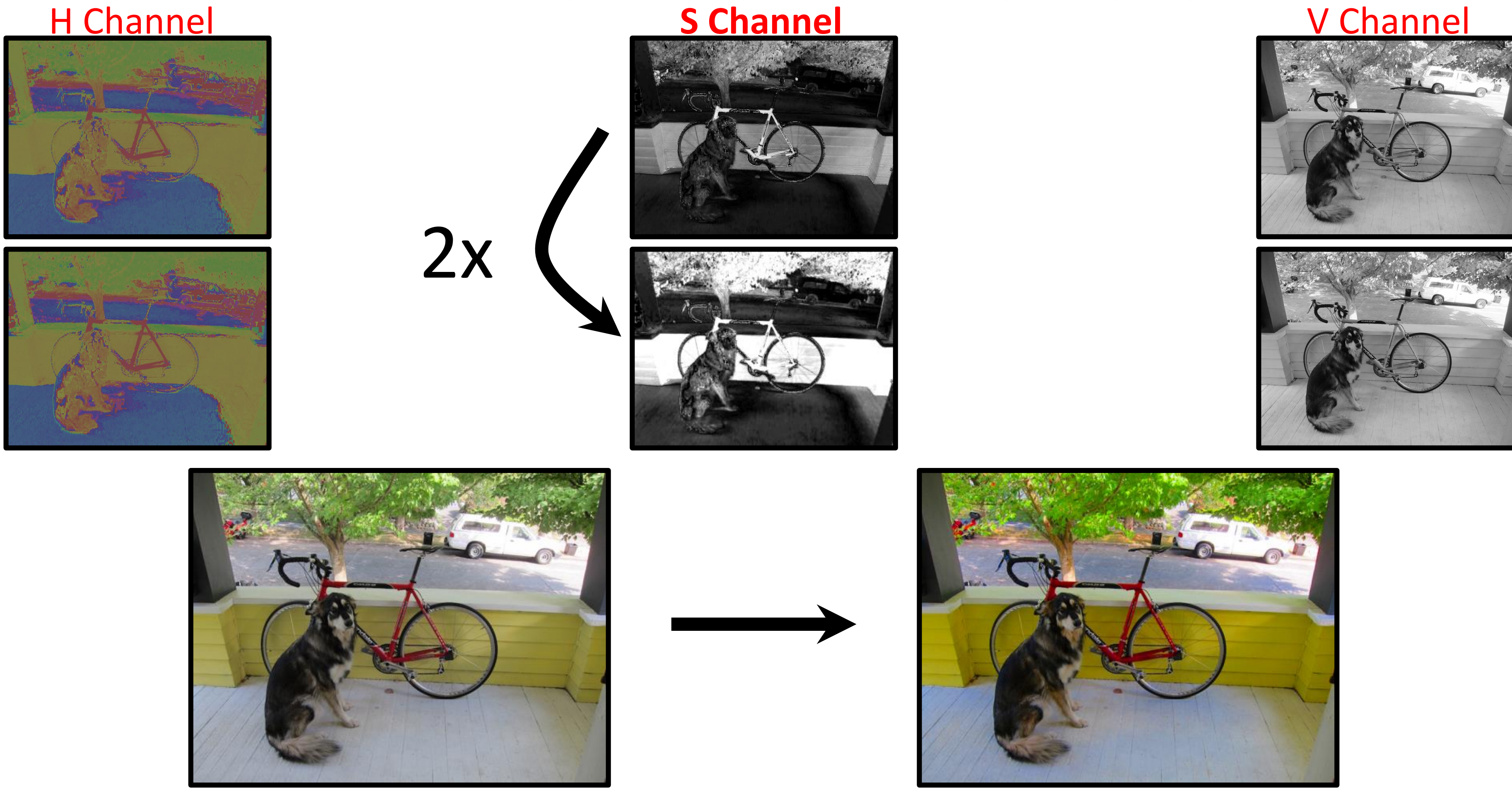
Hue

Saturation

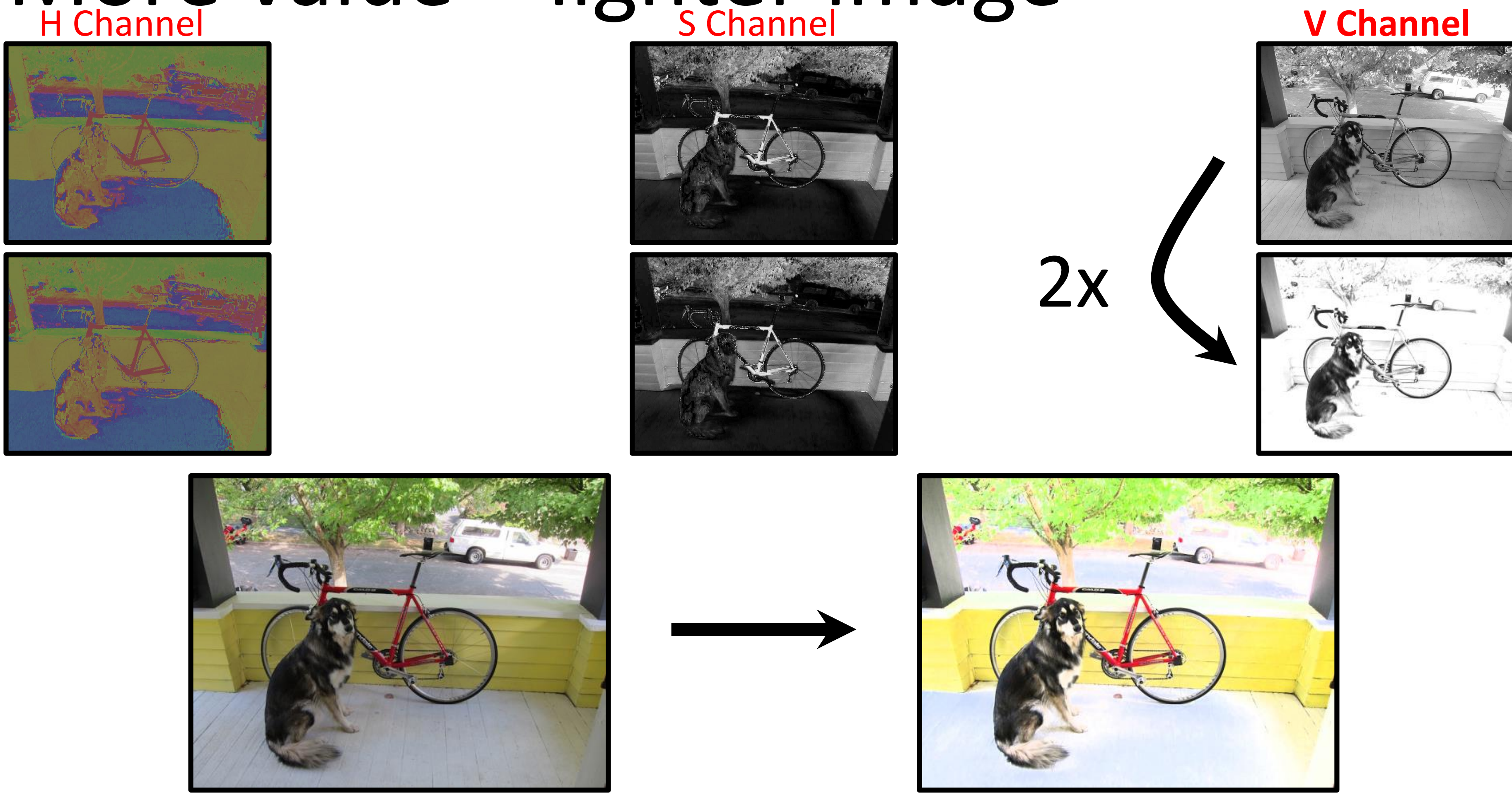
Value



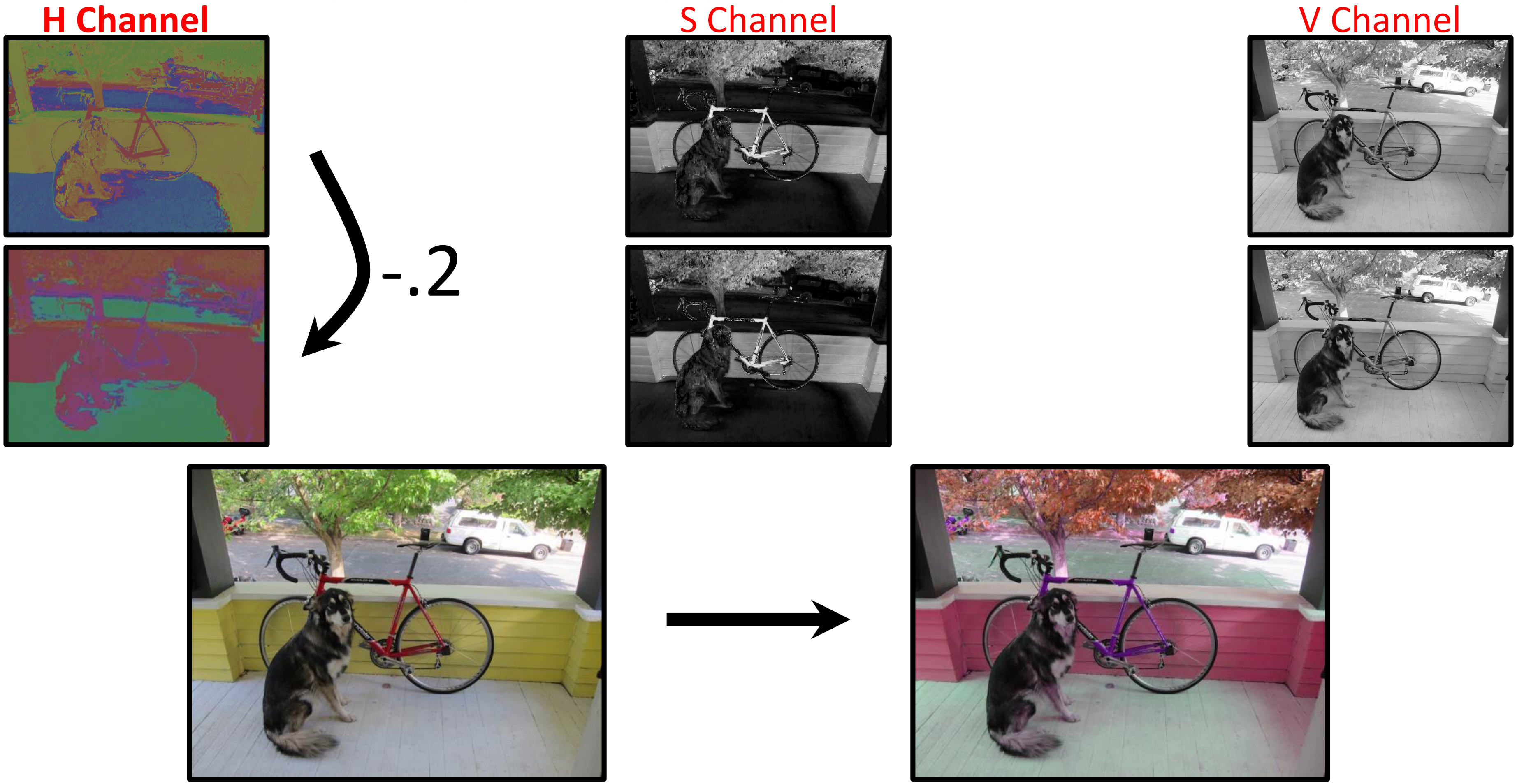
More saturation = intense colors



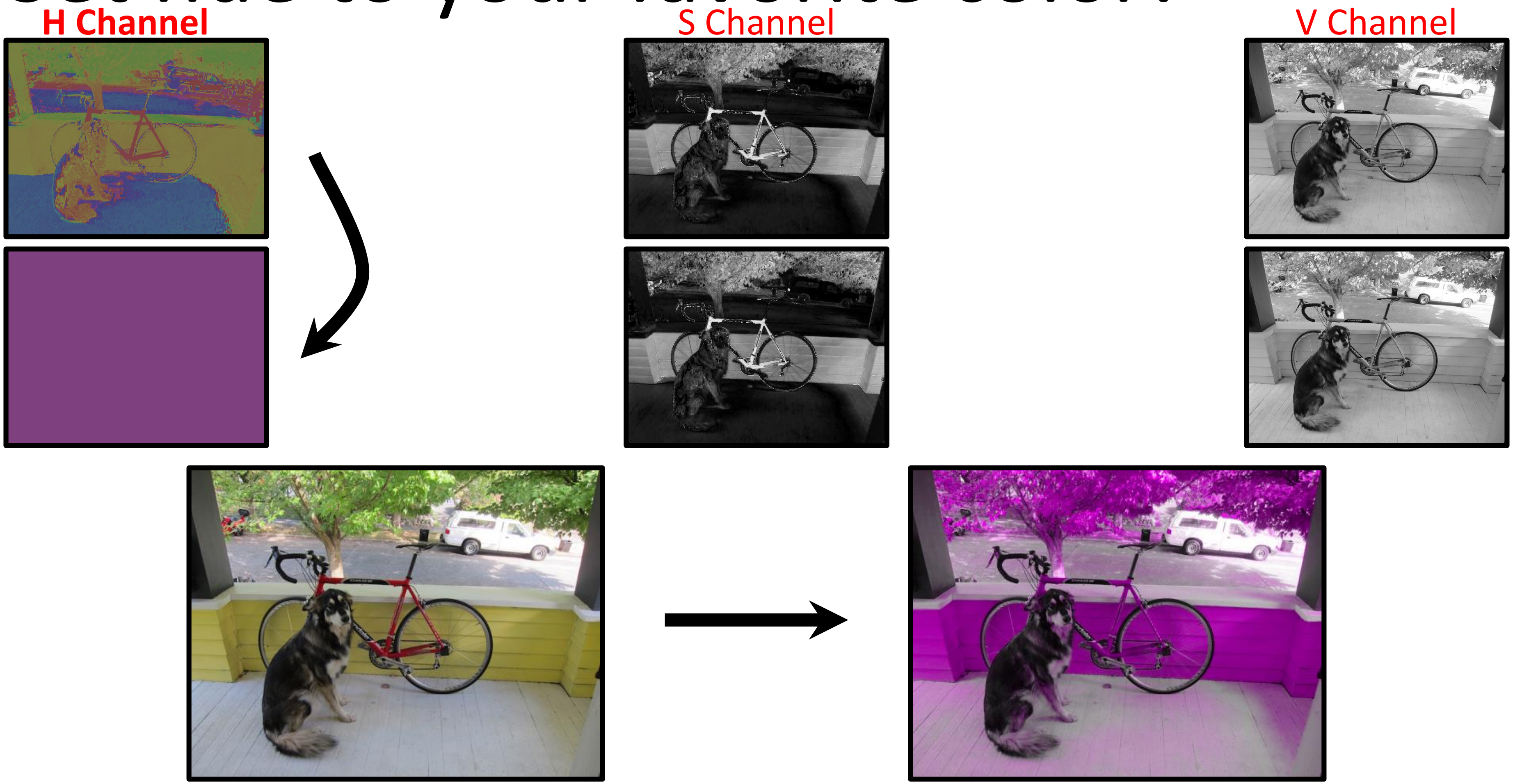
More value = lighter image



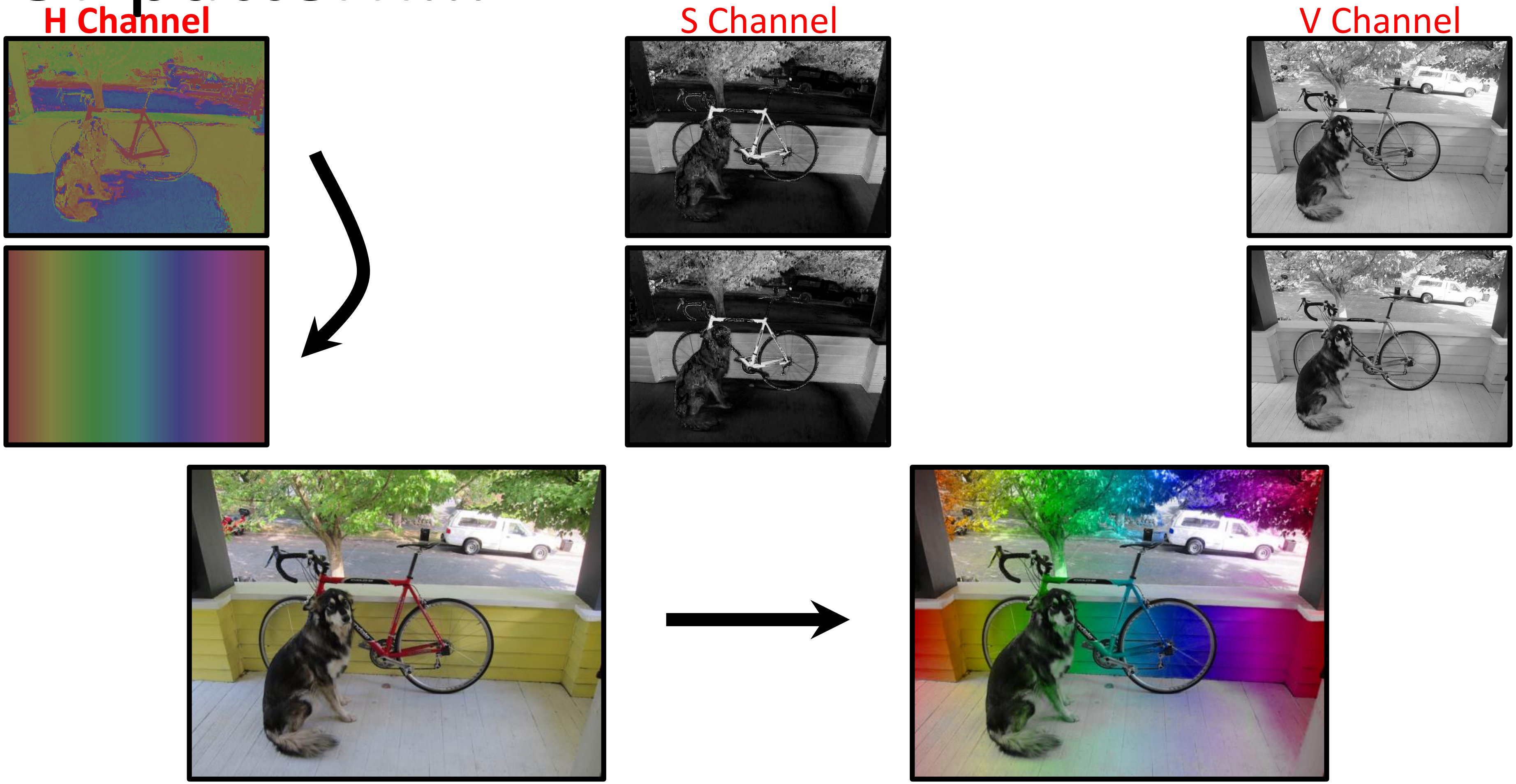
Shift hue = shift colors



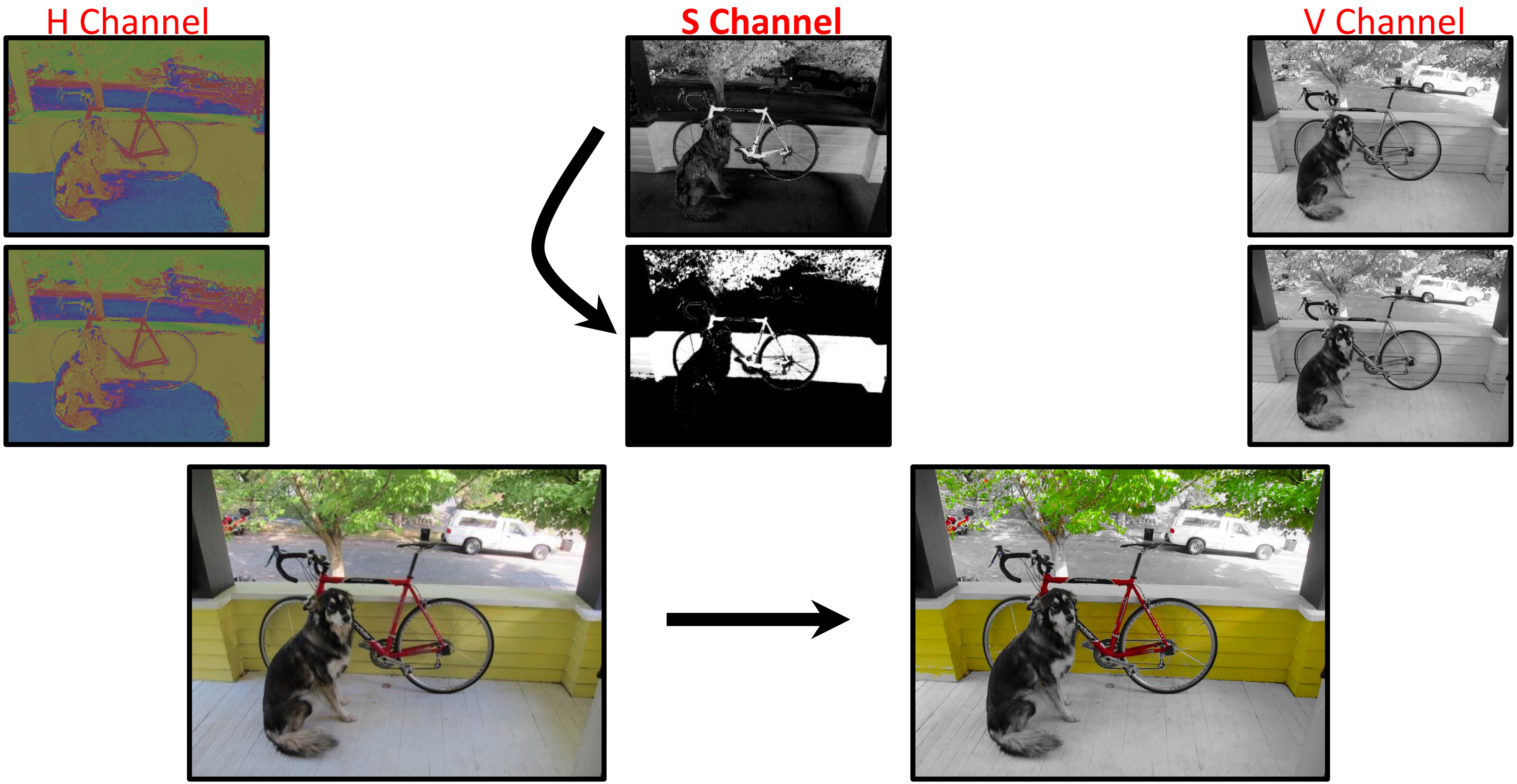
Set hue to your favorite color!



Or pattern…



Increase and threshold saturation

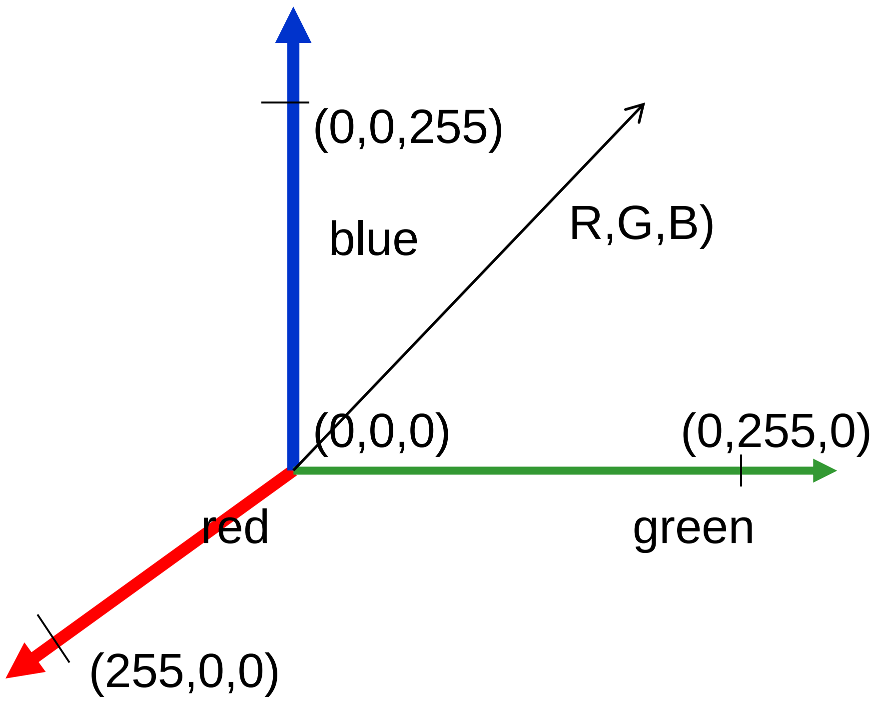


More Details on Color Spaces

• RGB	standard for cameras
• HSI/HSV	allows us to separate
• CIE L*a*b	intensity plus 2 color channels
• YIQ	color TVs, Y is intensity
• Opponent	used in Swain & Ballard work

RGB Color Space

Absolute



Normalized

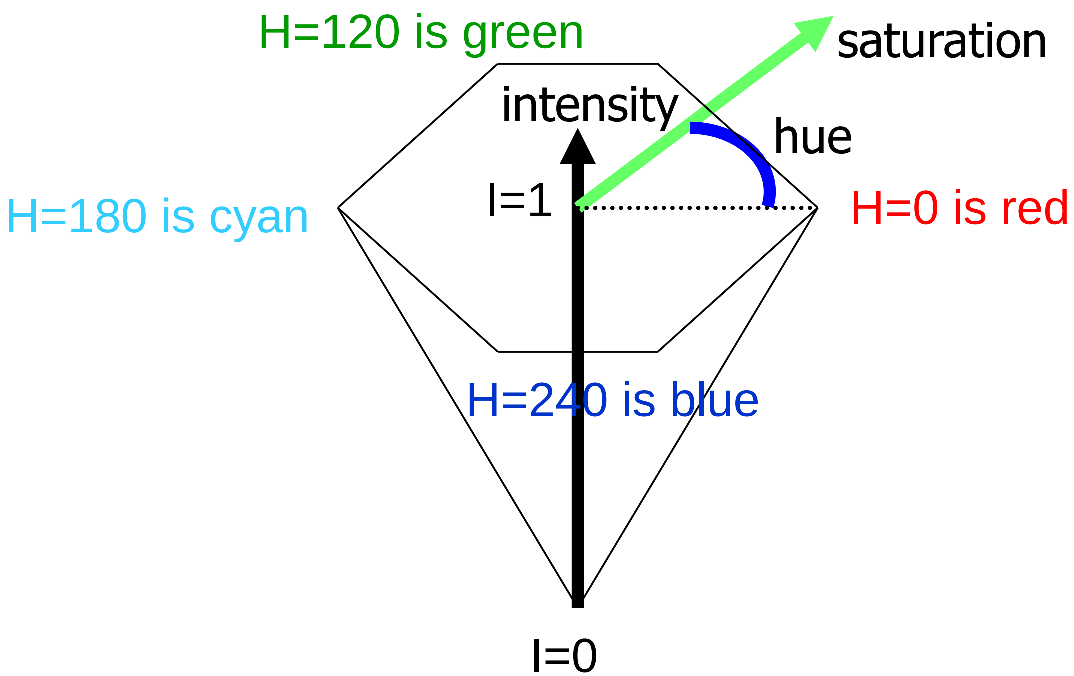
Normalized red **r = R/(R+G+B)**

Normalized green **g = G/(R+G+B)**

Normalized blue **b = B/(R+G+B)**

Color hexagon for HSI (HSV)

- Hue is encoded as an angle (0 to 2π).
- Saturation is the distance to the vertical axis (0 to 1).
- Intensity is the height along the vertical axis (0 to 1).



Conversion from RGB to YIQ

An approximate linear transformation from RGB to YIQ:

$$\begin{array}{rrcrcrcr}
luminance & Y & = & 0.30R & + & 0.59G & + & 0.11B \\
R - cyan & I & = & 0.60R & - & 0.28G & - & 0.32B \\
magenta - green & Q & = & 0.21R & - & 0.52G & + & 0.31B
\end{array}$$

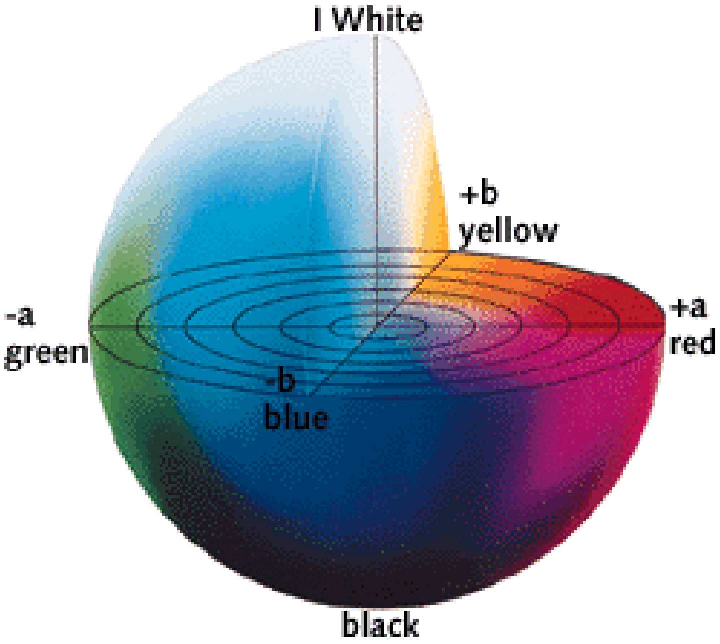
We often use this for color to gray-tone conversion.

CIELAB, Lab, L*a*b

- One luminance channel (L)
  and two color channels (a and b).

- In this model, the color differences which you perceive correspond to Euclidian distances in CIELab.

- The a axis extends from green (-a) to red (+a) and the b axis from blue (-b) to yellow (+b). The brightness (L) increases from the bottom to the top of the three-dimensional model.



CIE *L*a*b** (CIELAB) is a color space specified by the International Commission on Illumination (French *Commission internationale de l'éclairage*, hence its *CIE* initialism).

Histograms

- A histogram of a gray-tone image is an array H[*] of bins, one for each gray tone.

- H[i] gives the count of how many pixels of an image have gray tone i.

- P[i] (the normalized histogram) gives the percentage of pixels that have gray tone i.

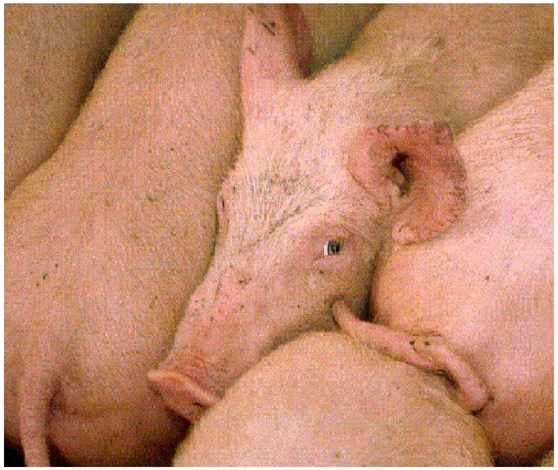
Color histograms can represent an image

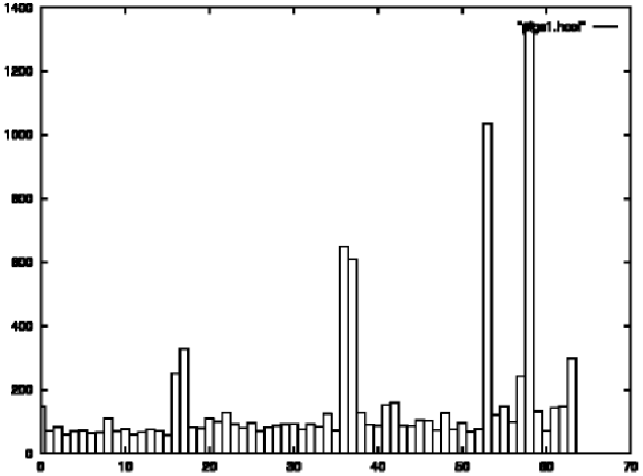
- Histogram is fast and easy to compute.

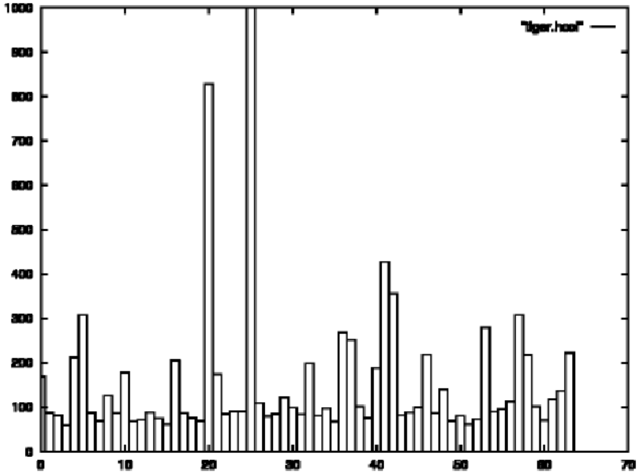
- Size can easily be normalized so that different image histograms can be compared.

- Can match color histograms for database query or classification.

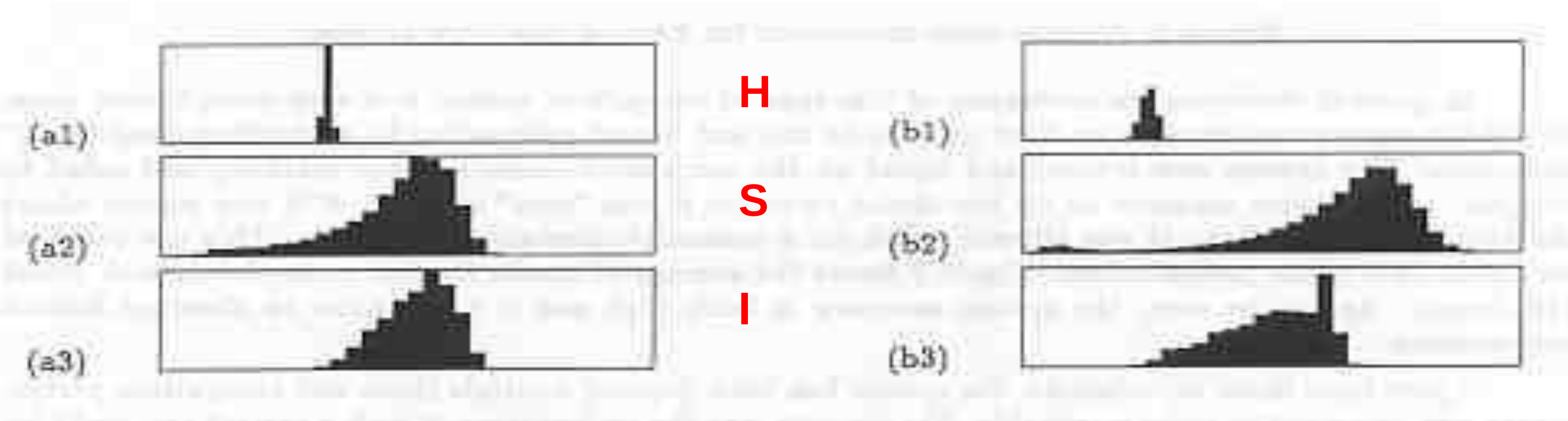
Histograms of two color images







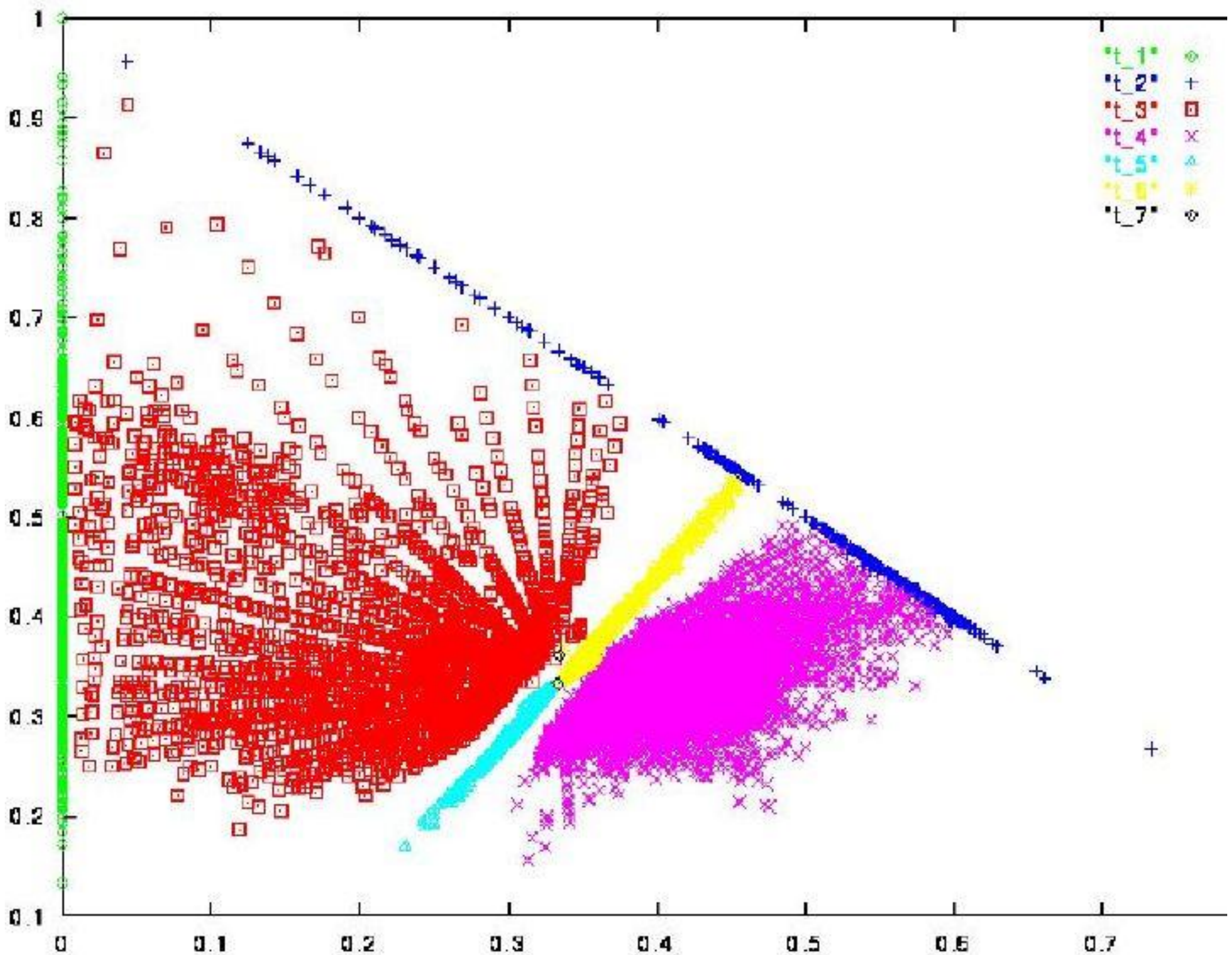
Apples versus Oranges



Separate HSI histograms for apples (left) and oranges (right) used by IBM's VeggieVision for recognizing produce at the grocery store checkout station (see Ch 16).

Skin color in Normalized R-G Space

G normalized



R normalized

Purple region shows skin color samples from several people. Blue and yellow regions show skin in shadow or behind a beard.

Finding a face in video frame

- (left) input video frame
- (center) pixels classified according to RGB space
- (right) largest connected component with aspect similar to a face (all work contributed by Vera Bakic)

Swain and Ballard's Histogram Matching for Color Object Recognition (IJCV Vol 7, No. 1, 1991)

Opponent Encoding:

- wb = R + G + B
- rg = R - G
- by = 2B - R - G

Histograms: 8 x 16 x 16 = 2048 bins

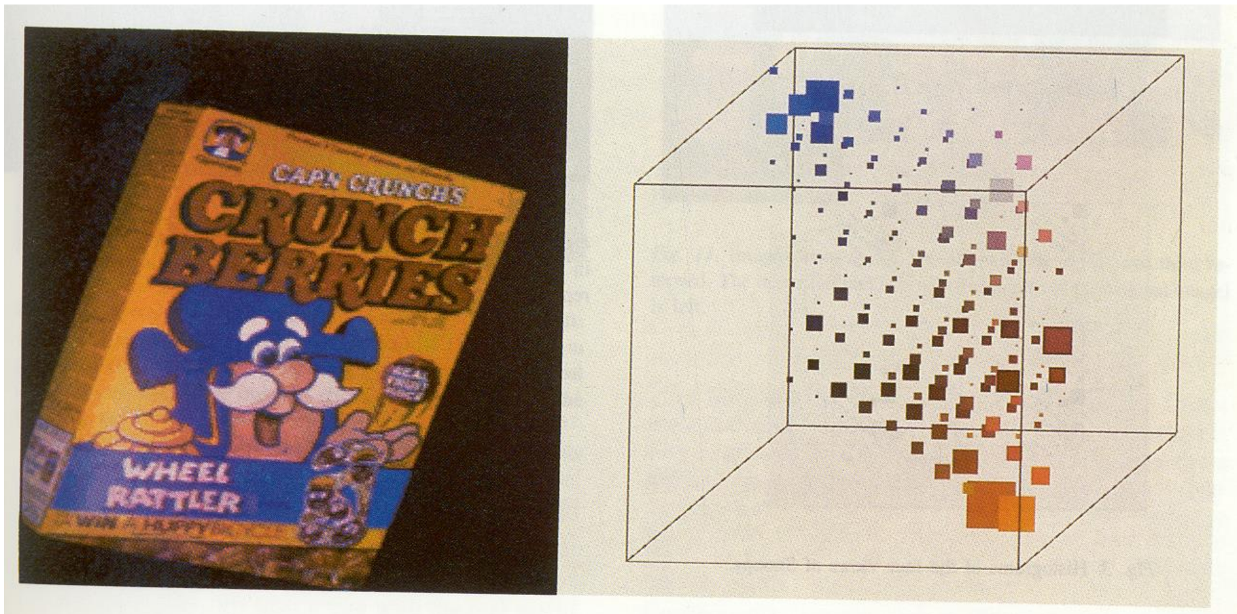
Intersection of image histogram and model histogram:

$$\mathrm{intersection}(h(I),h(M)) = \sum_{j=1}^{\mathrm{numbins}} \min\{h(I)[j],h(M)[j]\}$$

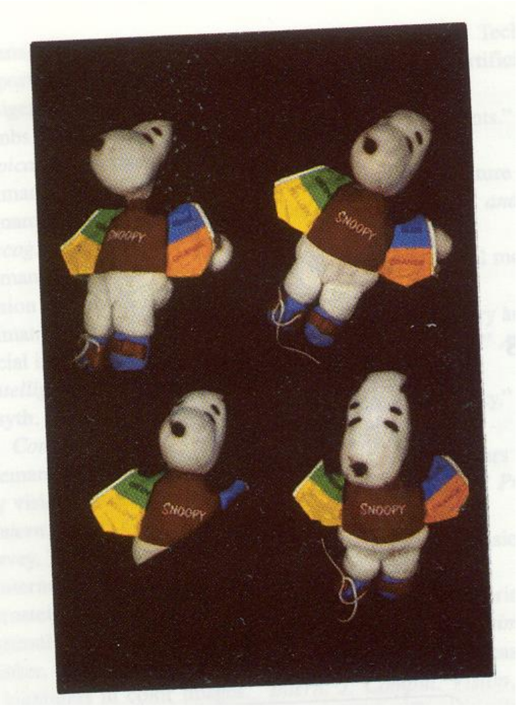
Match score is the normalized intersection:

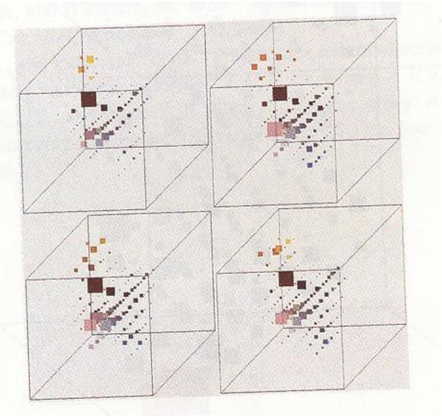
$$\mathrm{match}(h(I),h(M)) = \mathrm{intersection}(h(I),h(M)) \,/\, \sum_{j=1}^{\mathrm{numbins}} h(M)[j]$$

(from Swain and Ballard)



cereal box image 3D color histogram

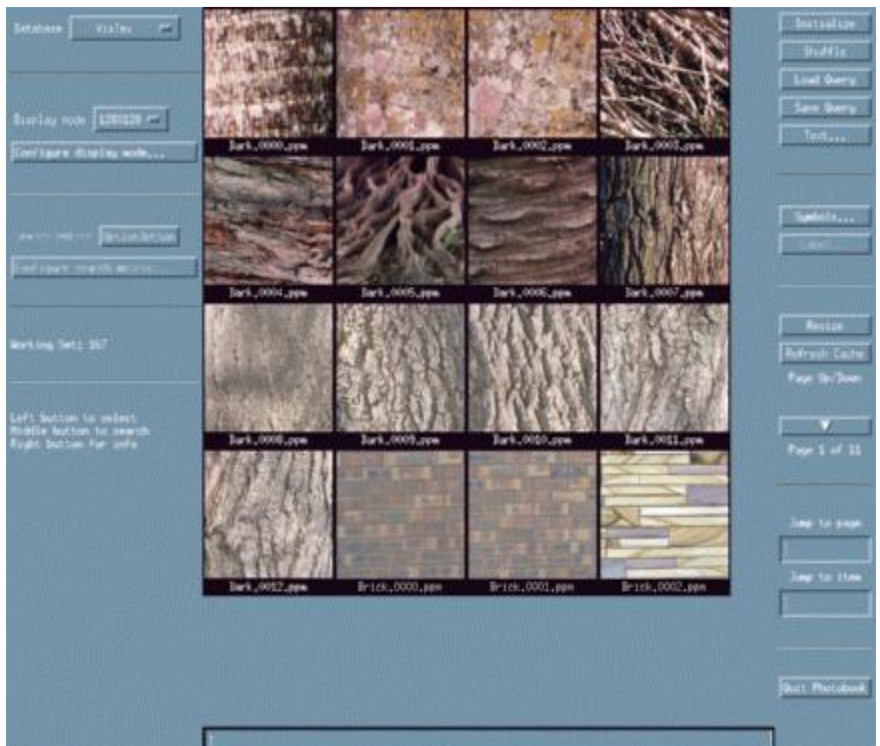




Four views of Snoopy Histograms

Texture

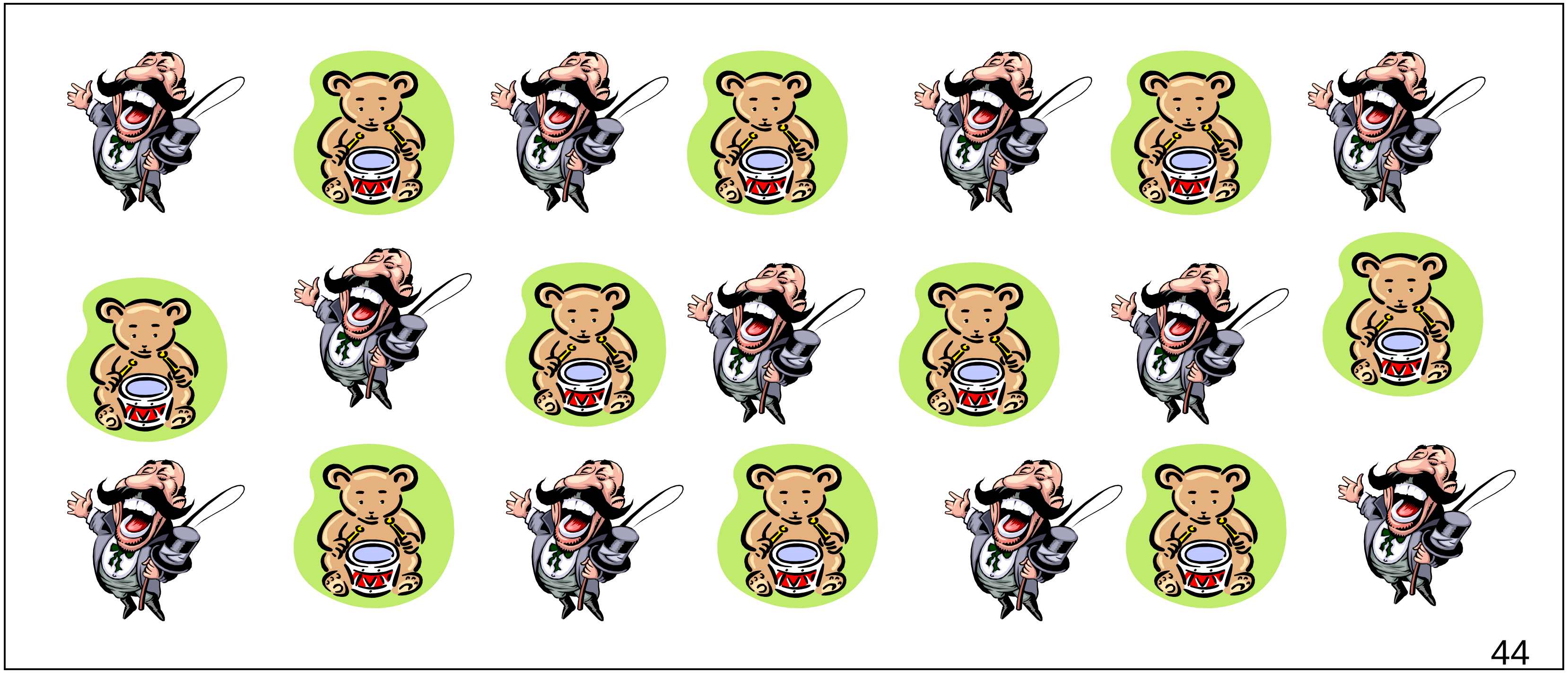
- Color is well defined.
- But what *is* texture?



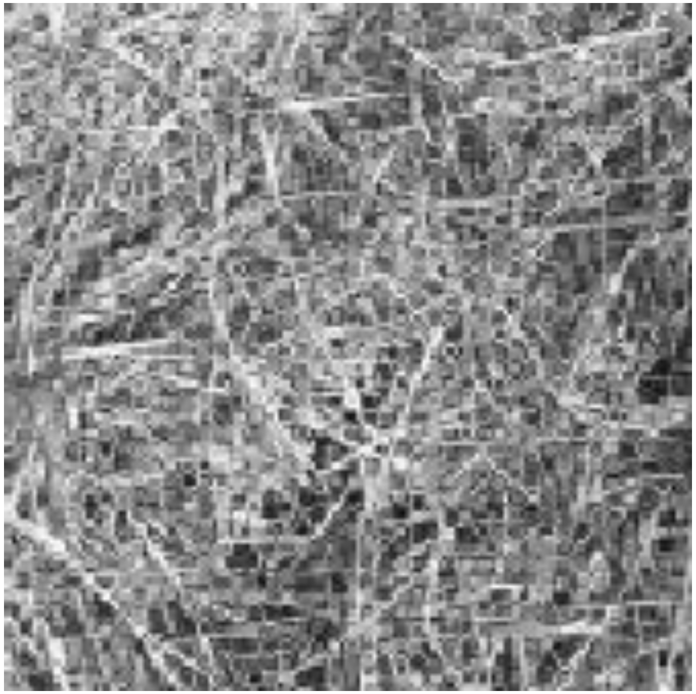
Structural Texture

Texture is a description of the spatial arrangement of color or intensities in an image or a selected region of an image.

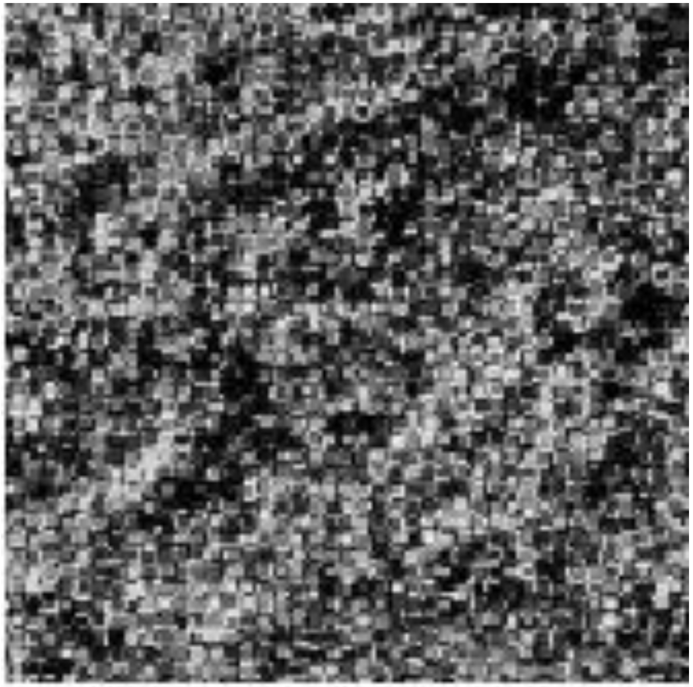
Structural approach: a set of texels in some regular or repeated pattern



Natural Textures from VisTex



grass



leaves

What/Where are the texels?

The Case for Statistical Texture

- Segmenting out texels is difficult or impossible in real images.
- Numeric quantities or statistics that describe a texture can be computed from the gray tones (or colors) alone.
- This approach is less intuitive, but is computationally efficient.
- It can be used for both classification and segmentation.

Local Binary **Pattern** Measure

- For each pixel p, create an 8-bit number $b_1\ b_2\ b_3\ b_4\ b_5\ b_6\ b_7\ b_8$, where $b_i = 0$ if neighbor i has value less than or equal to p's value and 1 otherwise.

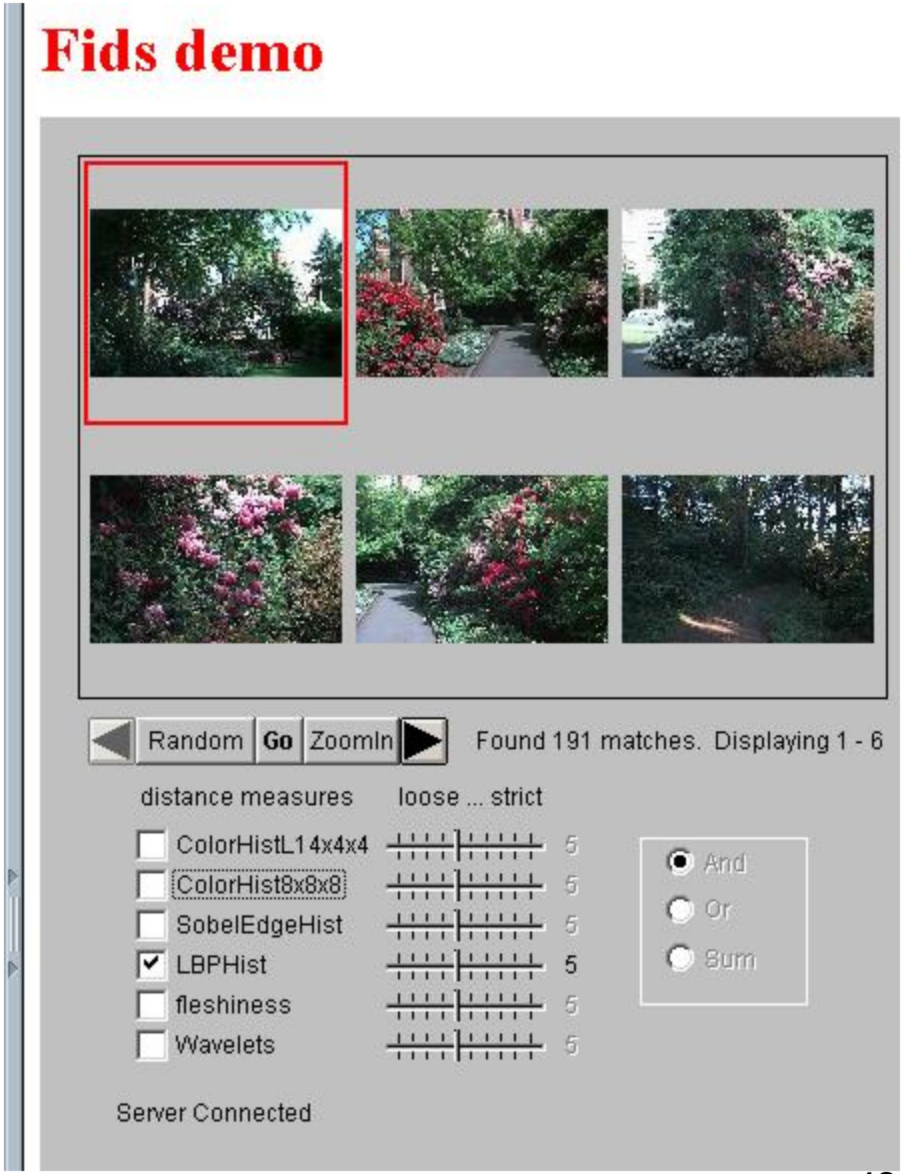
- Represent the texture in the image (or a region) by the histogram of these numbers.

	1	2	3	
	100	101	103	
8	40	50	80	4
	50	60	90	5
	7	6		

$\longrightarrow$ 1 1 1 1 1 1 0 0

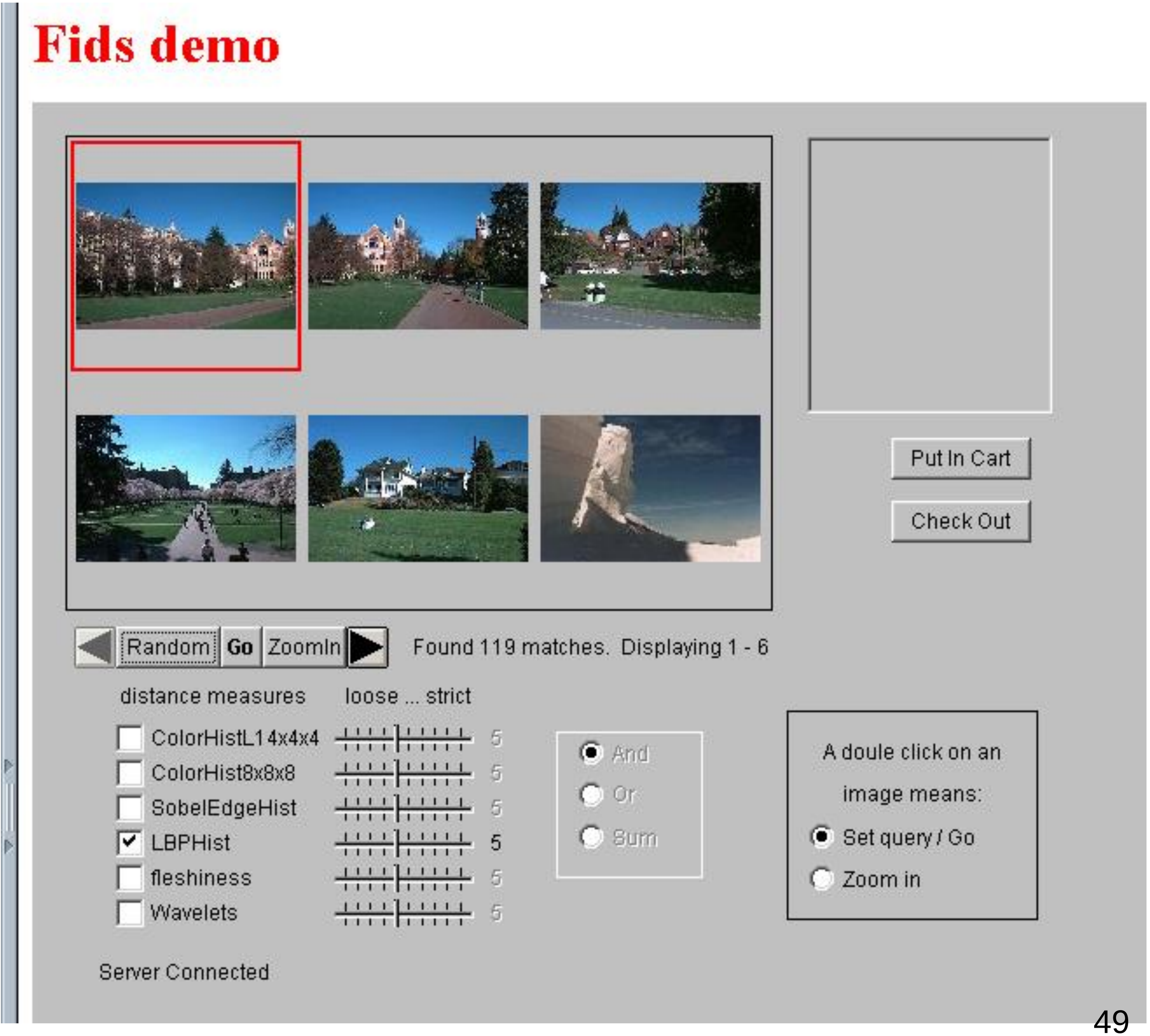
Example

Fids (Flexible Image Database System) is retrieving images similar to the query image using LBP texture as the texture measure and comparing their LBP histograms



Example

Low-level measures don't always find semantically similar images.



What else is LBP good for?

- We found it in a paper for classifying deciduous trees.
- We used it in a real system for finding cancer in Pap smears.
- We are using it to look for regions of interest in breast and melanoma biopsy slides.

Co-occurrence Matrix Features

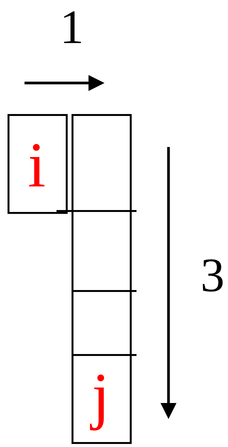
A co-occurrence matrix is a 2D array C in which

- Both the rows and columns represent a set of possible image values.
- $C_d(i,j)$ indicates how many times value i co-occurs with value j in a particular spatial relationship d.
- The spatial relationship is specified by a vector d = (dr,dc).

Co-occurrence Example

$$\begin{array}{cccc} 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \\ 0 & 0 & 2 & 2 \end{array}$$

gray-tone image



$d = (3,1)$

$$C_d = \begin{array}{c|ccc} & 0 & 1 & 2 \\ \hline 0 & 1 & 0 & 3 \\ 1 & 2 & 0 & 2 \\ 2 & 0 & 0 & 1 \end{array}$$

co-occurrence matrix

From C_d we can compute N_d, the normalized co-occurrence matrix, where each value is divided by the sum of all the values.

Co-occurrence Features

What do these measure?

$$Energy = \sum_i \sum_j N_d^2(i,j) \quad (7.7)$$

$$Entropy = -\sum_i \sum_j N_d(i,j) log_2 N_d(i,j) \quad (7.8)$$

$$Contrast = \sum_i \sum_j (i-j)^2 N_d(i,j) \quad (7.9)$$

$$Homogeneity = \sum_i \sum_j \frac{N_d(i,j)}{1+ \mid i-j \mid} \quad (7.10)$$

$$Correlation = \frac{\sum_i \sum_j (i-\mu_i)(j-\mu_j) N_d(i,j)}{\sigma_i \sigma_j} \quad (7.11)$$

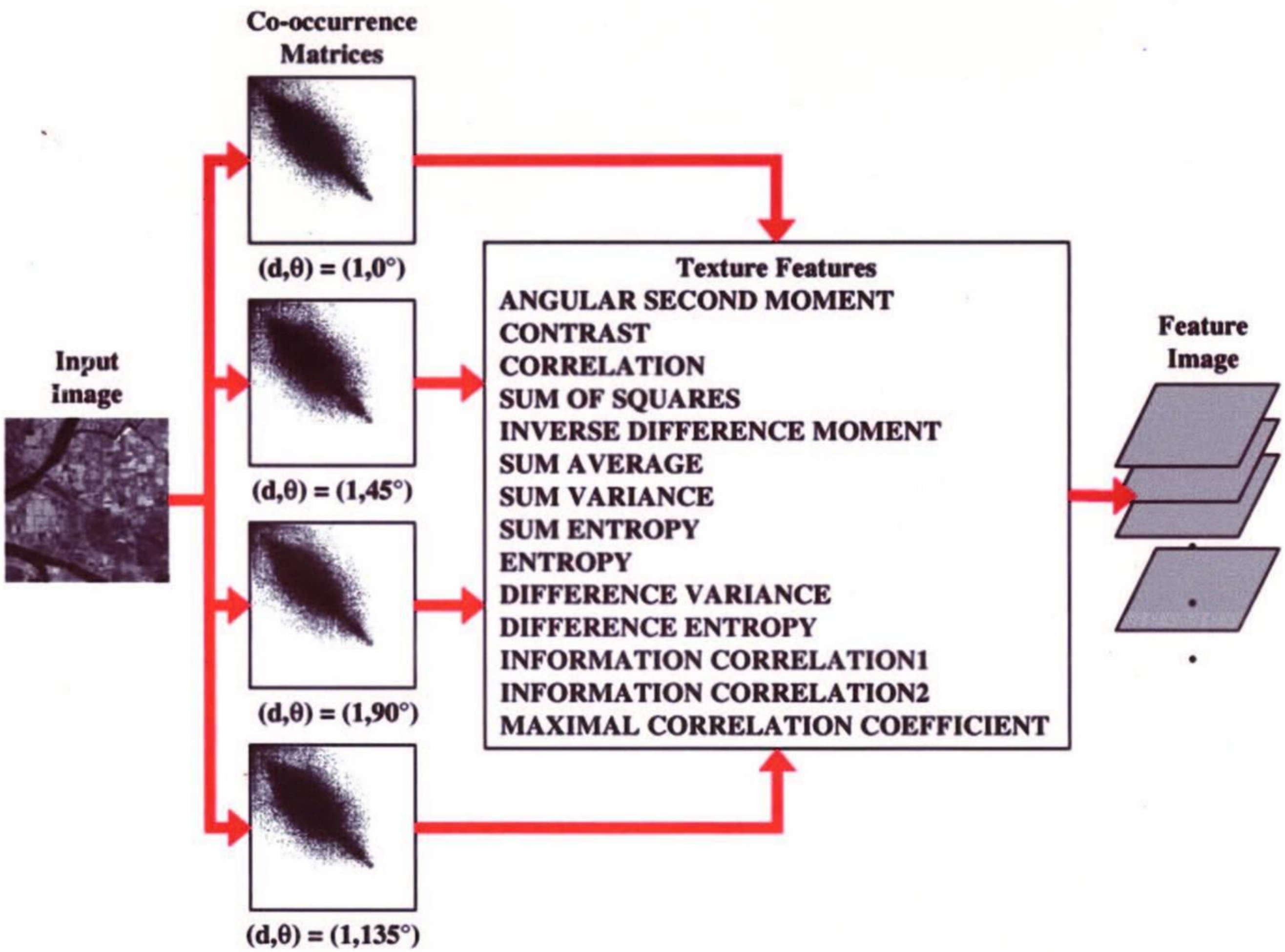
where μ_i, μ_j are the means and σ_i, σ_j are the standard deviations of the row and column sums.

$\boxed{\text{Energy measures uniformity of the normalized matrix.}}$

What are Co-occurrence Features used for?

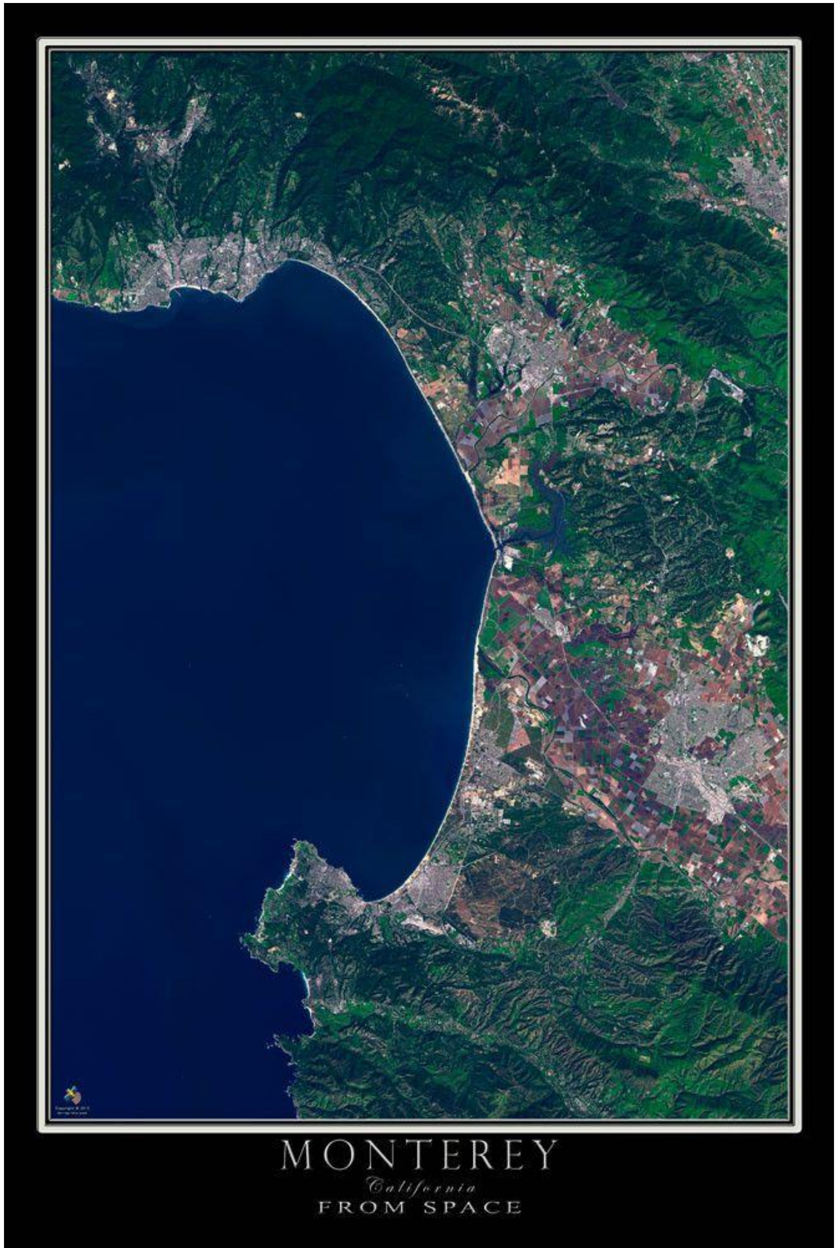
- They were designed for recognizing different kinds of land uses in satellite images.
- They are still used heavily in geospatial domains, but they can be added on to any other calculated features.

Example



Co-occurrence Matrices
Input Image
(d,θ) = (1,0°)
(d,θ) = (1,45°)
(d,θ) = (1,90°)
(d,θ) = (1,135°)
Texture Features
ANGULAR SECOND MOMENT
CONTRAST
CORRELATION
SUM OF SQUARES
INVERSE DIFFERENCE MOMENT
SUM AVERAGE
SUM VARIANCE
SUM ENTROPY
ENTROPY
DIFFERENCE VARIANCE
DIFFERENCE ENTROPY
INFORMATION CORRELATION1
INFORMATION CORRELATION2
MAXIMAL CORRELATION COEFFICIENT
Feature Image

Satellite Image of Monterey Bay



Assignment 0

FUN WITH COLOR!

Image basics

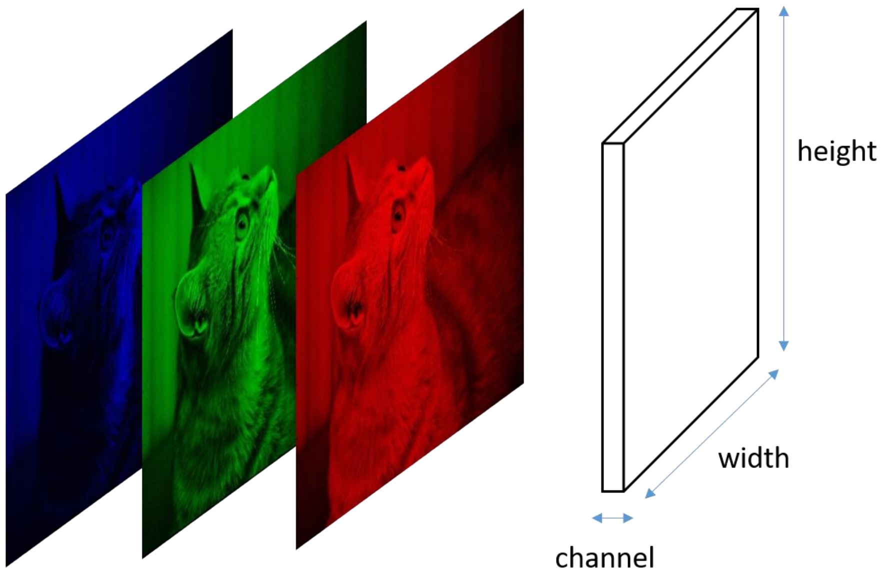
- Data structure for an image

```
typedef struct{
        int h, w, c;
        float *data;
} image;
```

Image basics

- Data structure for an image

```
typedef struct{
        int h, w, c;
        float *data;
} image;
```

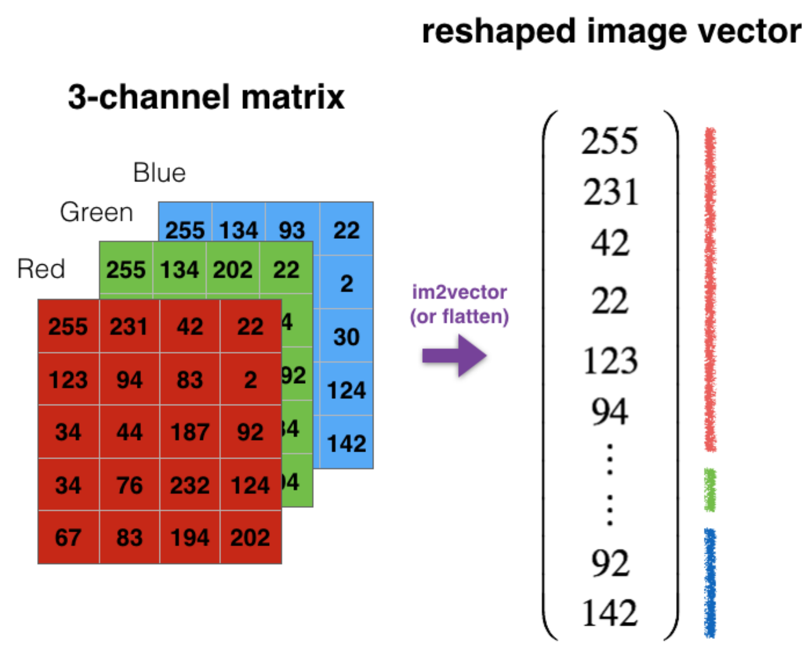


  - h = height
  - w = width
  - c = number of channels
    - c = 3 for RGB image; c = 1 for grayscale image

Image basics

- Data structure for an image

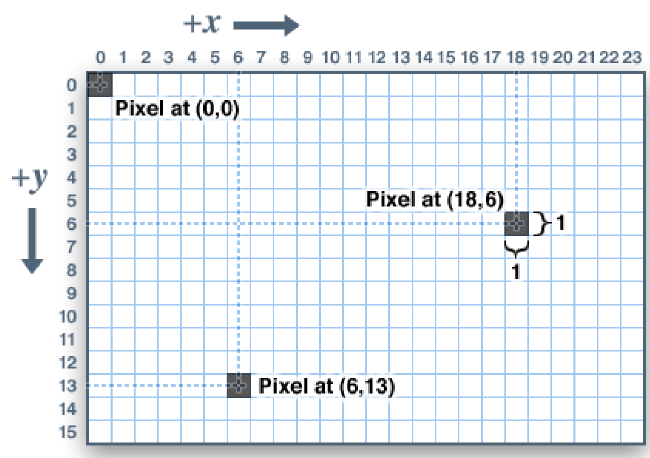
```
typedef struct{
        int h, w, c;
        float *data;
} image;
```



  - data = array of floats
  - floats in the range [0,1]
  - 3D image matrix linearized into 1D array

To Do #1

- Fill in:
  - `float get_pixel(image im, int x, int y, int c)`



  - `void set_pixel(image im, int x, int y, int c, float v)`
    - assign value `v` to `im.data[coord(x,y,c)]`

To Do #2

- Fill in:
  - `image copy_image(image im)`
    - create a separate image that is a copy of the input image `im` and return the copy
    - Useful functions:
      - `make_image()` in load_image.c
      - `memcpy()`
    - You can use `get_pixel()` and `set_pixel()` functions you implemented from now on

To Do #3

- Fill in:
  - `image rgb_to_grayscale(image im)`
    - create a new image with 1 channel
    - Get R,G,B values from input image `im`
    - Compute Y = 0.299 R + 0.587 G + .114 B
    - Assign Y to new image and return that image

To Do #4

- Fill in:
  - `void shift_image(image im, int c, float v)`

    - add value `v` to each pixel of `im` in channel `c`

    - change the input image in-place (do not create a separate image) wherever the return type of the function is `void`

To Do #5

- Fill in:
  - `void clamp_image(image im)`
    - clamp the pixel values of input image `im` to be in the range [0,1]

To Do #6

- Fill in:
  - `void rgb_to_hsv(image im)`
    - Convert R,G,B values of image `im` pixelwise into H,S,V values using the given formulas
    - You can use the `three_way_max()` and `three_way_min()` functions provided

To Do #7

- Fill in:
  - `void hsv_to_rgb(image im)`
    - Convert H,S,V values of image `im` pixelwise into R,G,B values using the given table

To Do #8 (extra credit)

- Fill in:
  - `void scale_image(image im, int c, float v)`
    - multiply each pixel of `im` in channel `c` with value `v`
    - add to the lines in image.h and other files as necessary to test the function. No need to submit the other edited files; write a comment in CAPS at the top of process_image.c if you attempt any extra credit

To Do #9 (super extra credit)

- Fill in:
  - `void rgb_to_lch(image im)`
    - Convert R,G,B values of image `im` pixelwise into L,C,H values using the formulas in the given link

Have Fun