Computer Science 631
Lecture 7: Colorspace, local operations

Ramin Zabih
Computer Science Department
CORNELL UNIVERSITY

Outline

- Color and surfaces
- How color is encoded in images
- Fast local operations
  - Box filtering
  - Crow’s algorithm

Color and surfaces

- From a physics point of view, a photon hits a surface, and (perhaps) a photon is emitted
  - Each photon has a wavelength and direction
  - For a small surface patch we establish a local polar coordinate system, relative to the surface normal

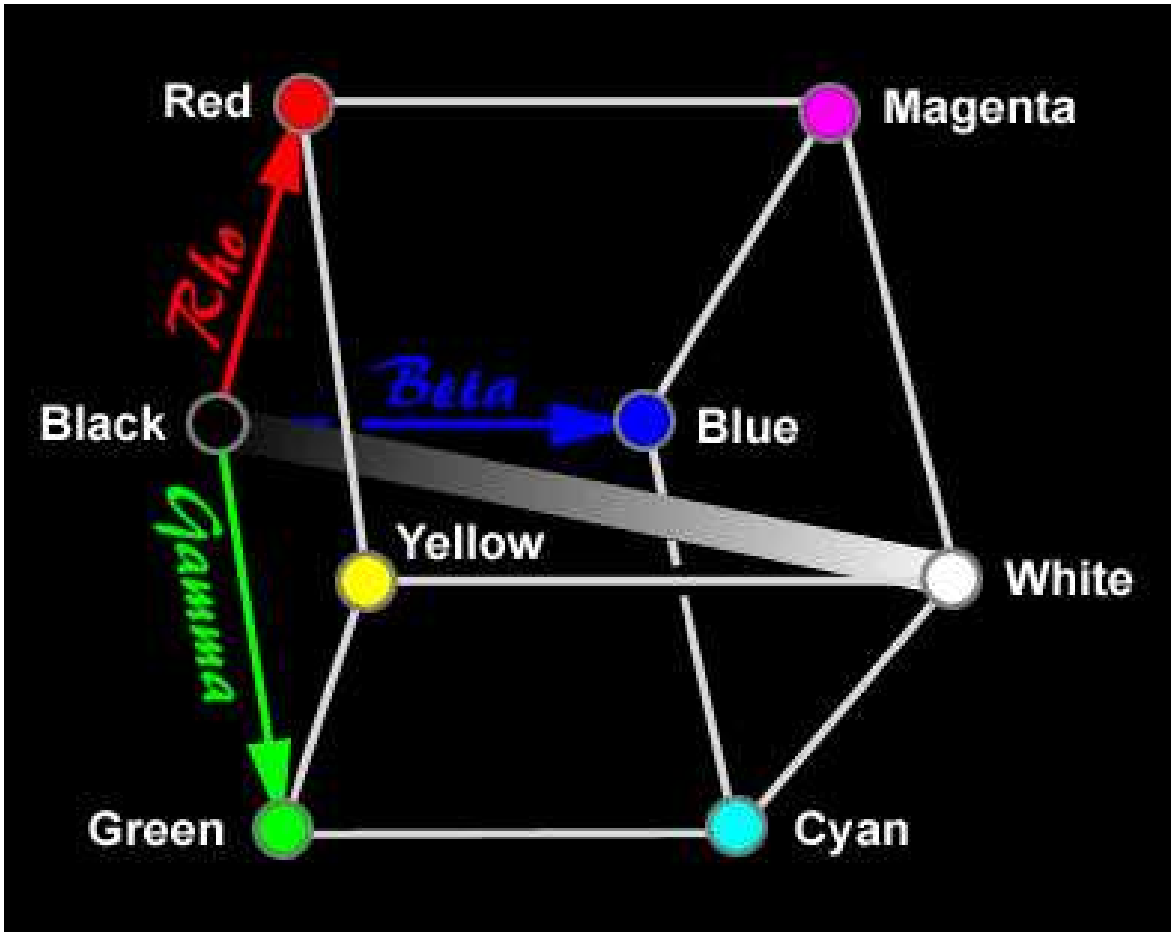
BRDF's map input energy to output

- Think of the brightness as the output energy
- A Bidirectional Reflectance Distribution Function (BRDF) specifies the ratio of output energy to input energy
  - As a function of the input and output photon directions

Specular plus diffuse components

- The true BRDF for a surface is very complex
- A common simplifying assumption is that there are two components
  - A *diffuse* component is uniform in all directions
  - A *specular* component covers “highlights”
    - Model the surface patch as a mirror
    - Incident angle = outgoing angle
  - There are (many) more complex models

RGB color space



Another way to think about color

- RGB maps nicely onto the way monitors phosphors are designed
  - Cameras naturally provide something like RGB
  - 3 different wavelengths
- But there is a more natural way to think about color
  - Hue, saturation, brightness

Hue, saturation and brightness

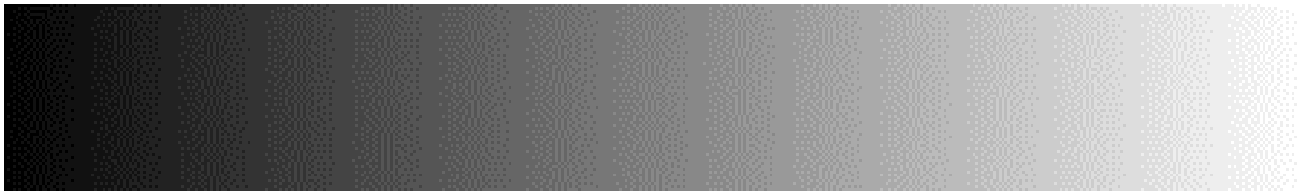
H

dominant wavelength

S

purity

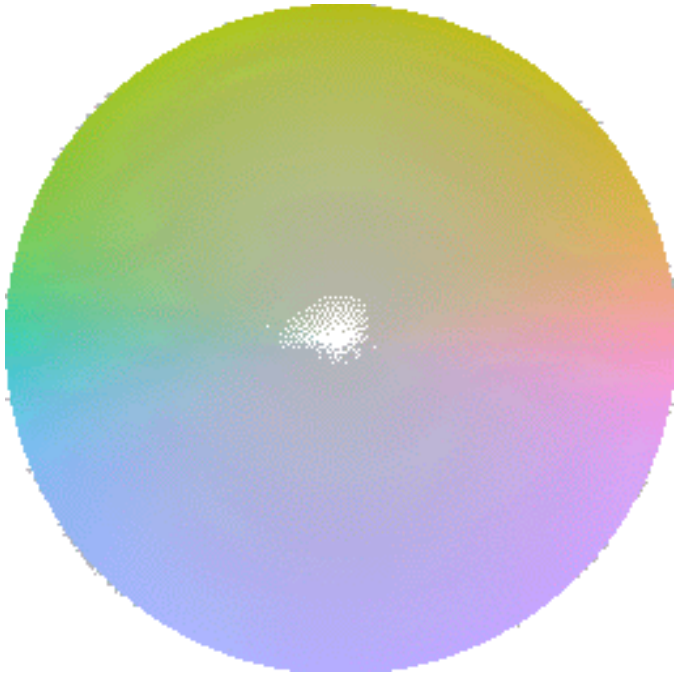
% white



B

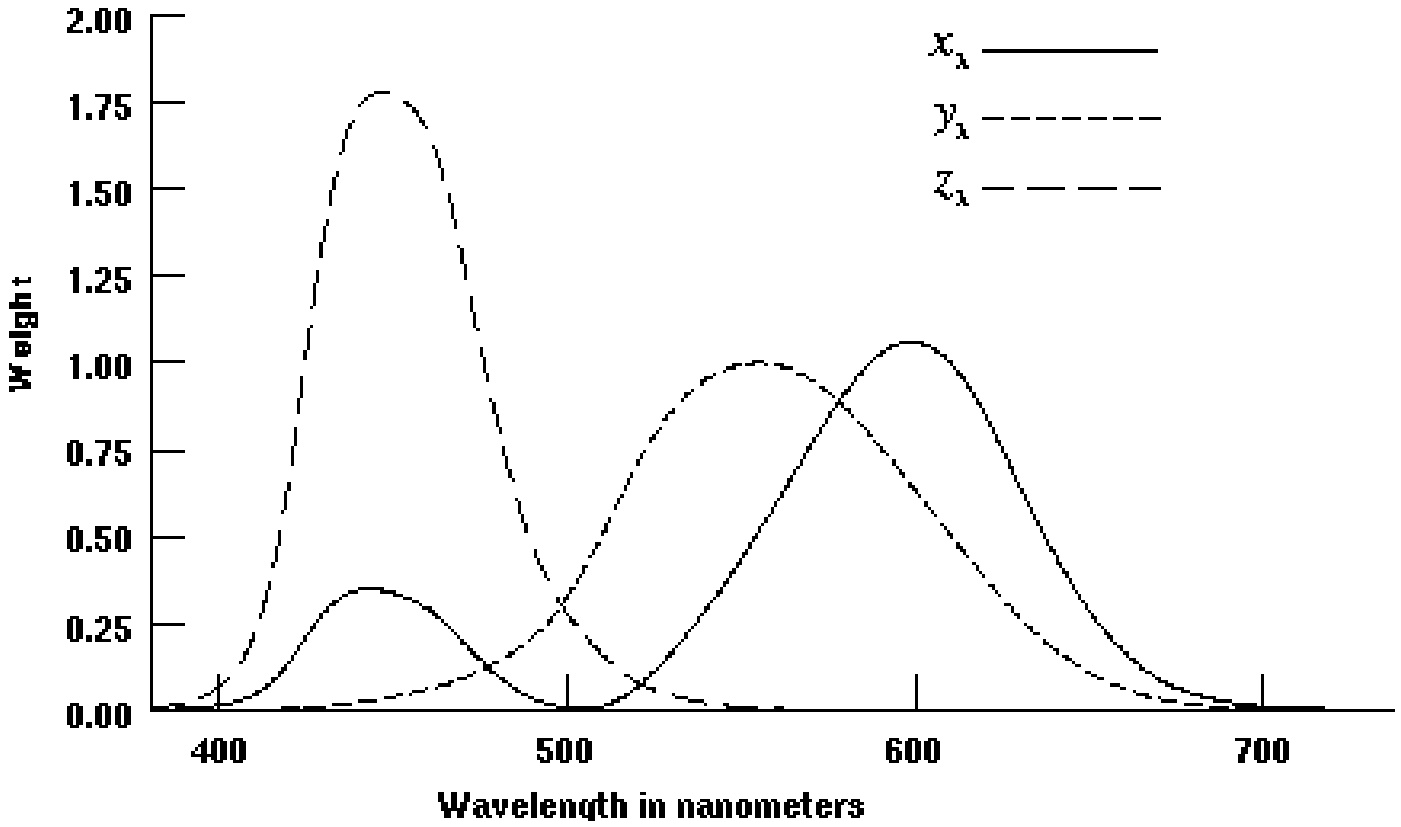
luminance

Color wheel (constant brightness)



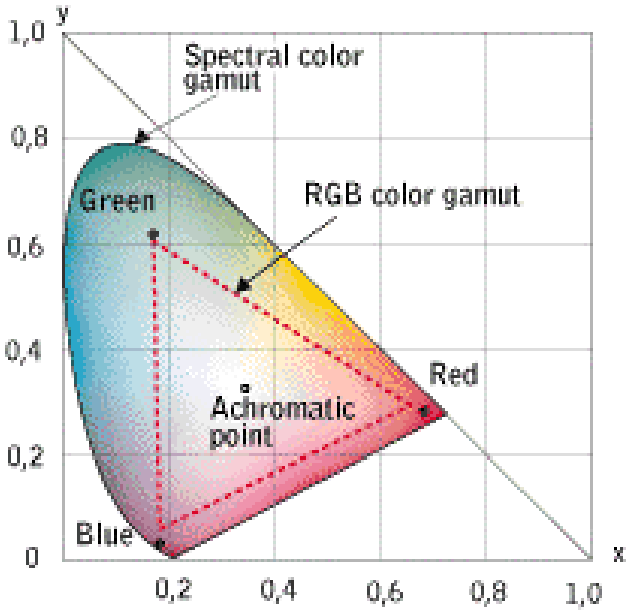
In this view of color,
there is a color *cone*
(this is a cross-section)

CIE colorspace



CIE color chart

- X+Y+Z is more or less luminosity
  - Let’s look at the plane X+Y+Z = 1



CIE chromaticity diagram properties

- Pure wavelengths along the edges, white in the center (almost)
- Adding two colors gives a new one along the line between them
  - This makes it easy to compute the dominant wavelength and %white of a given color
    - Note that we are looking at a constant luminance “slice”
  - Allows computation of complements
    - What about colors with no complement? (non-spectral)

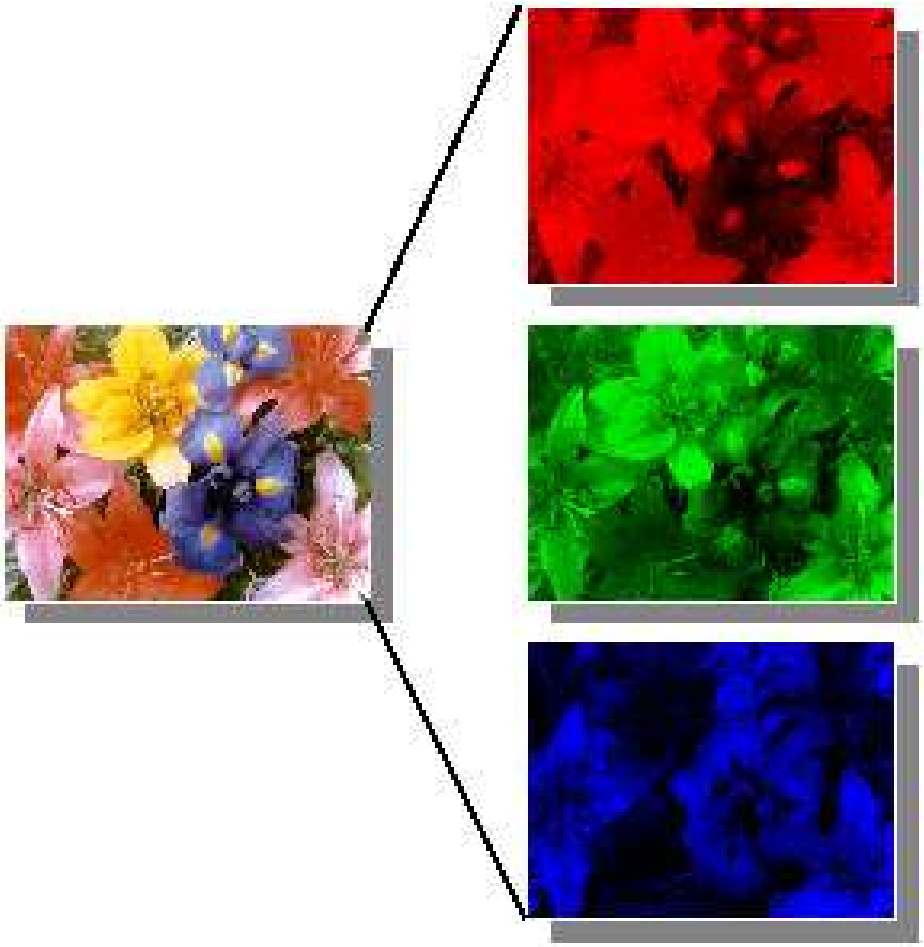
Gamuts

- Start with three colors (points on CIE chart)
  - Which colors can be displayed by adding them?
  - The triangle is called the gamut
- The RGB gamut isn’t very big
  - So, there are lots of colors that your monitor cannot display!

Perceptual uniformity

- The CIE XYZ colorspace is not perceptually uniform
  - Due to changes in JND as a function of wavelength
- In 1976 the CIE LUV colorspace was defined
  - L is more or less brightness, and is non-linearly related to Y
  - u,v linear scaled versions of X,Y

RGB example



YIQ colorspace (used in NTSC)

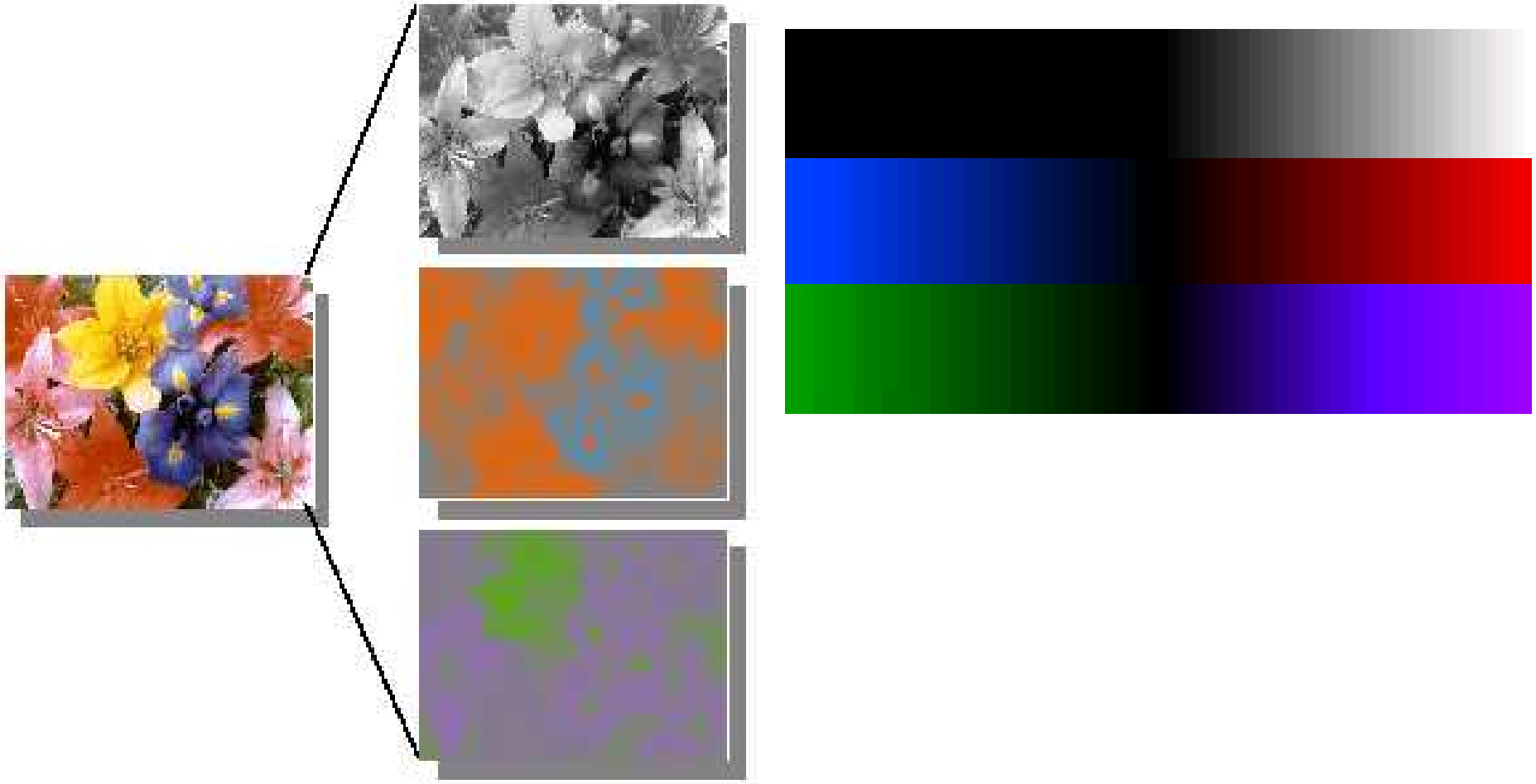
- Basic idea: Y is luminance, I and Q are in descending order of importance
- Y lies along the diagonal in the RGB cube

  $Y = 0.299\,R + 0.587\,G + 0.114\,B$
- For the other two vectors we use

  $I = 0.596\,R - 0.275\,G - 0.321\,B$

  $Q = 0.212\,R - 0.528\,G + 0.311\,B$
- I axis lies along red-orange, Q at a right angle

YIQ example



CCIR 601

- 1982 digital video standard
  - Based on fields (even and odd)
  - Colorspace is Y Cr Cb = Y U V

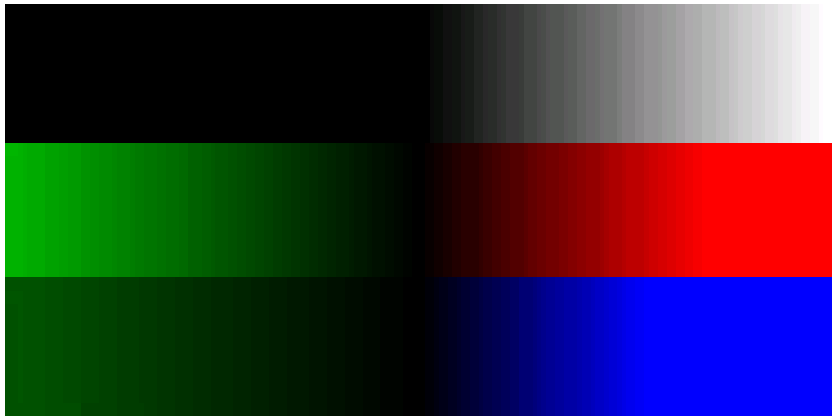
$$Y = 0.299\,R + 0.587\,G + 0.114\,B$$

$$U = k1(R - Y)$$

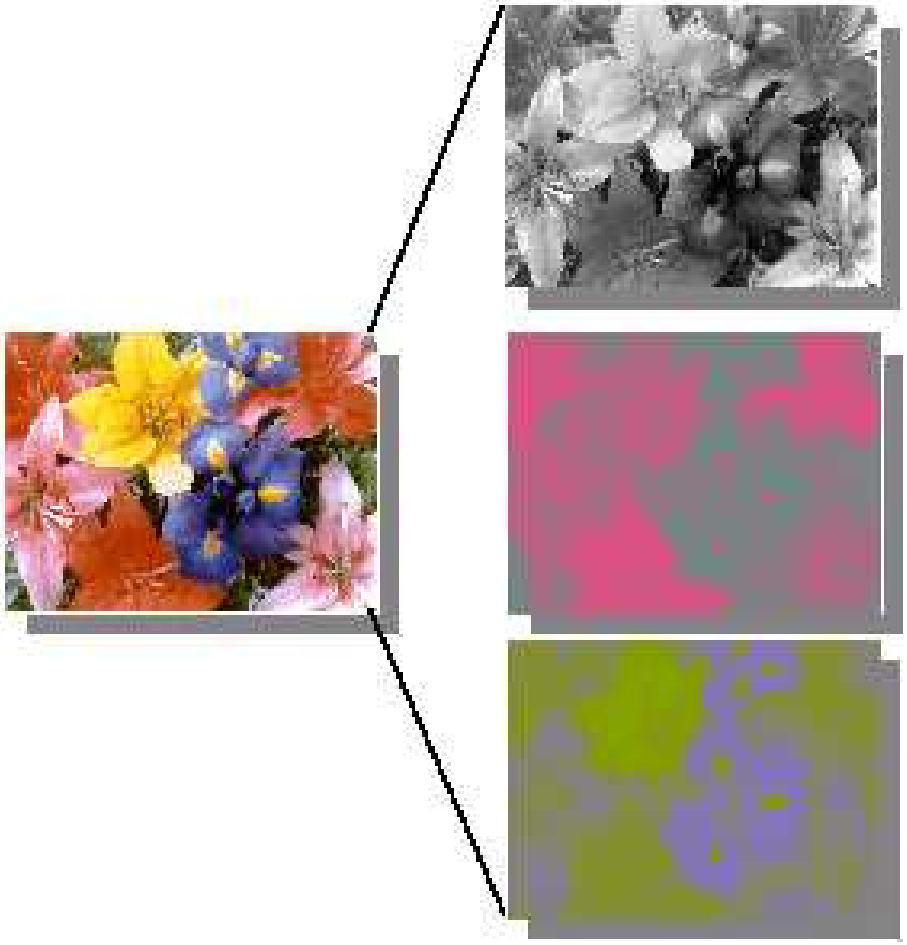
$$V = k2(B - Y)$$

CCIR 601 image sizes

- Luminance (Y) is 720 by 243 at 60 hertz
- Chrominance is 360 by 243
  - Split between U and V (alternate pixels)
- Two cables for SVHS!



YUV example



Local operations

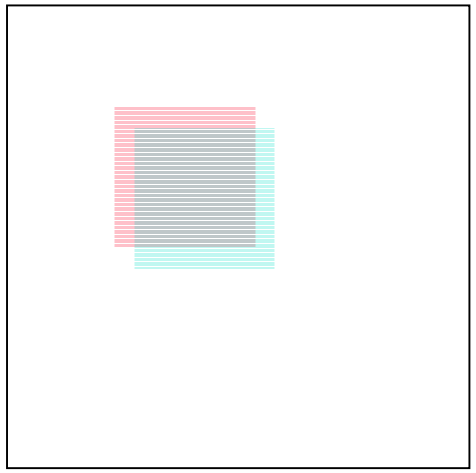
- Most image distortions involve
  - Coordinate changes
  - Color
  - Different spatial frequencies
- These last class of distortions center on local operations
  - Every pixel computes some function of its local neighborhood (window)
  - We will assume a square of radius *r*

Uniform local operations

- Many operations involve computing the sum over the window
  - Obvious example: local averaging
  - Convolution (weighted average)
  - Less obvious: median filtering, or any other local order operation
  - There are some tricks to make these fast!

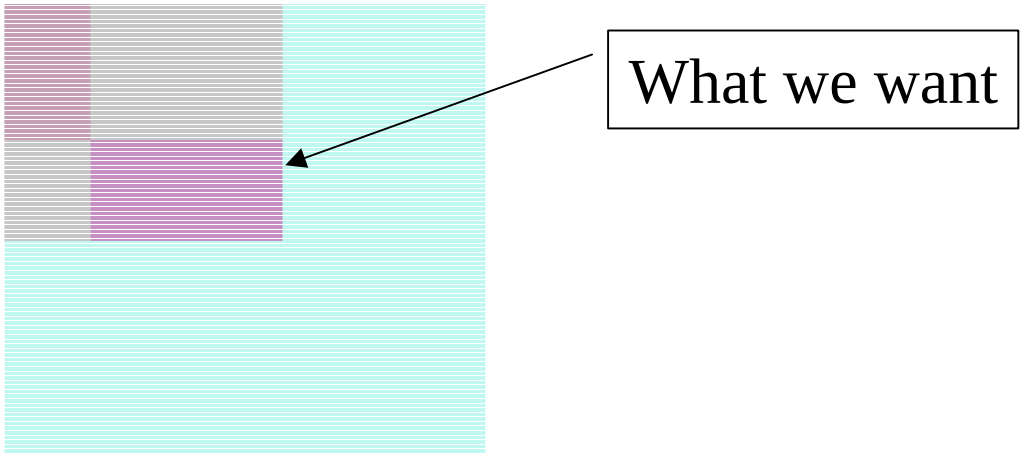
Local averaging as an example

- Assume that we process the image in a fixed order (row major)
- There is a lot of repeated work involved!
  - For example, sum in red versus green area



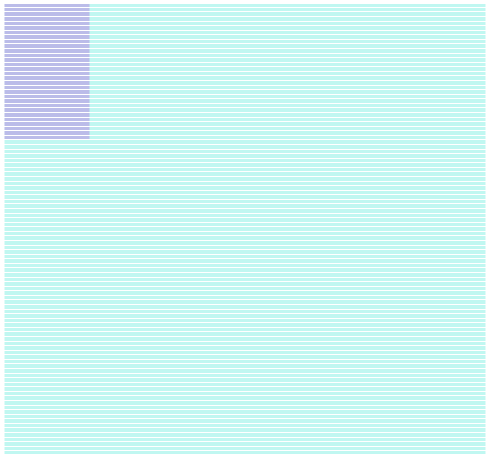
Crow's method (1984)

- With some simple pre-processing, we can compute the sum in *any* rectangle very rapidly
  - Add the purple, subtract the yellows



Preprocessing step

- At every pixel (x,y), we will compute the sum of the intensities in the rectangle (0,0,x,y)



This step can also be sped up

- Consider the problem of computing the "next" rectangle sum
  - It's the old rectangle sum plus a column
  - That column is the rectangle sum directly above, minus the rectangle sum to its left

rect[x,y] = rect[x-1,y] + col[x,y]

col[x,y] = col[x,y-1] + I[x,y]

col[x,y-1] = rect[x,y-1] - rect[x-1,y-1]

Sliding sums

- There is a similar trick for computing the sum in all fixed-size rectangles
  - Exactly what we need for local averaging
  - To get the new sum, start with the old,
  - Then add (at right) and subtract (left) a column sum
  - To get a new column sum, take the column sum directly above
    - Then add (below) and subtract (above) an intensity