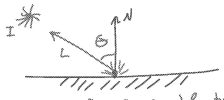
Surface reflectance

* Relate light source intensity to reflected light intensity
* Lambertian surface → Diffuse reflection

$\cos\theta = N \cdot L$

$N \cdot L < 0 \Rightarrow$ surface not visible



surface albedo $\in [0, 1]$ (reflection coefficient)

$$I_{ref} = I \cdot \rho \cdot \cos\theta = I \cdot \rho \cdot (N \cdot L)$$

I_{ref}: intensity of reflection; I: intensity of source

Lambertian surface --> when source of light falls on the surface, the light is reflected unformly in all direction - >it does not matter where the viewer is.

specular reflection -->like a mirror reflection, the viewer position changes what they see

Lambertian surface-->reflects light in accordance with lambert's cosine law

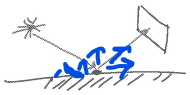
surface albedo-->how good the surface is at reflection.

N is the vector normal to the surface to give the surface orientation
I is the vector in the direction of source

Radiosity model

* Relate light in the scene (surface radiance) to light in the image (image irradiance)

$L(p)$ = Power of light per unit area reflected from surface (surface radiance)



$E(p)$ = power of light per unit area received at the image (image irradiance)

Fundamental equation of radiometric image formation

$$E(p) = L(p) \frac{\pi}{4} \left(\frac{d}{f}\right)^2 (\cos\alpha)^4$$

$E(p)$: light at image; $L(p)$: light at surface; d: diameter of lens; f: focal length; α: angle between principal axis and surface normal

$d \uparrow \Rightarrow E(p) \uparrow$

$f \uparrow \Rightarrow E(p) \downarrow$

$\alpha \uparrow \Rightarrow E(p) \downarrow$

Finding E(p) through L(p)

Alpha is the angle between the camera principal axis and the normal vector of the surface

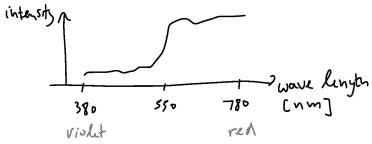
camera/viewer

Cos (α) = V.N

If the viewer is directly above the surface, alpha=0 making E(p) increase, receiving more light

Color

* visible light: 380 - 780 nm
  - higher frequency
  - lower frequency
* e.g. Brown banana



* Human vision: R, G, B receptors

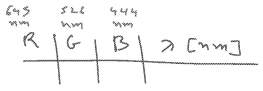
The color brown comes from the combination of multiple wavelengths

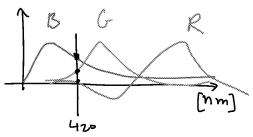
How are these wavelengths matched to the RGB values?

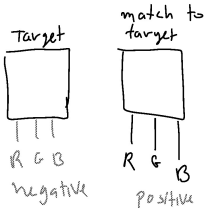
It was done by experimentation, trying various values for RGB and try to bring out the same color so as to match the physical world.

CIE tables

* Map wavelength to RGB intensities





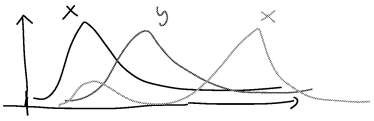


negative values are added to target to help match

Hypothetical light source

use X,Y,Z instead of R,G,B so that negative weights are not necessary.

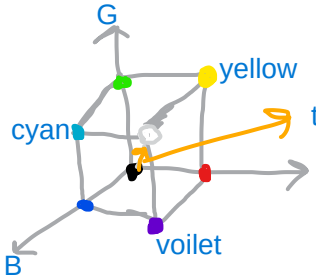
Removing negatives



$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.49 & 0.31 & 0.20 \\ 0.177 & 0.813 & 0.011 \\ 0.00 & 0.01 & 0.99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

RGB Colorspace

RGB - Additive
CMYK - subtractive



the line connecting - black to white-->grayscale

Please see Notes doc for more color scheme types like NTSC, LAB

CIE - LAB

*Euclidean distance in RGB space does not correspond to human perception.

whereas Euclidean distance in LAB space does correspond to perception.

RGB → $L^* a^* b^*$ → compare colors

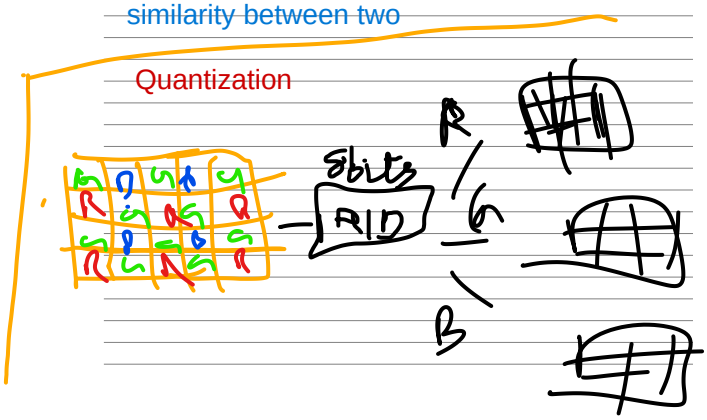
$|c_1 - R| < |c_2 - R|$

⇓

c_1 is more similar to R (eg. skin color)

This is used whenever we want to measure the color similarity between two

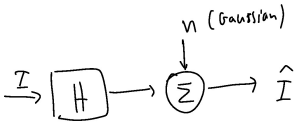
Quantization



The world is continuous but we get a discrete representation(2000x2000 image) from a discrete sensor (which has predominant green)-->quantization --->spatial quantization- can lead to loss of detail or resolution because continuous variations are approximated by discrete values. This can result in artifacts like pixelation in images.

Noise

- sampling noise (aliasing)
- color quantization
- Noise model:



color quantization--> although there are multiple ranges of intensities in real world, the camera can sense only a few intensities (8 bits-->256 green,256, red, 256 blue). The analog values of the intensities are converted using the 8 bit analog to digital converter in sensor. This results in converting the voltage values t0 between 0 and 1.

On the whole, number of color combination will be 256^3

Apart from the noise from quantization (electrically), we have noise/distortion when the image is taken in low light, because the camera is trying to count the number of photons and there are less in the dim lighted photo.

Signal to noise ratio

Strong Variance

$$SNR = \frac{E_s}{E_n} = \frac{\sigma_s^2}{\sigma_n^2} = \frac{\frac{1}{n}\sum_{i,j}\left(I(i,j) - \bar{I}\right)^2}{\sigma_n^2}$$

σ_n^2 = variance for multiple frames of a static scene

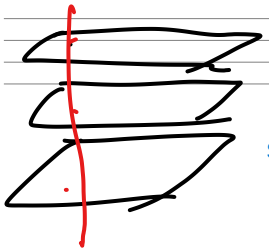
OR

variance in a uniform image region

To find how good a camera is or comapre 2 cameras, we use SNR

The Variance of signal is found by subtracting every pixel intensity with the average value and sum it and then divide by n(no.of pixels) for average = np.var()

The variance of noise is calculated by taking multiple picture of the same static scene and see the different values of the pixels at the same point. Or by taking a uniform area in the image and see the pixel difference in that area. refer hw2 1st question



see value diff at one point across pictures

SNR - generally calculated in db

SNR [db]

$$SNR[db] = 10 \log_{10} \frac{E_s}{E_n}$$

10 db $\Rightarrow$ E_s is 10 times larger than E_n this value is actually low, it should be greater generally

13 db $\Rightarrow$ E_s is 20 times larger than E_n

3 db $\Rightarrow$ E_s is 2 times larger than E_n poor quality

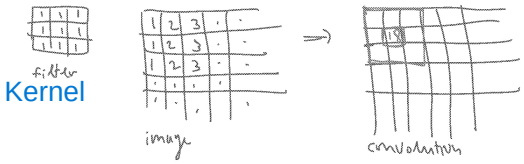
See Notes for a picture - Gaussian and impulsive noise

Noise filtering

- Remove noise using smoothing
- Smooth using convolution

$$I_A(i,j) = I(i,j) * A(i,j) = \sum_{h=-\frac{M}{2}}^{\frac{M}{2}} \sum_{k=-\frac{N}{2}}^{\frac{N}{2}} A(h,k)\, I(i-h, j-k)$$

(filter dimensions; convolution)



Kernel

convolution = Sum of the product of corresponding elements of the image and kernel

we can also do average convolution, by dividing the sum by the total number of elements -->this results in float values

if we dont take average, the image becomes more and more brighter with higher values as we convolve more

Convolution properties

$$f * g = g * f$$

$$f * (g * h) = (f * g) * h$$

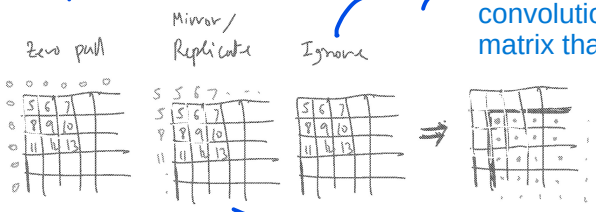
$$f * (g + h) = (f * g) + (f * h)$$

$$\frac{d}{dx}(f * g) = \frac{d}{dx} f * g = f * \frac{d}{dx} g$$

Convolution and derivative are linear, so it is possible to change the order

We add zeros to the edges and calculate the sum, disadvantage - we create artificial edges with black (0)

Convolution boundaries



We do not calculate the convolution in the edges, so the convolution result matrix will be smaller than the original one. If there are more number of convolution layers or if the kernel is bigger, then we keep shrinking the image matrix that results in very small matrix finally

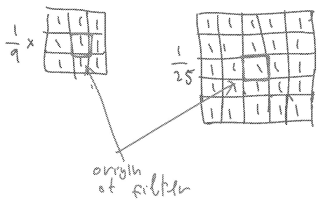
copy the nearby row and column values for edges instead of zeros. Disadvantage - more computational effort needed by the hardware

- store result in new image (if we overwrite, we may use the modified value for the nearby element calculation
- store result in float array

Both zero padding and Mirror/Replicate(less inaccuracy) has inaccuracies (due to addition of values that do not exist in original image) , if accuracy is important, we can use the ignore method

Smoothing using convolution

- convolution is a linear filter
- Simple smoothing filter:

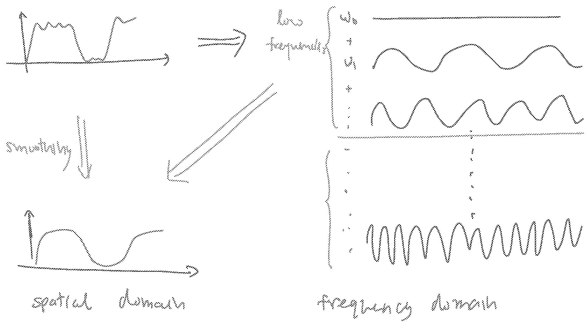


See Notes for images explanation of Noise

smoothing removes the noise as well as the details of the image -->making the image blurred upon smoothing

Low pass filter interpretation

Smoothing = removing high frequencies in image



Another method of smoothing ->low pass filter

1. Convert the image to frequency space through fourier Transformation
2. ignore the high frequencies and take the weighted sum of the low frequencies alone
3. Convert the frequency back to image space.

Definition of derivation - --->

$$\frac{d}{dx}f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

If we ignore the h, we see that derivative is actually the difference between the values of the neighbors. How much have changed from one position to another is the derivation.

Other applications of convolution

Blurring/smoothing $\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$

Sharpening: $\frac{1}{9}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 18 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

opposite of smoothing-->sharpen-->gives enhanced edges-->makes use of derivative (corresponds to -1)

Vertical edge detection: $\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$

derivative along x direction

we take the difference b/w the differences

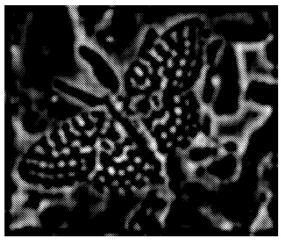
Horizontal edge detection: $\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

derivative along y direction

Similarity using convolution (1/2)

See more image examples in Notes doc

input



result

Convolution can be used to find correlation in the images.

If we take a part of the image as the kernel, we can detect the same pattern across the image as the intensity will be higher when it is multiplied by itself.

How do we decide such filters?

Use prior knowledge -->smoothen if there is noise
For detecting patterns, we can let the n/w decide the kernel values by learning