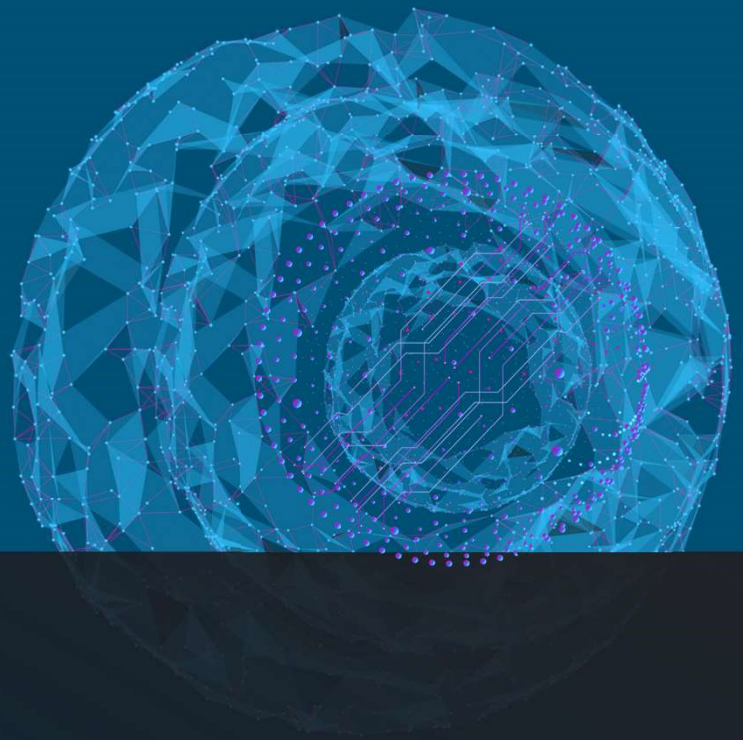
COMPUTATIONAL IMAGING

L 2. Image Processing and Computational Illumination



Dmitry Dylov

Associate Professor

Skoltech

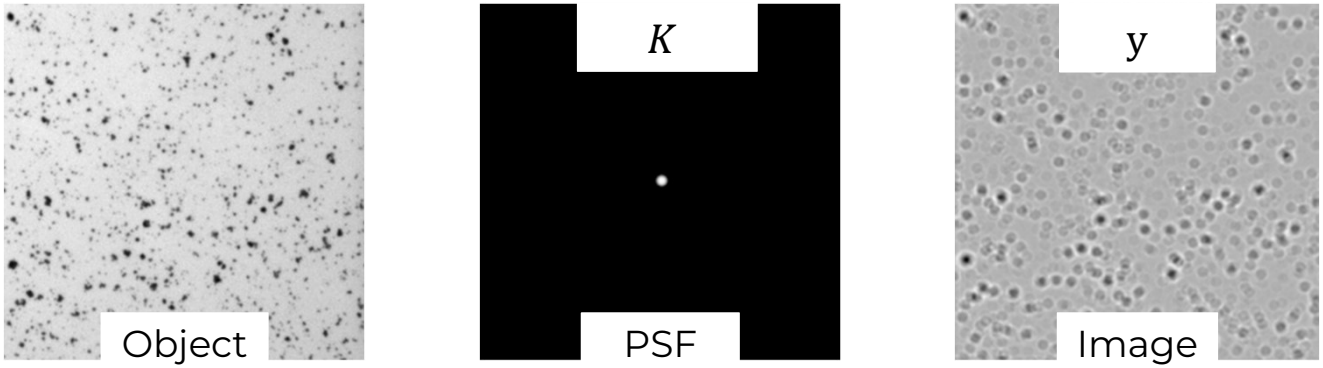
Week 1 – Understand Imaging Math & Camera Basics

Week 2 – Image Processing & Computational Illumination

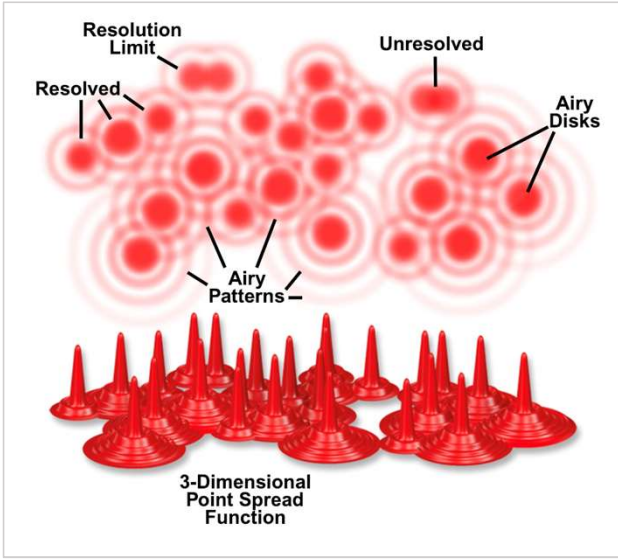
- Smoothing and Sharpening
- Filtering with convolution kernels
- Edge enhancement (Canny Filter)
- Computational Illumination:
  - structured light
  - multi-flash exposure for Canny

Week 3 – Specialized Cameras & Smart Control

Example: Microscopy & Gaussian PSF

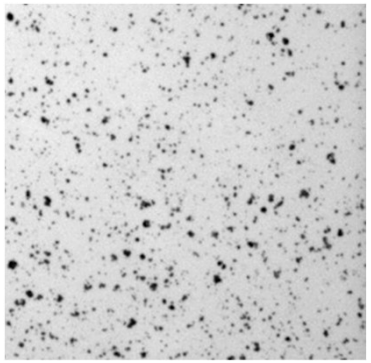


$$f(x,y) = \mathrm{FT}^{-1}\left[\frac{Y(u,v)}{K(u,v)}\right]$$



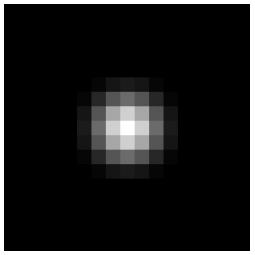
Resolution = 0.61 λ/NA
λ = color
NA = Numerical Aperture

How to de-blur?



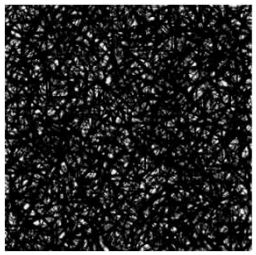
True scene x

$*$



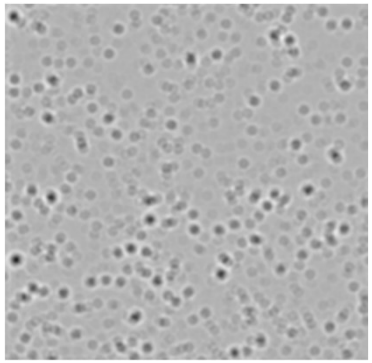
Blurring kernel

$+$



Measurement noise (additive Gaussian)

$=$



Distorted image *y*

$$\mathbf{y} = \mathbf{K}\mathbf{x} + \mathbf{n}$$

Solution of the optimization problem is obtained by minimizing the objective function

$$\hat{\mathbf{x}} = \underset{x}{\operatorname{argmin}} \frac{1}{2} ||\mathbf{y} - \mathbf{K}\mathbf{x}||_2^2$$

Basic Deblurring by Deconvolution

```python
import numpy as np
import matplotlib.pyplot as plt

from skimage import color, data, restoration

astro = color.rgb2gray(data.astronaut())
from scipy.signal import convolve2d as conv2
psf = np.ones((5, 5)) / 25
astro = conv2(astro, psf, 'same')
astro += 0.1 * astro.std() * np.random.standard_normal(astro.shape)

deconvolved, _ = restoration.unsupervised_wiener(astro, psf)

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(8, 5),
                       sharex=True, sharey=True)

plt.gray()

ax[0].imshow(astro, vmin=deconvolved.min(), vmax=deconvolved.max())
ax[0].axis('off')
ax[0].set_title('Data')

ax[1].imshow(deconvolved)
ax[1].axis('off')
ax[1].set_title('Self tuned restoration')

fig.tight_layout()

plt.show()
```

before

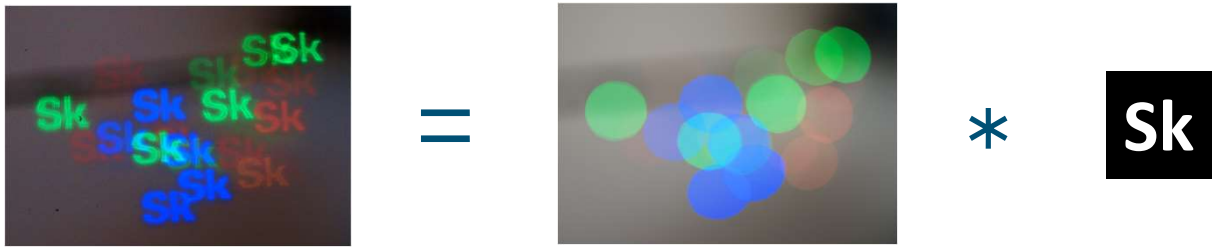
after

Know blurring kernel? → Deconvolution
No? → Blind Deconvolution (iterative)

https://scikit image.org/docs/dev/auto_examples/filters/plotrestoration.html

Another Example: Camera shake as a convolution process

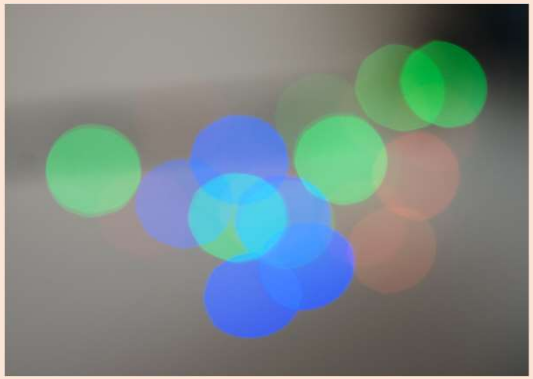
Camera shake

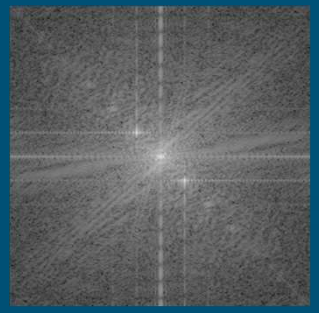


Bokeh experiments

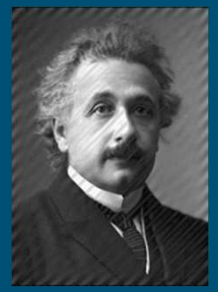
Hope you will enjoy the lab!

Bokeh: Blur in out-of-focus regions of an image.





Filtering in Fourier Domain



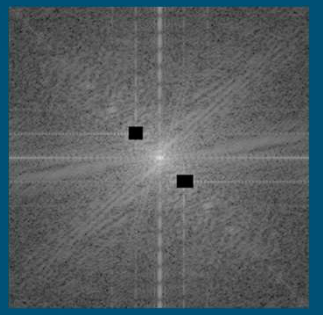
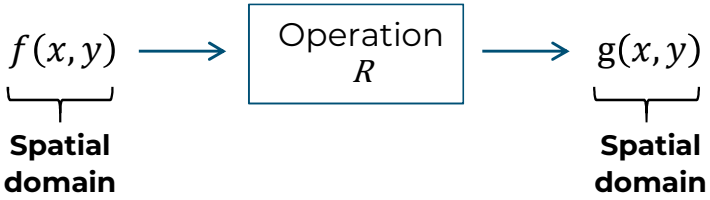
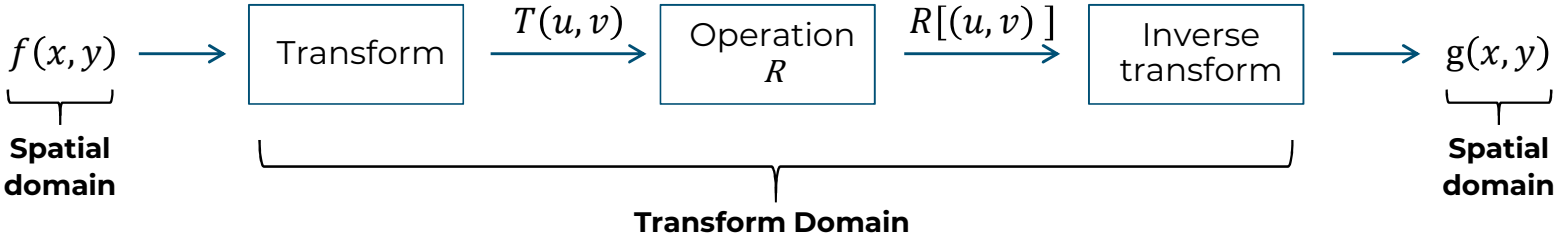


Image Filtering in the Transform Domain

- Spatial Domain

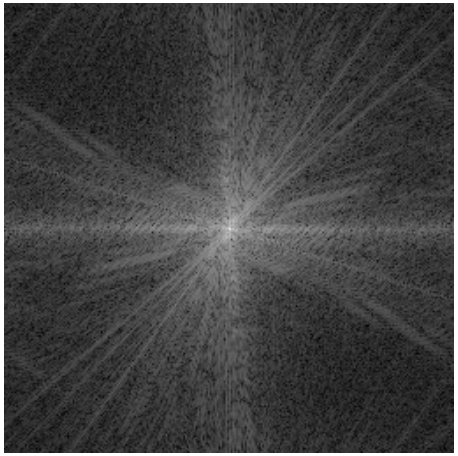


- Frequency Domain (i.e., uses Fourier Transform)



Linear filtering and convolution

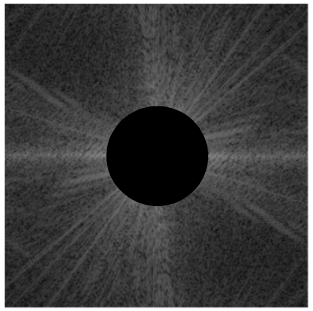
$$log(1 + |F(u, v)|)$$



Linear filtering and convolution

- As an example, suppose $h(x, y)$ corresponds to a linear filter with frequency response defined as follows:

$$H(u, v) = 0 \quad \text{for} \quad u^2 + v^2 < R^2, \qquad H(u, v) = 1 \text{ otherwise}$$

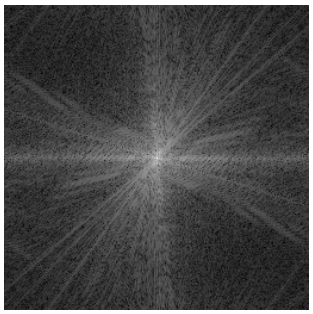


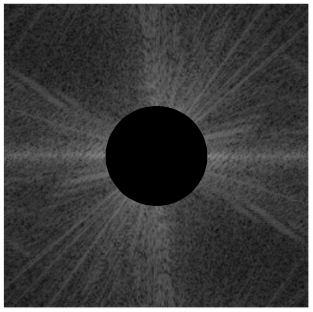
- Removes low frequency components of the image

  (Remember the rectangular slit calculation?)

Fourier Filtering vs Convolution

DFT →





IDFT →

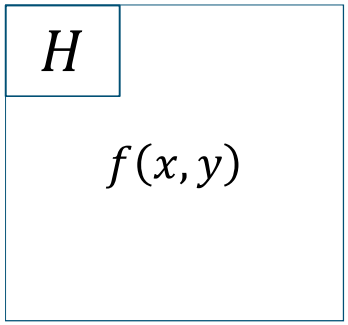
Convolve with Edge Kernel

$$H = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

Digital Filtering
with Convolution

Image Filtering

Convolution (now, digitally)



$$g(x,y) = h(x,y) * f(x,y)$$

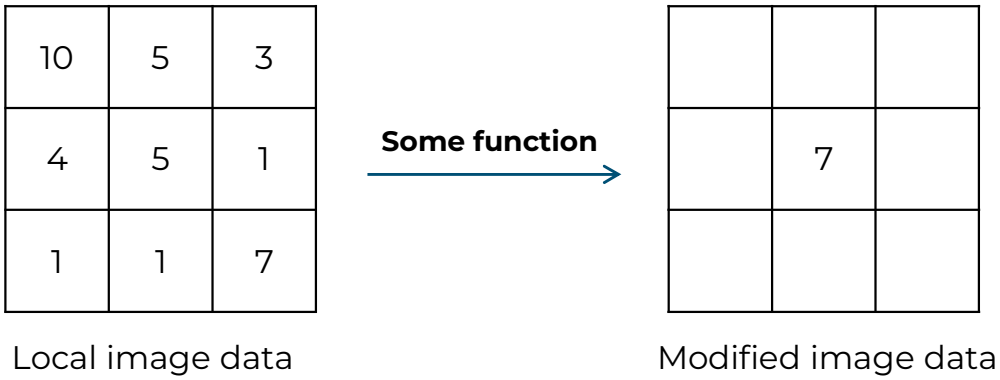
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

Convolution is **commutative** and **associative**

Now that you know what it does, use convolution for various digital image processing.

Image filtering

- Modify the pixels in an image based on some function of a local neighborhood of each pixel

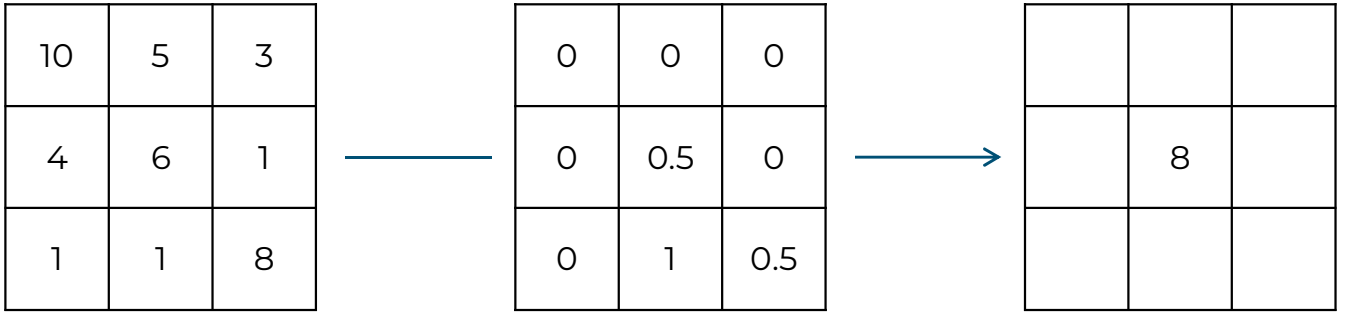


Local image data

Modified image data

Linear filtering

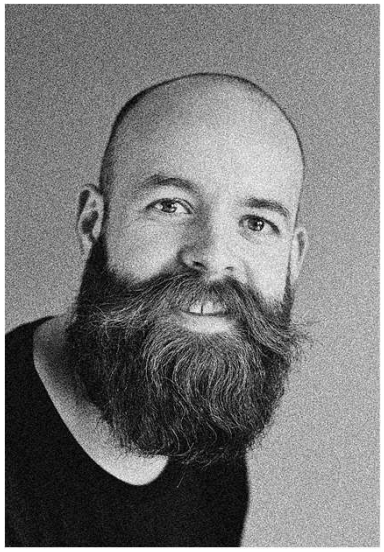
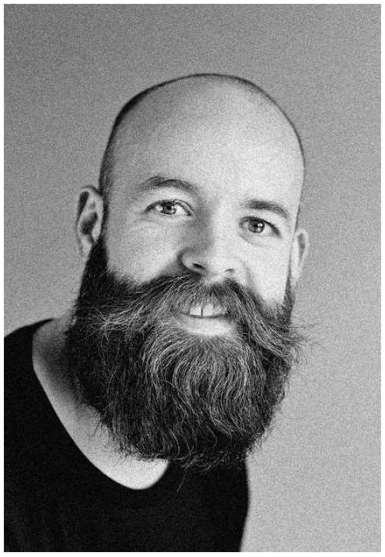
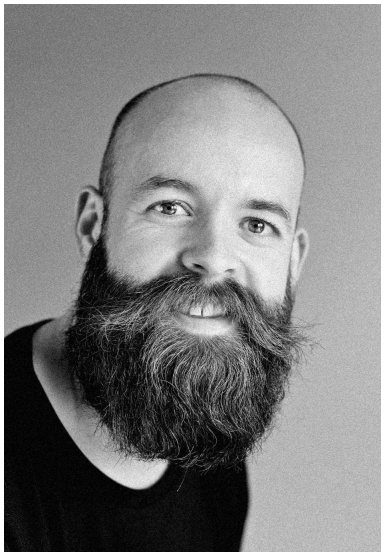
- One simple version: linear filtering (cross-correlation, convolution)
  - Replace each pixel by a linear combination of its neighbors
- The recipe for the linear combination is called the “kernel” (or “mask”, “filter”) – SAME ACTION AS PSF!



Local image data Kernel Modified image data

Question: Noise reduction

- Given a camera and a still scene, how can you reduce noise?

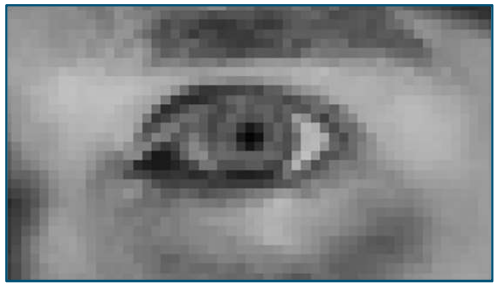
Take lots of images and average them!

Mean filtering

$$\frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} * \begin{bmatrix}0&0&0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&0&90&90&90&90&90&0&0\\0&0&0&90&90&90&90&90&0&0\\0&0&0&90&90&90&90&90&0&0\\0&0&0&90&0&90&90&90&0&0\\0&0&0&90&90&90&90&90&0&0\\0&0&0&0&0&0&0&0&0&0\\0&0&90&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0&0&0\end{bmatrix} = \begin{bmatrix}&&&&&&&&&\\&0&10&20&30&30&30&20&10&\\&0&20&40&60&60&60&40&20&\\&0&30&60&90&90&90&60&30&\\&0&30&50&80&80&90&60&30&\\&0&30&50&80&80&90&60&30&\\&0&20&30&50&50&60&40&20&\\&10&20&30&30&30&30&20&10&\\&10&10&10&0&0&0&0&0&\\&&&&&&&&&\end{bmatrix}$$

H F G

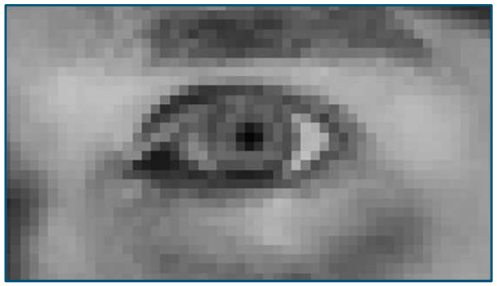
Linear filters: examples



$*$

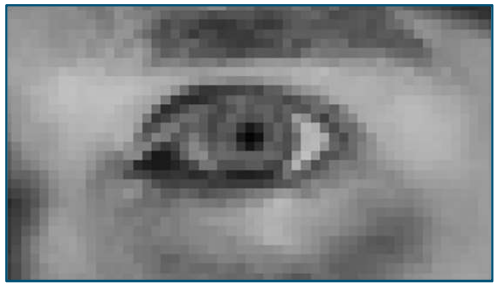
0	0	0
0	1	0
0	0	0

$=$



Original **Identical image**

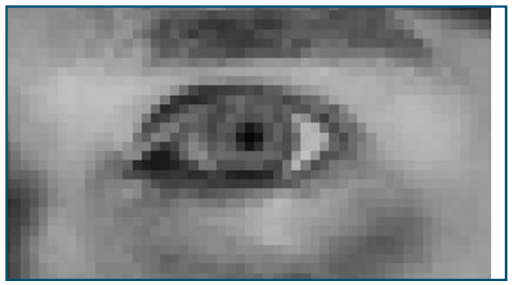
Linear filters: examples



$*$

0	0	0
1	0	0
0	0	0

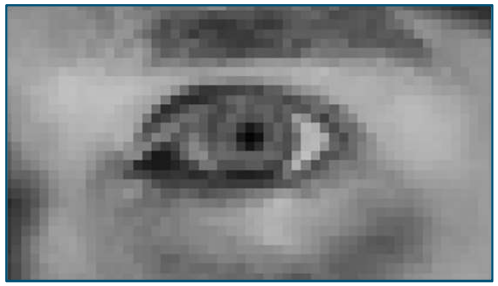
$=$



Original

Shifted left By 1 pixel

Linear filters: examples

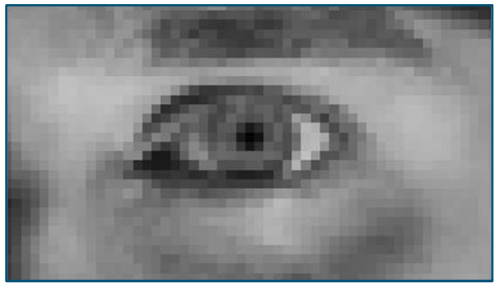


$$* \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} =$$

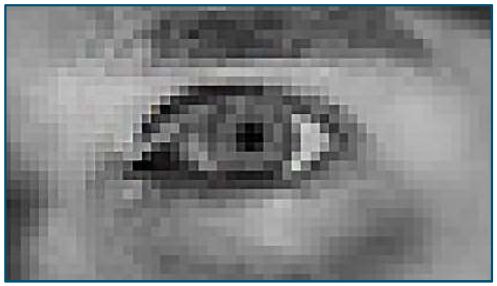
Original

Blur (with a mean filter)

Linear filters: examples



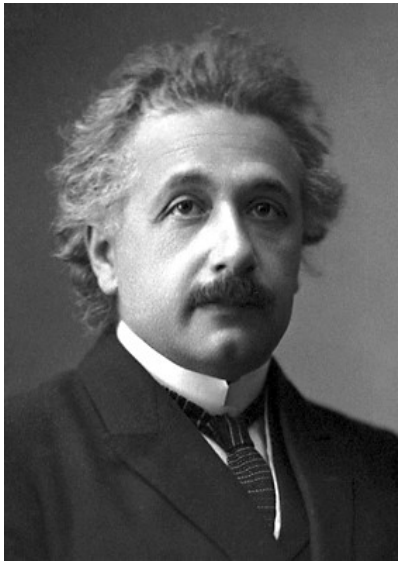
$$* \left(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \right) =$$



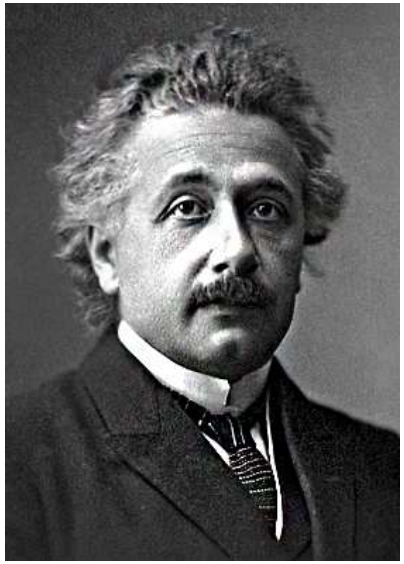
Original

Sharpening filter
(accentuates edges)

Sharpening



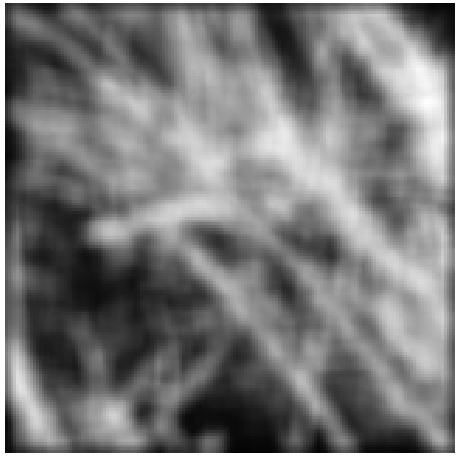
Before



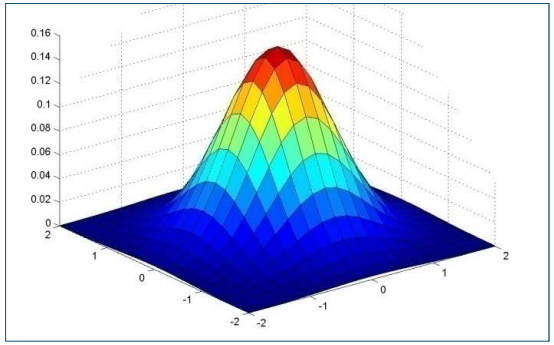
After

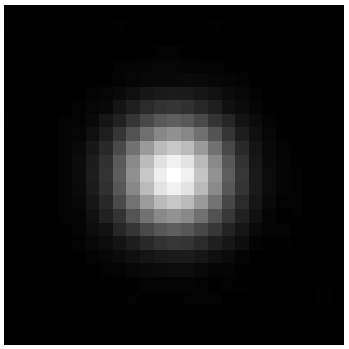
Smoothing with box filter revisited

$*$ $\square$ $=$



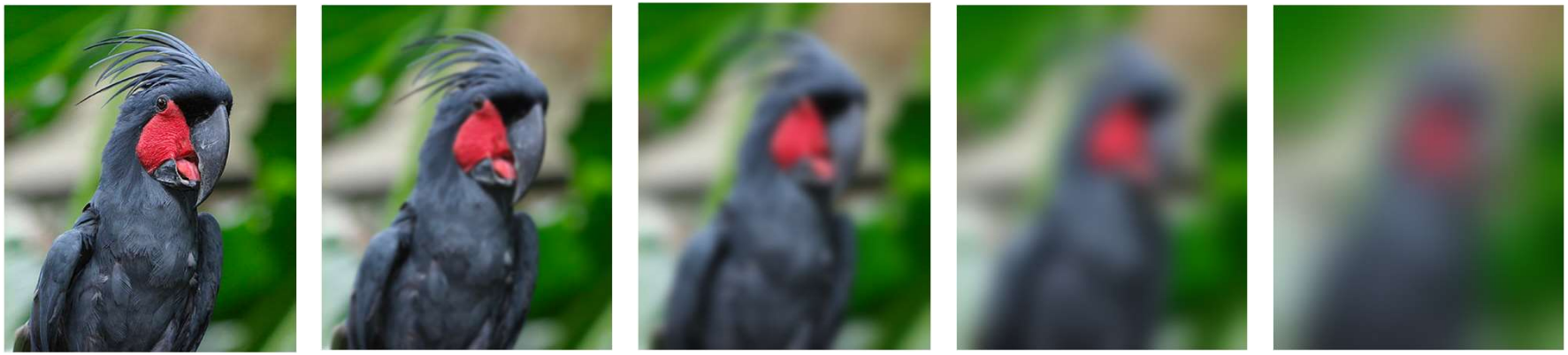
Gaussian Kernel





$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Gaussian filters



$\sigma = 1$ pixel $\quad$ $\sigma = 5$ pixel $\quad$ $\sigma = 10$ pixel $\quad$ $\sigma = 30$ pixel

- Removes “high-frequency” components from the image (low-pass filter)
- Convolution with self is another Gaussian

Gaussian filters – optimal amount

Original

Noisy

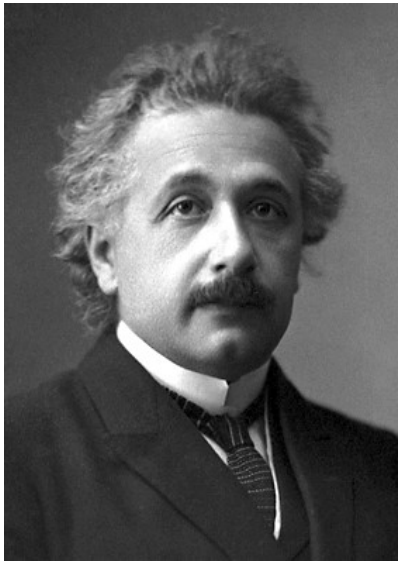
Filtered

$\sigma = 1.5$

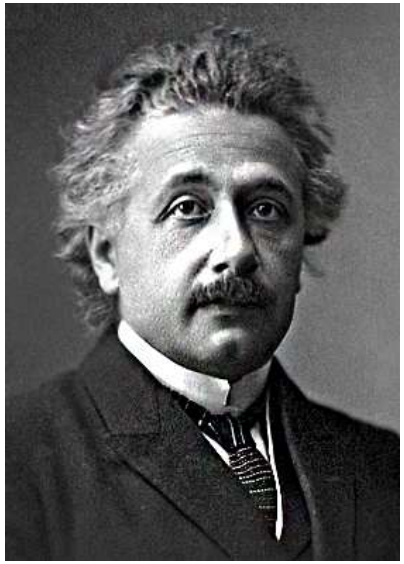
Filtered

$\sigma = 3.0$

Sharpening



Before



After

Sharpening revisited

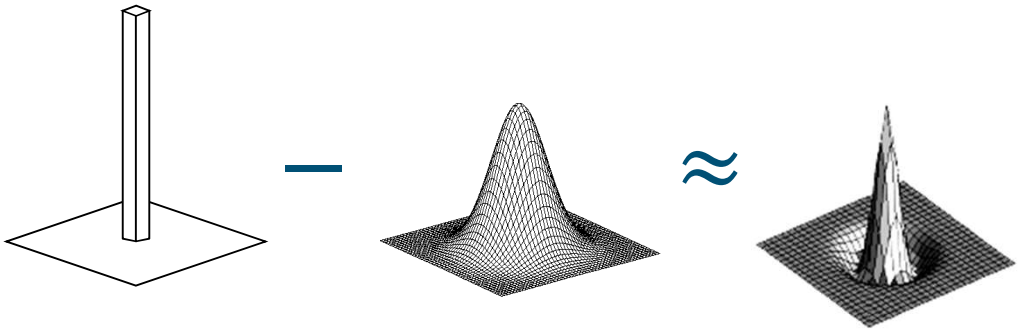
- What does blurring take away?

- Let's add it back

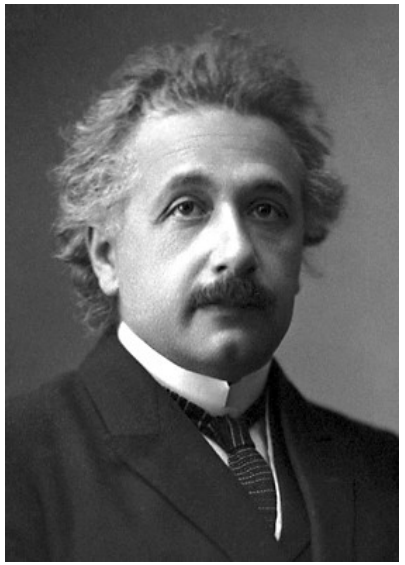
Sharpen filter

$$F + a(F - F * H) = (1 + a) - a(F * H) = F * ([1 + a]e - H)$$

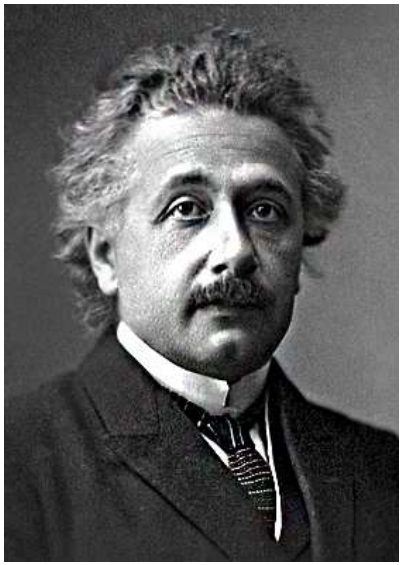
Image — Blurred image — Image



Scaled impulse — Gaussian — Laplacian of Gaussian



Before



After

Cross-correlation

Cross-correlation

- **Let F be the image, H be the kernel (of size 2k+1 x 2k+1), and G be the output image**

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

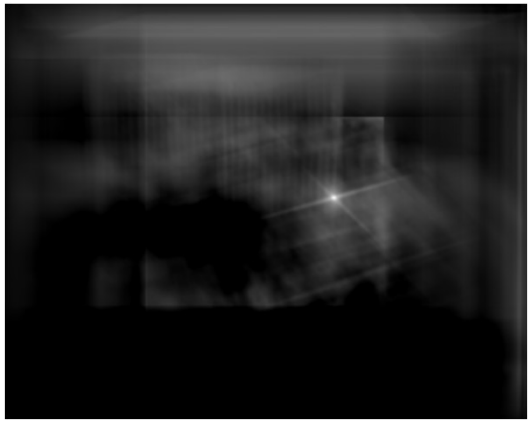
This is called a **cross-correlation** operation:

$$G = H \otimes F$$

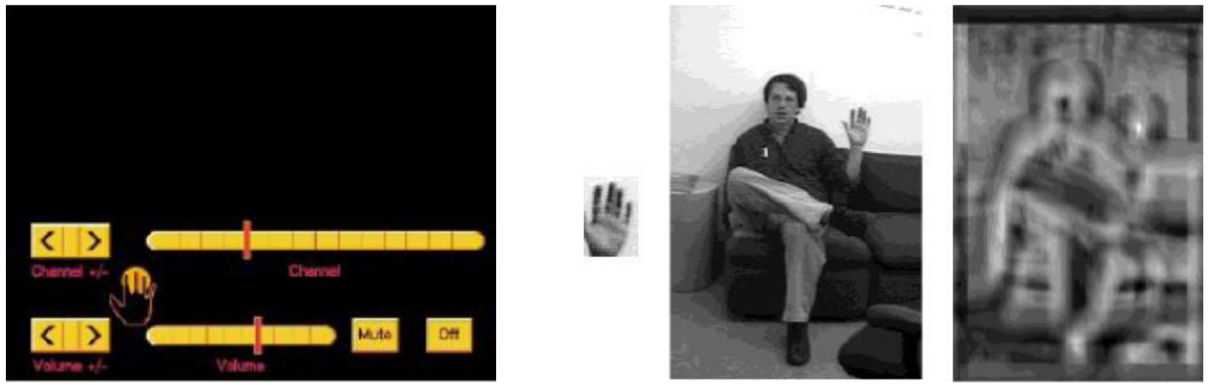
Normalized Correlation

- Measure the similarity between images or parts of images.

mask



Application: TV Remote Control



Template (left), image (middle), normalized correlation (right)

Note peak value at the true position of the hand

https://www.spiedigitallibrary.org/conference-proceedings-of-spie/10452/104520Y/

Summary: Digital Filtering

Convolution = most basic go-to linear filtering approach

Correlation = shows similarity between images

Fourier Domain = sometimes it is easier to know what to Filter there

Image Enhancement 2: Edge Detection

Sharpen filter

$$F + a(F - F * H) = F\,(1 + a)\; - a(F * H) = F * ([1 + a]e - H)$$

Image — Blurred image — Unit impulse (identity)

Convolve with Edge Kernel

(high-pass filter)

$$H = \begin{pmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{pmatrix}$$

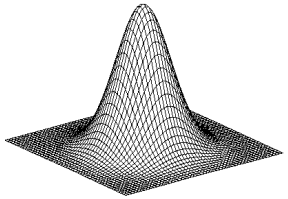
Original

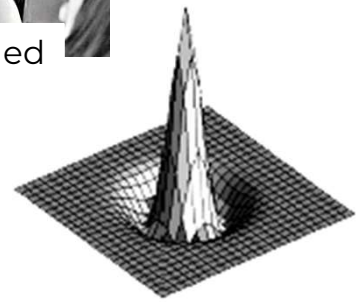
$+ \alpha$

Detail

$=$

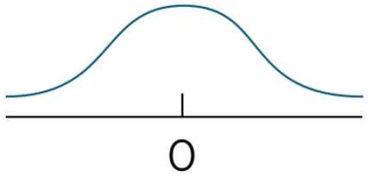
sharpened



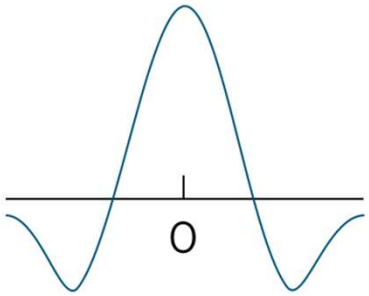


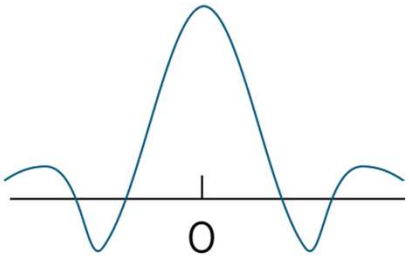
Kernel collection

- Depends on the application.
- Usually by sampling certain functions and their derivatives.



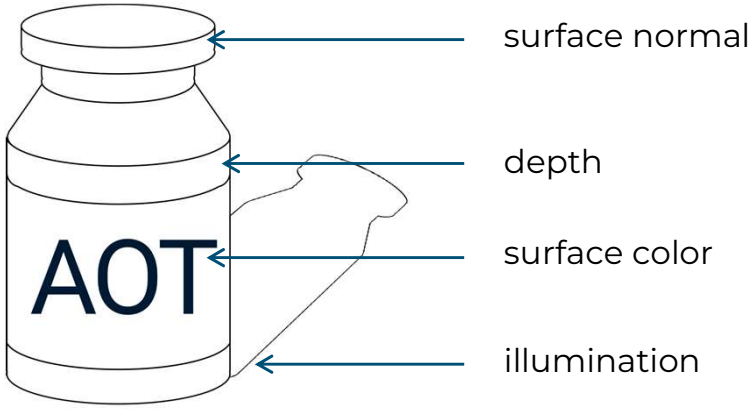
Good for image smoothing





Good for image sharpening

Edge detection



- **Convert a 2D image into a set of curves**

— Extracts salient features of the scene

— More compact than pixels

- **Origins: discontinuity in…**

Characterizing edges

- An edge is a place of rapid change in the image intensity function

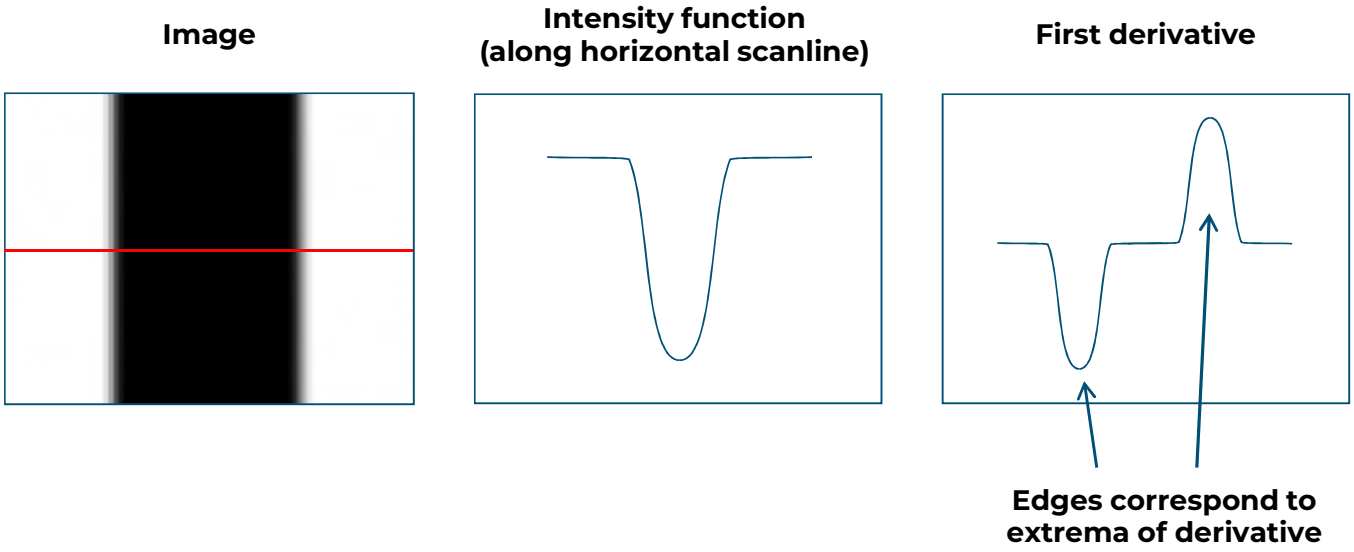


Image derivatives

- **How can we differentiate a digital image F[x,y]?**

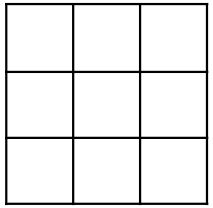
— Option 1: reconstruct a continuous image, f, then compute the derivative

— Option 2: take discrete derivative (finite difference)

$$\frac{\partial f}{\partial x}[x,y] \approx F[x+1,y] - F[x,y]$$

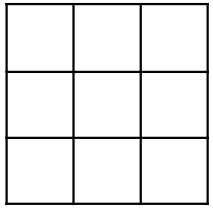
How would you implement this as a linear filter?

$\frac{\partial f}{\partial x}$:



H_x

$\frac{\partial f}{\partial y}$:



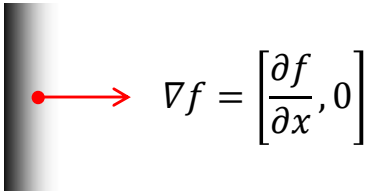
H_y

Image gradient

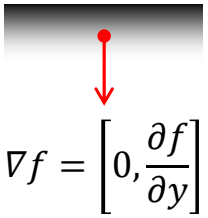
- **The gradient of an image:**

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

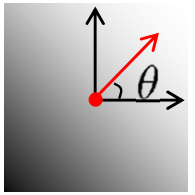
The gradient points in the direction of most rapid increase in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The edge strength is given by the gradient magnitude:

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

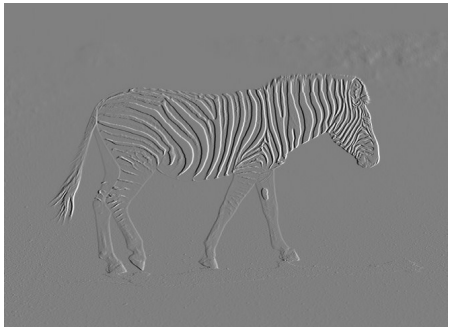
The gradient direction is given by:

$$\emptyset = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

- **how does this relate to the direction of the edge?**

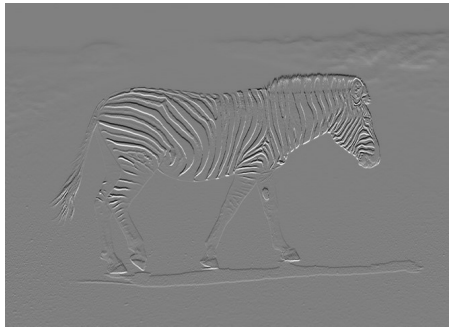
Image gradient

f



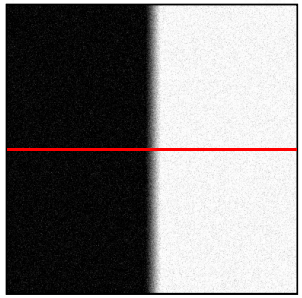
$\frac{\partial f}{\partial x}$

$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

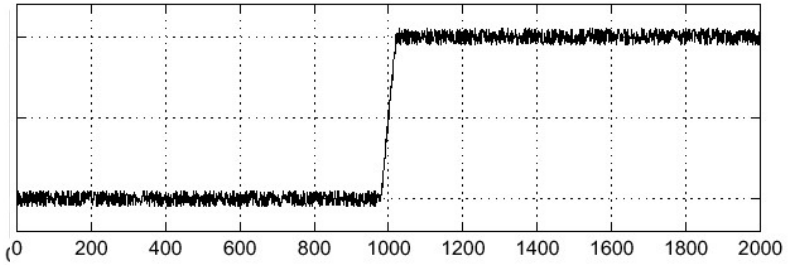


$\frac{\partial f}{\partial y}$

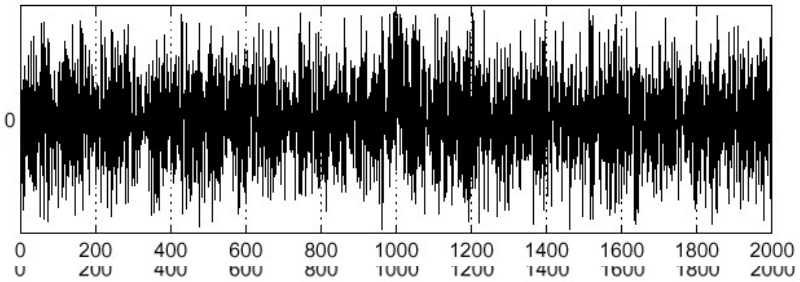
Effects of noise



Noisy input image



$f(x)$



$\frac{d}{dx}f(x)$

Where is the edge?

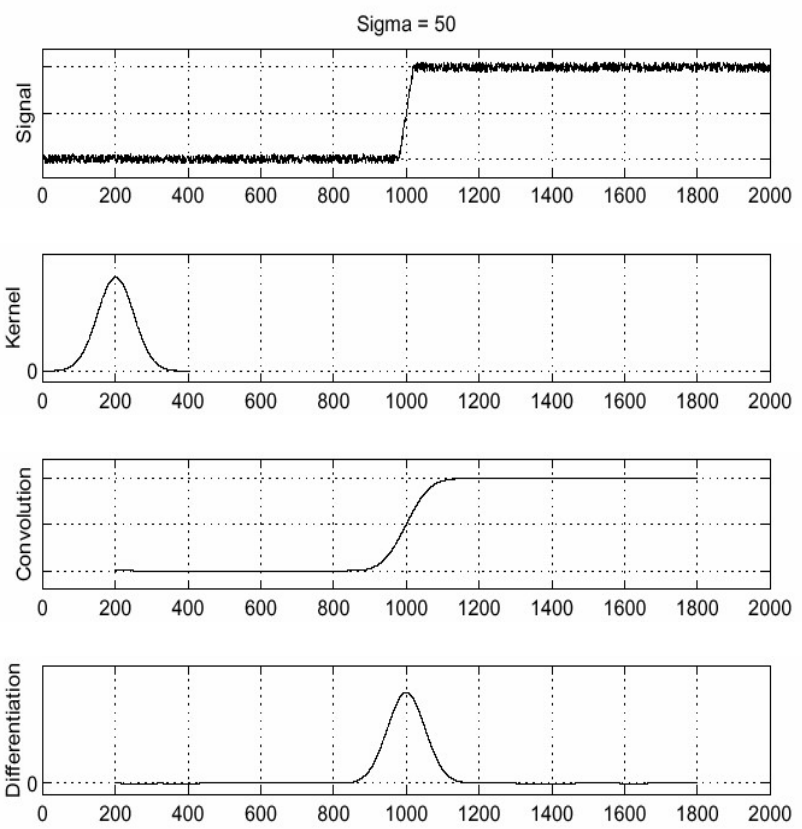
Solution: smooth first

f

h

$f * h$

$\frac{d}{dx}(f * h)$



To find edges, look for peaks in

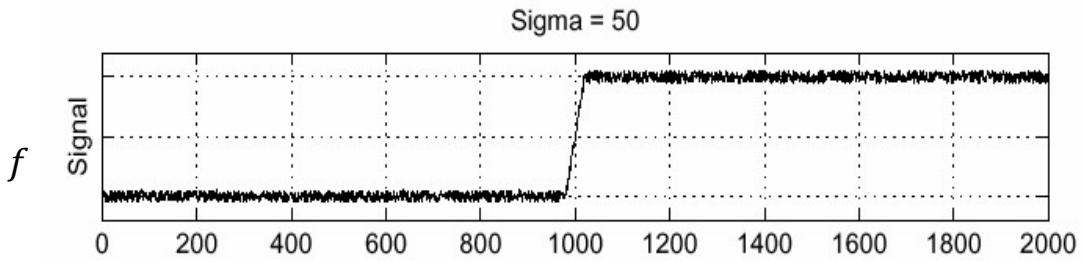
$$\frac{d}{dx}(f * h)$$

Associative property of convolution

Differentiation is convolution, and convolution is associative:

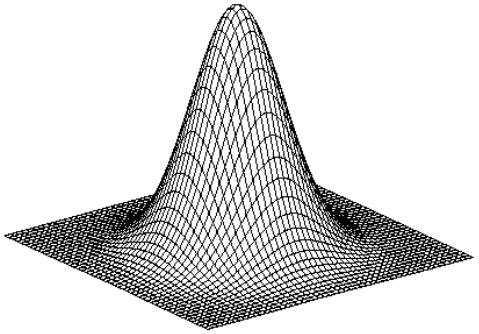
$$\frac{\partial}{\partial x}(f * h) = f * \frac{d}{dx} h$$

This saves us one operation:



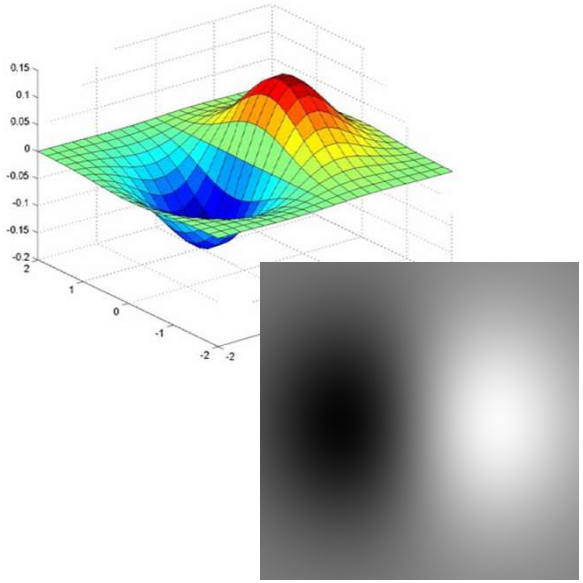
Instead, compute derivatives of a Gaussian kernel and then convolve

2D edge detection filters



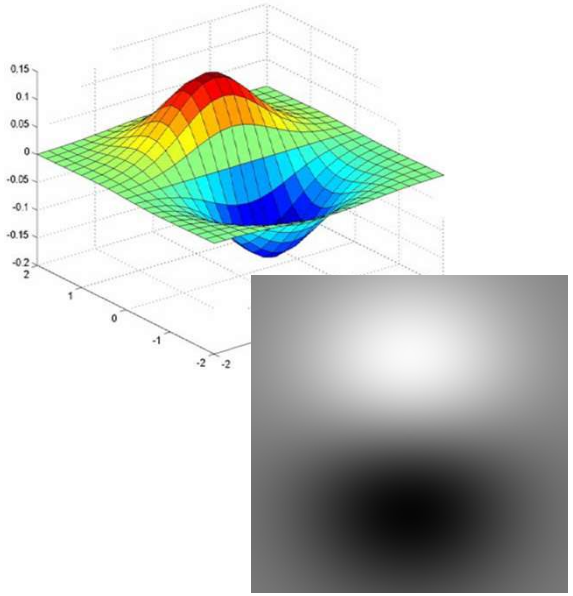
Gaussian

$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

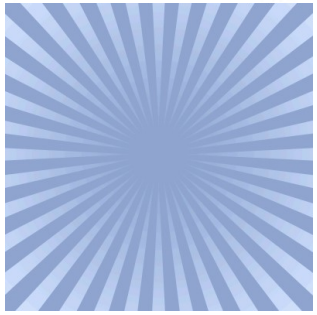
Derivative of Gaussian (*x*)
***x*–direction**



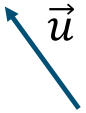
$$\frac{\partial}{\partial y} h_\sigma(u,v)$$

Derivative of Gaussian (*y*)
***y*–direction**

Side note: How would you compute a directional derivative?



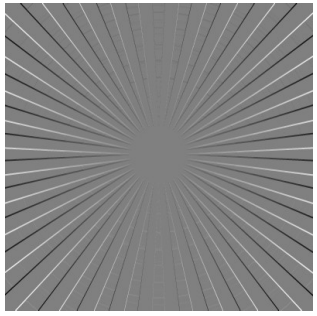
f



$\vec{u}$

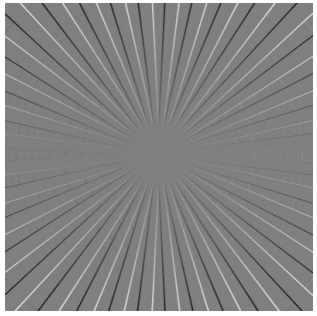
$\nabla_{\vec{u}} f = ?$

Directional derivative is a linear combination of partial derivatives



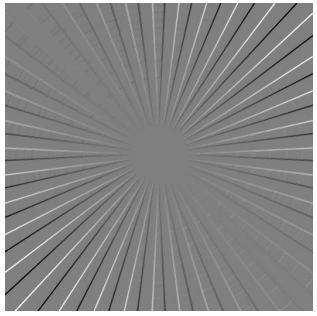
$\frac{\partial f}{\partial x} \cdot u_x$

+



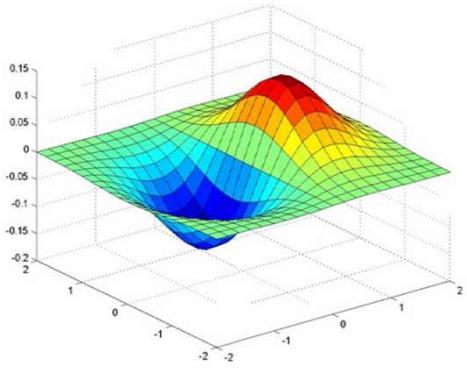
$\frac{\partial f}{\partial y} \cdot u_y$

=

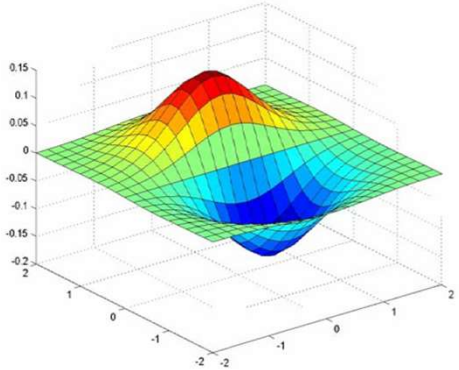


$\nabla_{\vec{u}} f$

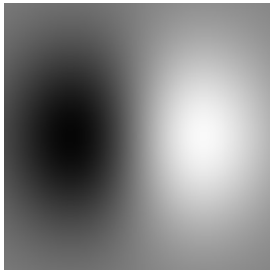
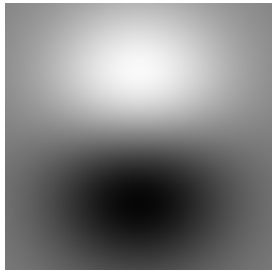
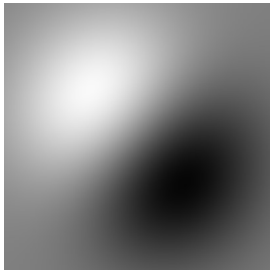
Derivative of Gaussian filter



***x*-direction**



***y*-direction**

$$\cos(\theta)$$  $$+ \sin(\theta)$$  $$=$$ 

The Sobel operator

- Common approximation of derivative of Gaussian

$$\frac{1}{8}$$

-1	0	1
-2	0	2
-1	0	1

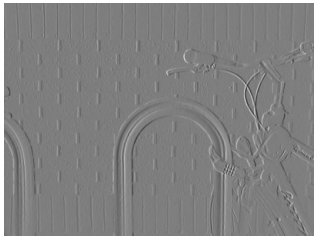
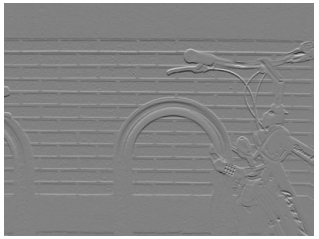
$$S_x$$

$$\frac{1}{8}$$

1	2	1
0	0	0
-1	-2	-1

$$S_y$$

Sobel operator: example

Example

Original image (Lena Söderberg)

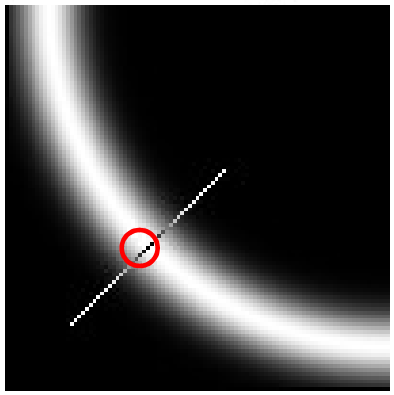
Finding edges

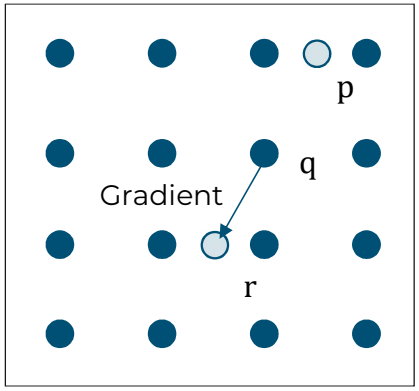
Gradient magnitude

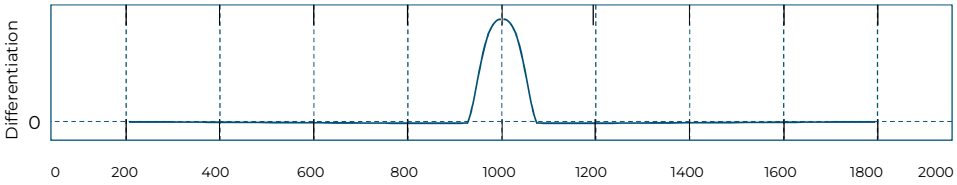
Finding edges

thresholding

Non-maximum suppression







- Check if pixel is local maximum along gradient direction
  - requires *interpolating* pixels p and r

Finding edges

thresholding

Finding edges

thinning (non-maximum suppression)

Canny edge detector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression
4. Linking and thresholding (hysteresis):
    - Define two thresholds: low and high
    - Use the high threshold to start edge curves and the low threshold to continue them

cv2.Canny()

Canny edge detector

Original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

- The choice of σ depends on desired behavior
  - — large σ detects “large-scale” edges
  - — small σ detects fine edges

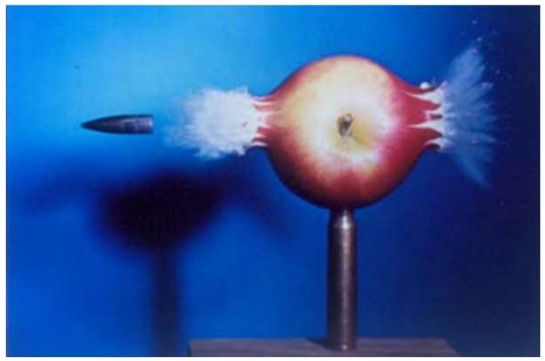
Computational Illumination

Computational Illumination

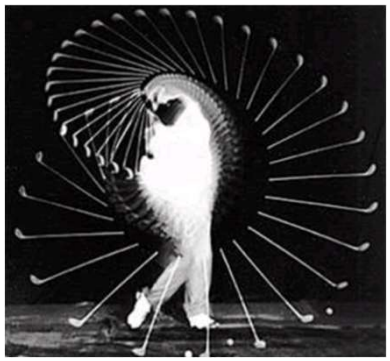
Programmable 4D Illumination Field + Time + Wavelength

- **ON or OFF, Duration, Brightness**
  — Flash/No-flash
- **Light position**
  — Relighting: Programmable dome
  — Shape enhancement: Multi-flash for depth edges
- **Light color/wavelength**
- **Spatial Modulation**
  — Synthetic Aperture Illumination
- **Temporal Modulation**
- — Motion Tracking
- **Exploiting natural lighting condition**
  — Day/Night Fusion, Time Lapse, Glare

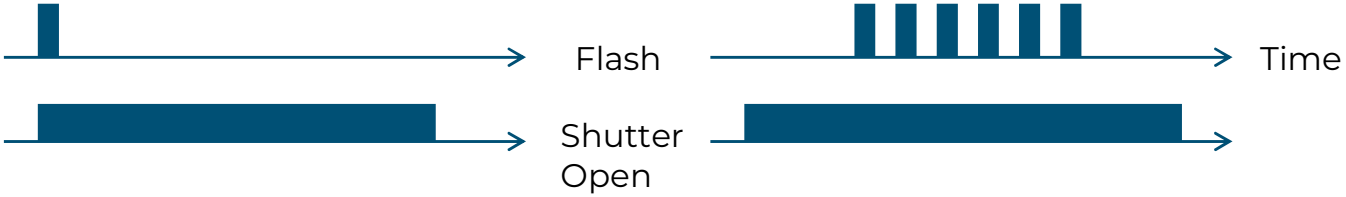
Edgerton, 1930 (MIT museum)



Stroboscope (Electronic Flash)

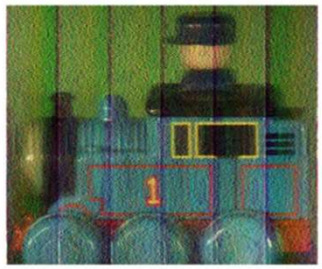


Multi-Flash Sequential Photography

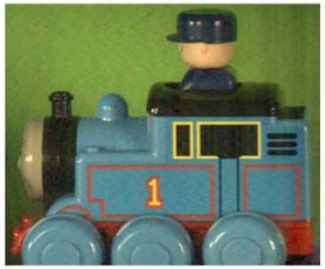


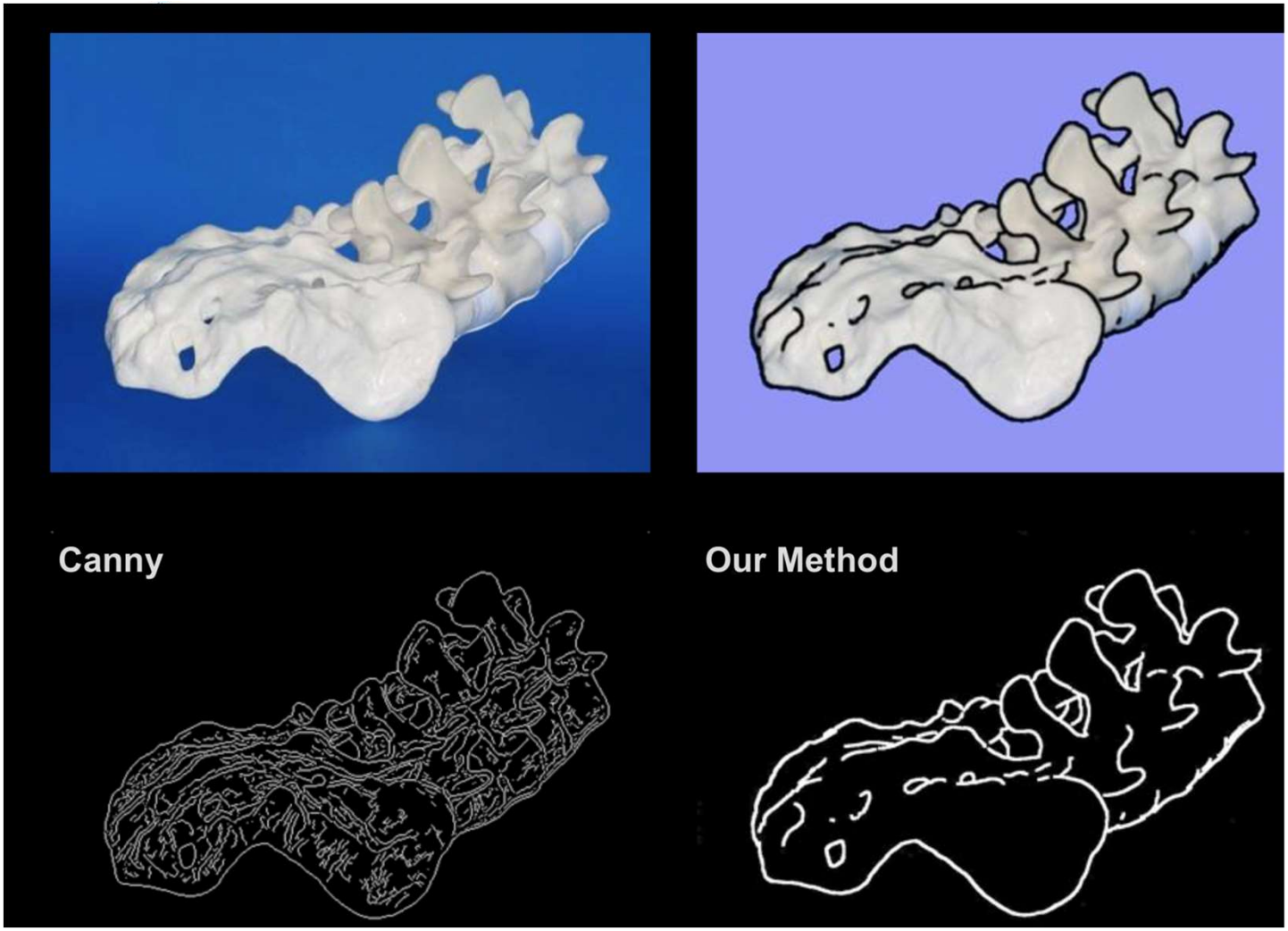
Flutter Shutter: deconvolution in time

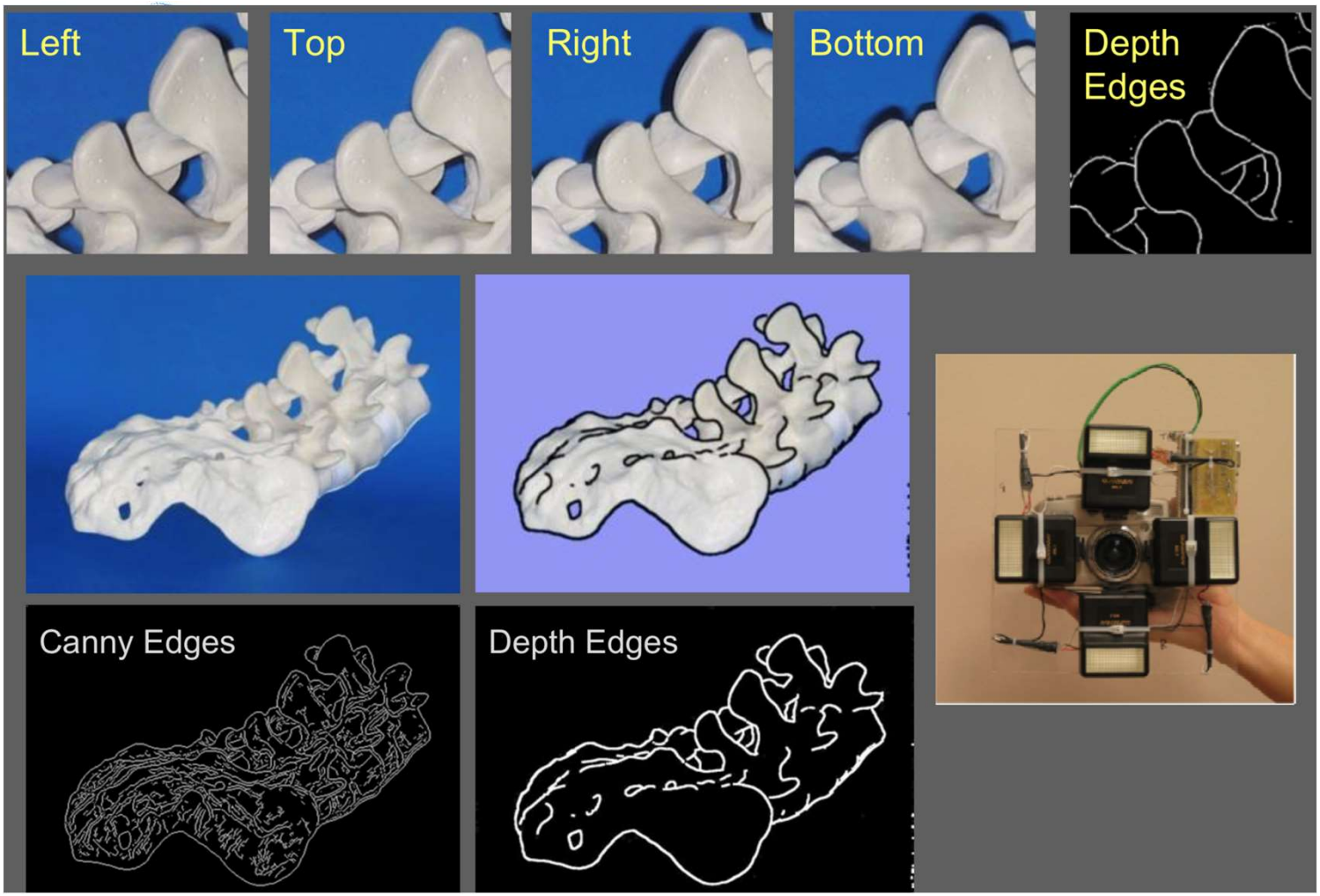
Long Exposure



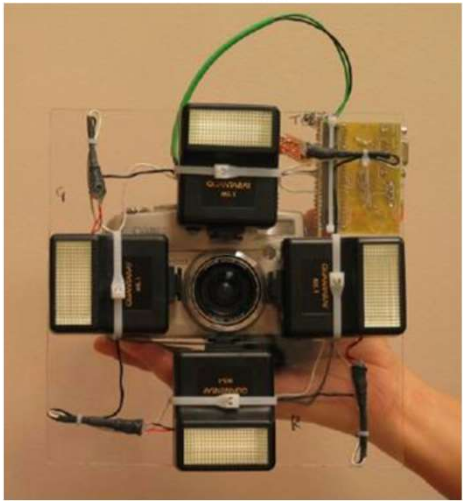
Coded sequence




Canny
Our Method

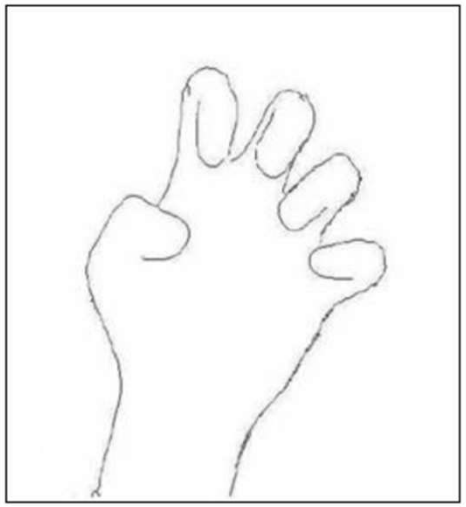

Left
Top
Right
Bottom
Depth Edges
Canny Edges
Depth Edges

Example of computational illumination: Illuminate 4 times, then reconstruct



Input Photo

Canny Edges



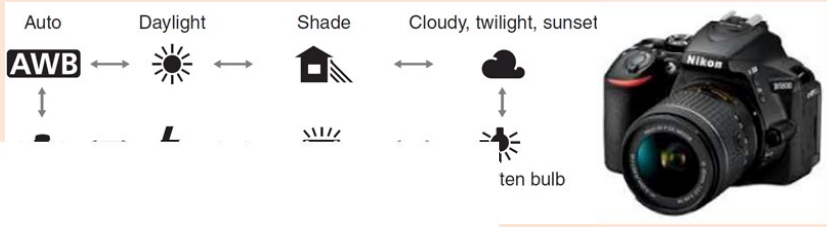
Milti-flash camera image

Color Imaging

Color imaging

“The Dress”

White Balance



Untitled Poll

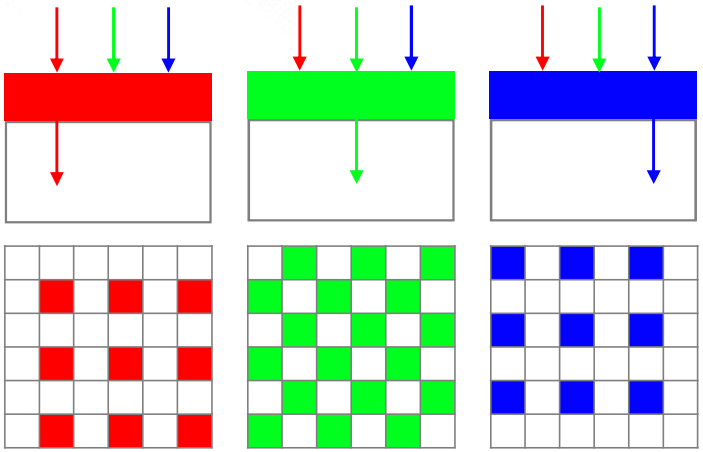
00:01:35 | 1 question | 21 of 27 (77%) participated

1. Which color is the dress (Single Choice) *

21/21 (100%) answered

Gold	(9/21) 43%
Blue	(12/21) 57%

Color sensor

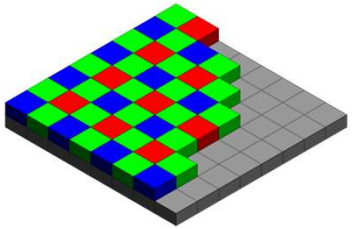


Incoming light

Filter layer

Sensor array

Resulting pattern



Bayer Filter

De-mosaicking

Incoming light

Filter layer

Sensor array

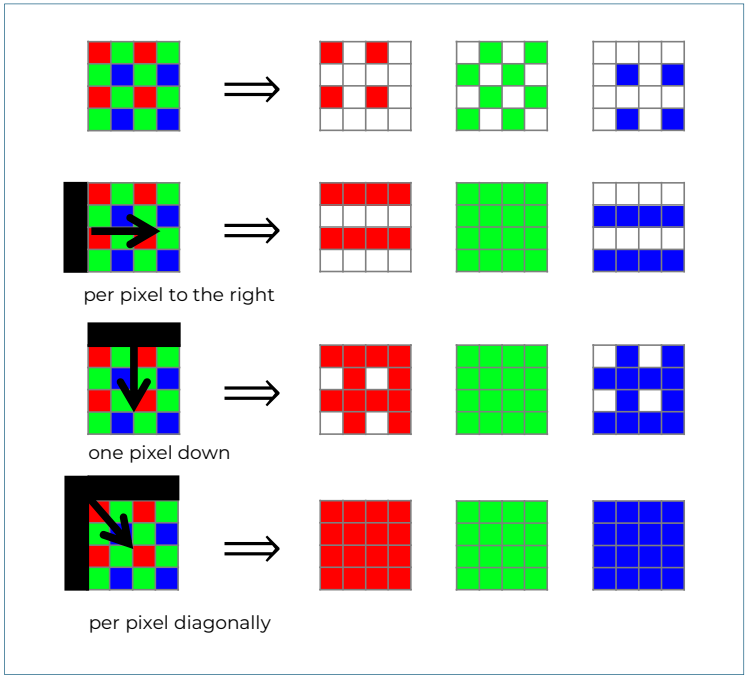
Resulting pattern

Interpolate missing colors!

Noise? -- Average

Pixel Shifting: in every mobile phone today

Pixel shifting



(your natural handshake is useful for averaging)

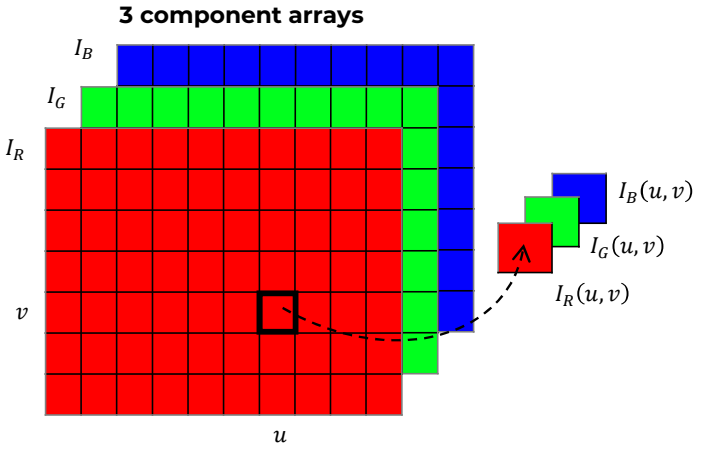
Color Images

- True color: Uses all colors in color space
- Indexed color: Uses only some colors
- Which subset of colors to use? Depends on application
- True color:
- used in applications that contain many colors with subtle differences
- e.g., digital photography or photorealistic rendering
  - Component ordering
  - Packed ordering

True Color: Ordering

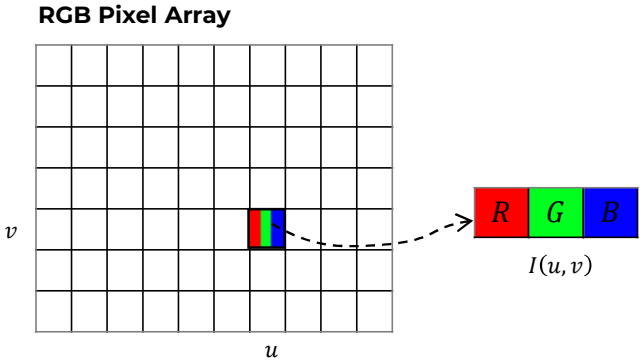
Component Ordering

Colors in 3 separate arrays of similar length Retrieve same location (u,v) in each R, G and B array



Packed Ordering

Component (RGB) values representing a pixel's color is packed together into single element



Indexed Images

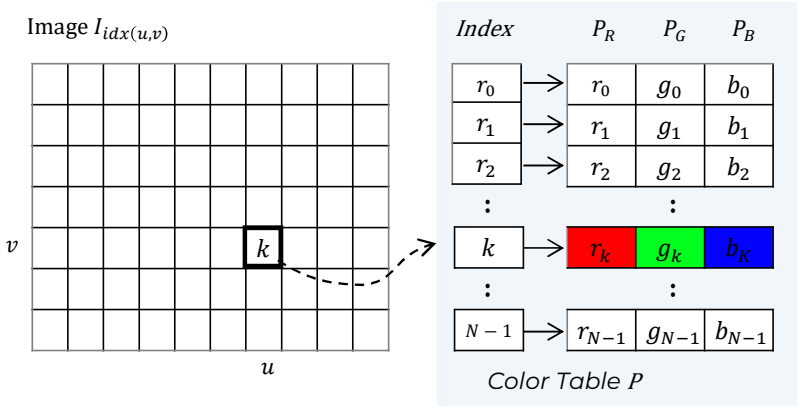
- Permit only limited number of distinct colors (N = 2 to 256)
- Used in illustrations or graphics containing large regions with same color
- Instead of intensity values, image contains indices into color table or palette
- Palette saved as part of image
- Converting from true color to indexed color requires quantization

RGB full-color images (24-bit “RGB color”)

- packed order
- Supports TIFF, BMP, JPEG, PNG and RAW

Indexed images (“8-bit color”)

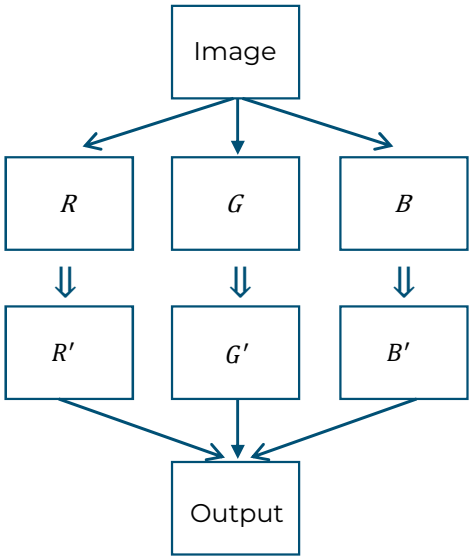
- Up to 256 colors max (8 bits)
- Supports GIF, PNG, BMP and TIFF (uncompressed)



General Strategies for Processing Color Images

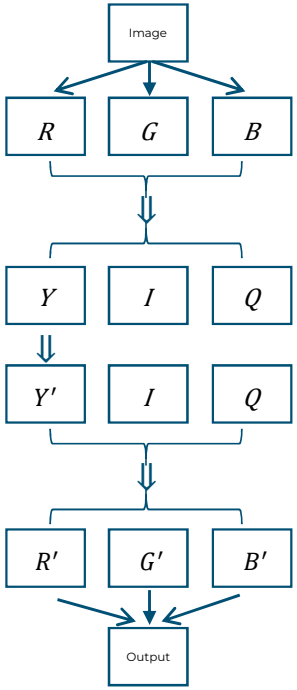
Strategy 1:

Process each RGB matrix separately



Strategy 2:

Compute luminance (weighted average of RGB), process intensity matrix



RGB to HSV Conversion

- **Define the following values**

$$C_{\text{high}} = max(R, G, B),\ C_{\text{low}} = min(R, G, B),\ C_{\text{rng}} = C_{\text{high}} - C_{\text{low}}$$

- **Find saturation of RGB color components** $(C_{\text{max}} = 255)$

$$S_{\text{HSV}} = \begin{cases} \frac{C_{high}}{C_{high}} & C_{\text{high}} > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Find **luminance** value

$$V_{\text{HSV}} = \frac{C_{\text{high}}}{C_{\text{max}}}$$

RGB to HSV Conversion

- Normalize each component using

$$R' = \frac{C_{\text{high}} - R}{C_{\text{rng}}} \quad G' = \frac{C_{\text{high}} - G}{C_{\text{rng}}} \quad B' = \frac{C_{\text{high}} - B}{C_{\text{rng}}}$$

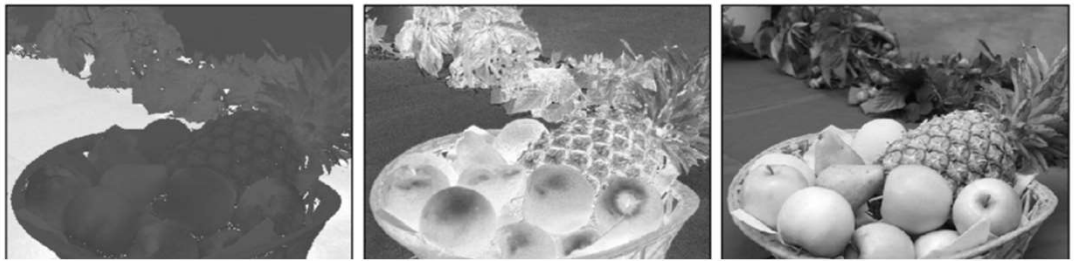
- Calculate preliminary hue value H' as

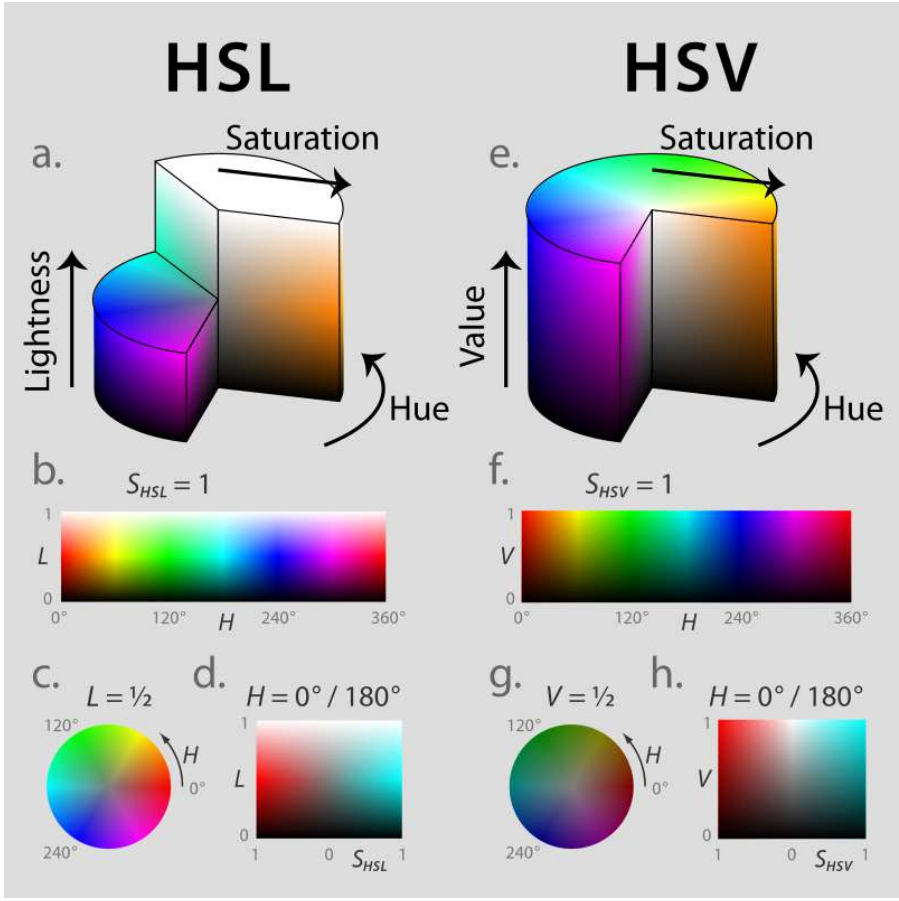
$$H' = \begin{cases} B' - G' & \text{if} \quad R = C_{high} \\ R' - B' + 2 & \text{if} \quad \mathrm{G} = C_{\text{high}} \\ G' - R' + 4 & \text{if} \quad \mathrm{B} = C_{\text{high}} \end{cases}$$

- Finally, obtain final **hue** value by normalizing to interval [0,1]

$$H_{\text{HSV}} = \frac{1}{6} \cdot \begin{cases} (H' + 6) & \text{for } H' < 0 \\ H' & \text{otherwise} \end{cases}$$

RGB to HSV (Hue, Saturation, Value)

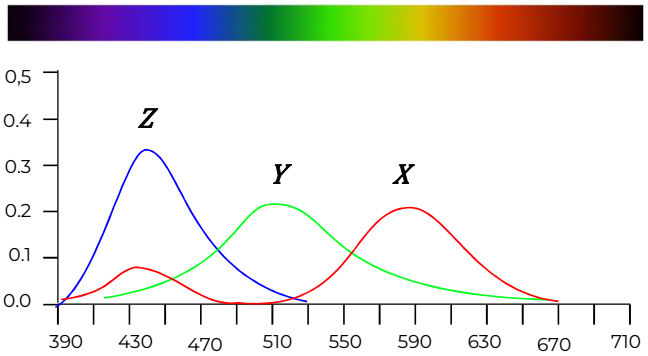




CIE Color Space

- *CIE* (Commission Internationale d'Eclairage) came up with

3 hypothetical lights X, Y, and Z with these spectra:

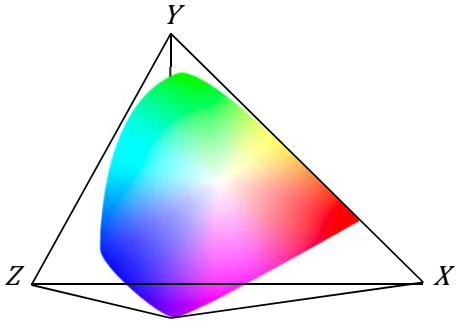


Note that

X ~ R

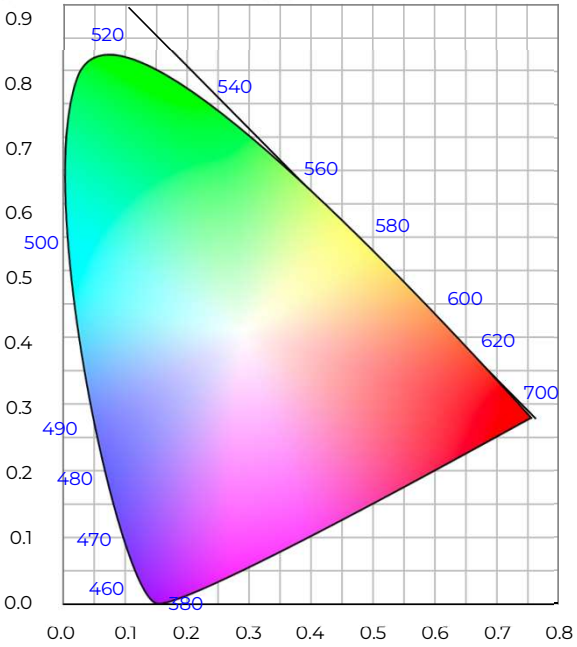
Y ~ G

Z ~ B



- **Idea:** any wavelength λ can be matched perceptually by *positive* combinations of X,Y,Z
- CIE created table of XYZ values for all *visible* colors

CIE Chromaticity Diagram (1931)



- For simplicity, we often project to the 2D plane
- Also normalize

$$x + y + z = 1$$

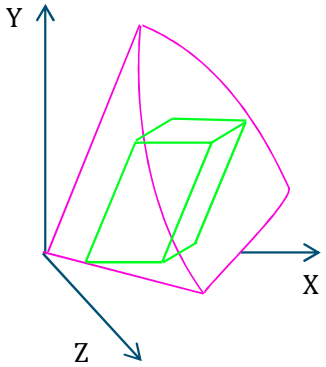
$$x = \frac{X}{X+Y+Z} \qquad y = \frac{Y}{X+Y+Z} \qquad z = \frac{Z}{X+Y+Z}$$

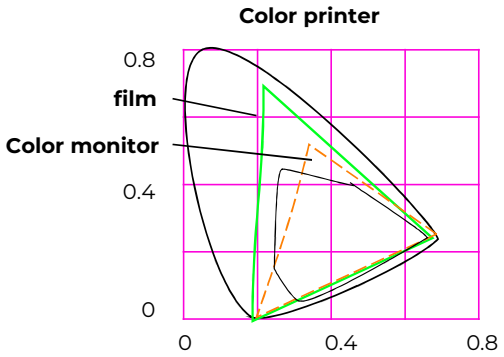
- **Note:** Inside horseshoe visible, outside invisible to eye

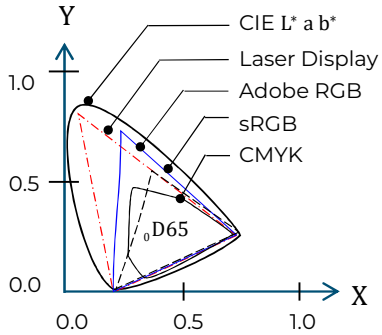
- **Note:** Look up x, y
  Calculate z as $1 - x - y$

Device Color Gamuts

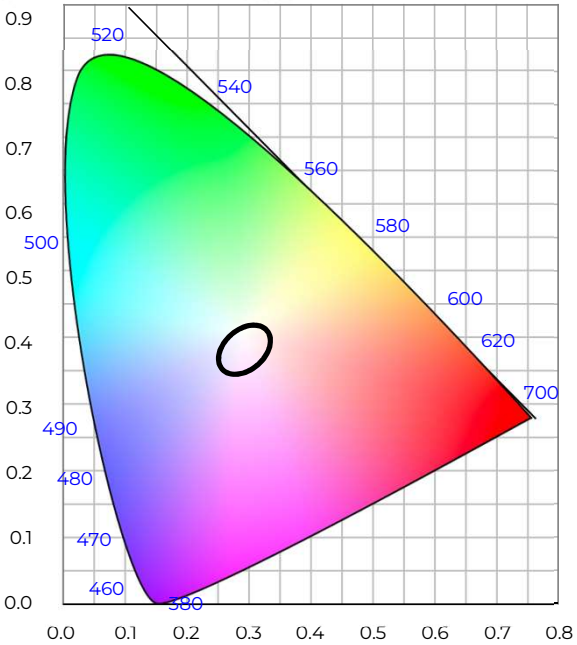
- The RGB color cube sits within CIE color space
- We can use the CIE chromaticity diagram to compare the gamuts of various devices
- E.g. compare color printer and monitor color gamuts







CIE Color classification



Human Vision is Trichromatic

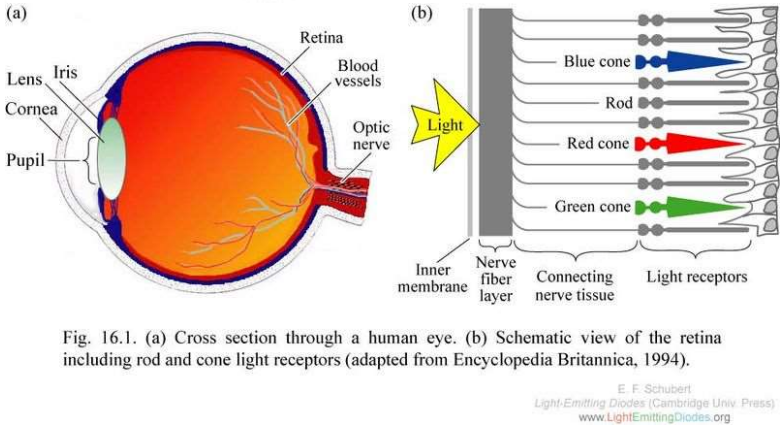
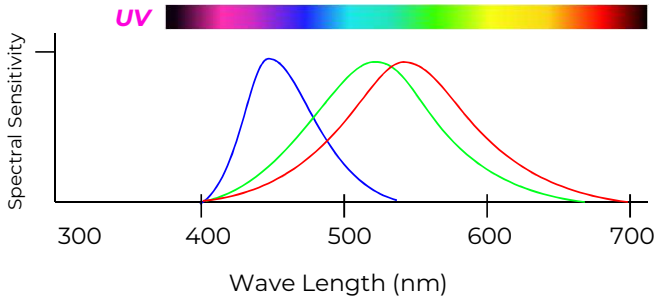


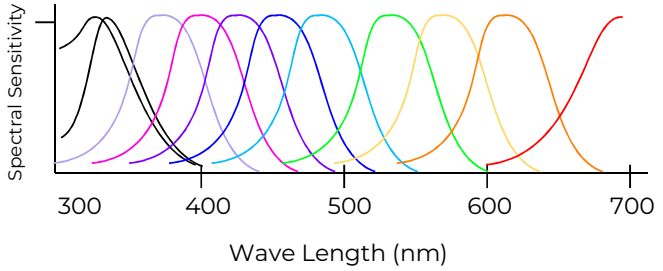
Fig. 16.1. (a) Cross section through a human eye. (b) Schematic view of the retina including rod and cone light receptors (adapted from Encyclopedia Britannica, 1994).

E. F. Schubert
Light-Emitting Diodes (Cambridge Univ. Press)
www.LightEmittingDiodes.org

Shrimp is better!

Neogonodactylus oestedii





Practical Machine Vision HOW TO's

- **Camera selection:**

https://www.baslerweb.com/en/vision-campus/vision-systems-and-components/camera-selection/

- **Lens & Lighting Selection:**

https://www.baslerweb.com/en/vision-campus/vision-systems-and-components/find-the-right-lens/

Resources

- Richard Szeliski, “Computer Vision” ISBN-13: 978-1848829343
- Rastislav Lukac, “Computtional Photography” ISBN-13: 978-1439817490
- Rick Nolthenius, Noah Snavely, George Bebis, S. Seitz, L. Zhang, D. Lowe, L. Fei-Fei, Antony Lam
- Pics: Wiki commons & Khan academy
- http://cvcl.mit.edu/hybridimage.htm
- MIT media lab materials, MERL edge detection
- https://vas3k.ru/blog/computational_photography/
- https://www.baslerweb.com/en/vision-campus/vision-systems-and-components/find-the-right-lens/
- http://www.pptsing.com
- https://www.cambridgeincolour.com/tutorials/cameras-vs-human-eye.htm
- https://ai.googleblog.com/2014/10/hdr-low-light-and-high-dynamic-range.html
- http://web.media.mit.edu/~raskar/Talks/ETCVparis08/raskarCompPhotoEpsilonCodedETVC08paper.pdf
- https://graphics.stanford.edu/talks/compphot-publictalk-may08.pdf
- https://www.cambridgeincolour.com/tutorials/cameras-vs-human-eye.htm

© Credits

- **Slide |2, 38, 47|** Image credit by MetalSunde94;
- **Slide |3,4,6|** https://en.wikipedia.org/wiki/The_Tower_of_Babel_(Bruegel) Skolkovo Institute of Science and Technology;
- **Slide |9, 10|** Albert Einstein, official 1921 Nobel Prize in Physics photograph
- **Slide |14|** http://www.normankoren.com/Tutorials/MTF.html photo by Andrew Hutchings
- **Slide |20|** https://cheezburger.com/6847687168photo by Denise Johnson
- **Slide |30-34|** https://en.wikipedia.org/wiki/Saint_Basil%27s_Cathedral#/media/File:St._Basil_Cathedral.jpg photo by Anton Zelenov
- **Slide |54|** photo by Craig McKay
- **Slide |56-59|** photo by Jordan Whitfield
- **Slide |60,65|** Albert Einstein, official 1921 Nobel Prize in Physics photograph
- **Slide |63|** https://en.wikipedia.org/wiki/Palm_cockatoo
- **Slide |66|** Lena Söderberg
- **Slide |67|** source lazebnik
- **Slide |68|** https://en.wikipedia.org/wiki/Pattern#/media/File:Aloe_polyphylla_spiral.jpg
- **Slide |71|** Boston City Hall, designed by Kallmann, McKinnell, & Knowles
- **Slide |77|** Credit W. Freeman et al, “Computer Vision for Interactive Computer Graphics”, IEEE Computer Graphics and Applications, 1998
- **Slide |80|** Fergus, et al. “Removing Camera Shake from a Single Photograph”, SIGGRAPH 2006, MIT Ray and Maria Stata Center

© Credits

- **Slide |3,23|** Image credit by Stevacer
- **Slide |14|** https://www.wikihow.com/Use-Your-Digital-Camera%27s-ISO-Setting
- **Slide |22|** https://www.pyimagesearch.com/2018/12/17/image-stitching-with-opencv-and-python
- **Slide |24|** kainz https://ai.googleblog.com/2014/10/hdr-low-light-and-high-dynamic-range.html
- **Slide |26|** ai.googleblog.com
- **Slide |27|** Ignatov, et al, ICCV2017: arXiv: 1704.02470, 2017
- **Slide |29|** Yuan SIGGRAPH 2007
- **Slide |32, 49|** Lena Söderberg
- **Slide |33|** Good collection of kernels in python: https://gist.github.com/jdumont0201/4327a1c26293b44e8d9c1cf1deef25bb
- **Slide |47|** Source: Wikipedia
- **Slide |55|** Source: S. Seitz
- **Slide |58|** https://www.buzzfeed.com/
- **Slide |60|** https://vas3k.ru/blog/computational_photography/
- **Slide |67|** source lazebnik
- **Slide |76|** E F Schubert Light Emitting Diodes (Cambridge Univ Press https://www.freepngs.com/shrimp-pngs?pgid=ivyg5wny-25516ff4-5461-11e8-a9ff-063f49e9a7e4
- **Slide |91|** Ramesh
- **Slide |102|** Albert Einstein, official 1921 Nobel Prize in Physics photograph

Thank you!

TV Color Spaces – YUV, YIQ, YC_bC_r

- YUV, YIQ: color encoding for analog NTSC and PAL
- YC_bC_r: Digital TV encoding
- Key common ideas:

  Separate luminance component Y from 2 chroma components
  Instead of encoding colors, encode color differences between components (maintains compatibility with black and white TV)

YUV color space

- Basis for color encoding in analog TV in North America (NTSC) and Europe (PAL)
- Y components computed from RGB components as

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

- UV components computed as

$$U = 0.492 \cdot (B - Y) \text{ and } V = 0.877 \cdot (R - Y)$$

YUV color space

- **Entire transformation from RGB to YUV**

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

- **Invert matrix above to transform from YUV back to RGB**

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.000 & 0.000 & 0.140 \\ 1.000 & -0.395 & -0.581 \\ 1.000 & -0.515 & -0.100 \end{pmatrix} \cdot \begin{pmatrix} Y \\ U \\ V \end{pmatrix}$$