— CM50248 — 2017/2018 —

Visual Understanding 1

Dr Christian Richardt

The course

- **Lectures:**
  - Monday 9:15 (3E 3.1) and Friday 15:15 (EB 0.9) – weeks 1 to 3 only
- **Labs:**
  - Monday 17:15 (1E 3.9) – from week 3
  - Flexible content: coursework and discussions
- **Assessment:**
  - 100% based on coursework
  - Part 1: using MATLAB — deadline: 13 November 2017
  - Part 2: using any language you like — deadline: 19 January 2018

Rough plan

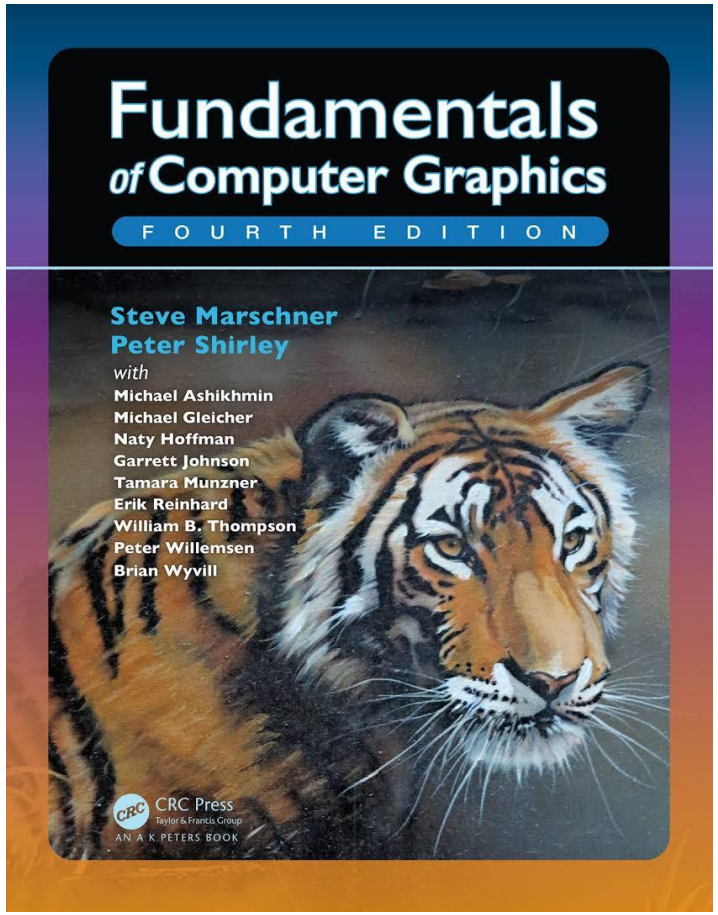
- introduction + motivation
- image formation, colours
- image filtering, convolution
- Fourier transform, convolution theorem
- scale space, pyramids, blobs
- Canny edges, Harris corners, SIFT
- planar geometry, feature matching, homographies
- RANSAC, image stitching
- camera geometry, projection, calibration, undistortion
- epipolar geometry, fundamental matrix estimation
- triangulation
- dense correspondence: stereo + optical flow
- introduction to object recognition + learning for vision

Resources: Moodle page

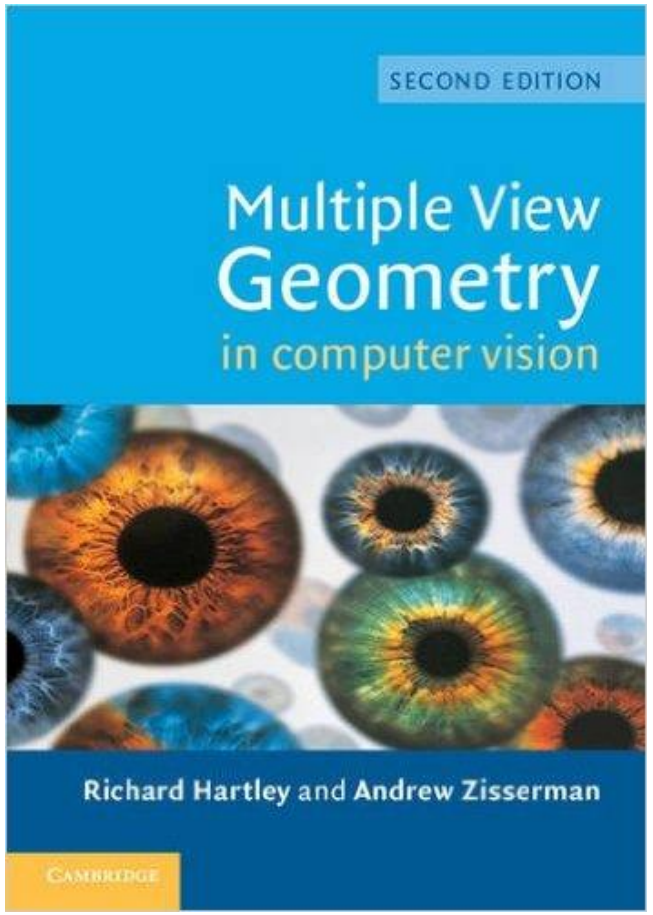
https://moodle.bath.ac.uk/course/view.php?id=56417

- Basic unit information
- Course notes
  - CM50248 (34 pages)
  - CM20219 (~80 pages, soon)
- Coursework information (soon)
- Lecture slides (after lectures)
- Q&A forum

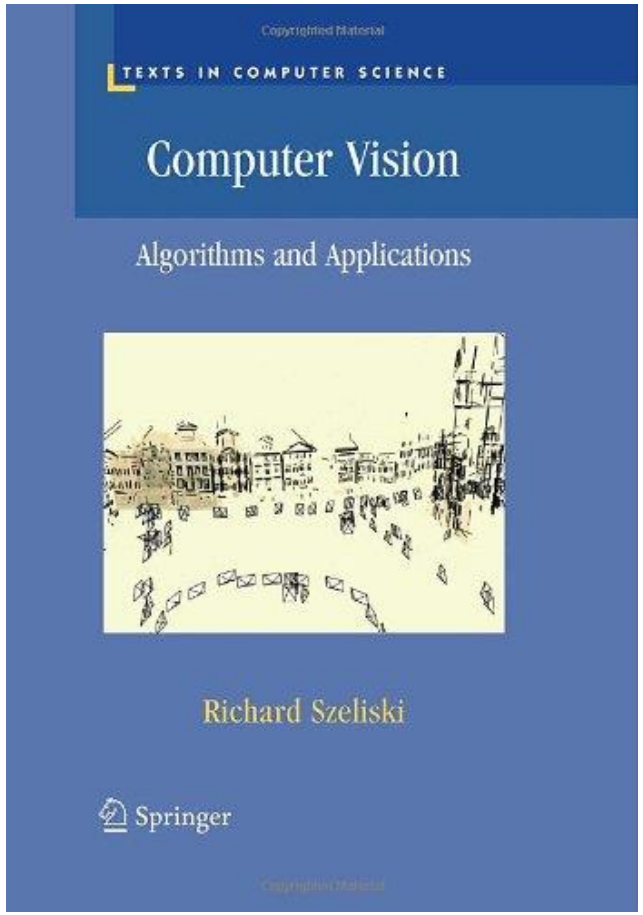
Resources: textbooks



Fundamentals of Computer Graphics
Marschner and Shirley et al., 2015

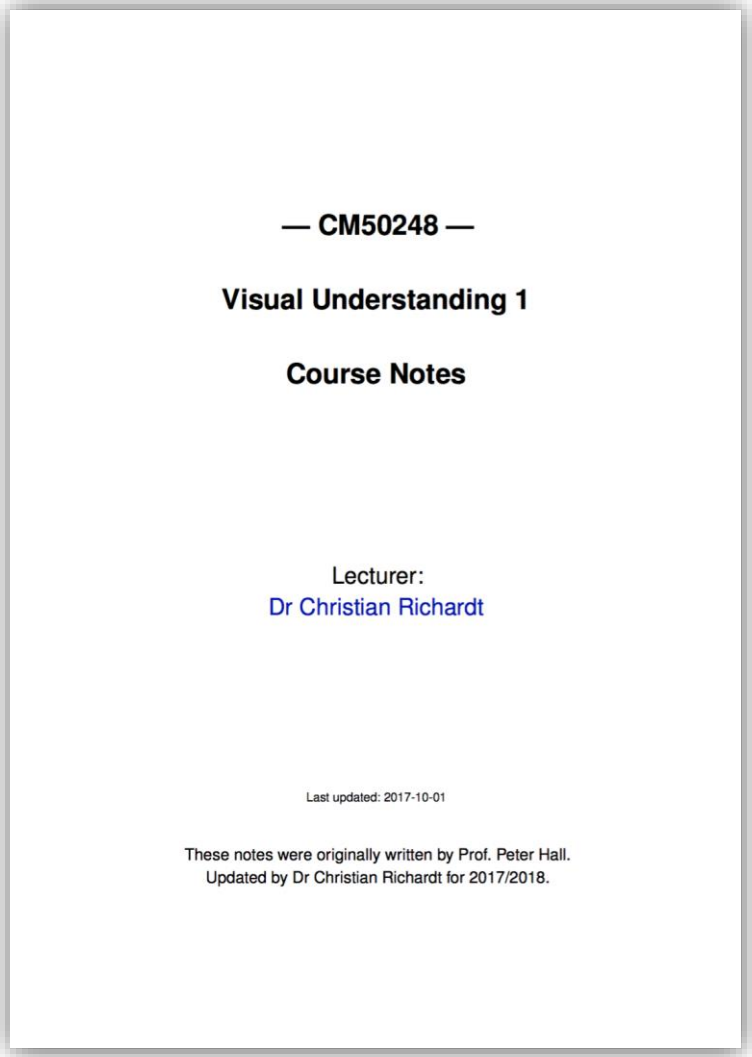


Multiple View Geometry in Computer Vision
Hartley & Zisserman, 2004



Computer Vision: Algorithms and Applications
Szeliski, 2010
Free: http://szeliski.org/Book/

Resources: CM50248 course notes



— CM50248 —

Visual Understanding 1

Course Notes

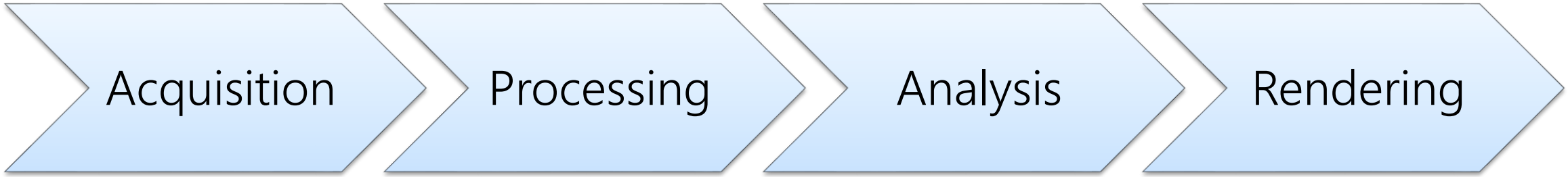
Lecturer:
Dr Christian Richardt

Last updated: 2017-10-01

These notes were originally written by Prof. Peter Hall.
Updated by Dr Christian Richardt for 2017/2018.

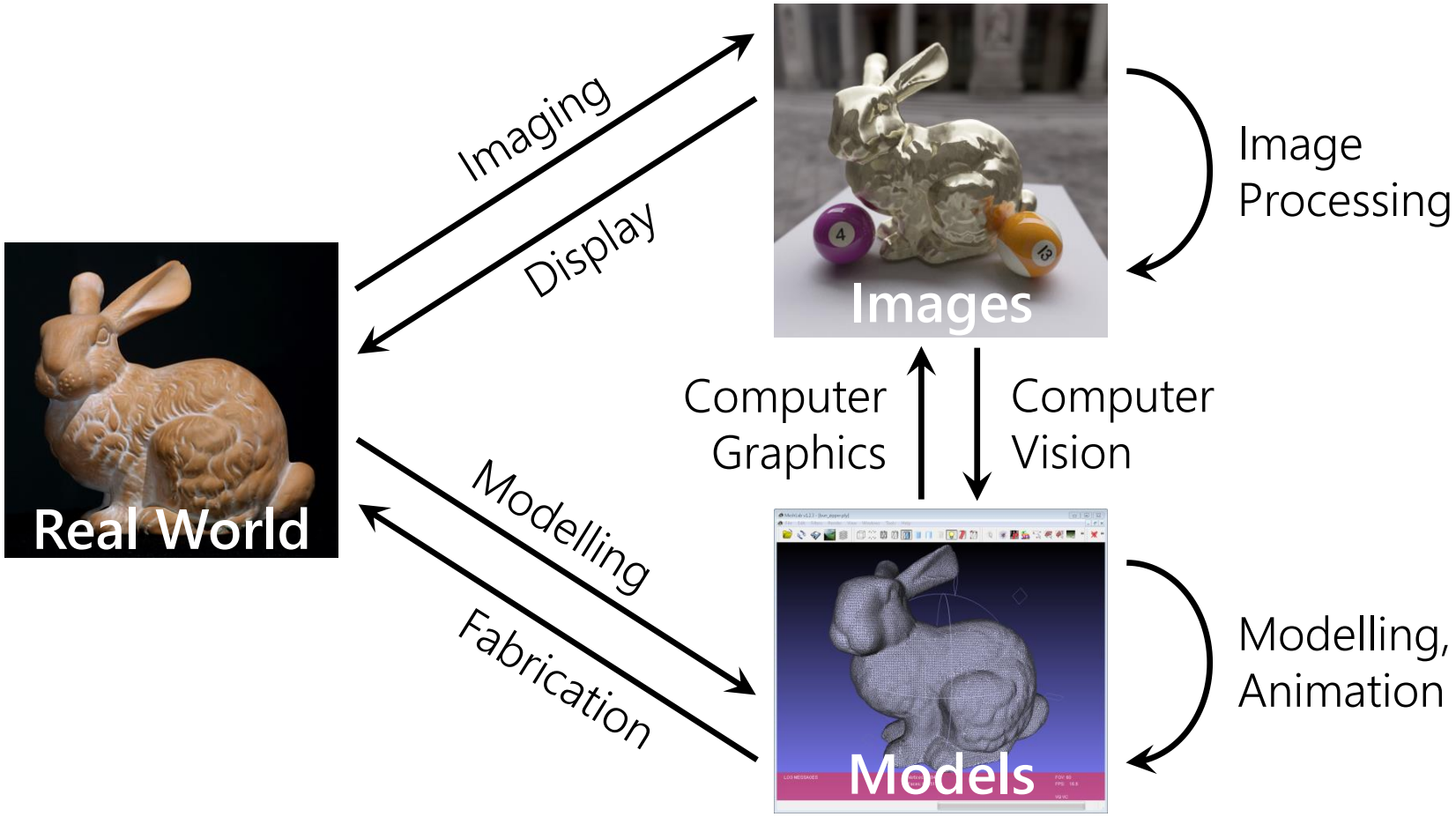
- Notes written for a previous version of this course
- freshly updated for 2017/2018
- Good coverage of a various topics covered in our course
  - But some parts are incomplete
  - Other parts are missing
  - If in doubt, refer to textbook, paper(s), online or slides
- Available on Moodle page

What is visual computing?



What is visual computing?

What is visual computing?



Digital imaging

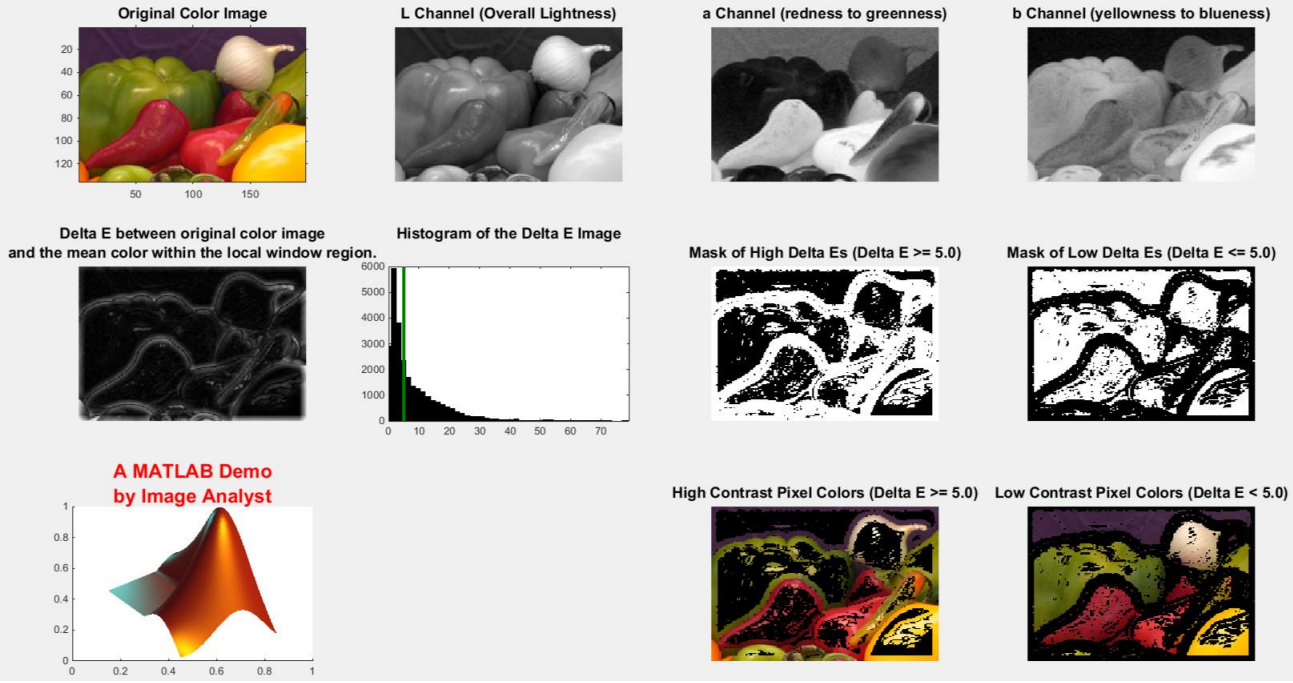
Capturing digital photographic images, usually in the visual spectrum

Paul Debevec et al., SIGGRAPH 2002

Image processing

Performing operations on digital images to produce another image

10steps.sg



By Image Analyst, MathWorks File Exchange

Displays and printers

Making a digital representation of an image visible using emissive (displays) or reflective media (paper)

Computer vision

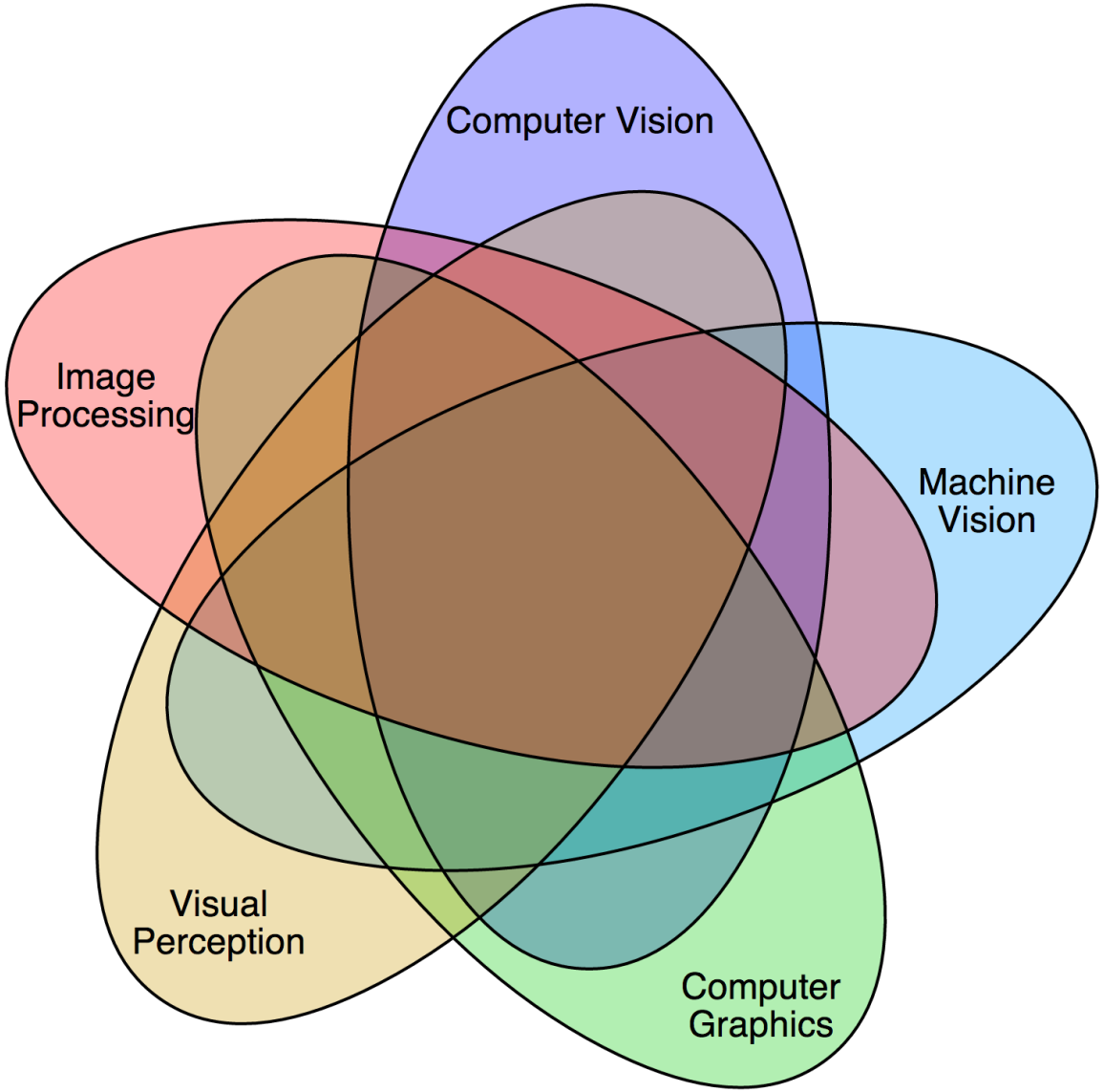
Understanding, interpreting and reconstructing information about real-world scenes from image and video data

Insafutinov et al., ECCV 2016

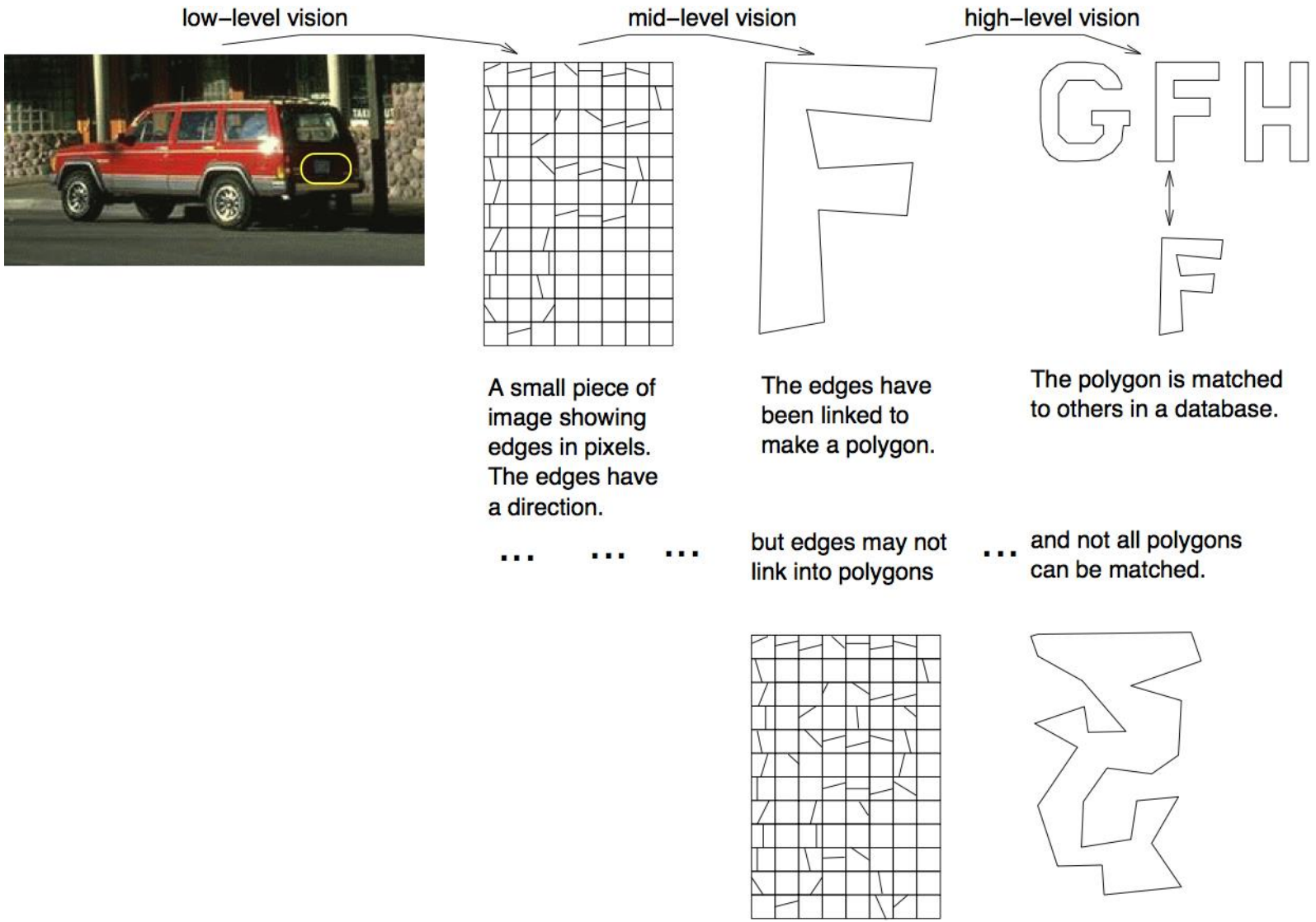
Neill Campbell

Computer vision

- Object/scene:
  - Detection, classification, recognition
  - Reconstruction
- Images:
  - Stitching
  - Enhancement
  - Restoration
- Motion tracking
- Segmentation



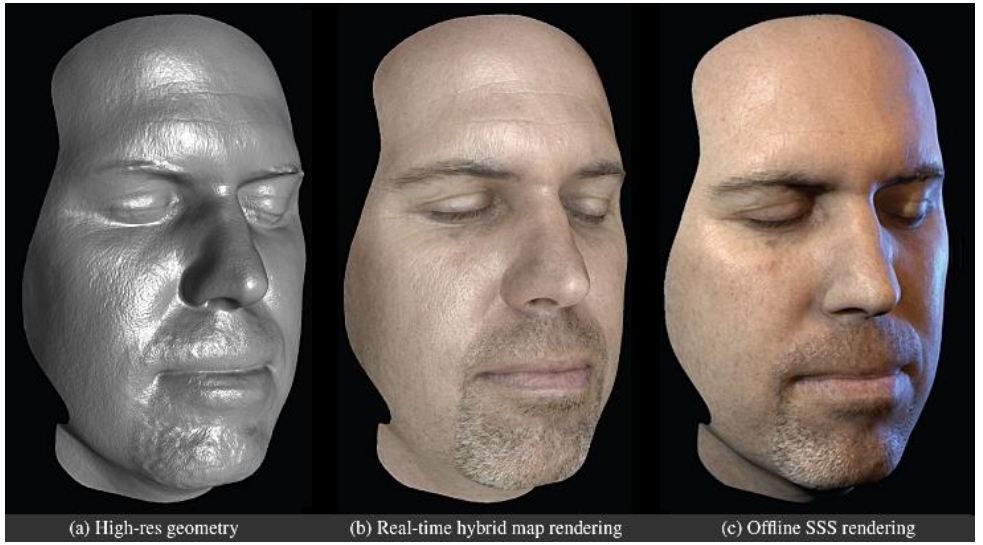
Bottom-up computer vision



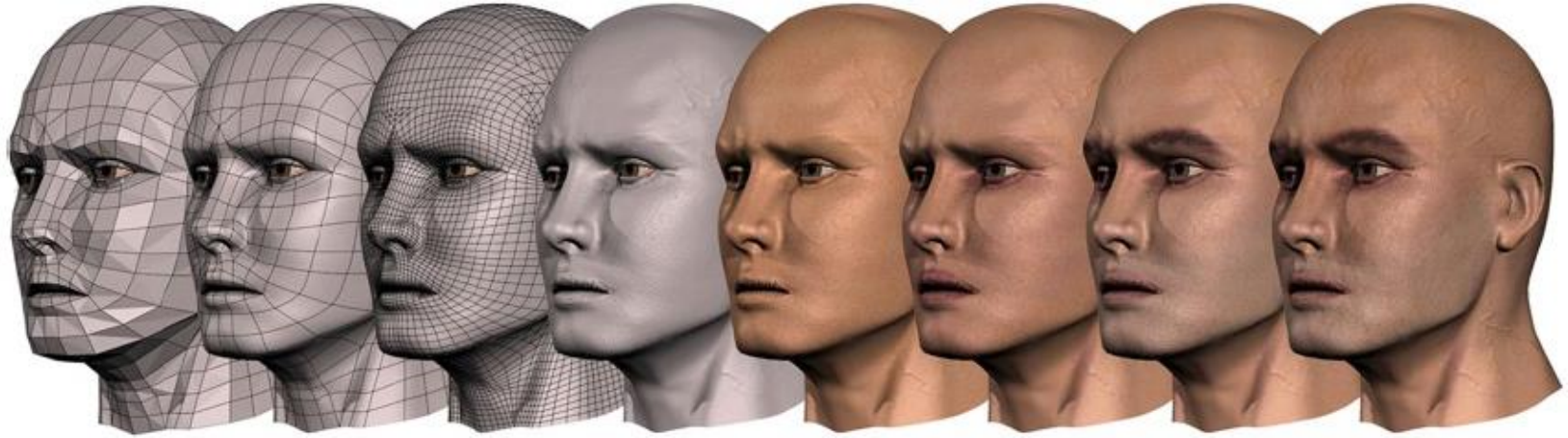
Why vision is hard

Modelling

Representing and manipulating virtual objects, including geometry, appearance and lighting properties



Ma et al., EGSR 2007



Animation

Putting virtual objects into motion using physical models and/or motion capture, e.g. to breathe life into inanimate objects

Pixar

Avatar, 20th Century Fox

Computer graphics

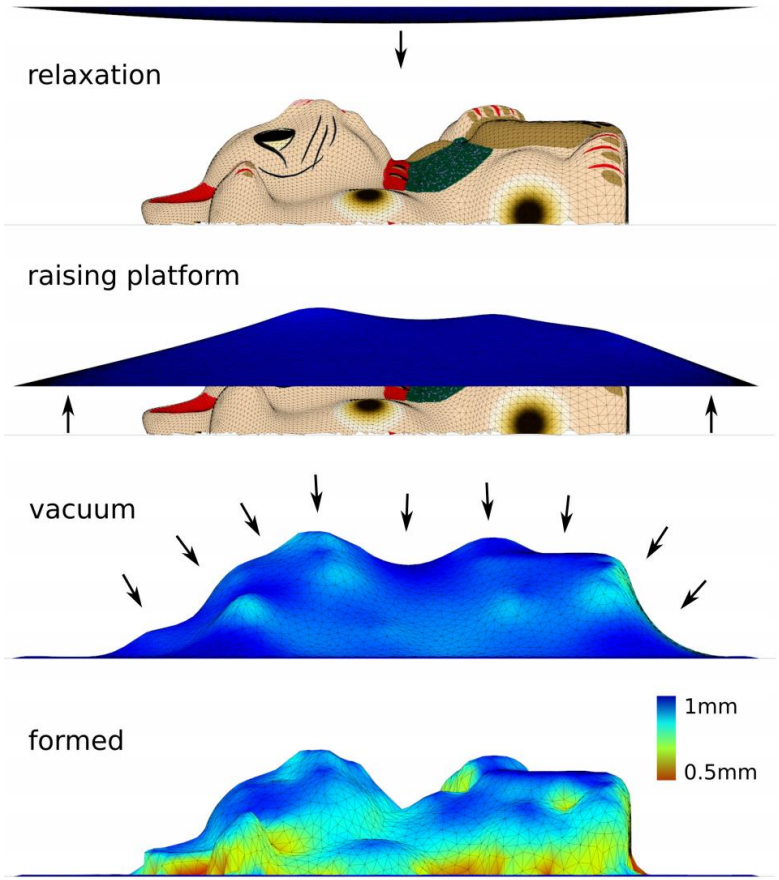
Synthesising an image from a model, generally models of the geometry, appearance and lighting properties of a scene

*"The screen is a window through which one sees a virtual world. The challenge is to **make** that world look real, act real, sound real, feel **real**."*

— Ivan Sutherland, 1965

Fabrication

Turning computational models into physical 3D objects using additive (e.g. 3D printing) or subtractive (e.g. laser cutting) processes

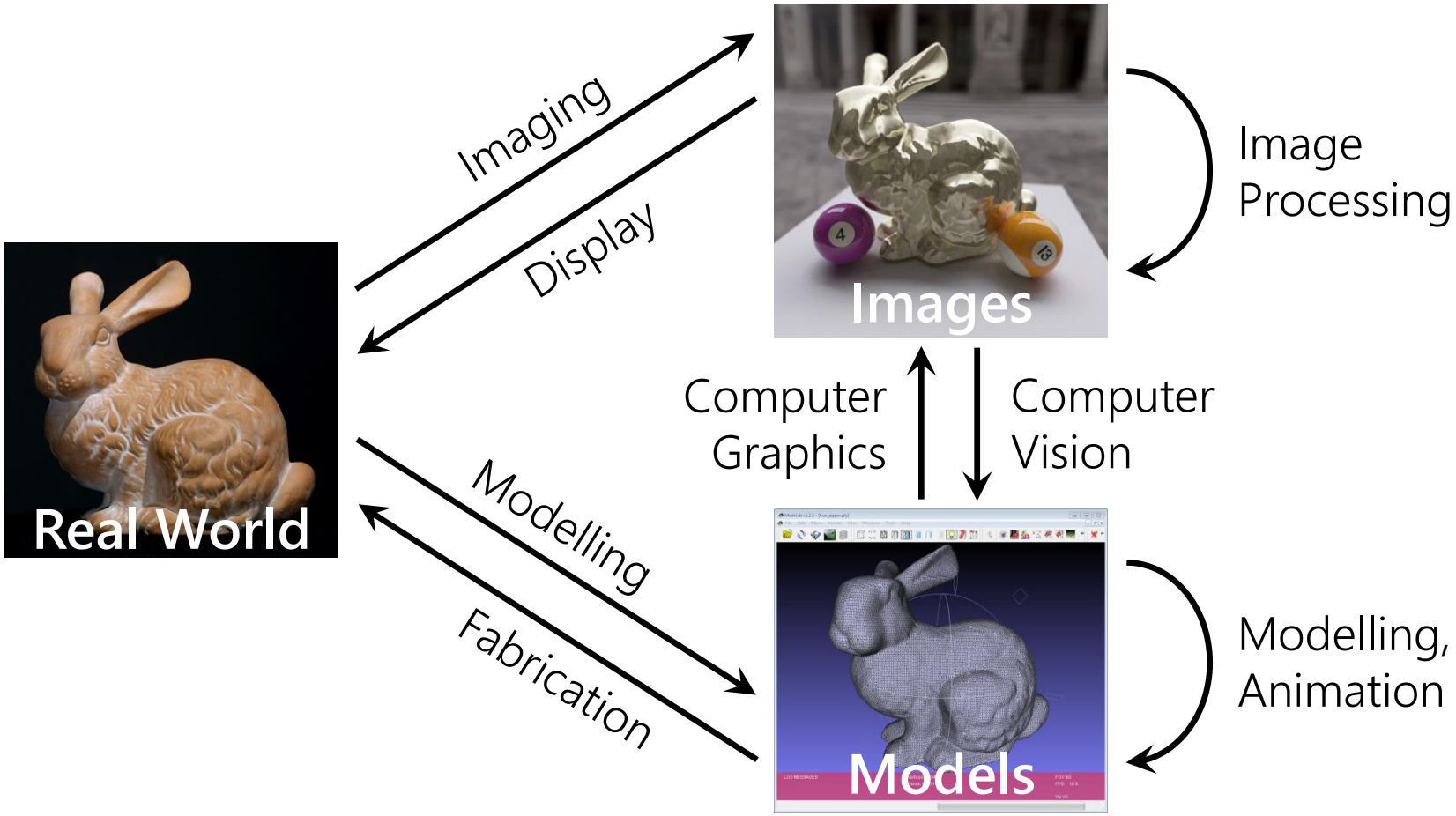


Schüller et al., SIGGRAPH 2016

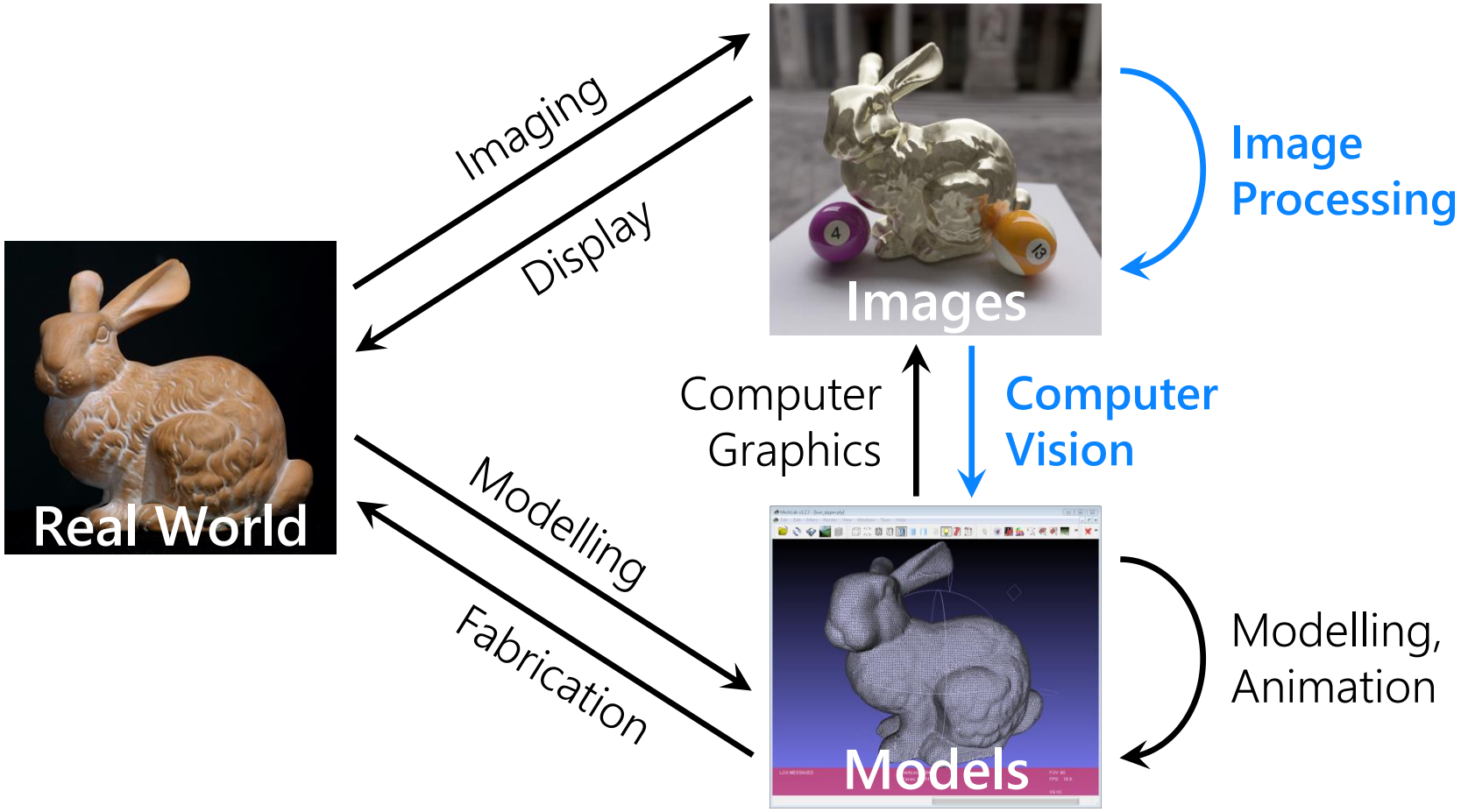
Wei et al., SIGGRAPH 2016

Brunton et al., SIGGRAPH 2016

Visual computing



This unit: Visual Understanding I



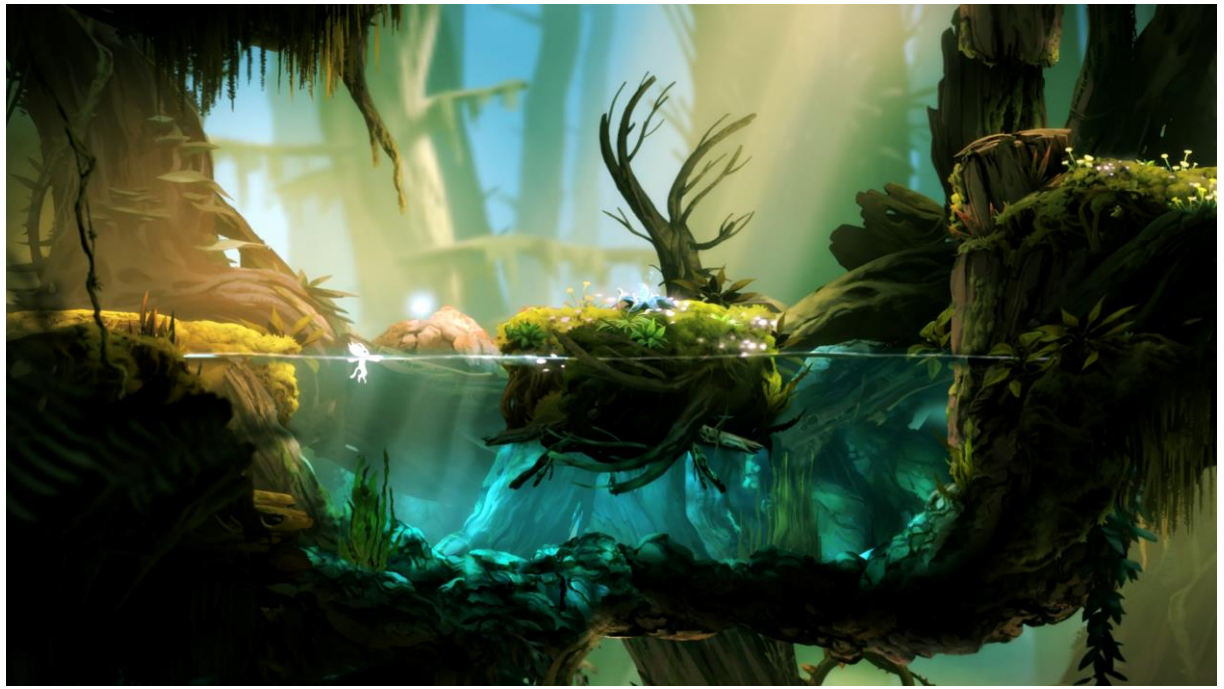
Applications – Movies

Inside Out
(Disney Pixar)

The Jungle Book
(Disney)

Applications – Computer games

Call of Duty: Advanced Warfare
(Activision)



Ori and the Blind Forest
(Microsoft Studios)

Applications – Virtual and augmented reality

HTC Vive

(Valve)

HoloLens

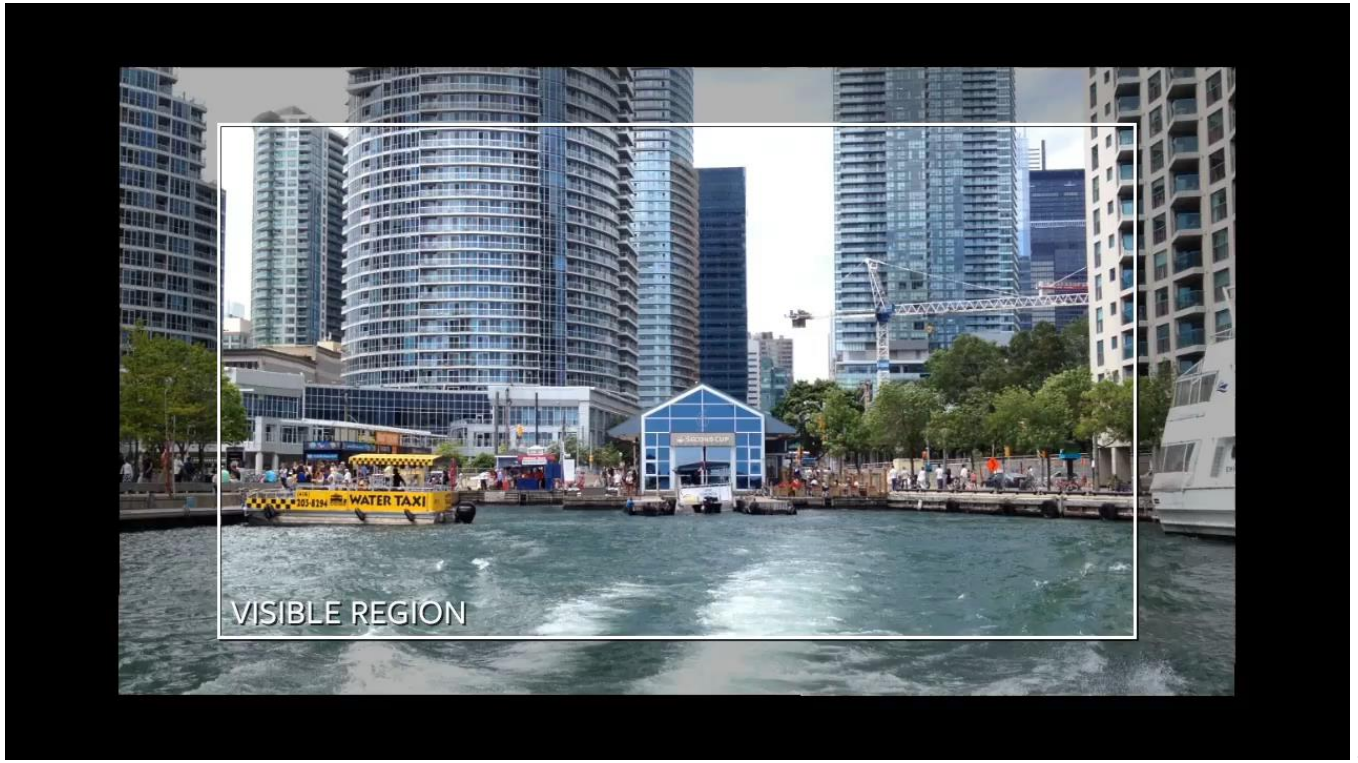
(Microsoft)

Applications – Computational photography

https://youtu.be/oQpg4TM96Ow?t=11s

Time lapses

(Martin-Brualla et al., 2015)



https://vimeo.com/104515109

Hyperlapses

(Instagram, 2014)

Applications – Computer-aided design (CAD)

London King's Cross Station

(John McAslan + Partners)

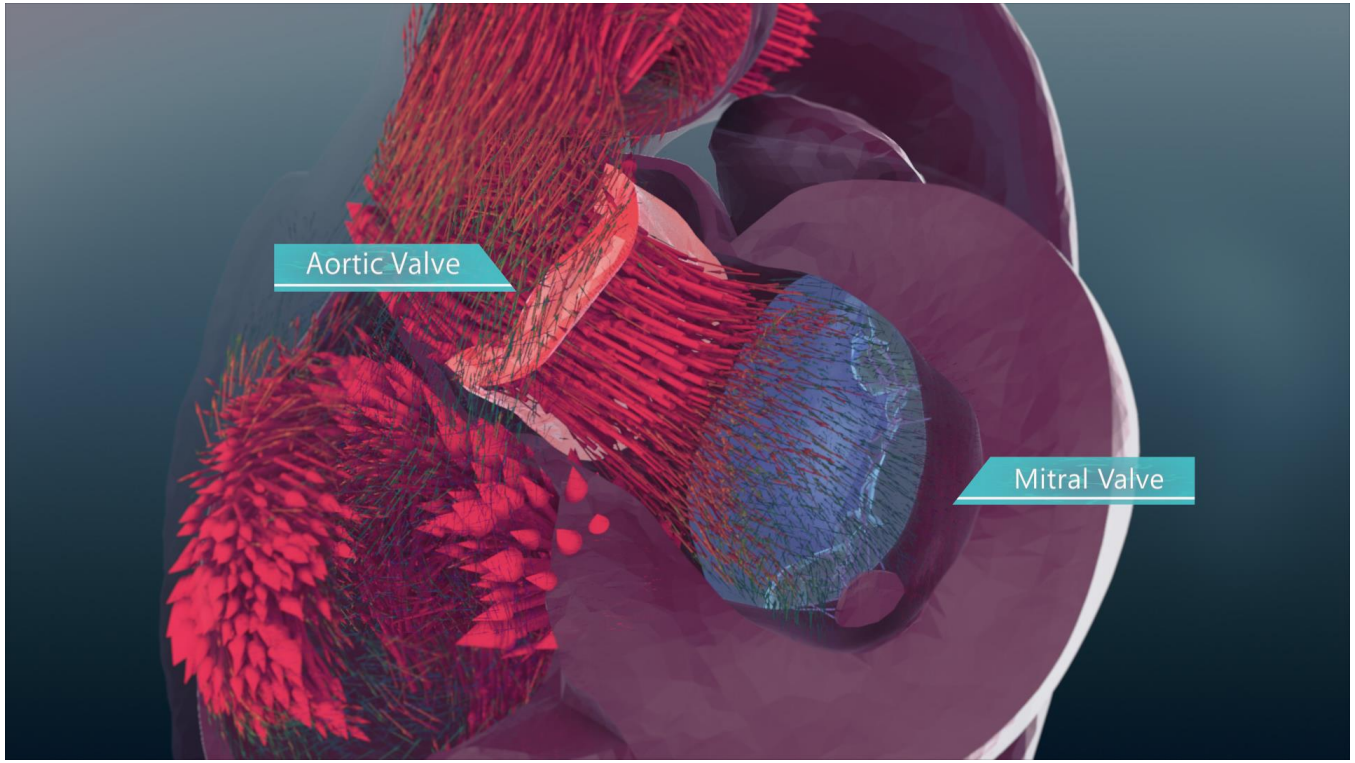
Airbus A380

(Airbus S.A.S.)

Applications – Scientific visualisation

Surgery Simulation

(U. Aarhus)



Multi-scale Multi-physics Heart Simulator, UT-Heart

(Sciement, Inc.)

Applications – Advertising

iPhone X

(Apple)

Vive

(HTC)

Leica X

(Leica)

Surface Pro

(Microsoft)

Applications – Advertising

(TT Games, Stephen Bate)

Vectors recap

- Points (usually in 2D or 3D) are represented by **vectors** in Cartesian coordinates

$$\mathbf{p} = [x, y]^\top \qquad \mathbf{q} = [x, y, z]^\top$$

- Points can be represented as row or column vectors

$$\mathbf{p}' = \mathbf{p}^\top = [x, y]$$

- We use column vectors unless otherwise noted

Vector algebra

- Vectors have a norm (aka length or magnitude):

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top \quad \|\mathbf{x}\| := \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$$

- Vectors are linear: they can be scaled, added and subtracted:

$$\mathbf{x} = [a, b]^\top \quad \mathbf{y} = [c, d]^\top$$

$$k\mathbf{x} = [ka, kb]^\top$$

$$\mathbf{x} + \mathbf{y} = [a + c, b + d]^\top$$

$$\mathbf{x} - \mathbf{y} = \mathbf{x} + (-\mathbf{y}) = [a - c, b - d]^\top$$

Vector algebra

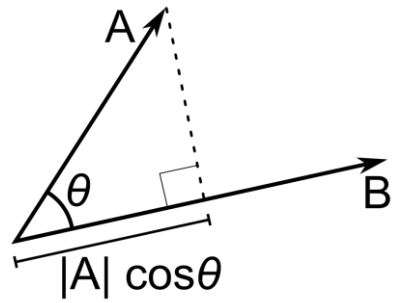
- Dot product / scalar product

$$\mathbf{x} = [x_1, x_2, \ldots, x_n]^\top$$

$$\mathbf{y} = [y_1, y_2, \ldots, y_n]^\top$$

$$\mathbf{x} \cdot \mathbf{y} = x_1 y_1 + x_2 y_2 + \ldots + x_n y_n$$

$$= \sum_{i=1}^{n} x_i y_i = \|\mathbf{x}\| \|\mathbf{y}\| \cos\theta$$

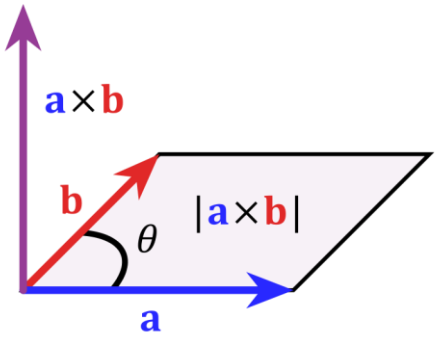


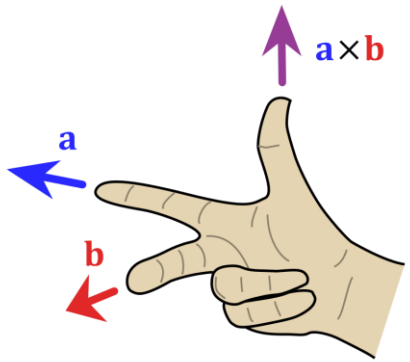
- Cross product / vector product

$$\mathbf{x} = [x_1, x_2, x_3]^\top$$

$$\mathbf{y} = [y_1, y_2, y_3]^\top$$

$$\mathbf{x} \times \mathbf{y} = \begin{bmatrix} x_2 y_3 - x_3 y_2 \\ x_3 y_1 - x_1 y_3 \\ x_1 y_2 - x_2 y_1 \end{bmatrix}$$





Matrices

- A matrix is a rectangular array of numbers:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- Matrices can be added, subtracted and scaled:

$$\mathbf{A} + \mathbf{B} = \begin{bmatrix} (a_{11} + b_{11}) & (a_{12} + b_{12}) \\ (a_{21} + b_{21}) & (a_{22} + b_{22}) \end{bmatrix}$$

$$s\mathbf{A} = \begin{bmatrix} sa_{11} & sa_{12} \\ sa_{21} & sa_{22} \end{bmatrix}$$

Matrix-matrix multiplication

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

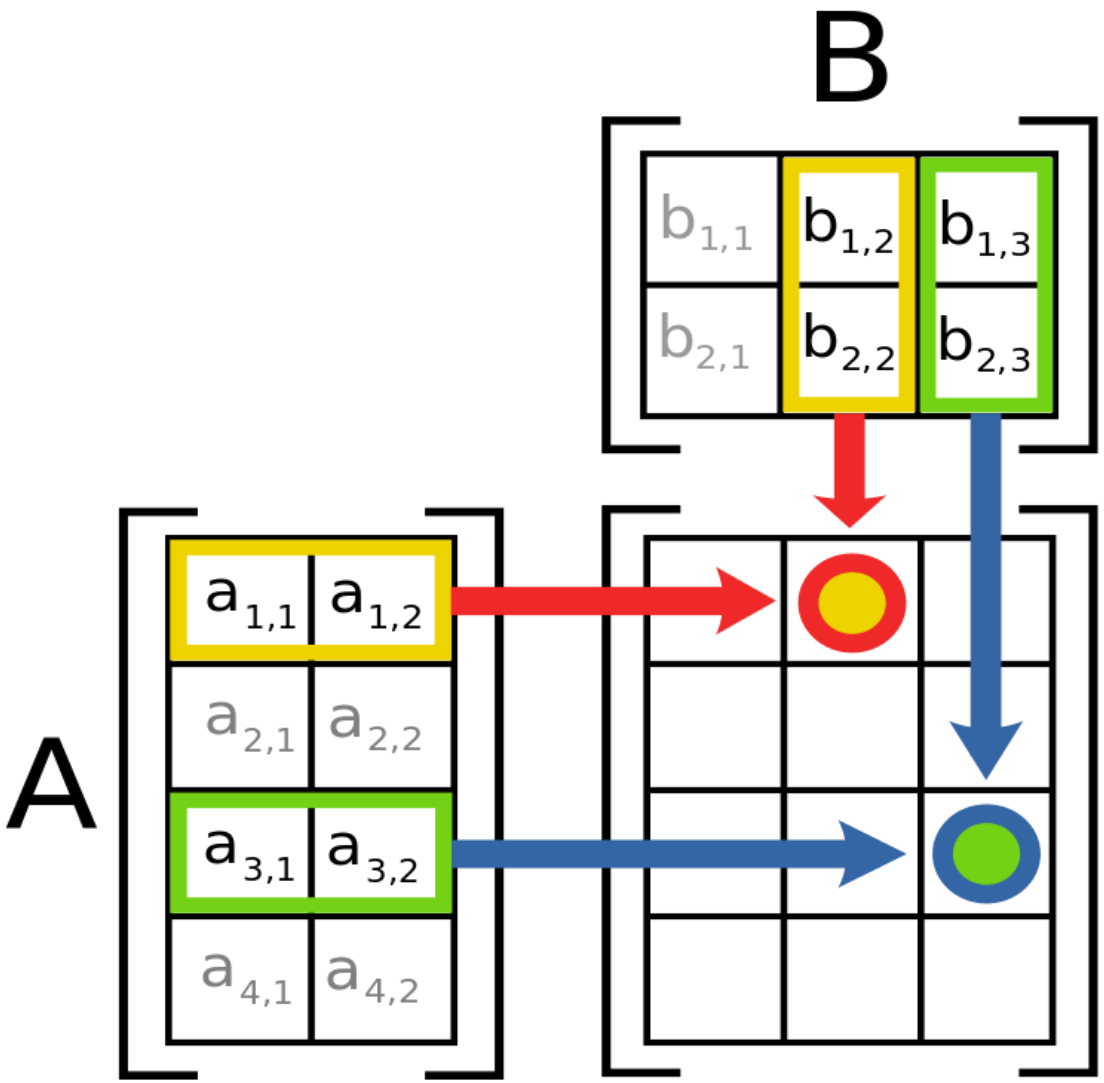
$$\mathbf{C} = \mathbf{AB} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} (a_{11}b_{11} + a_{12}b_{21}) & (a_{11}b_{12} + a_{12}b_{22}) \\ (a_{21}b_{11} + a_{22}b_{21}) & (a_{21}b_{12} + a_{22}b_{22}) \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Note: The number of columns in **A** must equal the number of rows in **B**.

Matrix-matrix multiplication



$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

$$c_{12} = \sum_{k=1}^{2} a_{1k} b_{k2}$$

$$= a_{11} b_{12} + a_{12} b_{22}$$

Note: The number of columns in **A** must equal the number of rows in **B**.

Some observations about matrices

- Addition behaves as expected:

$$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A}$$

$$s(\mathbf{A} + \mathbf{B}) = s\mathbf{A} + s\mathbf{B}$$

$$(\mathbf{A} + \mathbf{B})^\top = \mathbf{A}^\top + \mathbf{B}^\top$$

- Multiplication behaves mostly:

$$(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC})$$

$$(\mathbf{A} + \mathbf{B})\mathbf{C} = \mathbf{AC} + \mathbf{BC}$$

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC}$$

- But it is not commutative, i.e. in general $\mathbf{AB} \neq \mathbf{BA}$

- Note the transposition rule:

$$(\mathbf{ABC})^\top = \mathbf{C}^\top \mathbf{B}^\top \mathbf{A}^\top$$

Matrix inverse

- Some matrices can be inverted:

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I}$$

- Requires a square matrix and non-zero determinant (i.e. non-singular matrix)

- Linear system of equations

$$2x + 3y = 6$$

$$4x + 9y = 15$$

- Express in matrix form: $\mathbf{Ax} = \mathbf{b}$

$$\mathbf{A} = \begin{bmatrix} 2 & 3 \\ 4 & 9 \end{bmatrix} \mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} \mathbf{b} = \begin{bmatrix} 6 \\ 15 \end{bmatrix}$$

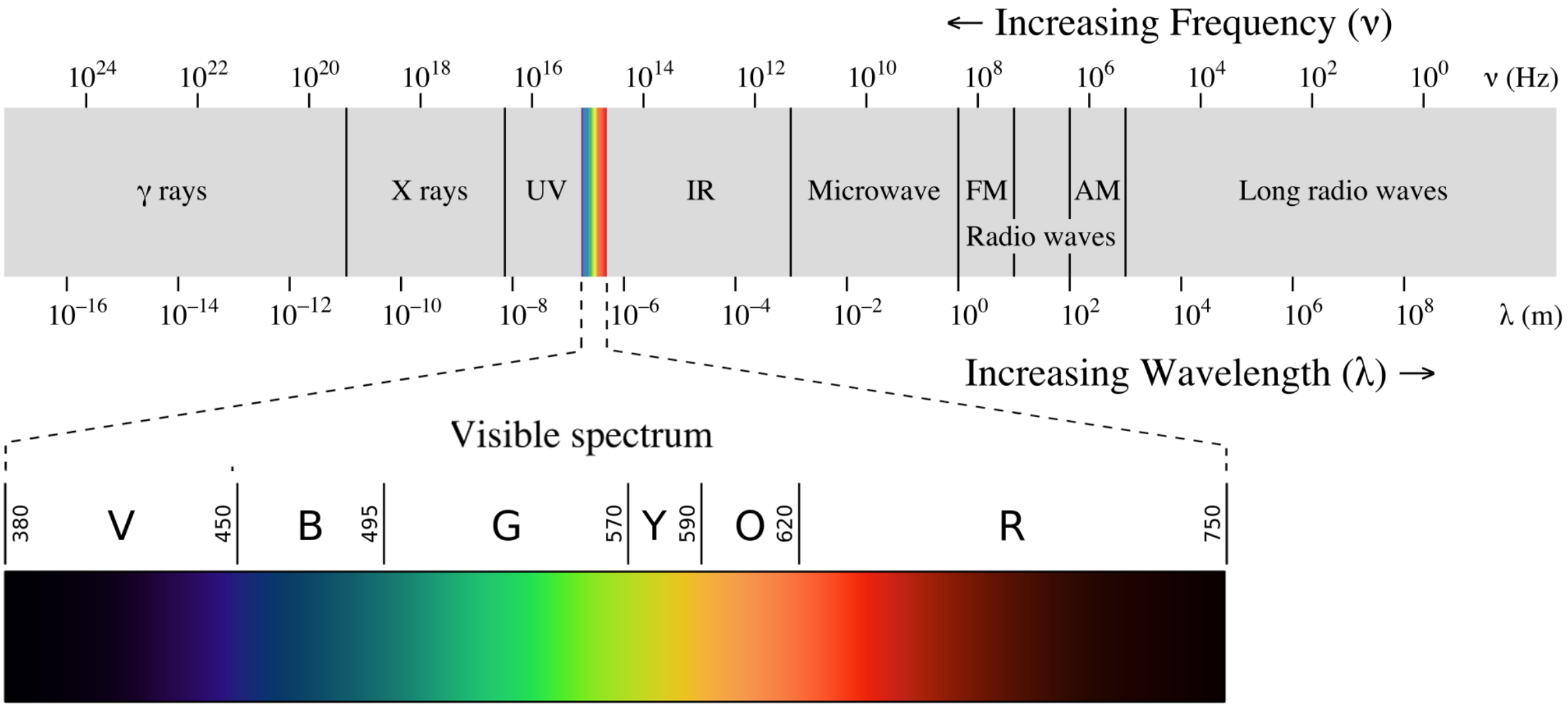
- Solve for unknowns:

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

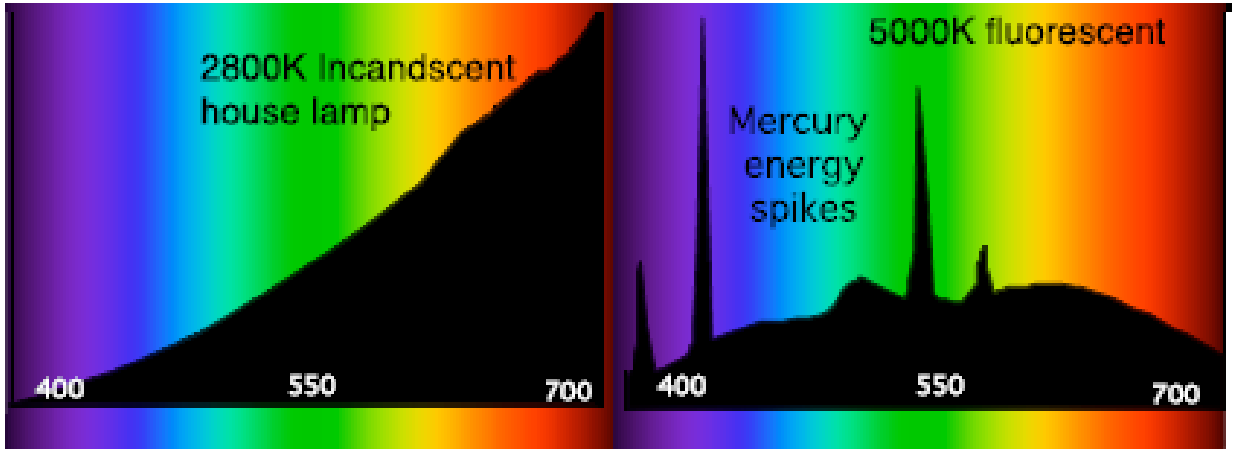
Let’s have a break

Visible light spectrum



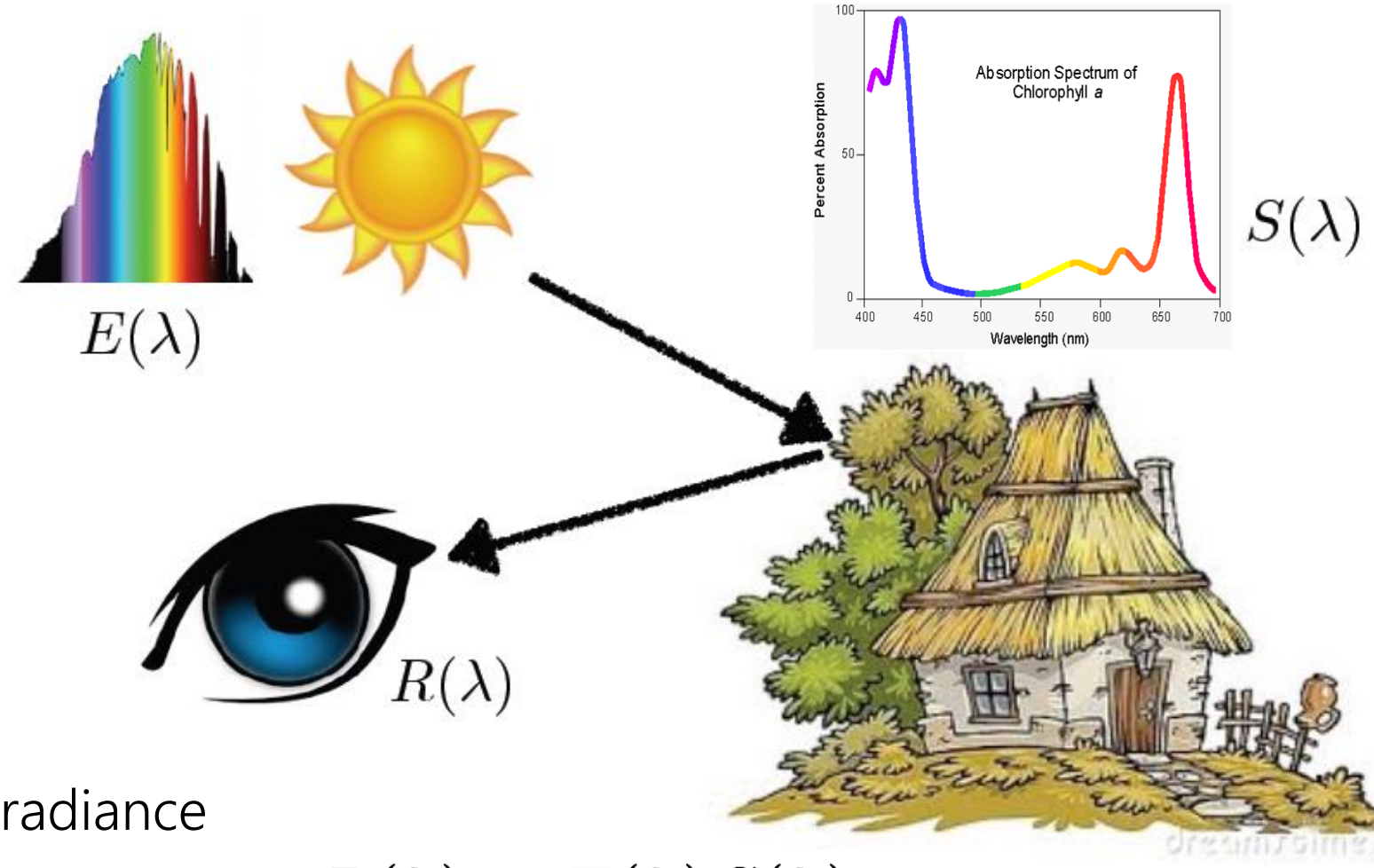
Spectral power distribution

- The light is usually a spectral distribution of energy
- The overall distribution in a light ray determines its colour



- Surfaces reflect light according to a spectral distribution as well
- The combination of incident spectrum and reflectance spectrum determines the light colour

Reflectance example

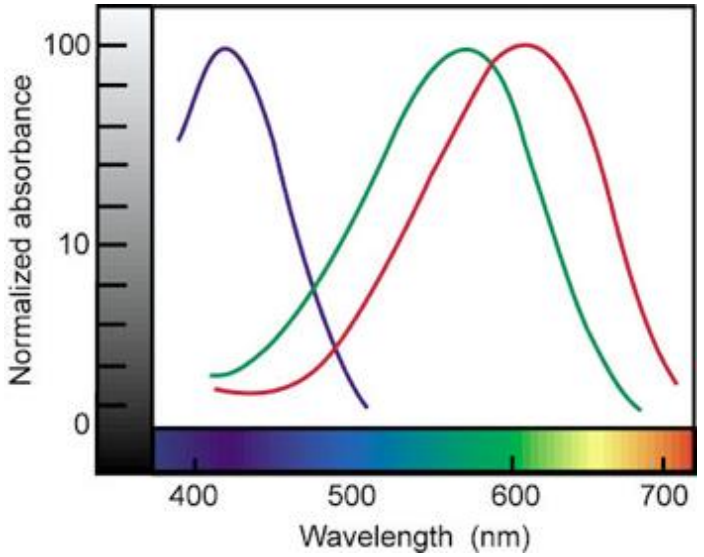


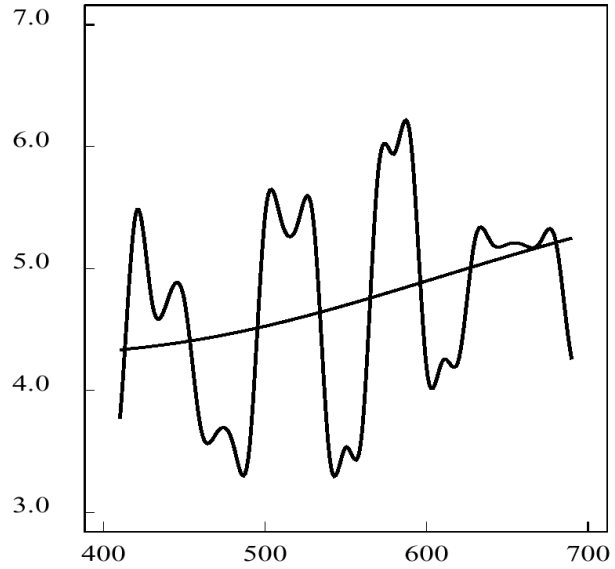
E: incident radiance
S: reflectance
R: outgoing radiance

$$R(\lambda) = E(\lambda)S(\lambda)$$

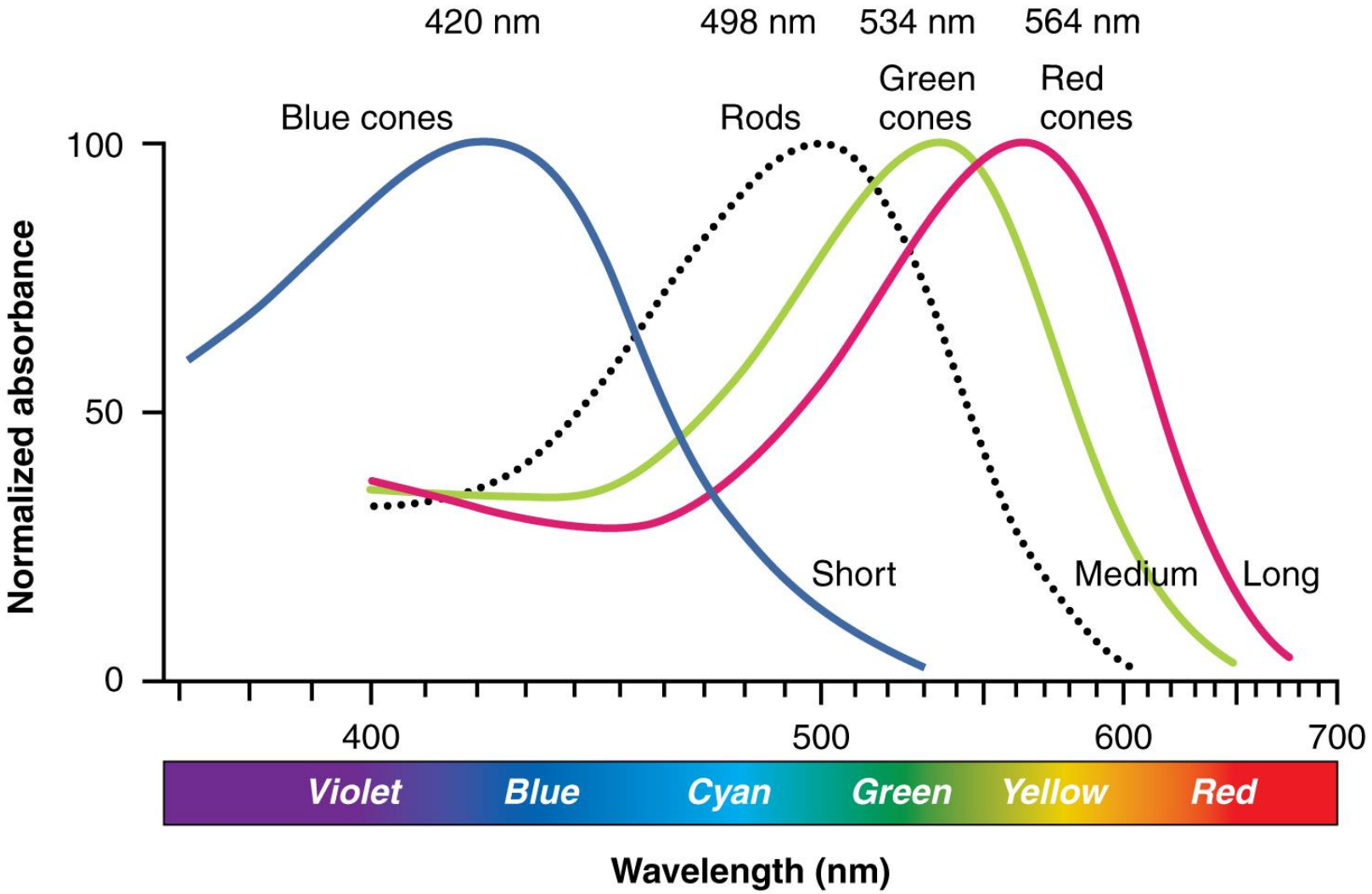
Cones and colour displays

- Humans have trichromatic vision (3 cone types)
  - sensitive to long, medium and short wavelengths (peaks near red, green and blue, respectively)
- We don't need displays capable of generating arbitrary spectral power distributions – humans are insensitive to much of the fine details
  - metamers: colours with different spectra, but the same cone responses
- We just need to generate spectral power distributions for all cone responses …



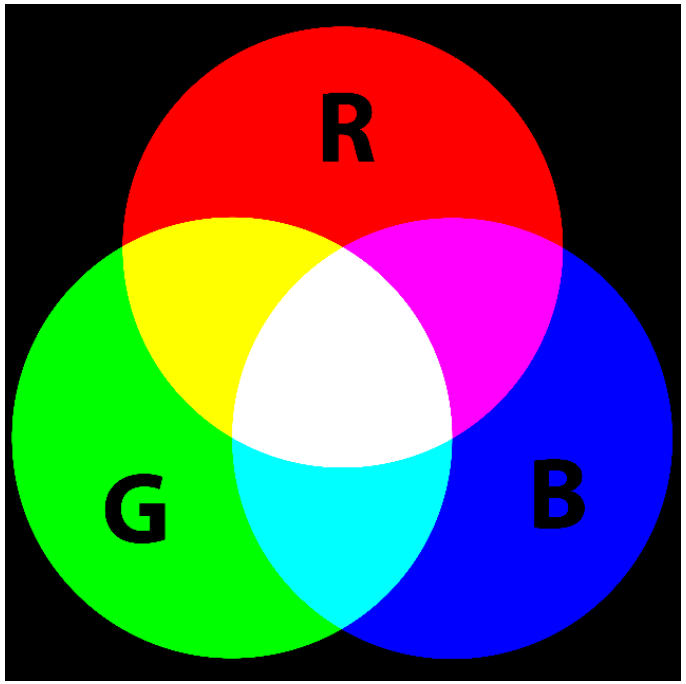


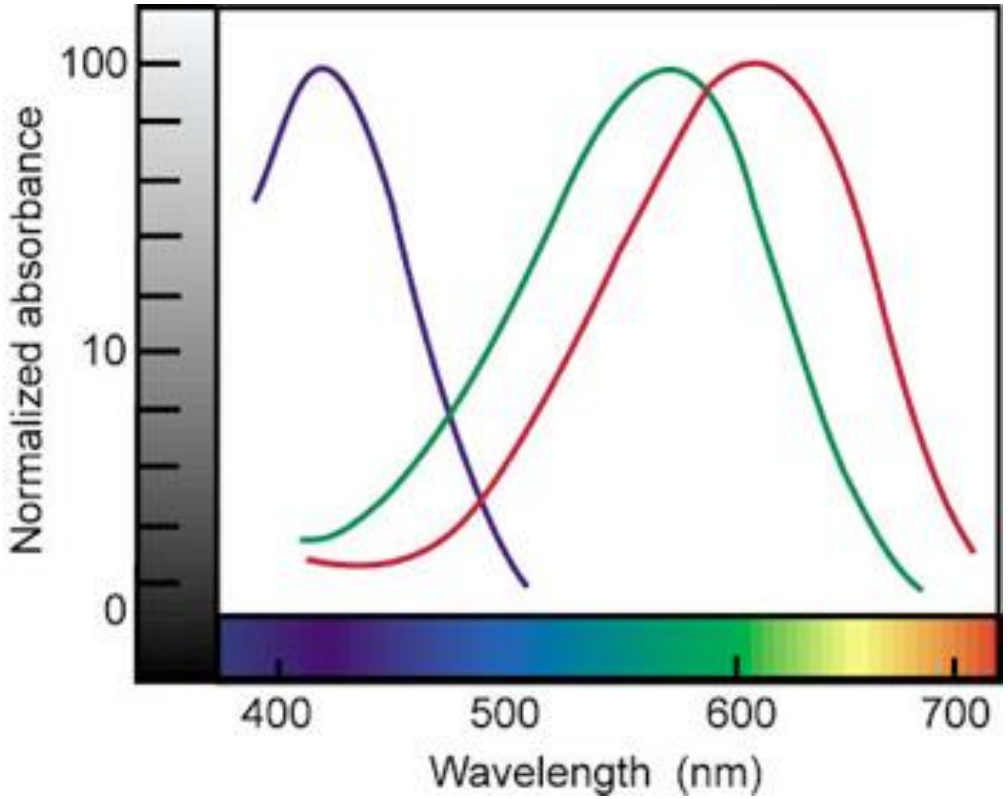
Cone responses in the human eye



- human retina has:
  - ~6 million cones (colour vision)
  - ~120 million rods (night vision)
- cones ≠ colours
- cones = short, medium, long wavelengths

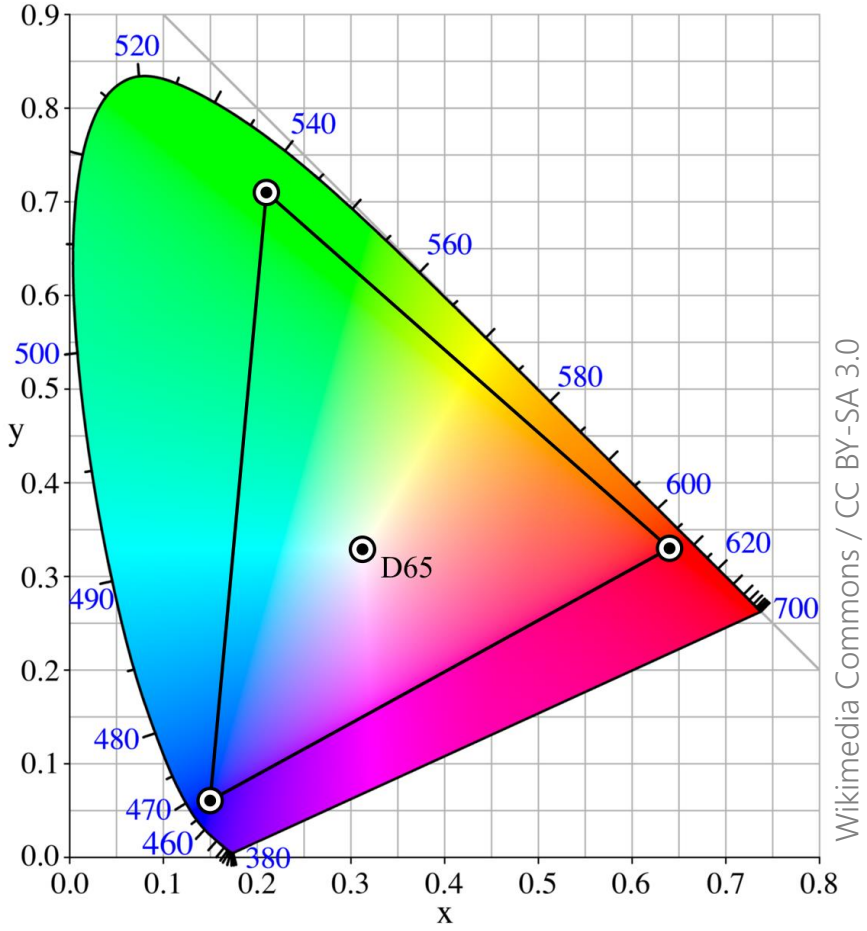
RGB colours


R
G
B


100
10
0
Normalized absorbance
400
500
600
700
Wavelength (nm)

RGB colour model

- **Red**, **Green** and **Blue** are often used to present other colours
- Can these generate every perceivable colour?
  - Yes, in theory, but some weights would need to be negative …
  - In practice, R, G, B can generate a good fraction of all visible colours

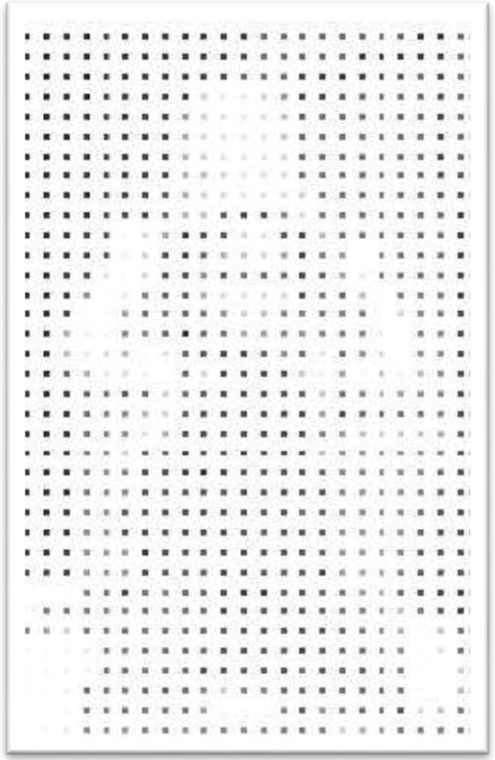


Adobe RGB (1998) primaries in the CIE xy chromaticity diagram

Digital images

- **Digital images** are 2D **sampled** representations (a 2D array) of some continuous function, like a real scene

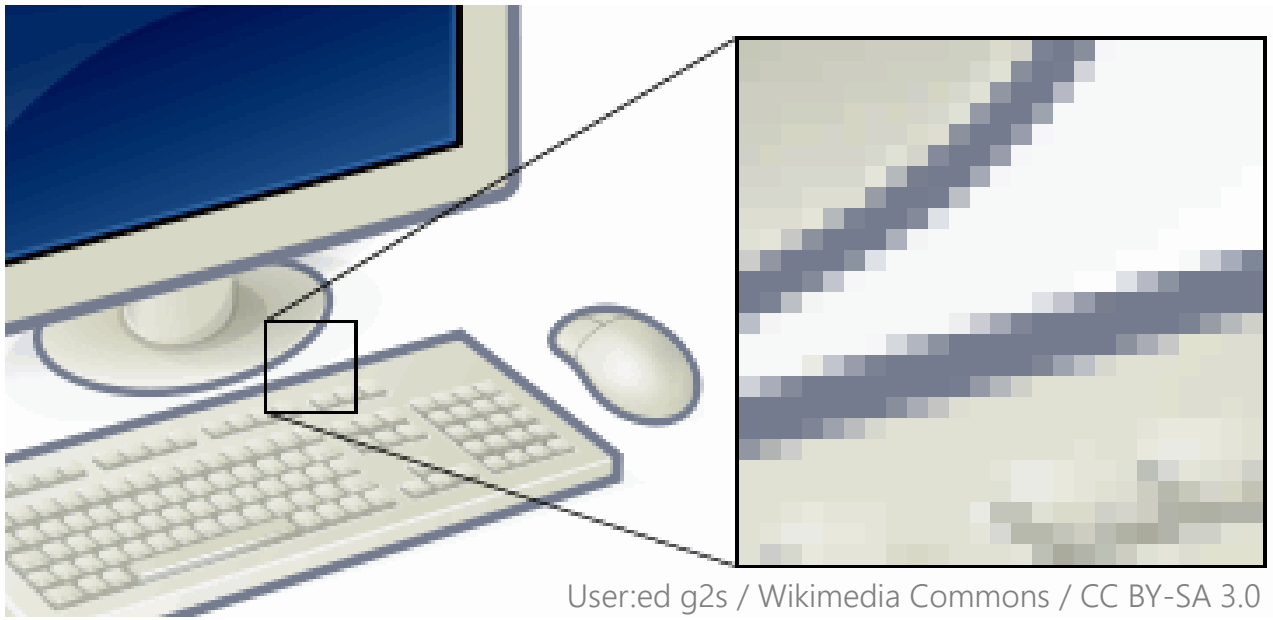
continuous function



discrete samples

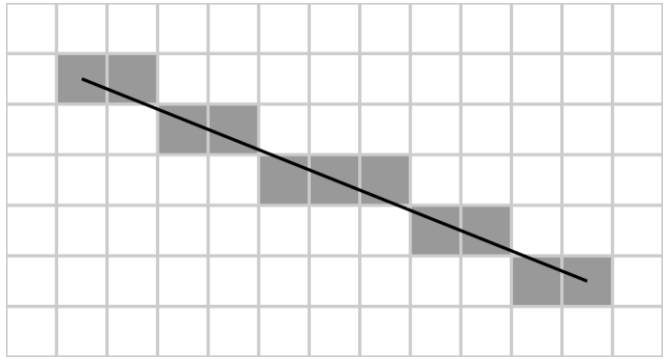
Pixels

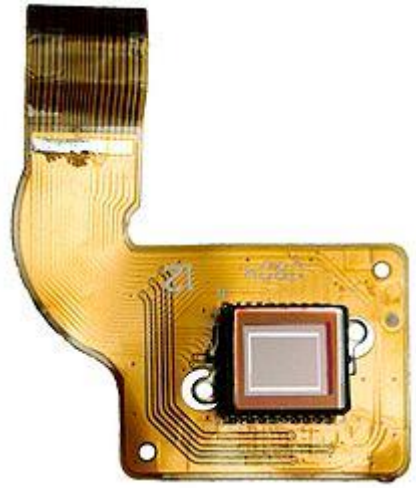
- The sampled digital values are called **pixels**
  - The smallest individual element in an image
- 1 pixel = single sampled colour
  - Usually represented in Red, Green, Blue (RGB)
  - Can represent many different colours



Digital image formation

- **Rasterisation**: convert a continuous or vector image representation to a rectangular sampled grid of pixels
- Rasterising vector graphics
  - e.g. Bresenham line drawing
- Sampling an analogue signal
  - e.g. in a digital camera
  - Digital cameras have a CCD or CMOS sensor with light sensitive elements, typically arranged in a “Bayer” pattern





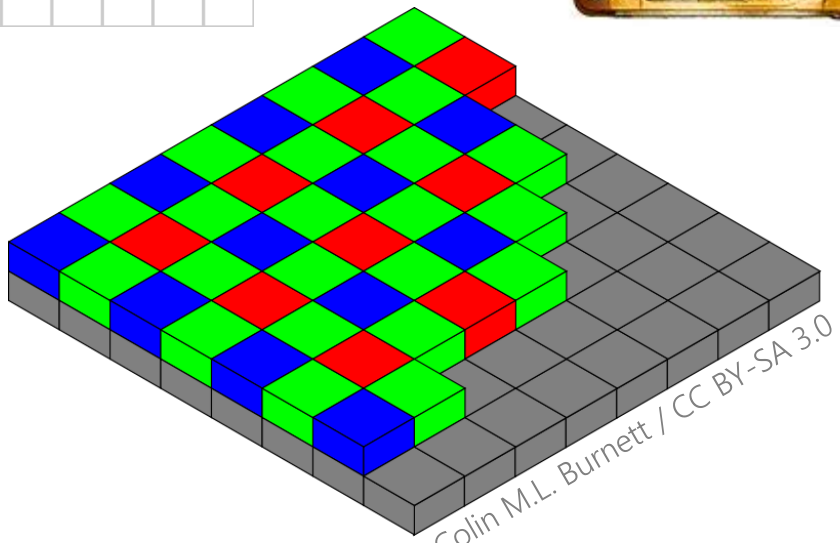
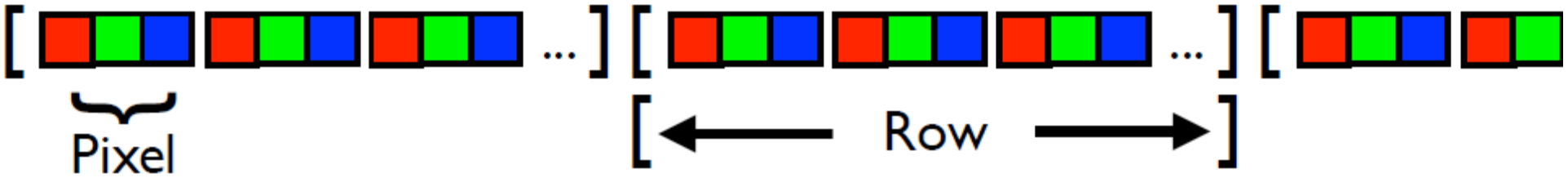


Image storage

- Digital images are arranged in memory with rows of RGB pixels:



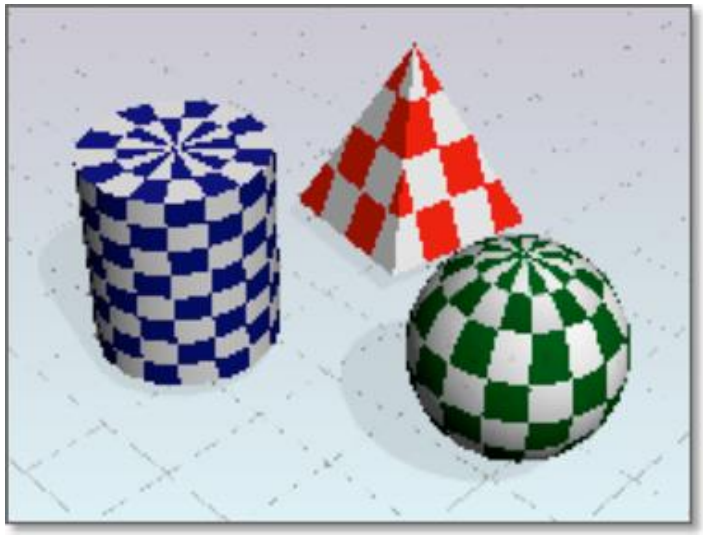
- However, there are other possibilities:
  - 1 value (channel) per pixel: black & white (“bitmap”), grayscale, indexed
  - 4 values per pixel (RGBA: red, green, blue and “alpha” for opacity)

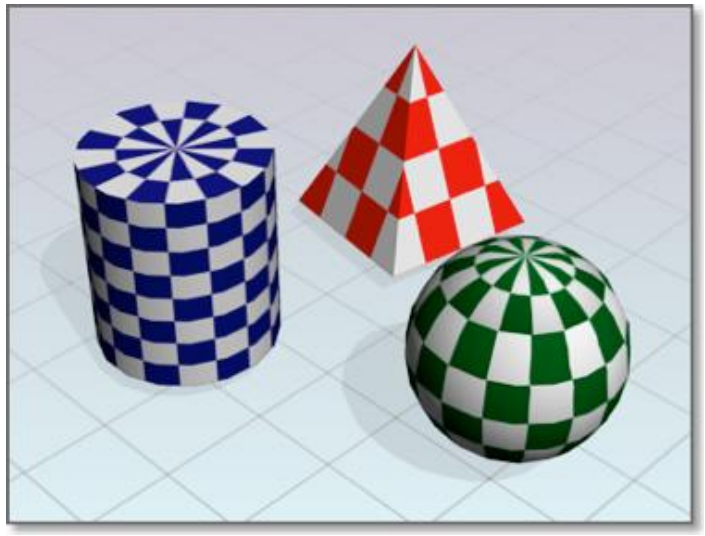
Image coding

- Consider a 10 megapixel image, stored with RGB colours and 32 bits per pixel: 10×10^6 pixels × 4 bytes/pixel = **40 MB**
- Need **image compression** in practice:
  - **Lossless** compression: e.g. PNG/GIF using LZW (Lempel-Ziv-Welch) coding (enumerate frequent strings)
  - **Lossy** compression: e.g. JPEG uses the DCT (discrete cosine transform) to encode high-frequency components with fewer bits

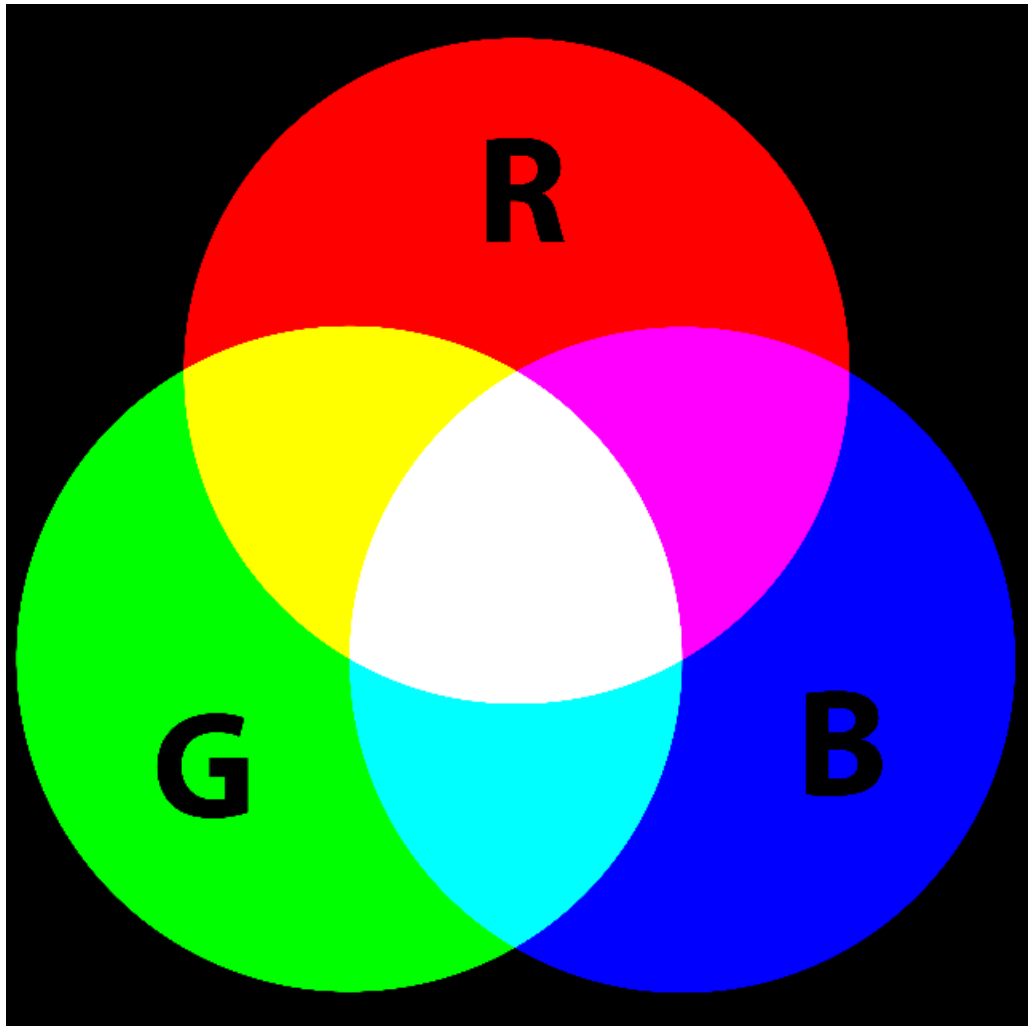
Image sampling

- Many image manipulations require accessing images at non-integer coordinates, i.e. not aligned with the pixel grid
- We typically need to **interpolate** between pixels, e.g. bilinear
- Also, we have to understand **sampling theory** to avoid aliasing (under-sampling) of the signal



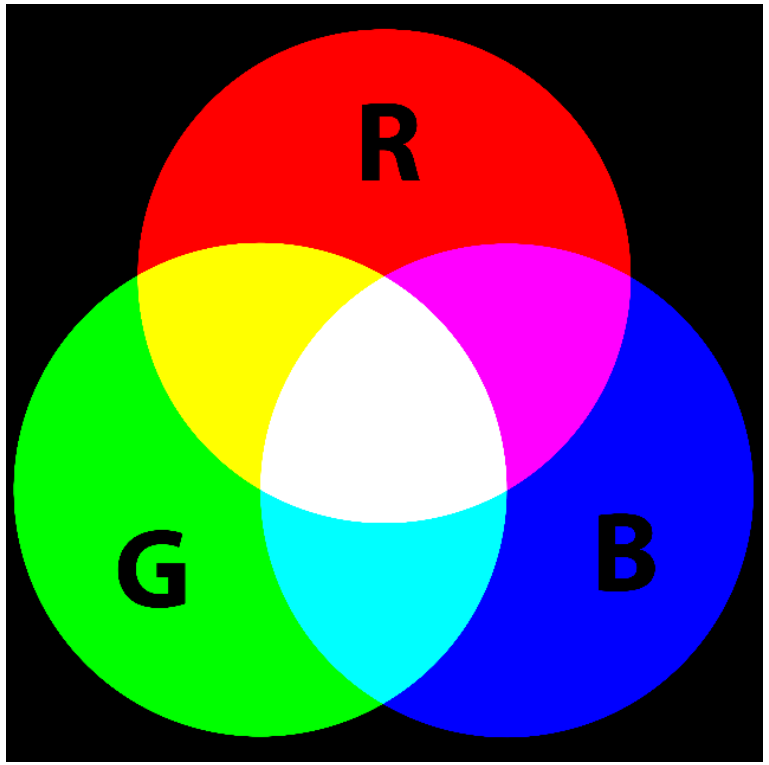


RGB colours: additive primaries



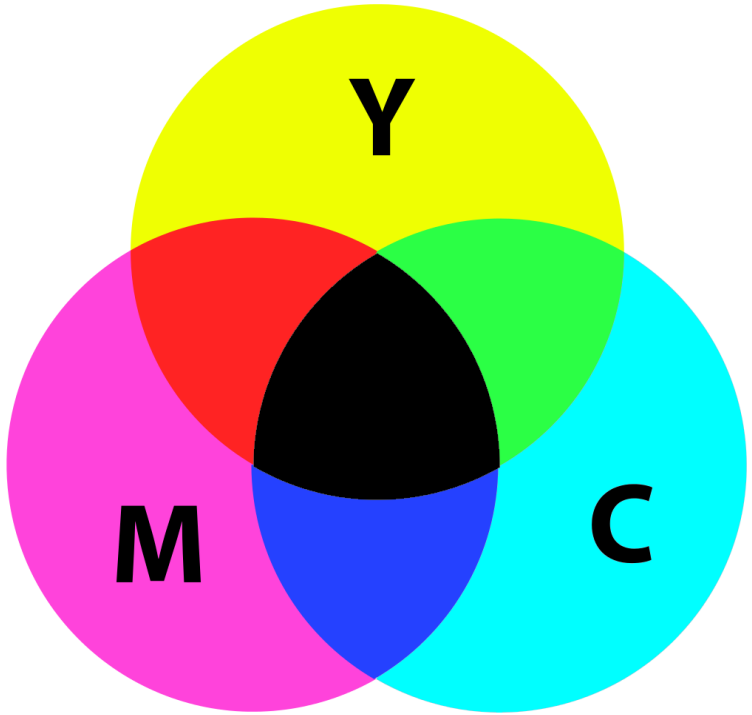
Additive and subtractive primary colours

RGB – additive



Mixing lights
e.g. LCD monitors

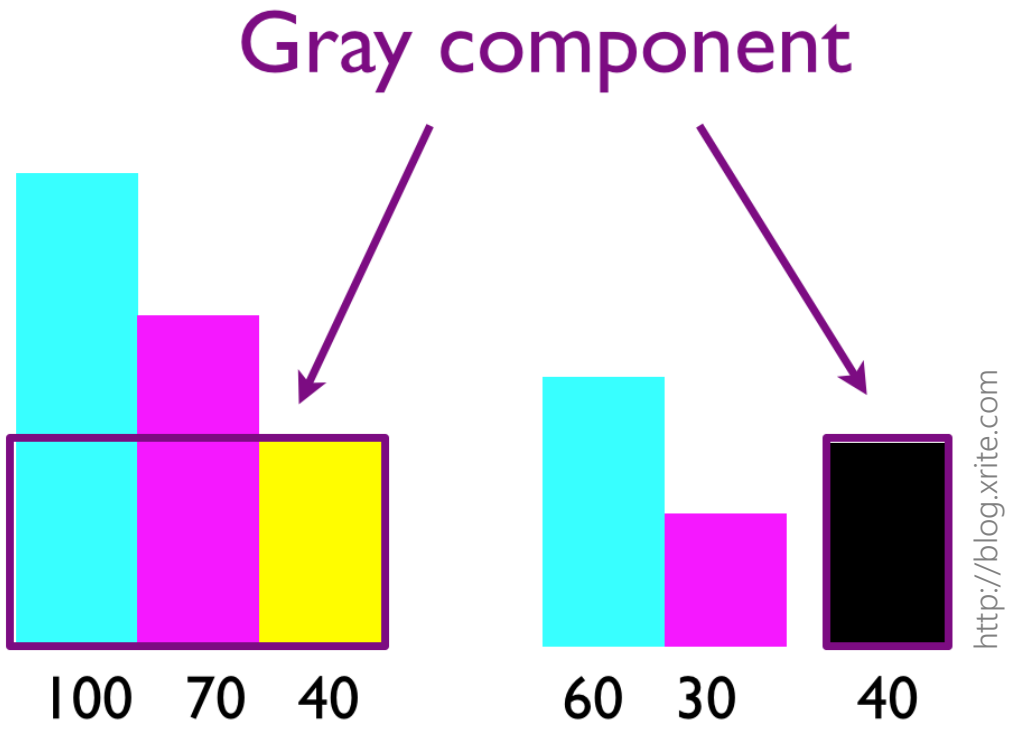
CMY – subtractive



Filters and paints
e.g. printers

CMY vs CMYK

Target colour



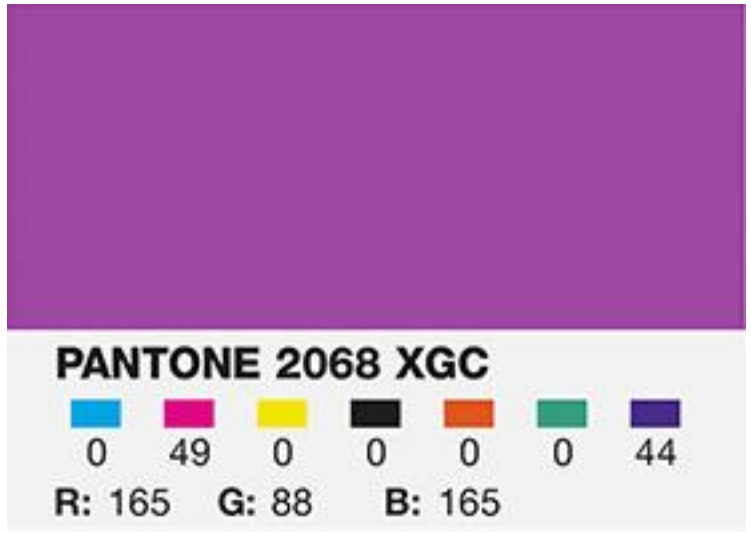
CMY CMYK

- add black (K) to:
  - accurately print black (C+M+Y = muddy grey)
  - save ink compared to C+M+Y (lots of ink soaks paper)
  - print fine black text or lines without having to align colours carefully
  - save money: black ink is cheaper than coloured

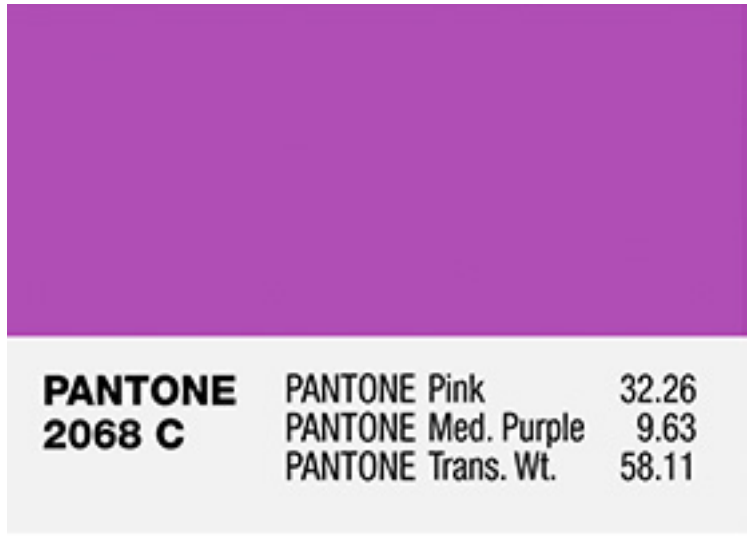
Beyond CMYK

- larger gamut: reproduce more colours
- accuracy: more precise colour reproduction
- spot colour: reproduce one colour perfectly

CMYK



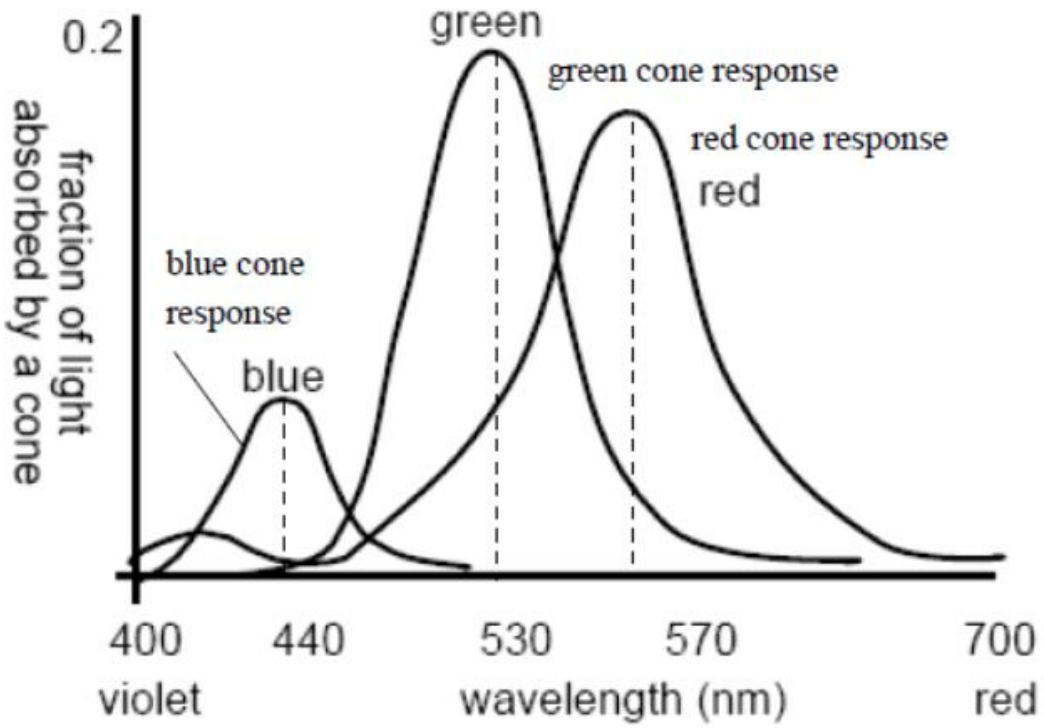
Extended gamut

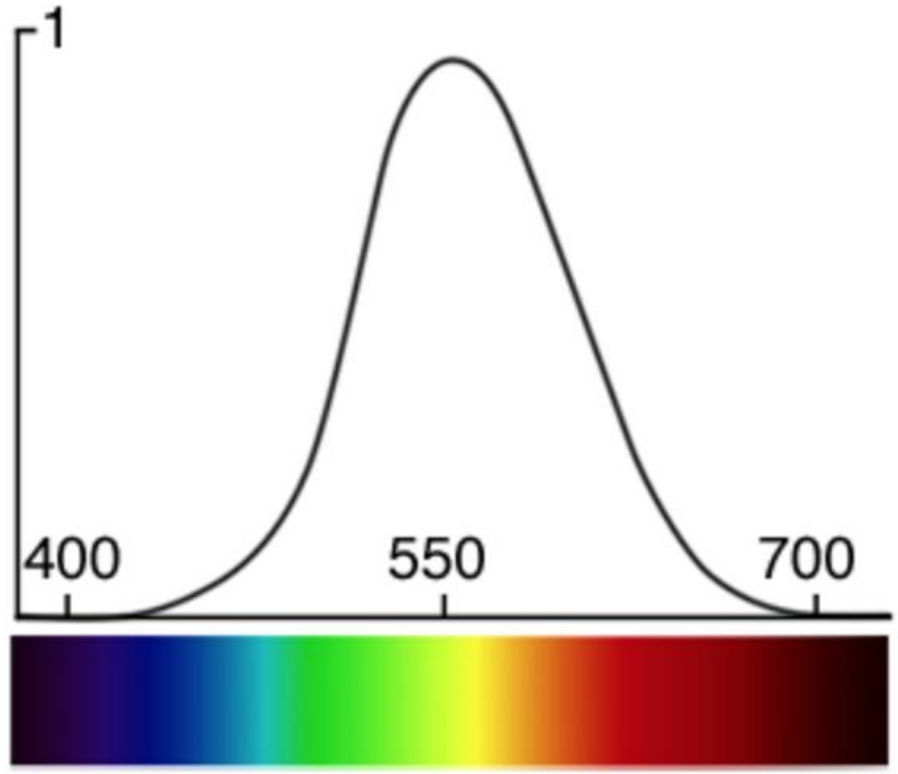


spot colour

https://www.pantone.com/what-is-extended-gamut

Greyscale





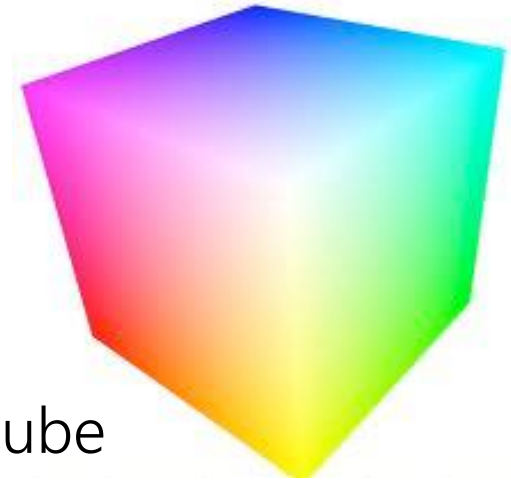
$$l(r, g, b) = 0.2126r + 0.7152g + 0.0722b$$

"Impression, Sunrise" by Claude Monet, 1872

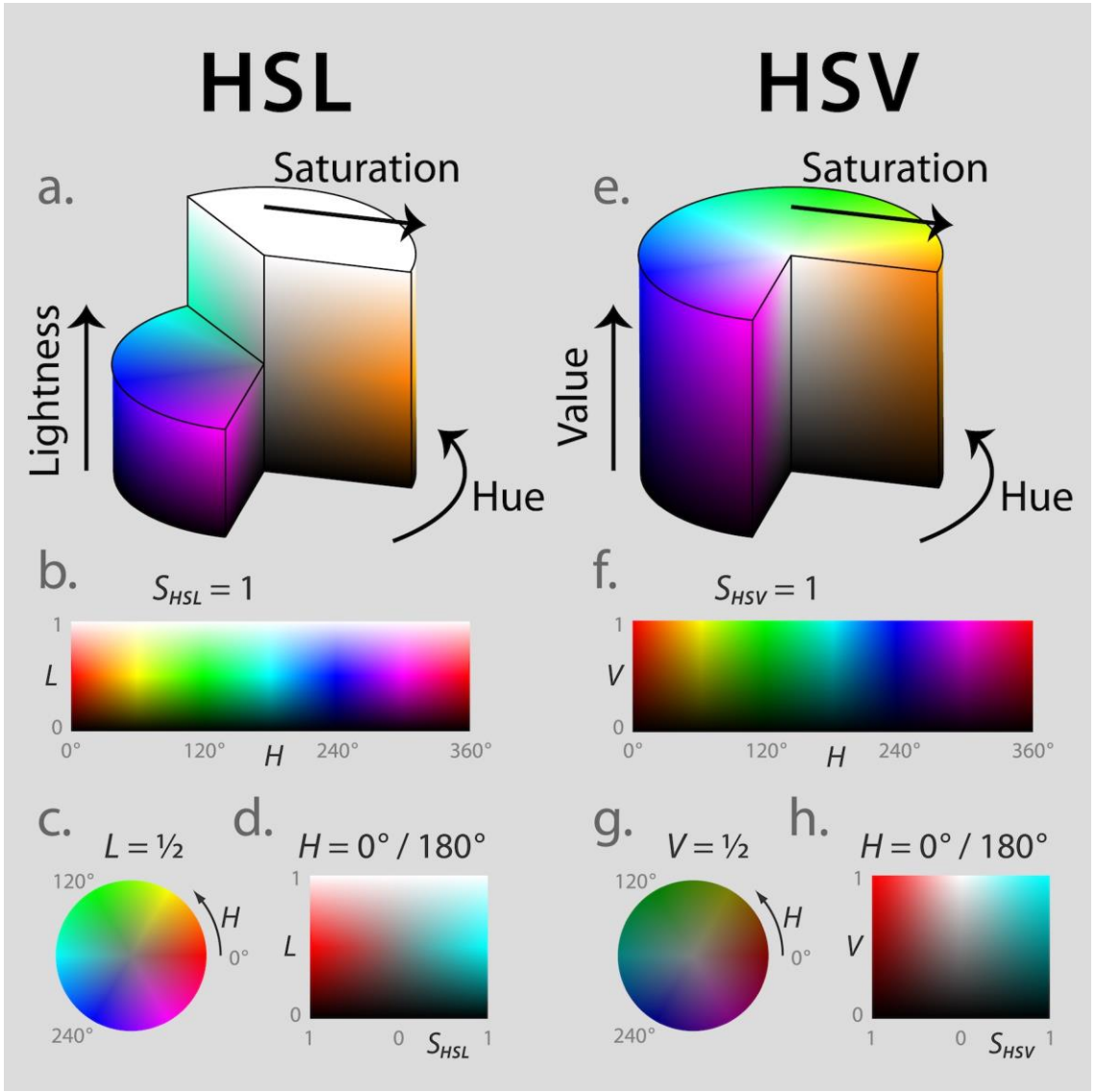
"Impression, Sunrise" by Claude Monet, 1872

HSV/HSL colour spaces

- Often used for colour selection in applications
  - Hue, Saturation, Value (HSV)
  - Hue, Saturation, Lightness (HSL)
  - See details in course notes



RGB Cube

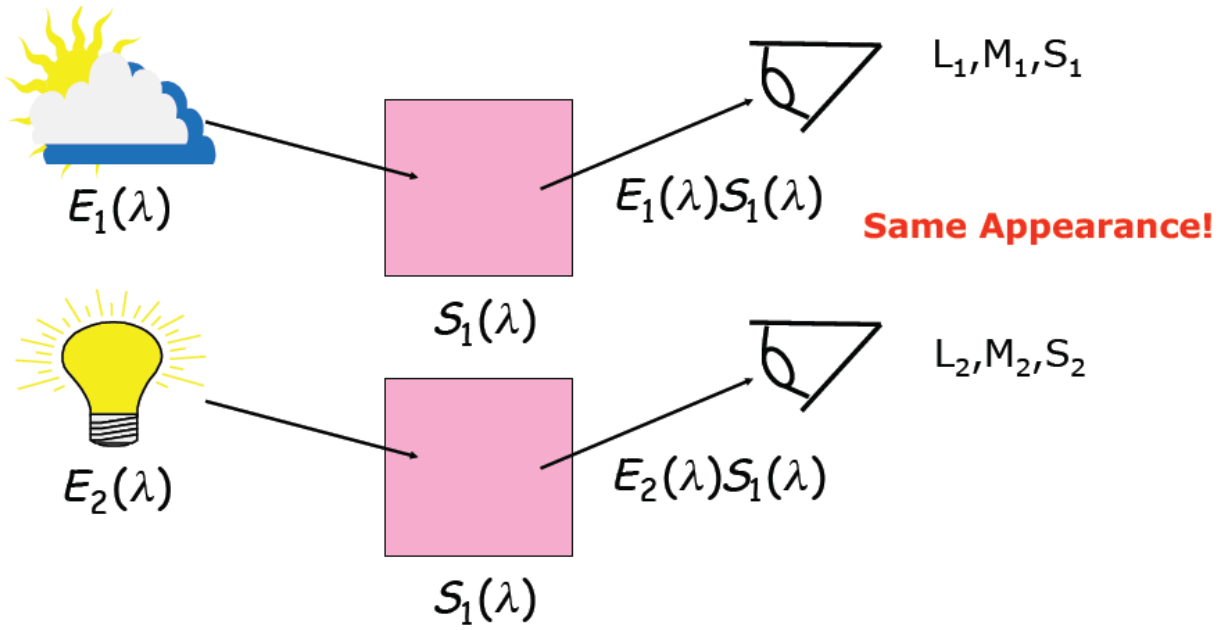


Colour spaces

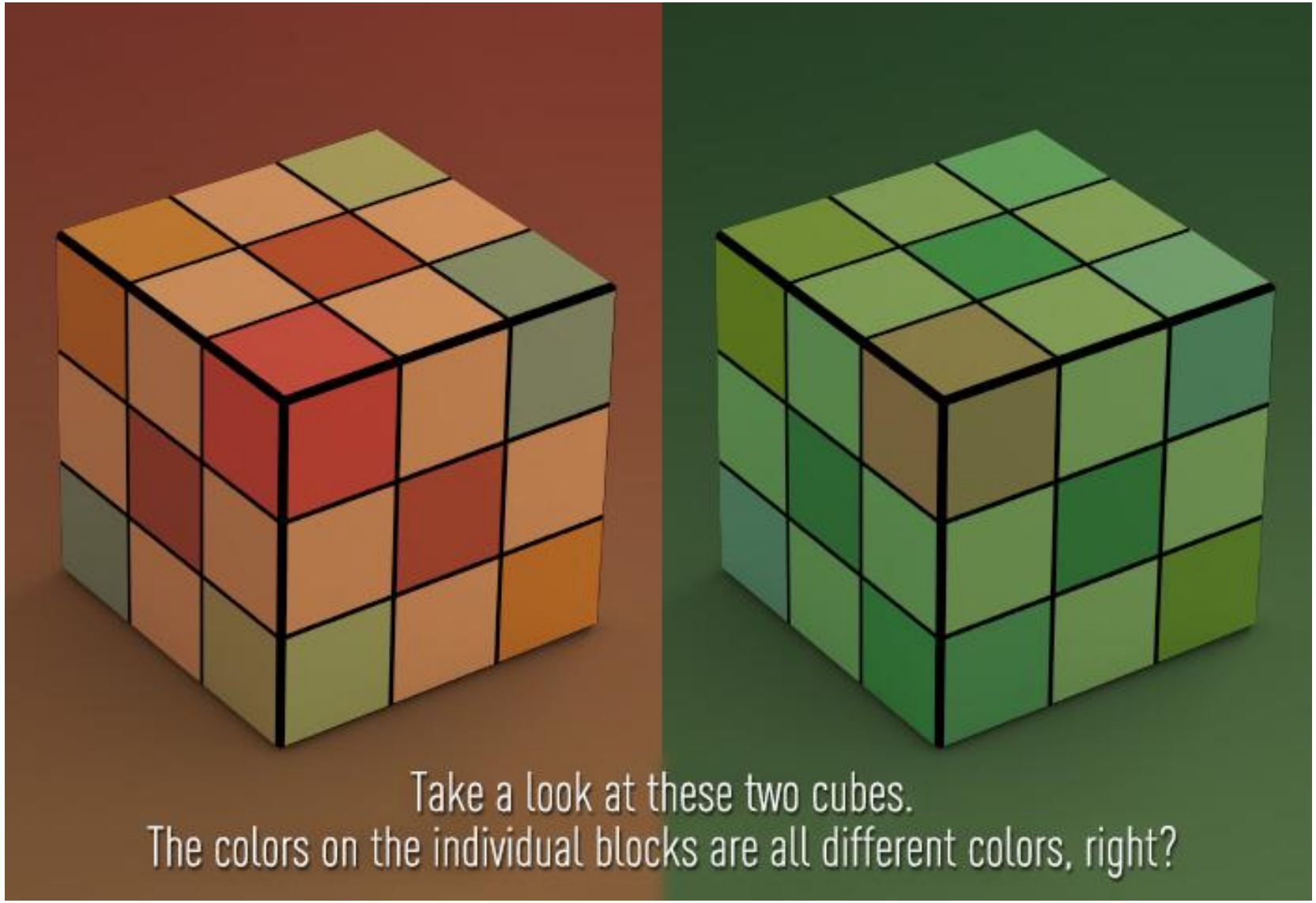
- RGB (Red, Green, Blue)
  - monitors
- CMY (Cyan, Magenta, Yellow)
  - printers
- HSV/HSL (Hue, Saturation, Value/Lightness)
  - user-friendly, common for colour pickers in graphics packages
- CIEXYZ and CIE xyY – see CM20219 notes
  - specifying colours and their ranges absolutely

Colour constancy

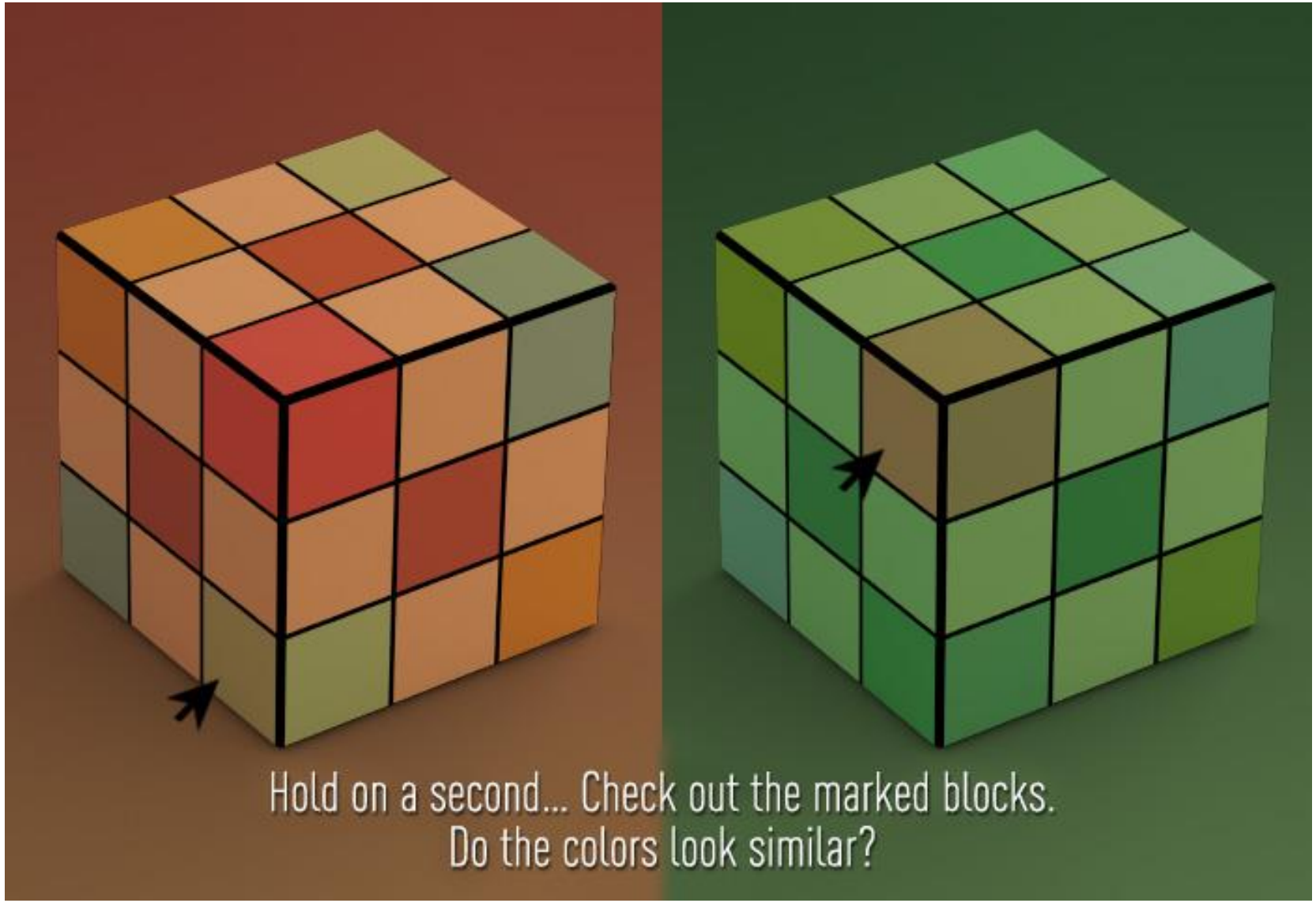
- Humans adapt to the global illumination conditions
  - observe objects under different illumination
  - objects appear to have approximately the same appearance



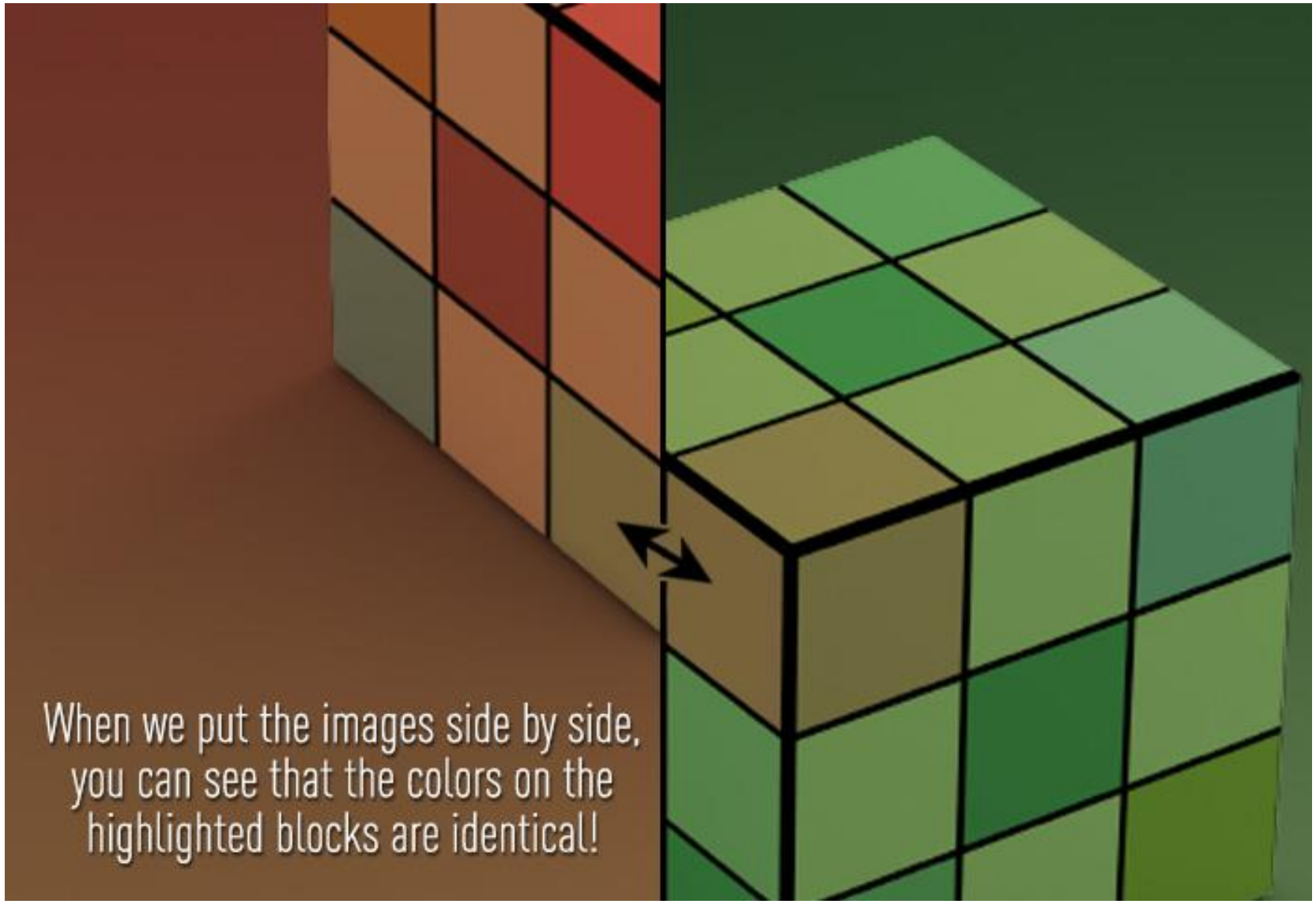
Colour constancy – example



Colour constancy – example



Colour constancy – example



Colour constancy – The dress

http://www.iflscience.com/brain/explaining-perceptions-dress/

Let’s have another break

Quantisation

- Pixel values are represented by a number (or more for colours)
- Values must be quantised for use in computers
  - This limits precision and the number of intensity levels
- How many bits do we need?
  - 8 is convenient (number of bits in a byte) and usually sufficient
  - some applications use 10, 12, 16 or 32 bits

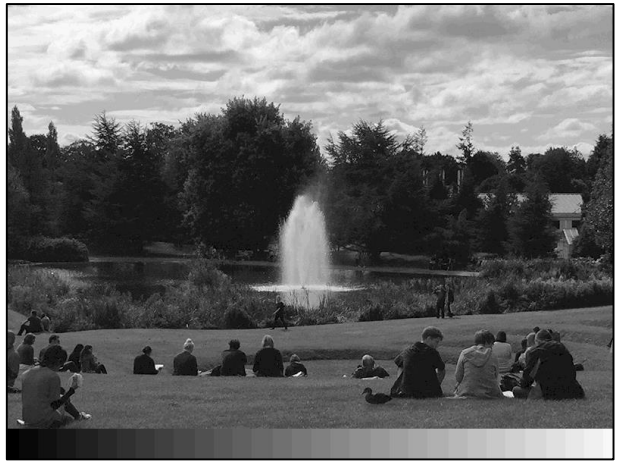
Quantisation

1 bit = 2 levels

2 bits = 4 levels

3 bits = 8 levels

4 bits = 16 levels



5 bits = 32 levels

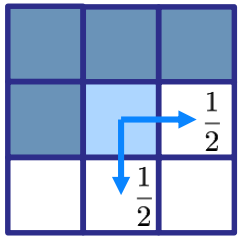
6 bits = 64 levels

7 bits = 128 levels

8 bits = 256 levels

Error diffusion

- Achieves more visually pleasing results by distributing quantisation errors
- Algorithm:
  - Iterate over pixels in scanline order
  - Quantise pixel value
  - Pass on quantisation error to pixels on the right and below



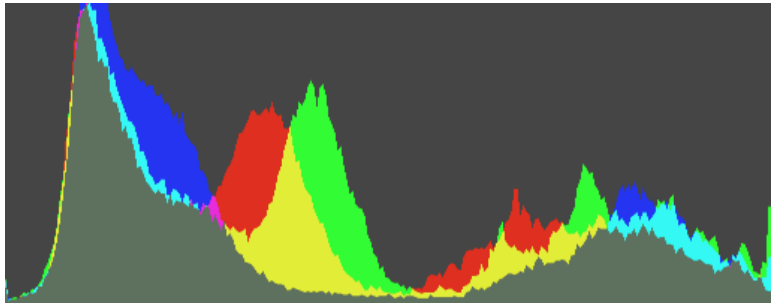
8-bit input value $I(x,y)$	1-bit output $O(x,y)$	quant. error $E(x,y)$
$[0, 127]$	0	$i(x,y)$
$[128, 255]$	1	$i(x,y) - 255$

Input image

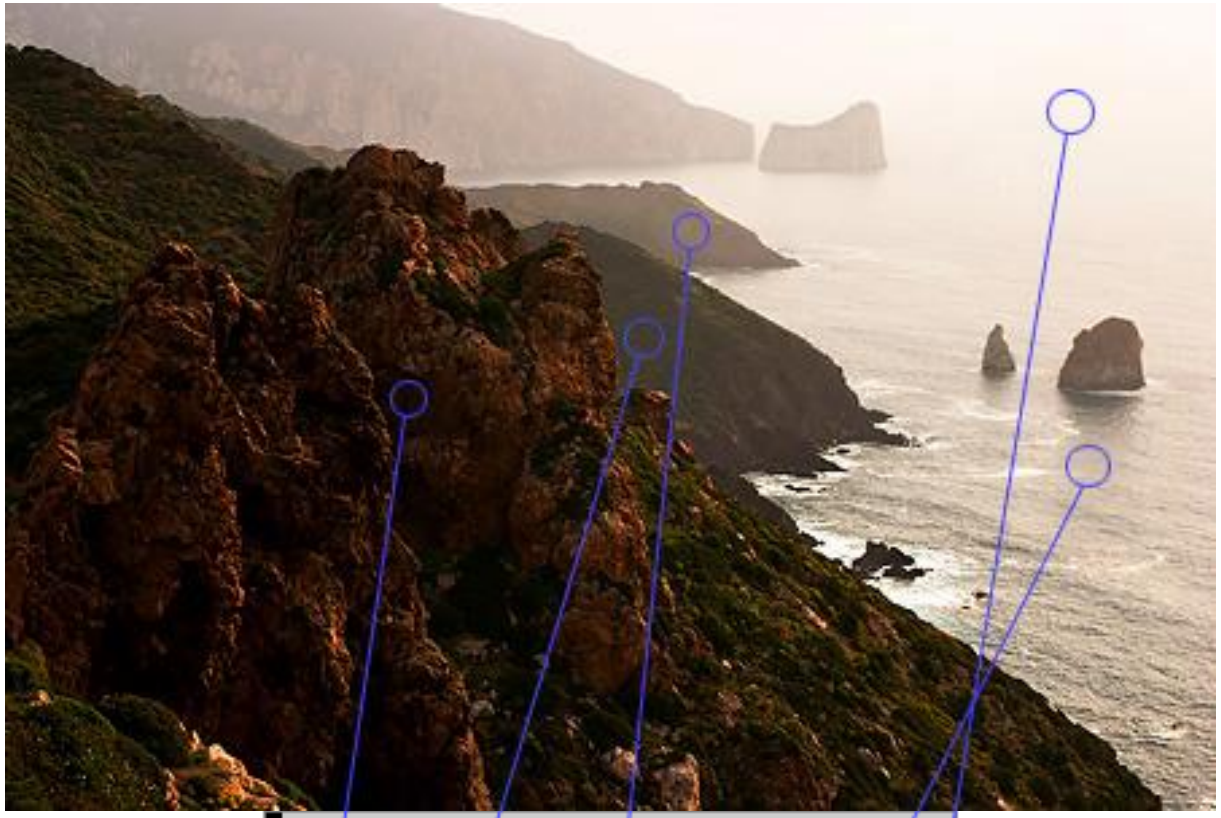
Error diffusion

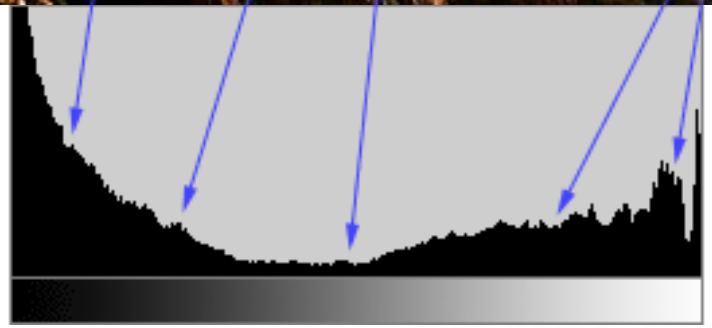
Histograms

- Summarise distribution of intensities in an image



RGB histogram (Photoshop)





intensity histogram

Point processing – inversion

$$O(x, y) = 255 - I(x, y)$$

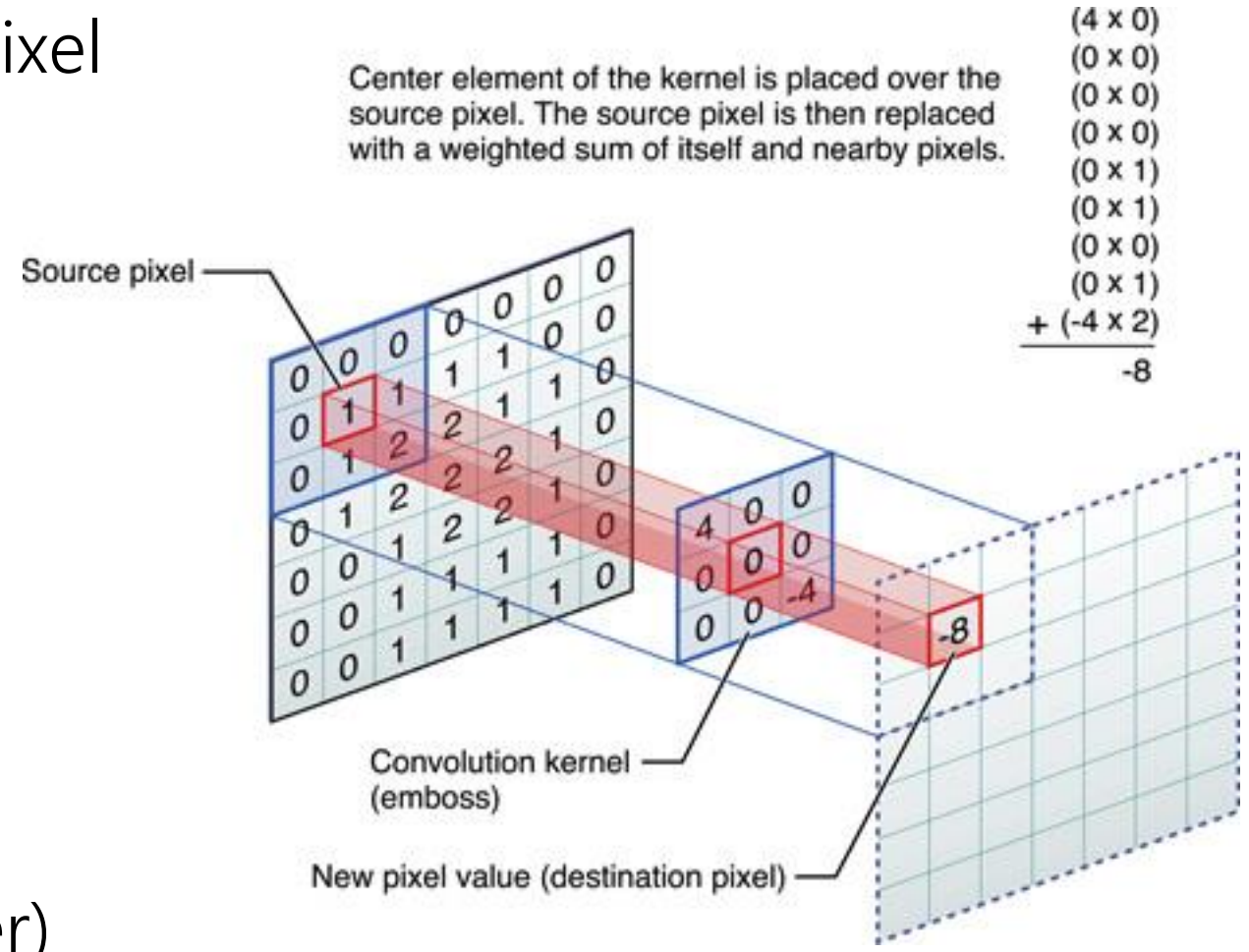
Point processing – contrast enhancement

$$O(x,y) = \left[\frac{I(x,y) - 51}{0.6} \right]_0^{255}$$

Linear filtering (convolution)

- Slide a filter over the image to compute a new value for each pixel
  - Linearly combine pixel values within a window/neighbourhood
  - Weight of each neighbouring pixel defined by a **kernel**
- **Convolution**: general filtering approach that supports arbitrary kernels
- (We'll look at speeding it up later)

$$(K * I)(x, y) = \sum_i \sum_j K(i, j) I(x - i, y - j)$$



Linear filtering (convolution): blurring

- Box filter: fast and simple

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

=

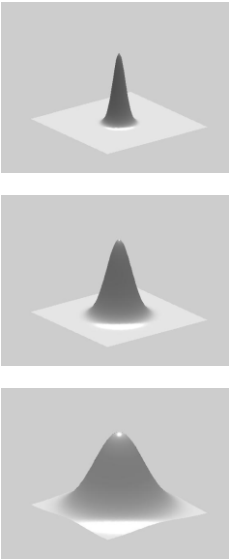
$$\frac{1}{9} \times$$

1	1	1
1	1	1
1	1	1

- Gaussian blur: smooth blur

$$\frac{1}{16} \times$$

1	2	1
2	4	2
1	2	1



- Low-pass filter: removes high frequencies (details)
  - Defined by standard deviation σ
  - Usually cut off kernel at 2 σ

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

Linear filtering (convolutions): gradient filters

Input image

0	0	0
0	1	0
0	0	0

Horizontal gradients

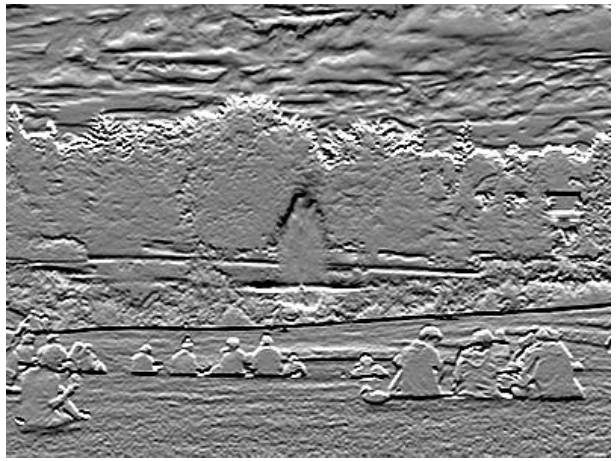
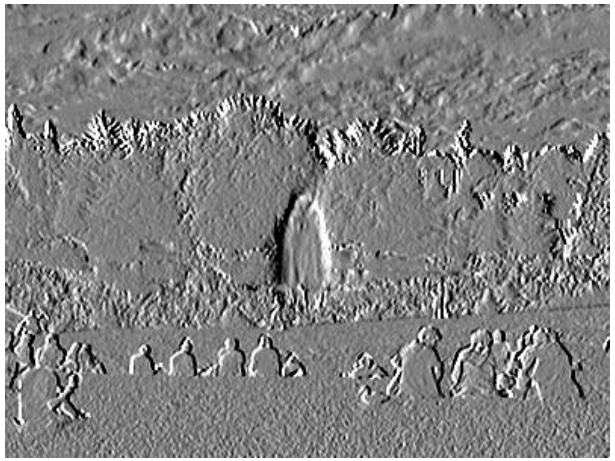
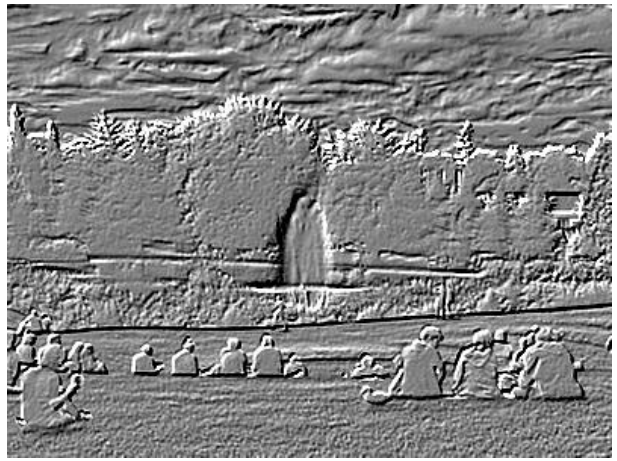
1	2	1
0	0	0
–1	–2	–1

Vertical gradients

1	0	–1
2	0	–2
1	0	–1

Diagonal gradients

2	1	0
1	0	–1
0	–1	–2

These filters are known as **Sobel** filters. The resulting images are shown here with an offset of 127, so as to show both negative and positive gradients.

Linear filtering (convolution): additional effects

- Motion blur

$$\frac{1}{5} \times \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

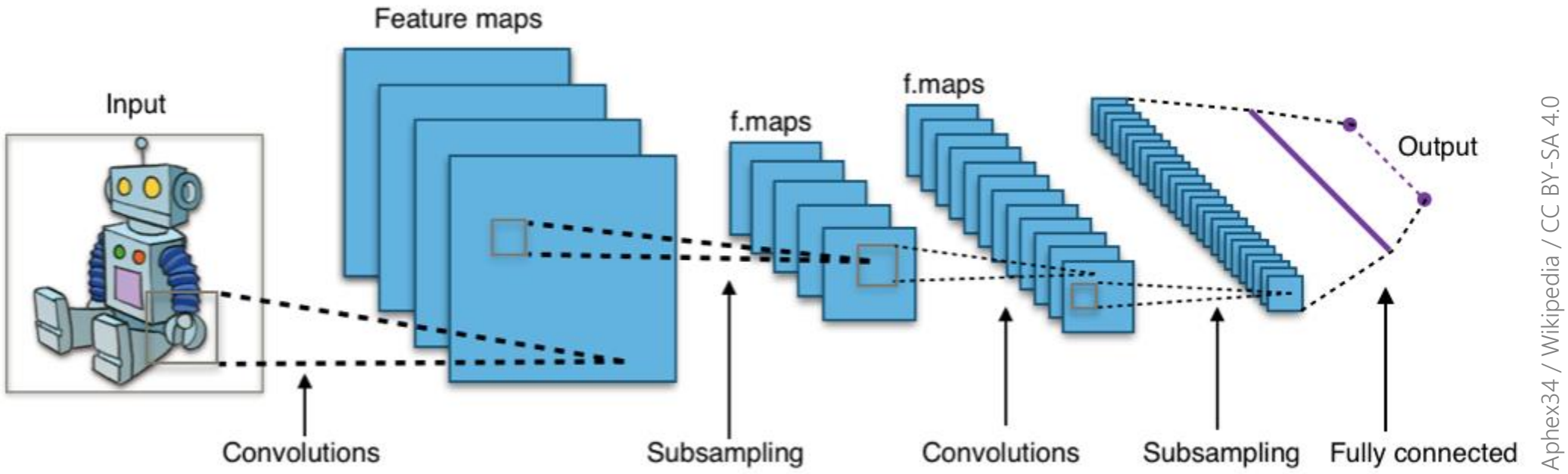
- Sharpening using unsharp masking

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$O = I + x \cdot (I - G_\sigma * I)$$

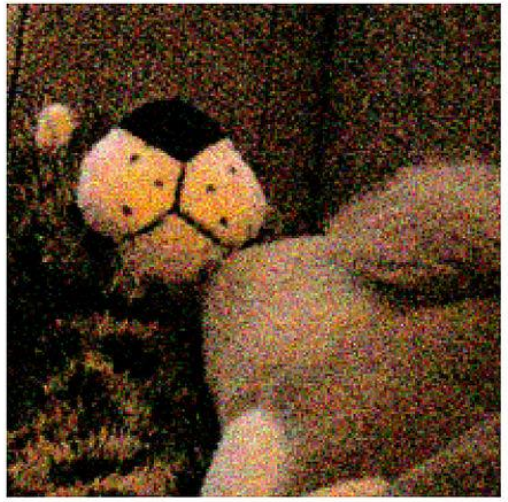
Application: convolutional neural networks (CNNs)

- Artificial neural network inspired by organization of the visual cortex
- Multiple layers of convolutions, subsampling and other operations
- Learning convolution kernels from large amounts of training data



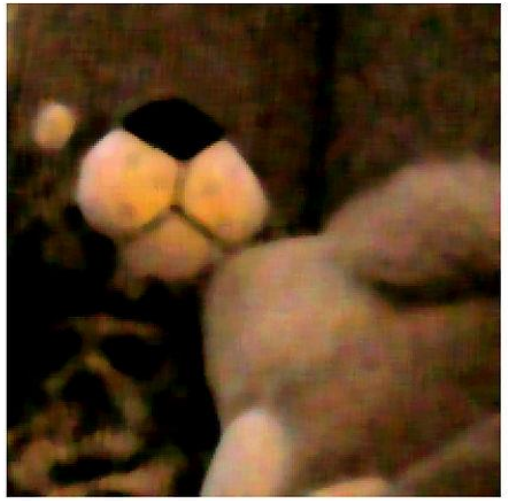
Median filtering

- Not a convolution filter
  - Good against ‘salt & pepper’ noise (sparse corrupt pixels)
- Replace each pixel with the median value of all pixels in its neighbourhood
  - Naïve implementation: sort all pixel values + pick middle value
  - “1px median” = window with radius 1 pixel = 3×3 window



original image

1px median filter



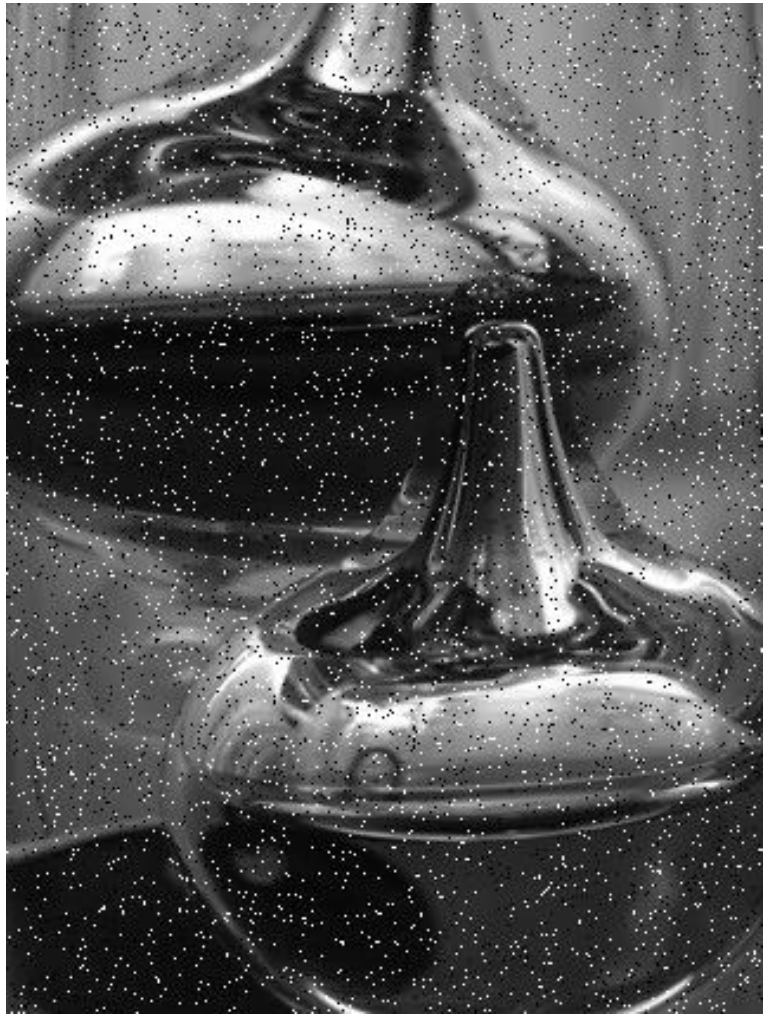
3px median filter

10px median filter

Median filtering

Input image

Corrupted image



Median-filtered image

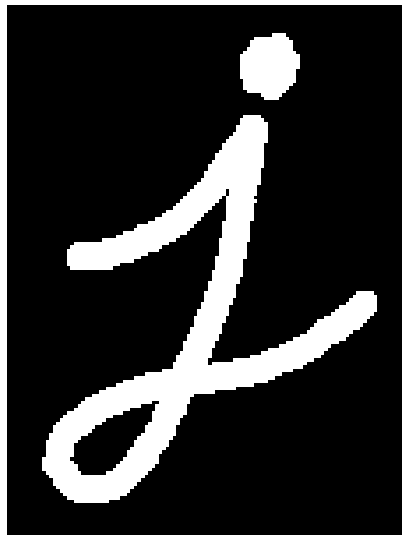
Marko Meza

Morphological filtering

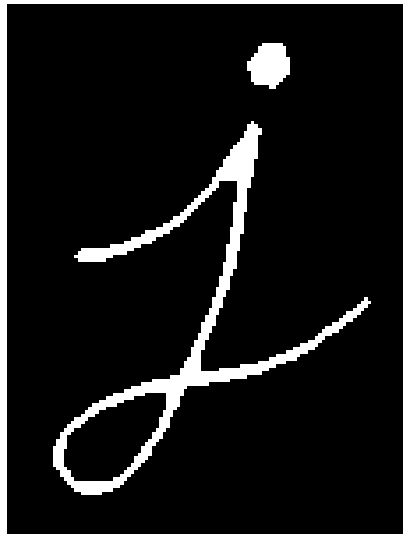
- Filtering of images using set theory
- Using a **structuring element** (set of pixel offsets),

  e.g. $B = \{(-2,-2), (-1,-1), (0,0), (1,1), (2,2)\}$

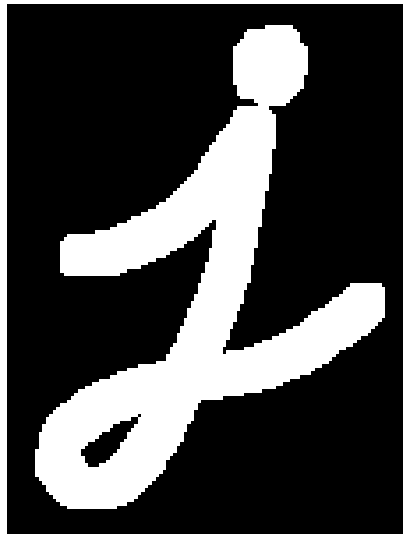
Input image



Erosion



Dilation



OpenCV documentation

Opening

Closing

0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	1	0	0	0
1	0	0	0	0

line

0	0	1	0	0
0	1	1	1	0
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0

disc/diamond

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

square

Morphological filtering

Erosion

- Reduces the width of bright objects

$$(I \ominus B)(x,y) = \min_{(i,j)\in B} I(x-i, y-j)$$

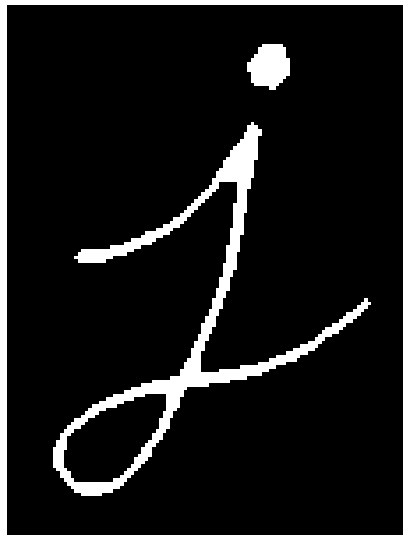
Dilation

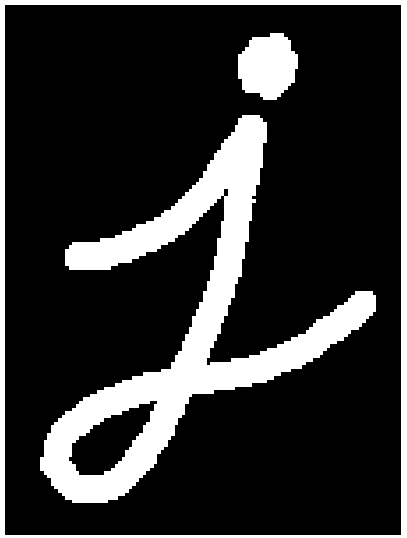
- Increases the width of bright objects

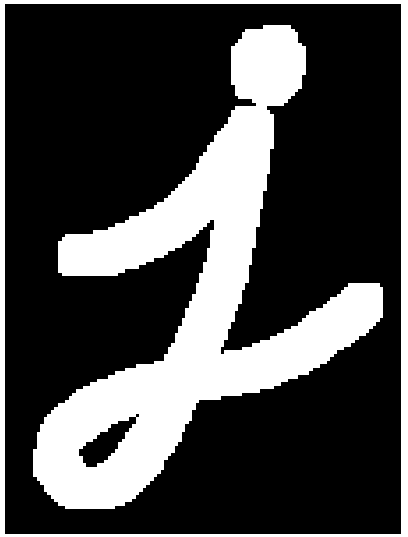
$$(I \oplus B)(x,y) = \max_{(i,j)\in B} I(x-i, y-j)$$

$B =$

1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1







OpenCV documentation

Input image

Morphological filtering

Opening: dilation after erosion

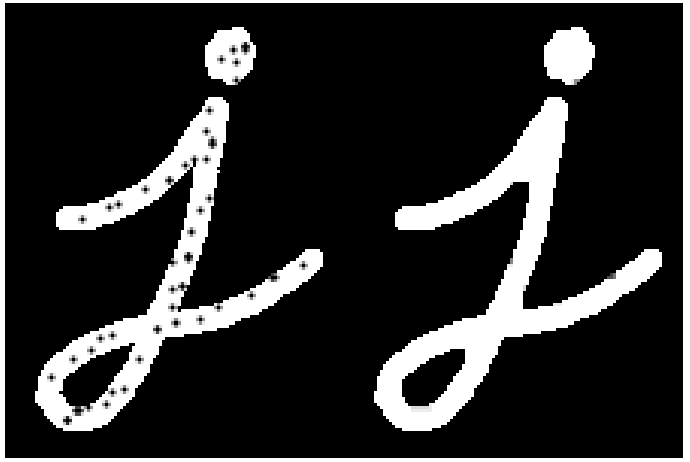
- Simplify bright objects while avoiding the shrinkage of boundaries caused by erosion

$$I \circ B = (I \ominus B) \oplus B$$

Closing: erosion after dilation

- Simplify bright objects while avoiding the expansion of boundaries caused by dilation

$$I \bullet B = (I \oplus B) \ominus B$$



OpenCV documentation

Any questions?