INTRODUCTION TO THE DIGITAL IMAGE PROCESSING

Lecture #2

Niels Volkmann
Professor

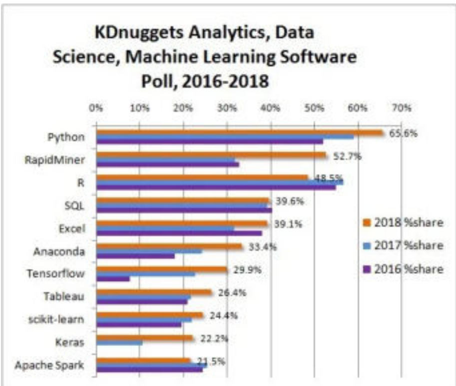
ECE Department
Department of Bioengineering
Quantitative Biosceince Program

WHY PYTHON?

Worldwide, May 2019 compared to a year ago:

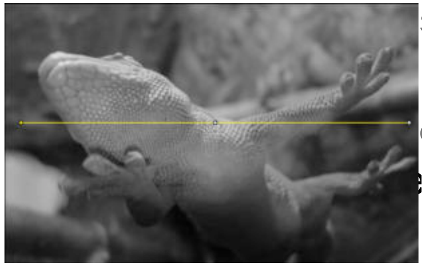
Rank	Change	Language	Share	Trend
1		Python	27.34 %	+4.5 %
2		Java	20.25 %	-2.1 %
3		Javascript	8.51 %	-0.0 %
4	↑	C#	7.38 %	-0.5 %
5	↓	PHP	7.34 %	-0.9 %
6		C/C++	6.01 %	-0.3 %
7		R	4.16 %	-0.1 %
8		Objective-C	2.91 %	-0.6 %
9		Swift	2.5 %	-0.3 %
10		Matlab	2.03 %	-0.3 %

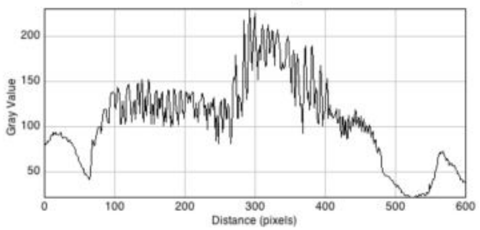
WHY PYTHON?



REPRESENTING IMAGES

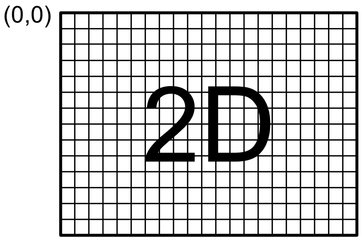
- Images are simply 2D functions parameterized by *x* and *y*
  - i.e., color as a function of *x* and *y*
  - We can call this the image "signal"
  - And write this as, for example: *C(x,y)*





REPRESENTING IMAGES

- But how do you actually represent these functions?
- Different choices…

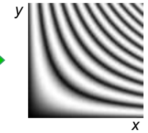


REPRESENTING IMAGES

- Many ways to represent image signals:

1. Explicitly, with mathematical equations

e.g., $I(x,y) = sin(10\pi\ x\ y)$



Unfortunately, this is difficult to generalize…
What's the mathematical equation for this?

Spoiler: Later in the quarter we discuss how to represent arbitrary images as sums of weighted sine and cosine functions (Fourier theory)!

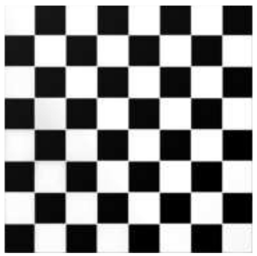
REPRESENTING IMAGES

- Many ways to represent image signals:

1. Explicitly, with mathematical equations
2. Implicitly, with a program that computes *I(x,y)* everywhere on the domain

```
import math

def checkboard(x, y):
   x = math.floor(x * 8)
   y = math.floor(y * 8)
   color = (x % 2) == (y % 2)
   return color
```



Again, difficult to generalize to arbitrary images…

REPRESENTING IMAGES

- Many ways to represent image signals:

1. Explicitly, with mathematical equations
2. Implicitly, with a program that computes *I(x,y)* everywhere on the domain
3. Implicitly, with a composition of geometric shapes with well-defined mathematical representations

e.g., vector graphics

REPRESENTING IMAGES

- Many ways to represent image signals:

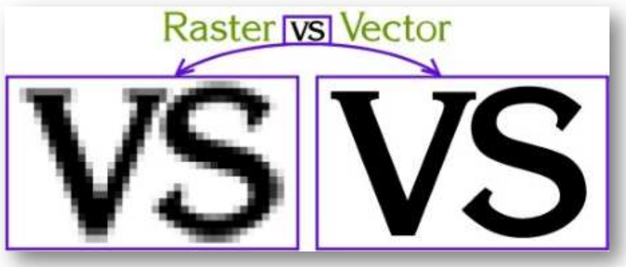
1. Explicitly, with mathematical equations
2. Implicitly, with a program that computes *I(x,y)* everywhere on the domain
3. Implicitly, with a composition of geometric shapes with well-defined mathematical representations
4. Approximately, using a discrete representation

REPRESENTING IMAGES

- Many ways to represent image signals:

1. Explicitly, with mathematical equations
2. Implicitly, with a program that computes *I(x,y)* everywhere on the domain
3. Implicitly, with a composition of geometric shapes with well-defined mathematical representations
4. Approximately, using a discrete representation

x	y	value
0	0	“red”
1	0	“blue”
2	0	“purple”



Pros:
- Very simple and fast to query
- Can represent complex images
- Easy to capture with cameras

Cons:
- Discrete representation, fixed resolution
- Cannot zoom-in arbitrarily
- Pixel artifacts from sampling
- Can be expensive to store

Pros:
- “Infinite” resolution, can zoom in arbitrarily
- In certain cases, more efficient to store than bitmaps

Cons:
- Hard to represent arbitrary images
- Requires computation to query
- Cannot be “captured” by a camera

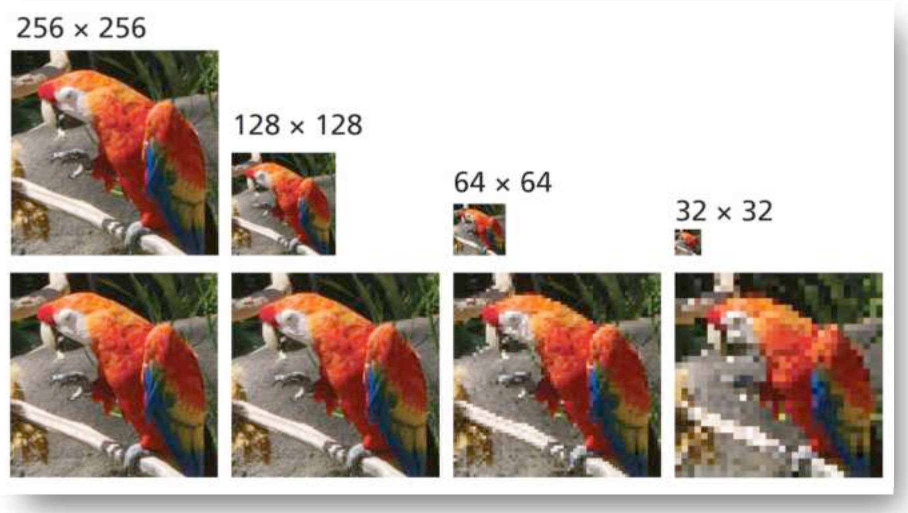
Name	Extension	vector/bitmap	compression	still/video
JPEG	.jpg, .jpeg	bitmap	lossy	still
TIFF	.tif, .tiff	bitmap	LZW nonlossy	still
JPEG2000	.jp2 .j2k	bitmap	lossy/nonlossy	still
PNG	.png	bitmap	no compression	still
GIF	.gif	bitmap	indexed color	still
BMP	.bmp	bitmap	no compression	still
AVI	.avi	bitmap	various	video
Apple Movie	.mov	bitmap	H.264,... JPEG2000,MPEG-4	video
Adobe Illustrator	.ai	vector	N/A	design
PostScript (PS)	.ps	vector	N/A	design
Encapsulated PS	.eps	vector	N/A	design
PDF	.pdf	vector	N/A	(e)print
SVG	.svg	vector	N/A	(e)print

BITMAPPED IMAGES

- Have a fixed resolution (usually specified by some *width* x *height*)
- This is the number of *picture elements* (aka "pixels") in the image
- In practice, we can think of this as an array of values:

Finite Arrays of pixels: $$F = \begin{bmatrix} F_{1,1} & \cdots & F_{1,C} \\ \vdots & \ddots & \vdots \\ F_{R,1} & \cdots & F_{R,C} \end{bmatrix} \in \mathbb{R}^R \times \mathbb{R}^C$$

IMAGE SIZE AND RESOLUTION



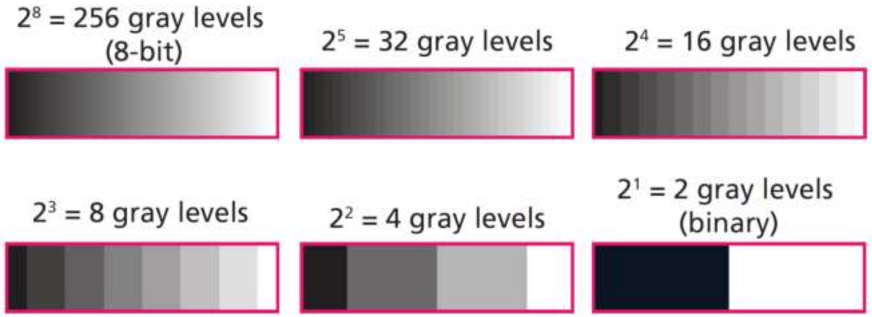
NAÏVE DOWNSAMPLING → ALIASING

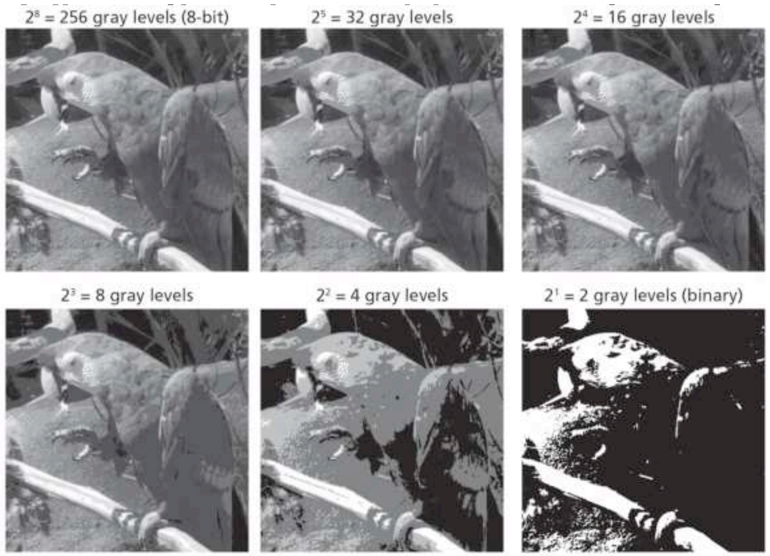
REPRESENTING GRAYSCALE VALUES

- We use a digital (binary) representation to represent grayscale values
- Lower values = darker, higher = brighter
- # of bits to encode value is the "bit depth"
- Common bit-depths include:
  - 8-bit: values range from 0 to 255
  - 10-bit: values range from 0 to 1,023
  - 12-bit: values range from 0 to 4,095
  - 16-bit: values range from 0 to 65,535
  - 32-bit: values range from 0 to 4,294,967,295

MOST COMMON!

VISUALIZATION WITH GRAY LEVELS





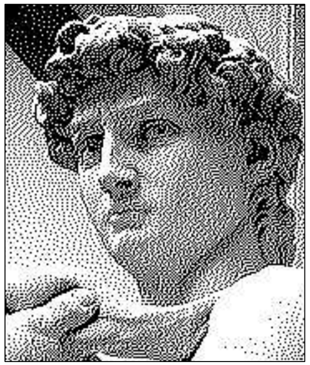
GETTING GOOD IMAGES WITH FEW LEVELS

- Even with small number of levels, we can get good images if we're smart about it
- The "smart way" is known as Dithering

Original

Simple thresholding (2 levels)

Dithering (2 levels)

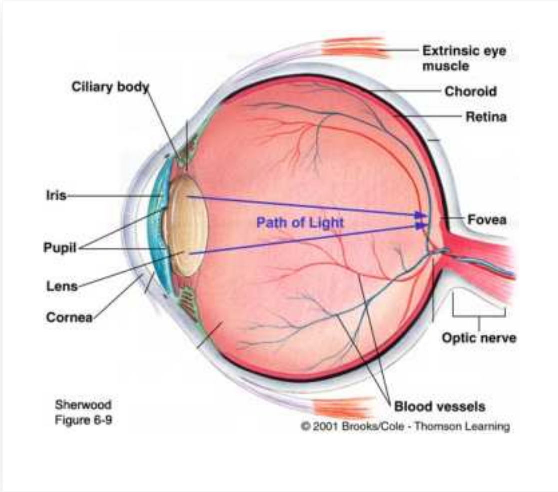


source: Wikipedia

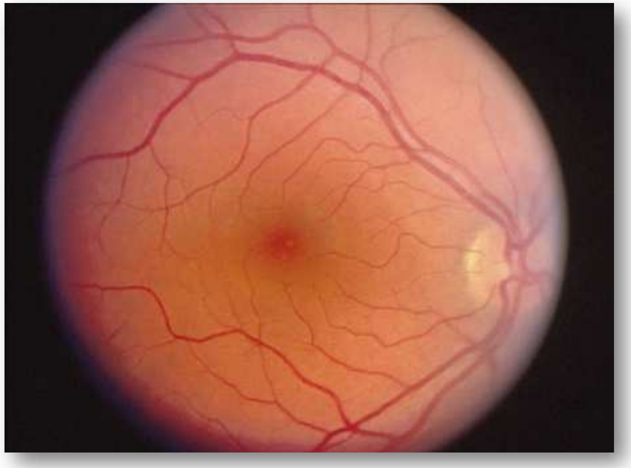
HOW DO WE REPRESENT COLOR?

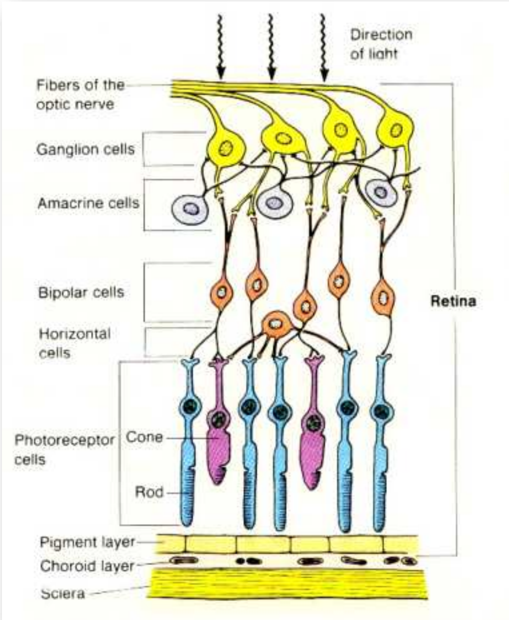
- To do this, we use three numbers:
  1. Red (R)
  2. Green (G)
  3. Blue (B)
- Why?
- The answer has to do with the human visual system…

THE EYE



RETINA





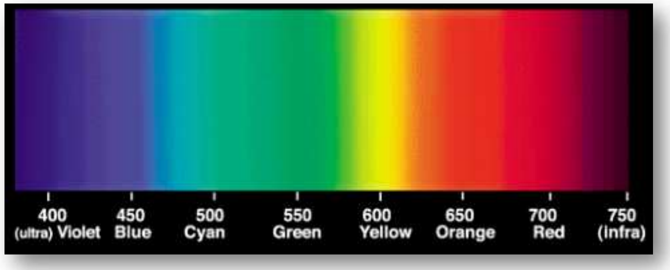
Source: http://hk.geocities.com/miko_nkwhk/cone_rod.gif

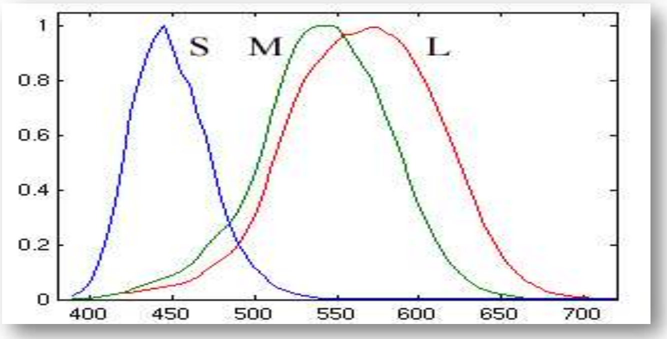
RODS

- Much higher density than cones (i.e., higher “resolution”)
- Better at seeing in low-light conditions
- Have higher frequency response (can detect fast-moving objects)
- Spread out throughout the retina
- Only “detect” grayscale (intensity values)

CONES

- Less sensitive to light
- Concentrated in the fovea
- Fewer in number (approx. 5 million)
- Responsible for color vision
- 3 different classes of cone cells:
  - S: 420 – 440nm
  - M: 534 – 545nm
  - L: 564 – 580nm





RGB-IMAGES

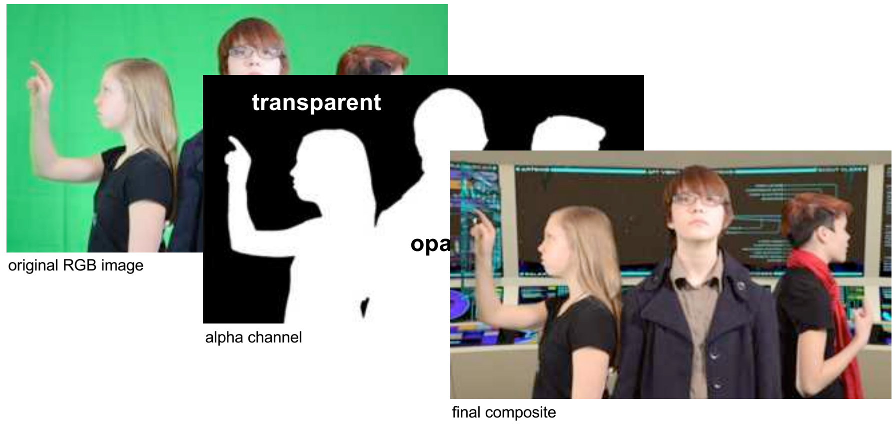
- Since our visual system has 3-primary components for color, we use 3 components to represent color in our images (RGB)
- So in an 8-bit image, we use 3 bytes per pixel (one byte for each of red, green, blue)
- An 8-bit, megapixel image (1M pixels) would take 3MB to encode (without compression)
- At this res, a 2-hour movie at 24 fps would be: 3MB x 120 mins x 60 secs/min x 24 fps = 518,400 MB (half a terabyte!)

ALPHA CHANNEL

- Some images have an additional channel (RGBA)
- This ***alpha channel*** is used to encode transparency (opacity)
- Alpha = 0 is fully transparent
- Alpha = 1 (or max value) is fully opaque
- Per-pixel value: every pixel can have different opacity

ALPHA CHANNEL: COMPOSITING

- Compositing: combining images with alpha channels together to form new images



COMPOSITING ALGEBRA [PORTER & DUFF 1984]

- Given images *A* and *B* to composite together
- Assume alpha channel at a pixel is α_A and α_B
- Think of alpha channel as the probability of hitting an object at the pixel
- e.g., if $\alpha_A = 0.3$ then the probability of being in object *A* is $P_A = 0.3$, probability of not being in object *A* is $1- \alpha_A = 0.7$
- Key: assume statistical independence, so no correlation between images when compositing!

COMPOSITING ALGEBRA

- So, if we have two alpha channel images:
  - Prob Background (neither image) = $(1- \alpha_A)(1- \alpha_B)$
  - Prob A only = $(\alpha_A)(1- \alpha_B)$
  - Prob B only = $(1 - \alpha_A)(\alpha_B)$
  - Prob Both = $(\alpha_A)(\alpha_B)$

description	*area*
$\bar{A} \cap \bar{B}$	$(1-\alpha_A)(1-\alpha_B)$
$A \cap \bar{B}$	$\alpha_A(1-\alpha_B)$
$\bar{A} \cap B$	$(1-\alpha_A)\alpha_B$
$A \cap B$	$\alpha_A \alpha_B$

source: Porter and Duff [1984]