Image Processing 101
Fredo Durand
MIT EECS
6.815/6.865

http://xkcd.com/1014/
Basics of Image processing
Digital Images

• Can be encoded as 2D arrays of RGB triplets

• Formally:
  - The domain is the 2D plane
  - The range is the RGB space

• Other color spaces are possible and will be used

• Values are often encoded as 8 or 16 byte integers
  - But we will use floats in \([0...1]\) to make life easier
Numpy & scipy

• Python’s numerical tools, and in particular efficient array class
• http://numpy.scipy.org/

• Docs:
• http://docs.scipy.org/doc/numpy/user/whatisnumpy.html
• and
• http://docs.scipy.org/doc/numpy/user/basics.html
Images as numpy arrays

- 3D array: \( \text{im}[y, x, c] \) means pixel at 2D coordinates \( x, y \) and color channel \( c \)
  - \( \text{im}[y, x, 0] \) is red, \( \text{im}[y, x, 1] \) is green, etc.
  - \( \text{im}[y, x] \) is a 1D array of size 3

```python
>>> im
array([[0.07592612, 0.1636467, 0.38377465],
       [0.07592612, 0.1636467, 0.38377465],
       [0.07806559, 0.1694451, 0.38377465],
       ....
       [0.06281671, 0.14189829, 0.36366993],
       [0.06281671, 0.14189829, 0.36366993],
       [0.06293649, 0.13902245, 0.35965931]],
      [[0.07592612, 0.1636467, 0.38377465],
       [0.07806559, 0.1694451, 0.38377465],
       [0.07806559, 0.1694451, 0.38377465],
       ....
       [0.06281671, 0.14189829, 0.36366993],
       [0.06281671, 0.14189829, 0.36366993],
       [0.06293649, 0.13902245, 0.35965931]],
      [[0.07806559, 0.1694451, 0.38377465],
       [0.07806559, 0.1694451, 0.38377465],
       [0.08219267, 0.1636467, 0.38918265],
       ....
       [0.06772459, 0.13902245, 0.36859092],
       [0.06972786, 0.14198829, 0.37631892],
       [0.07176145, 0.1487213, 0.3767625]],
      ....
      ....
      ....
      ....
      [[0.49950528, 0.35856391, 0.16044351],
       [0.50033247, 0.35856391, 0.16667227],
       [0.4936162, 0.34888483, 0.17343904],
       ....
       [0.52344316, 0.3636693, 0.18698651],
       [0.52952322, 0.36859092, 0.19846288],
       [0.53654316, 0.37631502, 0.19397217]],
      [[0.4877652, 0.34982566, 0.15728073],
       [0.46784113, 0.32359036, 0.15195805],
       [0.42590529, 0.29321615, 0.13698855],
       ....
       [0.53654316, 0.37631502, 0.19397217],
       [0.52344316, 0.3636693, 0.18698651],
       [0.51139782, 0.353749, 0.18814429]],
      [[0.4958528, 0.353749, 0.1636467],
       [0.4977652, 0.3482556, 0.1636467],
       [0.47617727, 0.3345761, 0.1636467],
       ....
       [0.6054843, 0.43134021, 0.23489526],
       [0.6054843, 0.43134021, 0.23489526],
       [0.6054843, 0.43134021, 0.23489526]]
```
Basic session

- `from imageIO import *`
- `im = imread('in.png')`
- `im2 = im*1.3`
- `imwrite(im2, 'out.png')`
Numpy array operations

- Operator overloading, e.g.
  - `out=im1 + im2 * 3 - im3**2.2`
- Very fast
- avoid for loops as much as possible
Array tools

• arrays are mutable, assignment doesn’t copy them.
  - use im.copy()

• New array:
  - numpy.empty([height, width, 3])
  - numpy.zeros([height, width, 3])

• Array size: numpy.shape(im) or im.shape

>>> height=numpy.shape(im)[0]
>>> height
85
>>> width = numpy.shape(im)[1]
>>> width
128
Indexing

- \( \text{im}[\text{height}/2, \text{width}/2, 0] = 0 \)
  - sets the red channel to zero in the center

- \( \text{im}[\text{height}/2, \text{width}/2] = [0, 0, 0] \)
  - black dot in the center
  - same as \( \text{im}[\text{height}/2, \text{width}/2] = 0 \)

- The ‘:’ operator slices an entire dimension
  - \( \text{im}[\text{height}/2, \text{width}/2, :) = [0, 0, 0] \)
    - same as above
  - \( \text{im}[:, \text{width}/2] = [0, 0, 0] \)
  - \( \text{im}[\text{height}/2, :) = [0, 0, 0] \)
    - black cross in the middle, full height, full width
Bad news

• No good support for pypy yet
Domain vs. range

• 2D plane: domain of images
• color value: range ($\mathbb{R}^3$ for us)
  - stores the red, green and blue components in $\text{im}[y, x, 0]$, $\text{im}[y, x, 1]$, $\text{im}[y, x, 2]$, respectively
Basic types of operations

Point operations: range only

Domain operations

Neighborhood operations: domain and range
Basic types of operations

Point operations: range only
- Pset 1

Domain operations
- Pset 2

Neighborhood operations: domain and range
- Pset 3
Light matter, eyes

• Light from sources is reflected by objects and reaches the eye.
• The amount of light from the source gets multiplied by the object reflectance
  - on a per-wavelength basis
Human perception

• Our eyes have an uncanny ability to discount the illumination
  - Only objects really matter for survival
  - Light is only useful to understand if you’re a photographer or to choose your sun lotion
Illusion by Adelson

• A & B have exactly the same tone
Illusion by Adelson

• A & B have exactly the same tone
Mechanism to discount light

• Light adaptation
  - We recenter our neural response around the current average brightness
  - neural + chemical + pupil

• Chromatic adaptation
  - eliminate color cast due to light sources
    - e.g. Daylight is white but tungsten is yellowish
  - Related to white balance - more soon
  - and Spanish Castle illusion
Contrast is about ratios

• Contrast between 1 & 2 is the same as between 100 & 200
• Useful to discount the multiplicative effect of light

0.1 to 0.2

0.4 to 0.8
Exposure

- On cameras, exposure (shutter speed, aperture\(^2\), ISO) has a multiplicative effect on the values recorded by the sensor.
- Changes the “brightness”, not contrast

http://photographystepbystep.com/exposure-2/auto-bracketing/
White balance
White balance & Chromatic adaptation

- Different illuminants have different color temperature
- Our eyes adapt to this: Chromatic adaptation
  - We actually adapt better in brighter scenes
  - This is why candlelit scenes still look yellow
White balance problem

• When watching a picture on screen or print, we adapt to the illuminant of the room, not that of the scene in the picture
• The eye cares more about objects’ intrinsic color, not the color of the light leaving the objects
• We need to discount the color of the light source

Same object, different illuminants
White balance & Film

- Different types of film for fluorescent, tungsten, daylight
- Need to change film!

- Electronic & Digital imaging are more flexible
Von Kries adaptation

- Multiply each channel by a gain factor
  - \( R' = R \times k_r \)
  - \( G' = G \times k_g \)
  - \( B' = B \times k_b \)

http://www.cambridgeincolour.com/tutorials/white-balance.htm
Von Kries adaptation

- Multiply each channel by a gain factor
- Note that the light source could have a more complex effect
  - Arbitrary 3x3 matrix
  - More complex spectrum transformation

http://www.cambridgeincolour.com/tutorials/white-balance.htm
White balance challenge

• How do we find the scaling factors for r, g, and b?
Best way to do white balance

• Grey card:
• Take a picture of a neutral object (white or gray)
• Deduce the weight of each channel
• If the object is recoded as $r_w$, $g_w$, $b_w$
use weights $k/r_w$, $k/g_w$, $k/b_w$
where $k$ controls the exposure
Lightroom demo

• Most photo editing software lets you click on a neutral object to achieve white balance
  • In “Levels” in Photoshop
  • In “basic” in Lightroom
  • In Adjustments in Aperture
• You also often have presets such as daylight, tungsten
Party name tags

- Provide excellent white references!
Without grey cards

• We need to “guess” which pixels correspond to white objects
Grey world assumption

• Assume average color in the image is grey
• Use weights proportional to

\[
\frac{1}{\int_{image} r}, \frac{1}{\int_{image} g}, \frac{1}{\int_{image} b}
\]

• Usually assumes 18% grey to set exposure
Brightest pixel assumption

• Highlights usually have the color of the light source
  – At least for dielectric materials

• White balance by using the brightest pixels
  – Plus potentially a bunch of heuristics
  – In particular use a pixel that is not saturated/clipped
Light with color control


- Gel on flash
Refs

• Recent work on color constancy
  – http://gvi.seas.harvard.edu/paper/color-subspaces-photometric-invariants

• Still an open problem!
Questions?

from xkcd

INSTEAD OF DRIVING ALL THIS WAY, WE COULDN'T HAVE JUST TAKEN OUR SUMMER PICTURES AND MESS WITH THE "HUE" SLIDER IN PHOTOSHOP.

HUSH.

Click
Take home messages

• Discounting the illumination is useful
• Ratios matter
• Optical illusions are not optical but fun
Gamma
Linearity and gamma

- Images are usually gamma encoded
- Instead of storing the light intensity $x$, they store $x^\gamma$
- to get more precision in dark areas for 8-bit encoding
Linearity and gamma

- Images are usually gamma encoded
- Instead of storing the light intensity $x$, they store $x^\gamma$
- to get more precision in dark areas

6 bit encoding for emphasis:

Linear
Gamma2.2
Linearity and gamma

• Images are usually gamma encoded
• Instead of storing the light intensity $x$, they store $x^\gamma$

• imageIO decodes gamma:
  our images are linear
Histograms
Histogram

• For each value (e.g. 0-255) how many pixels have this value?

• Cumulative histogram: for each value x, how many pixels have a value smaller than x

http://en.wikipedia.org/wiki/Histogram_equalization
Very useful on camera

- Allows you to tell if you use the dynamic range entirely
Bad: bright values under-used (underexposure)

Bad: bright values saturate (overexposure)

Histograms
equalization
Histogram equalization

- Given image with given histogram
- monotonic remapping to get a flat histogram

http://en.wikipedia.org/wiki/Histogram_equalization
Histogram equalization

• Ideal flat histogram:
  There are y% pixels with value less than y%
  - assuming everything is normalized to [0 1]
• Flip it: a pixel with y% value smaller than itself should have value y%
• For an old value x%, we know the number of pixels that have value < x%: cumulative histogram
• Therefore, we want x to be mapped to its cumulative histogram value.
Histogram matching

- Input image with given histogram
- Target histogram
- Monotonic remapping: cumulative histogram of input followed by inverse cumulative histogram of target
Pset 1
Pset 1

• Just a warm up. Due next Monday 2/13 at 9pm
• Brightness, contrast, color space manipulations
• Spanish castle illusion
• Histograms
Pset 1

- brightness(im, factor)
- contrast(im, factor, midpoint=0.3)
- BW(im, weights=[0.3, 0.6, 0.1])
- lumiChromi(im), fromLumiChromi(lumi, chromi)
- brightnessContrastLumi(im, brightF, contrastF, midpoint=0.3)
- RGB2YUV(im), YUV2RGB(im)
- saturate(im, k)
- spanish(im)
- histo(im, N), printHisto(im, N, scale)
Debugging
Debugging

- Doubt everything
- Debug pieces in isolation
  - Binary search/divide and conquer
- Display/print everything
  - In particular intermediate results
- Create simple inputs
  - where you can easily manually compute the result
  - e.g. constant image, edge image, etc.
  - use small images (e.g. 3x3)
  - including (especially) inputs to intermediate stages
  - use input that isolates different failure modes
- Change one thing at a time
  - e.g. to verify that a given command has the effect you want, modify it to break it