From: http://www.devx.com/projectcool/
Article/19987/0/page/2
The small green squares are the same color. Hold your hand to the screen and block out the adjacent blocks, showing only the two green square, and you'll see how dramatic the chromatic induction effect is.

From: http://www.devx.com/projectcool/
Article/19987/0/page/2
These two brown shades are different colors—yet the square blocks appear to be the same shade of brown. It is only when your eye follows the line down that you see how different the shades are. Cover the bottom part of the illustration with your hand and look only at the brown boxes to see the way the chromatic induction effect impacts the way you see color.
Administrivia

• Pset 1 is due tomorrow
• Photo session Thursday at 2:30
  – 32–D424
  – alternative: Thursday 4pm or Friday 3:30pm
  – let me know if you can’t make it
Domain vs. range

• 2D plane: domain of images
• color value: range (R³ 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

Each pixel is treated the same, doesn’t depend on other pixels, only on the input color
Basic types of operations

Point operations: range only

Domain operations

Only moves pixels around, doesn’t change colors, just where they are
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 & perception
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/
Gamma
Problem with linear encoding

- e.g., 8-bit values: 0..255
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.

![Gamma Compression Curve](gamma_curve.png)
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:
Gamma demo

- thank you Lili & Charlotte

See effects of gamma encoding for default : picture of skippy:

Pick a value for γ: 2.2

Pick how many bits: 2

Function plotted: $x^{(1/γ)}$

Original brightness

RGB value
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

• DO NOT USE ANY OTHER IMAGE READER FUNCTION
  – or do your own gamma decoding
How to capture linear images

• [http://www.mit.edu/~kimo/blog/linear.html](http://www.mit.edu/~kimo/blog/linear.html)
POINT
OPERATION
Point operations

- for each pixel: \( R_{\text{out}}, G_{\text{out}}, B_{\text{out}} = f(R_{\text{in}}, G_{\text{in}}, B_{\text{in}}) \)
  - and \( f \) is the same for all pixels

- Often discussed for greyscale version:
  - \( L_{\text{out}} = f(L_{\text{in}}) \)
Curves

- In many photo editors
Pset 1

• Exposure change (brightness)
• Contrast: change ratios
• Luminance chrominance
  – ability to change luminance/brightness independent of color
  – save pure-color information
  – need to respect multiplicative factors
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)

http://www.luminous-landscape.com/tutorials/understanding-series/understanding-histograms.shtml
Histograms

equalization
Histogram equalization

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

http://en.wikipedia.org/wiki/Histogram_equalization
Angle 1:

- where histogram is high, we need to spread out values in the output
- where histogram is low, we want to compress them
- => histogram gives the derivative of the remapping
- => remapping is the cumulative histogram
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
Application: BW digital

- with Soonmin Bae and Sylvain Paris [Siggraph 06]
- Users often disappointed by BW photos
High-quality black and white

✧ Can we “transfer” some of the low-level qualities?
  • Yes, with histogram matching ++
  • But not just on pixel colors (based on bilateral filtering)
Our result based on Adams' example

With Bae & Paris [Siggraph 06]
Better histogram matching

Domain operations
Domain transform

- Apply a function $f$ from $\mathbb{R}^2$ to $\mathbb{R}^2$ to the image domain
- If $(y, x)$ had color $c$ in the input, $f(y, x)$ has color $c$ in the output
Transformation

- Simple parametric transformations
  - linear, affine, perspective, etc

illustration by Rick Szeliski
Warping

• Imagine your image is made of rubber
• warp the rubber

No prairie dogs were armed when creating this image
Application of warping: weight loss

- Liquify in photoshop
figure 9.37
Selecting the entire left side of the image avoids potential artifacts.

figure 9.38
Dragging a Free Transform handle to narrow the selected area.

figure 9.39
The Liquify filter’s Warp tool pushes pixels forward as you drag.
Step Three:
Get the Push Left tool from the Toolbar (as shown here). It was called the Shift Pixels tool in Photoshop 6 and 7, but Adobe realized that you were getting used to the name, so they changed it, just to keep you off balance.

Step Four:
Choose a relatively small brush size (like the one shown here) using the Brush Size field near the top-right of the Liquify dialog. With it, paint a downward stroke starting just above and outside the love handle and continuing downward. The pixels shifts back in toward the body, removing the love handle as you paint. (Note: If you need to remove love handles on the left side of the body, paint upward rather than downward. Why? That's just the way it works.) When you click OK, the love handle repair is complete.
Domain transform issues

• Apply a function $f$ from $\mathbb{R}^2$ to $\mathbb{R}^2$ to the image domain
• looks easy enough

• But two big issues:
  – which direction do we transform
  – how do we deal with non-integer coordinates?

  – And for warping: how do we specify $f$?
Questions?
Basic resampling
**Naive scaling**

- Take all the pixels in input and transform them to their output location
  - \( \text{im}[y, x] \Rightarrow \text{out}[k \times y, k \times x] \)

```python
def scaleBAD(im, k):
    out = constantIm(im.shape[0]*k, im.shape[1]*k, 0)
    for y, x in imIter(im):
        out[k*y, k*x] = im[y, x]
    return out
```
Use the inverse transform!!!!!

- Main loop on output pixels
  - out[y, x] <= im[y/k, x/k]

```python
def scaleWithInverse(im, k):
    out = constantIm(im.shape[0]*k, im.shape[1]*k, 0.0)
    for y, x in imIter(out):
        out[y,x]=im[y/k, x/k]
    return out
```
Questions?
Remaining problem

• A little too “blocky”
• Because we round to the nearest integer pixel coordinates
  – called nearest neighbor reconstruction
Linear reconstruction

• im is now a 1D array along x
• reconstruct im[1.3]
• \[0.7 \times \text{im}[1] + 0.3 \times \text{im}[2]\]
Bilinear reconstruction

- Take 4 nearest neighbors
- Weight according to x & y fractional coordinates
- Can be done using two 1D linear reconstructions along x then y (or y then x)
Recall nearest neighbor
Bilinear
Take home messages

• Main loop over OUTPUT pixels
  – Makes sure you cover all of them
• Use inverse transform
• Reconstruction makes a difference
  – Linear much better than nearest neighbor
Questions?
Better reconstruction

- Consider more than 4 pixels:
  - bicubic, Lanczos, etc.
- Try to sharpen edges
- Use training database of low-res/high-res pairs
Bicubic (Photoshop)

Ignore small color issues
Bill Freeman’s super-resolution
Questions?
Basic programming tools
Loops & iterators

- Bad news: we’ll need some for loop for resampling
- iterators

```python
def imIter(im):
    for y in xrange(im.shape[0]):
        for x in xrange(im.shape[1]):
            yield y, x
```
Padding problems

- Sometimes, we try to read outside the image
  - e.g. $y, x$ are negative
  - For example, we try to rotate an image
Black Padding

def getBlackPadded(im, y, x):
    if (x<0) or (x>=im.shape[1]) or (y<0) or (y>= im.shape[0]):
        return numpy.array([0, 0, 0])
    else: return im[y, x]
def clipX(im, x): return min(width(im)-1, max(x, 0))
def clipY(im, y): return min(height(im)-1, max(y, 0))
def getSafePix(im, y, x):
    return im[clipY(im, y), clipX(im, x)]
Questions?
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 isolate 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