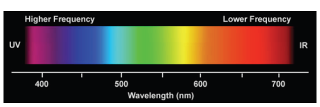
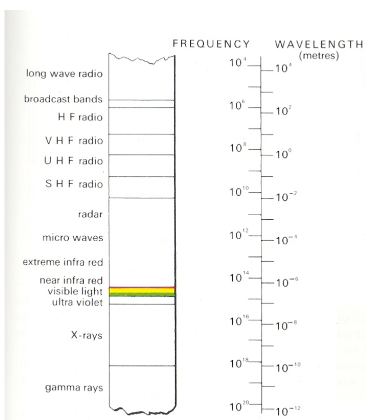
CSC 433/533
Computer Graphics

Lecture 05
Color and Perception

Recall: Light is Electromagnetic Radiation

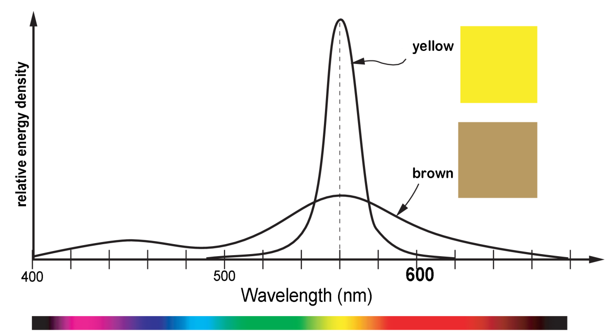
- Visible spectrum is "tiny"
- Wavelength range: 380-740 nm





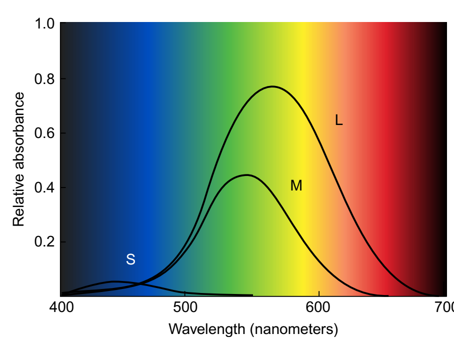
Recall: Color != Wavelength

- But rather, an integral over the wavelengths of the energy encoded of some **power spectrum**



Color and Perception

Recall: We have three types of cones (<u>S</u>hort, <u>M</u>edium, and <u>L</u>ong)



Colin Ware, Information Visualization: Perception for Design

Hunters

Gatherers

Trichromacy

- Our 3 cones cover the visible spectrum (theoretically, all we might are 2 though)
- Most birds, some fish, reptiles, and insects have 4, some as many as 12 (e.g. the mantis shrimp)
- This is a “reason” why many of our acquisition devices and displays use 3 channels, and why many of our color spaces are three dimensional

Mantis Shrimp

16 Photoreceptors, 12 for color sensitivity!

Key Idea: Perception of color

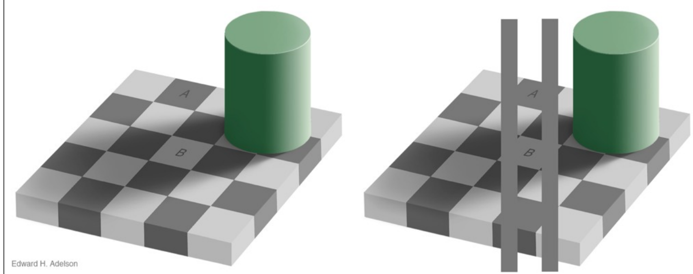
Ultimately, color is a perceptual phenomenon, we all perceive it differently

Color Models

Color Terminology

- **Color Model**
  - Is an abstract mathematical system for representing color.
  - Is often 3-dimensional, but not necessarily.
  - Is typically limited in the range of colors they can represent and hence often can't represent all colors in the visible spectrum
- **Gamut** or **Color Space**
  - The range of colors that are covered by a color model.

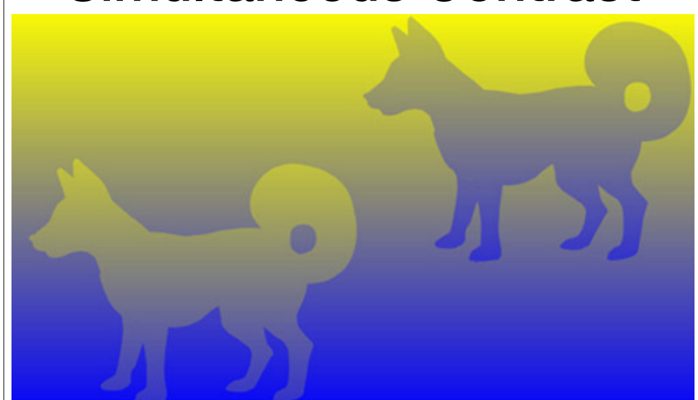
Simultaneous Contrast



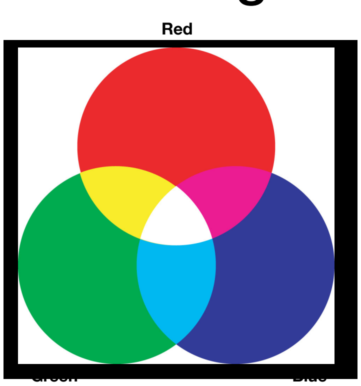
http://persci.mit.edu/_media/gallery/checkershadow_double_full.jpg

Simultaneous Contrast

Simultaneous Contrast



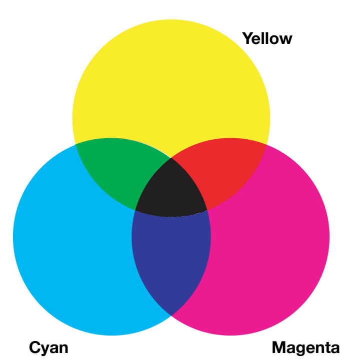
Light Mixing



- **Additive** mix of colored lights (start with black)
  - Add up wavelengths of light to make new colors
- Primary: RGB
- Secondary:
  CMY (cyan, magenta yellow)
- Neutral = R + G + B
- Commonly used by monitors, projectors, etc.

Ink Mixing

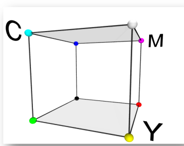
Different game, since we start with white page rather than a black screen
Each color filters the light that is reflected from the white page.



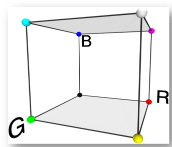
- **Subtractive** mix of transparent inks
  - Start with white and other wavelengths are selectively filtered.
  - The Yellow region does not completely prevent reflection of light from the white page. But it TENDS (depending on transparency) to filter others frequencies)
- Primary: **CMY (Cyan, Magenta, Yellow)**
- Secondary: RGB
- ~Black: C + M + Y
- In practice, we use **CMYK**, with some amount K of black ink, to get true black

Converting from RGB to CMY

- Assuming RGB values are normalized (all channels between [0,1]), the exact same color in CMY space can be found by inverting:



$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1-R \\ 1-G \\ 1-B \end{bmatrix}$$

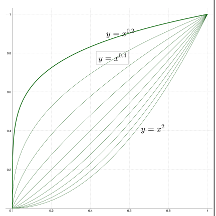


Color Spaces

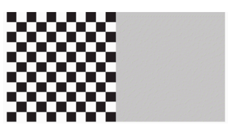
gamma-Correction

- Individual respond from the display (monitor) to every value of GrayScale
- Lets normalize the intensity by using float in [0,1] instead of 255 values of RGB
- such that
- 0 = black, and 1=white
- A pixel with input intensity 0.5 might look very different in different devices.
- Furthermore, the individual response is always monotonic but usually **not linear.**
- On top of it, viewer/illumination/other environmental factor
- So is there a subjective definition of what is gray (**middle** between white and black) ?
- Gamma-Correction. We will assume approximately that if the input is a then

$$\text{displayed intensity} = (\text{maximum intensity})a^{\gamma},$$



- a here is the input intensity to the monitor (between 0 to 1)
- γ is a constant the user could change,
- If no gamma-correction is needed, then the left and right should look the same (when viewed from a distance)
- Change a continuously to the right region, until the output a^{γ} looks like the left region.
- If this happens for some value a of input intensity, we deduce that
- $a^{\gamma} = 0.5$, or $\gamma = (\ln 0.5)/(\ln a)$
- Now every new image, with intensity a', will be displayed using intensity $(a')^{1/\gamma}$

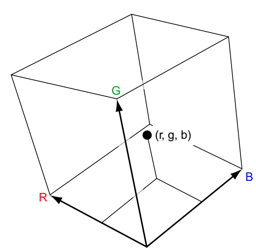


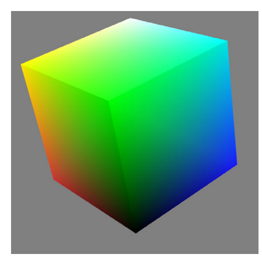
Chessboard of black/white pixels

Uniform region with grey pixels, all get input of 0.5 (before correction).

RGB Color Space

- Additive, useful for computer monitors
- Not perceptually uniform
  - For example, more “greens” than “yellows”





Converting from CMY to CMYK (less relevant to us)

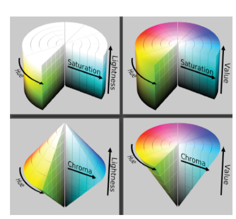
- Assuming CMY values are normalized (all channels between [0,1]), the exact same color in CMYK is

$$\langle C, M, Y, K\rangle = \begin{cases} \langle 0,0,0,1\rangle & \text{if } \min(C', M', Y') = 1, \\ \langle \frac{C'-K}{1-K}, \frac{M'-K}{1-K}, \frac{Y'-K}{1-K}, K\rangle & \text{otherwise where } K = \min(C', M', Y') \end{cases} \tag{3.2}$$

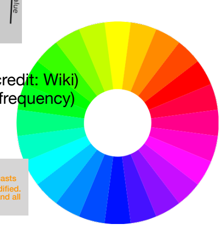
- K is a measure of the ‘blackness’ of the color and essentially serves as an offset after which the remaining amounts of cyan, magenta and yellow are ‘added’

HSL, HSV Color Space

- <u>**H**</u>ue - what people think of as color (color, normalized by sensitivity)
- <u>**S**</u>aturation - purity, distance from grey
  - Also called <u>**C**</u>hroma
- <u>**L**</u>ightness - from dark to light (how many photons, alternatively, add more sources of light)
  - Also <u>**B**</u>rightness or <u>**V**</u>alue



Hue wheel (credit: Wiki)
(not a single frequency)



The HSL color space was invented for television in 1938 by Georges Valensi as a method to add color encoding to existing monochrome broadcasts, allowing existing receivers to receive new color broadcasts (in black and white) without modification as the luminance (black and white) signal is broadcast unmodified. It has been used in all major analog broadcast television encoding including NTSC, PAL and SECAM and all major digital broadcast systems and is the basis for composite video.

Conversion from RGB to HSB

- Assuming RGB values are normalized (all channels between [0,1]), the exact same color in HSB space can be found by first figuring out which channel (R,G, or B) has the max intensity

$$H = \begin{cases} \text{undefined} & \text{if } \max = \min, \\ 60 \times \frac{G-B}{\max - \min} & \text{if } \max = R \text{ and } G \geq B, \\ 60 \times \frac{G-B}{\max - \min} + 360 & \text{if } \max = R \text{ and } G < B, \\ 60 \times \frac{B-R}{\max - \min} + 120 & \text{if } \max = G, \\ 60 \times \frac{R-G}{\max - \min} + 240 & \text{if } \max = B. \end{cases} \quad (3.3)$$

$$S = \begin{cases} 0 & \text{if } \max = 0, \\ 1 - \frac{\min}{\max} & \text{otherwise} \end{cases}$$

$B = \max$. **// 'B' for "brightness". Not 'B' for "blue"**

Note: this method returns H as a value between 0° and 360°

Encoding Color Images

- Could encode 256 colors with a single unsigned byte. But what convention to use?
- One of the most common is to use 3 **channels** or bands
- Red-Green-Blue or RGB color is the most common -- based on how color is represented by lights.
- Coincidentally, this just happens to be related to how our eyes work too.

NOTE : There are many schemes to represent color, most use 3 channels, but the same idea extends to >3 channels

CSC 433/533 Computer Graphics

Anti-Aliasing and
Signal Processing
Sampling, Smoothing and Convolutions

Recall: Images are Functions

Domains and Ranges

- All functions have two components, the **domain** and **range**. For the case of images, I: R → V
- The domain is:
  - R, is some rectangular area (R ⊆ ℝ²)
- The range is:
  - A set of possible values.
  - …in the space of color values we're encoding

Concept for the Day: Pixels are Samples of Image Functions

Image Samples

- Each pixel is a sample of what?
  - One interpretation: a pixel represents the intensity of light at a single (infinitely small point in space)
- The sample is displayed in such a way as to spread the point out across some spatial area (drawing a square of color)

Continuous vs. Discrete

- Key Idea: An image represents data in either (both?) of
  - Continuous domain: where light intensity is defined at every (infinitesimally small) point in some projection
  - Discrete domain, where intensity is defined only at a discretely sampled set of points.
  - This seem like a philosophical discussions without clear practical applications. Surprisingly, it has very concrete algorithmic applications.

Converting Between Image Domains

- When an image is acquired, an image is **sampled** from some continuous domain to a discrete domain.
- **Reconstruction** converts digital back to continuous.
- The reconstructed image can then be **resampled** and **quantized** back to the discrete domain.

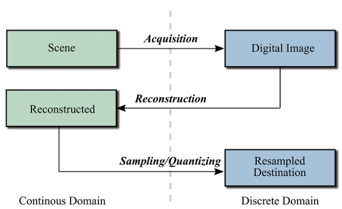


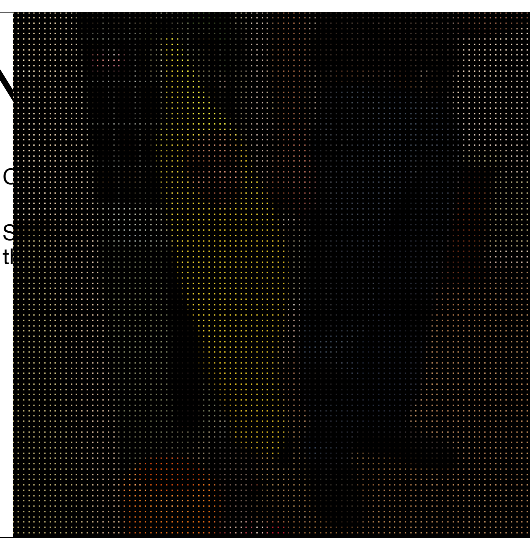
Figure 7.7. Resampling.

Naive Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//copy the pixels over
for (let row = 0, row < H; row++) {
   for (let col = 0; col < W; col++) {
      let index = row*W + col;
      let index2 = (k*row)*W + (k*col);
      output[index2] = input[index];
   }
}
```

N g

- C
- S t



What's the Problem?

- The output image has gaps!
- Why: we skip a many of the pixels in the output.
- Why don't we fix this by changing the code to at least put some color at each pixel of the output?

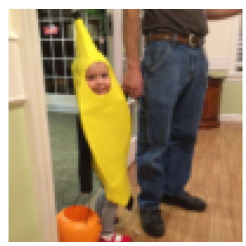
Naive Image Rescaling Code

```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//copy the pixels over
for (let row = 0, row < H; row++) {
   for (let col = 0; col < W; col++) {
      let index = row*W + col;
      let index2 = (k*row)*W + (k*col);
      output[index2] = input[index];
   }
}
```

"Inverse" Image Rescaling Code

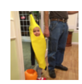
```
//scale factor
let k = 4;

//create an output greyscale image that is both
//k times as wide and k times as tall
Uint8Array output = new Uint8Array((k*W)*(k*H));

//Loop over each output pixel instead.
for (let row = 0, row < k*H; row++) {
   for (let col = 0; col < k*W; col++) {
      let index = (row/k)*W + (col/k);
      let index2 = row*k*W + col;
      output[index2] = input[index];
   }
}
```

Inverse Image Rescaling



400x400 image

Not great, but could become worse



100x100 image

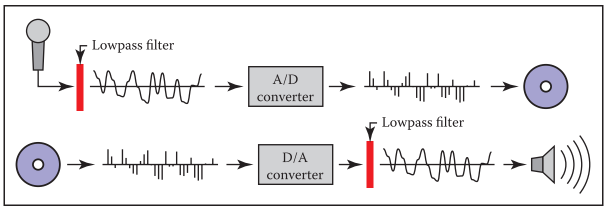
What's the Problem?

- The output image is too "blocky"
- Why: because our image reconstruction rounds the index to the nearest integer pixel coordinates
  - Rounding to the "nearest" is why this type of interpolation is called **nearest neighbor interpolation**

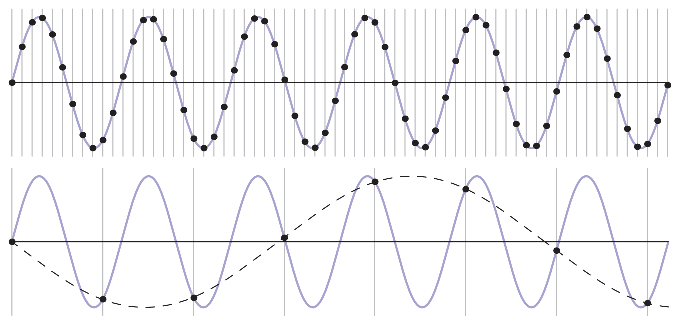
Sampling Artifacts / Aliasing

Motivation: Digital Audio

- Acquisition of images takes a continuous object and converts this signal to something digital
- Two types of artifacts:
  - **Undersampling** artifacts: on acquisition side
  - **Reconstruction** artifacts: when the samples are interpreted

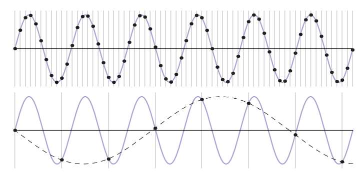


Undersampling Artifacts



Shannon-Nyquist Theorem

(not needed for the exam)

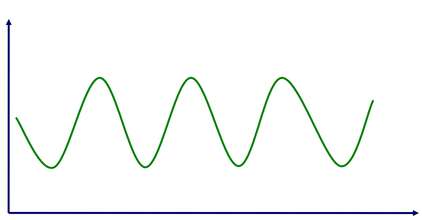


- The sampling frequency must be **double** the highest frequency of the content.
- If there are any higher frequencies in the data, or the sampling rate is too low, **aliasing**, happens
  - Named this because the discrete signal "pretends" to be something lower frequency

S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

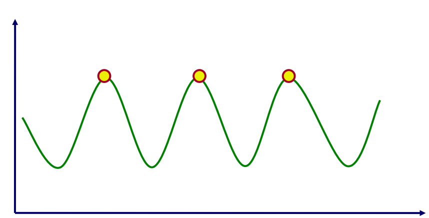
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

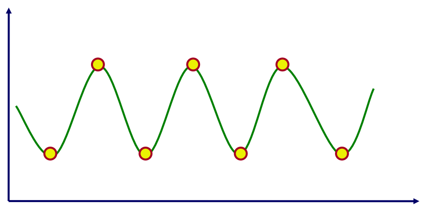
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

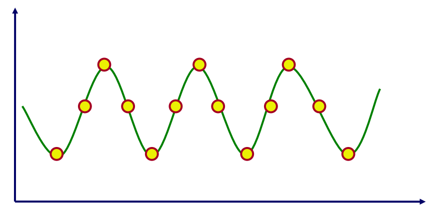
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?

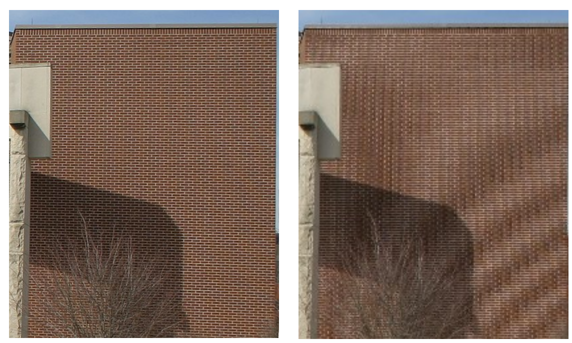


Aliasing in images

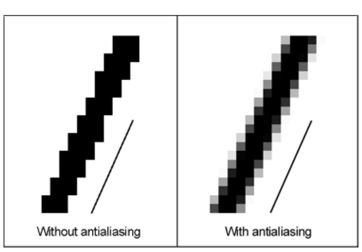
Two outcomes of under-sampling

1) Moire Pattern
2) Rasterization

Moire Patterns



Aliasing for edges



Each pixel is effected by nearby pixels
For example, even though the input image image is black/white,
We allow grey values for output pixels.

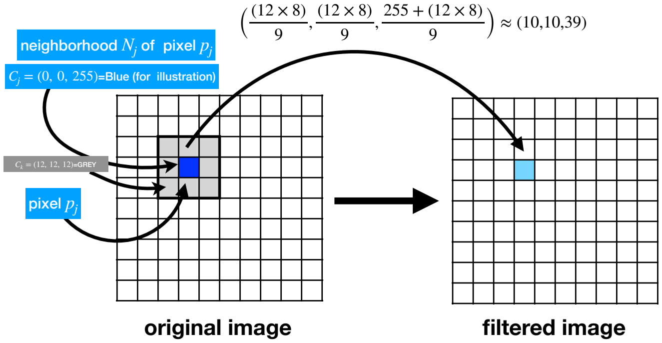
Convolution

Each pixel is effected by nearby pixels
For example, even though the image is black/white,
We allow grey values

Neighborhood Filtering (Schematic)

f(Nj)=average color in this region (neighborhood)=

$$\left(\frac{(12\times 8)}{9}, \frac{(12\times 8)}{9}, \frac{255+(12\times 8)}{9}\right) \approx (10{,}10{,}39)$$

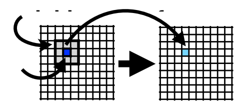


An Example: Mean Filtering

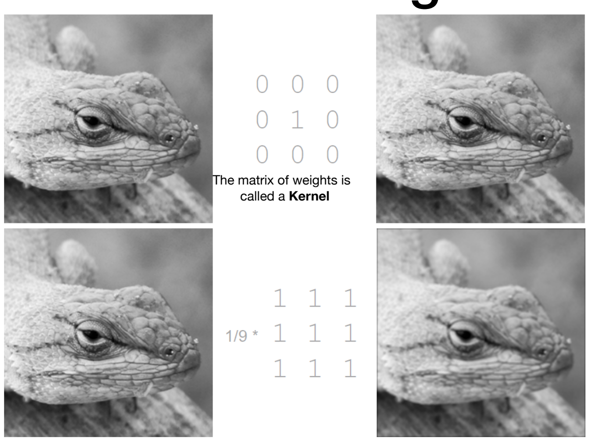
- Mean filters sum all of the pixels in a local neighborhood N_i and divide by the total number, computing the average pixel.
- Said another way, we replace each pixel as a linear combination of its neighbors (with equal weights!)
- To find the new color of a pixel *j,* we will look at N_j , defined as the (say) 3×3 neighborhood of the pixel p_j, and set
- Where the N_i is a square, we call these **box** filters
- Think about it as a weighted average:

$$f(N_i) = \sum_{\textbf{pixels } p_k \textbf{ in the region } N_j} w_k C_k = \sum_{\textbf{pixels } p_k \textbf{ in the region } N_j} \frac{1}{9} C_k$$

- The weights $w_1 \dots w_k$ are convex combination. Meaning that they are all positive, and $w_1 + w_2 + \dots w_k = 1$. For example, $w_1 = w_2 = w_3 = \frac{1}{3}$ **(convex combination)**
- Remember: The input matrix and the output matrix have the same size (in this case). This is **not** rescaling.
- Refer to the geogebra app https://www.geogebra.org/m/cetpvwaw
- The term filter is very common, but might be very confusing. We don't necessarily filter out anything.



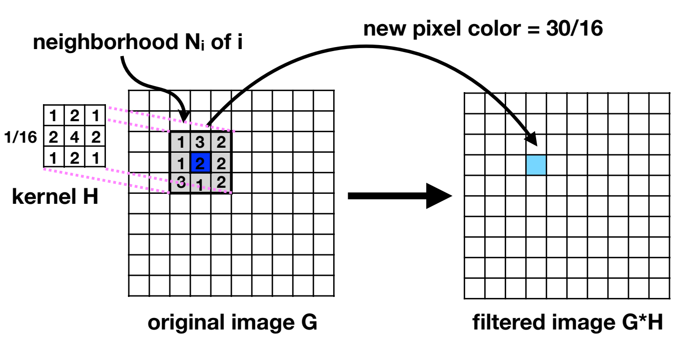
Box Filtering



Box Filtering

Convolution

- This process of adding up pixels multiplied by various weights is called **convolution.** We denote the result by **(confusion warning)** the symbol * **See example below.**



Kernels

- Convolution employs a rectangular grid of coefficients, (that is, weights) known as a **kernel**
- Kernels are like a neighborhood mask, they specify which elements of the image are in the neighborhood and their relative weights.
- A kernel is a set of weights that is applied to corresponding input samples that are summed to produce the output sample.
- For **smoothing** purposes, the sum of weights must be 1 (convex combination)

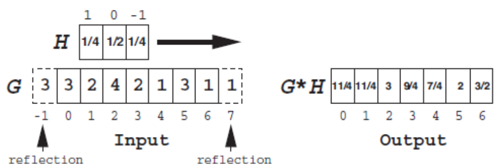
$$\frac{1}{9}\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix} \quad \frac{1}{13}\begin{pmatrix}1&1&1\\1&5&1\\1&1&1\end{pmatrix} \quad \frac{1}{37}\begin{pmatrix}1&1&1&1&1\\1&2&2&2&1\\1&2&5&2&1\\1&2&2&2&1\\1&1&1&1&1\end{pmatrix}$$

One-dimensional Convolution

- Can be expressed by the following equation, which takes a filter H and convolves it with G:

$$\hat{G}[i] = (G * H)[i] = \sum_{j=i-n}^{i+n} G[j]H[i-j], \; i \in [0, N-1]$$

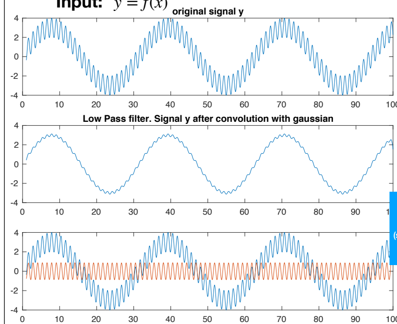
- Equivalent to sliding a window



Low-pass and high-pass filtering

The smoothing operation is always a low pass filter.
Only lower frequencies could pass.
It removes higher frequencies from the input.

Input: $y = f(x)$



We convolved the original signal f(x) a smoothing kernel H. For example

$$g(x) = \frac{f(x-1) + f(x) + f(x+1)}{3}$$

The output of the smoothing operation
$g(x) = f(x) * H$
The higher frequencies are less noticeable:
we need to move a lot (in x) to notice a large different in y

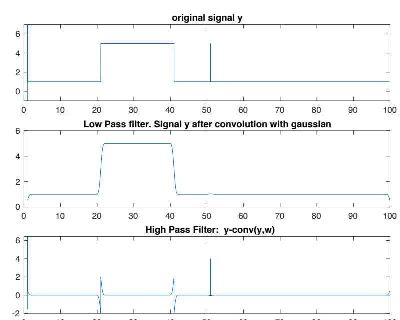
New idea: High-pass filter.
$h(x) = f(x) - g(x)$
Only high frequencies pass
(shown: Original signal (blue) and the result of the high pass filter (red))

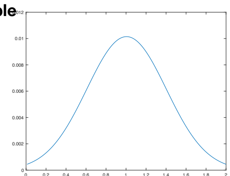
We remove (subtract) from the signal all lower frequencies

Twitter - could move very fast, but only small distances

Woofer - moves slowly but cold cover large distances

Low pass and hight pass filters - another example

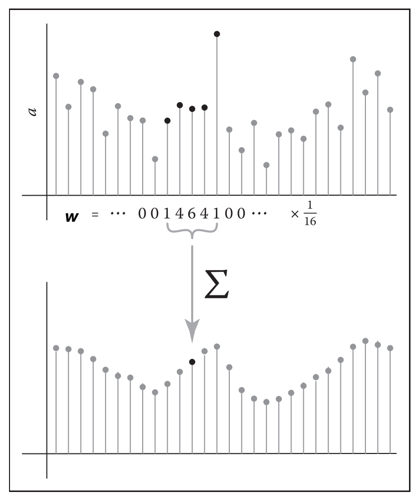




We convolved the original signal y with this gaussian

Convolution is a Moving, Weighted Average

- Getting used to the new notation:
  $b[i] = \frac{1}{3}(\, a[i-1] + a[i] + a[i+1] \,) \quad \forall i$
- is similar to writing $b = a \star w$, where
  $b[i] = (a \star w)[i] = \sum_{j=1}^{3} a[i-j+2] \cdot w[j]$ and
  w[1]=w[2]=w[3]=1/3
- Commonly $(a \star w)[i] = \sum_{j=i-r}^{j=i+r} a[j]w[i-j]$
- For example, w[-1]=w[0]=w[1]=1/3
- Note that we did not define exactly what are the first and last values



2-Dimensional Version

- Given an image a and a kernel b with (2r+1)² values, the convolution of a with b is given below as a*b:

$$(a \star b)[i,j] = \sum_{i'=i-r}^{i+r} \sum_{j'=j-r}^{j+r} a[i',j']\,b[i-i',j-j']$$

- The (i-i') and (j-j') terms can be understood as reflections of the kernel about the central vertical and horizontal axes.
- The kernel weights are multiplied by the corresponding image samples and then summed together.

A Note on Indexing

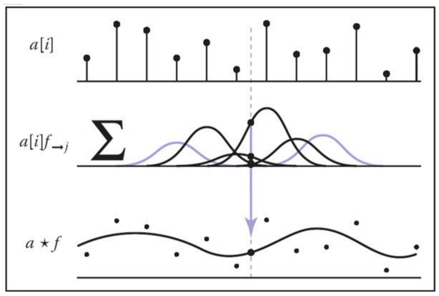
- Convolution **reflects** the filter to preserve orientation.
- **Correlation** does **not** have this reflection.
  - But we often use them interchangeably since most kernels are symmetric!!

Convolution reflects and shifts the kernel

Given kernel H =
$$\begin{matrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{matrix}$$

Convolution Can Also Convert from Discrete to Continuous

- Discrete signal a
- Continuous filter f
- Output a*f defined on positions x as opposed to discrete pixels i



$$(a \star f)(x) = \sum_{i=\lceil x-r \rceil}^{\lfloor x+r \rfloor} a[i] f(x-i)$$

B g

Filtering helps to reconstruct the signal better when rescaling

Inverse Rescaling

Reconstructed w/ Discrete-to-Continuous

Types of Filters: Smoothing

Smoothing Spatial Filters

- Any weighted filter with positive values will smooth in some way, examples:

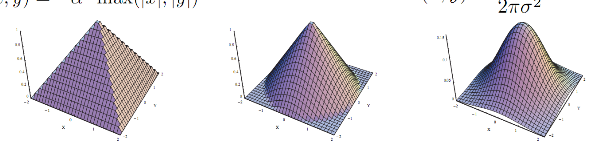
$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \qquad \frac{1}{16} \times \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

- Normally, we use integers in the filter, and then divide by the sum (computationally more efficient)
- These are also called **blurring** or **low-pass** filters

Smoothing Kernels

$$f(x,y) = -\alpha \cdot \max(|x|,|y|)$$

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



(a) Pyramid. (b) Cone. (c) Gaussian.

$$f(x,y) = -\alpha \cdot \sqrt{x^2+y^2}$$

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

(a) Pyramid.

0	0	1	0	0
0	2	2	2	0
1	2	5	2	1
0	2	2	2	0
0	0	1	0	0

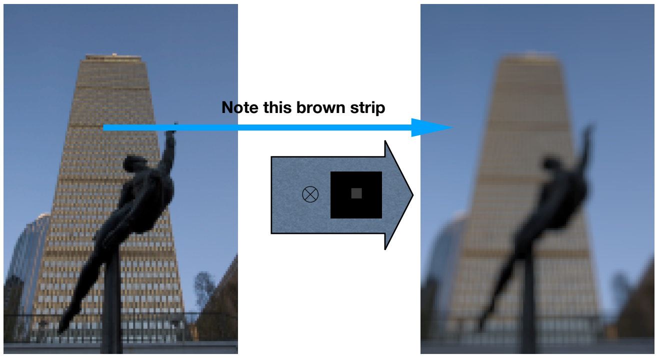
(b) Cone.

1	4	7	4	1
4	16	28	16	4
7	28	49	28	7
4	16	28	16	4
1	4	7	4	1

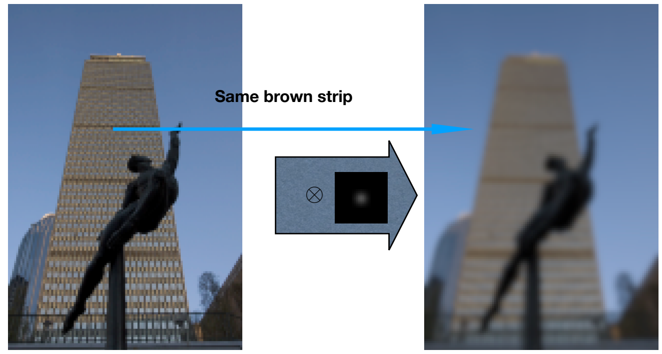
(c) Gaussian.

Table 6.1. Discretized kernels.

Box Filter



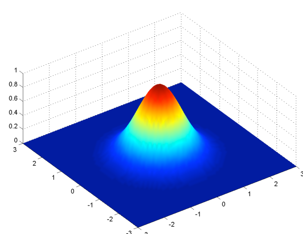
Gaussian Filter



Gaussians

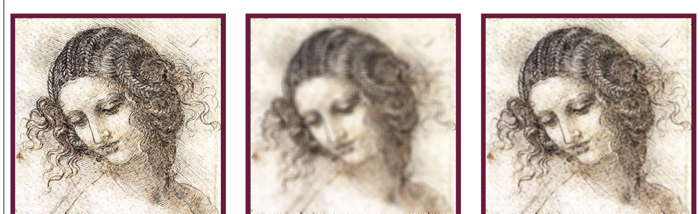
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

- Gaussian kernel is parameterized on the standard deviation σ
- Large σ's reduce the center peak and spread the information across a larger area
- Smaller σ's create a thinner and taller peak
- Gaussians are smooth everywhere.
- Gaussians have infinite **support**
  - >0 everywhere
- But often truncate to 2σ or 3σ
- Volume =1 (sum of weights =1)



http://en.wikipedia.org/wiki/Gaussian_function

Smoothing Comparison

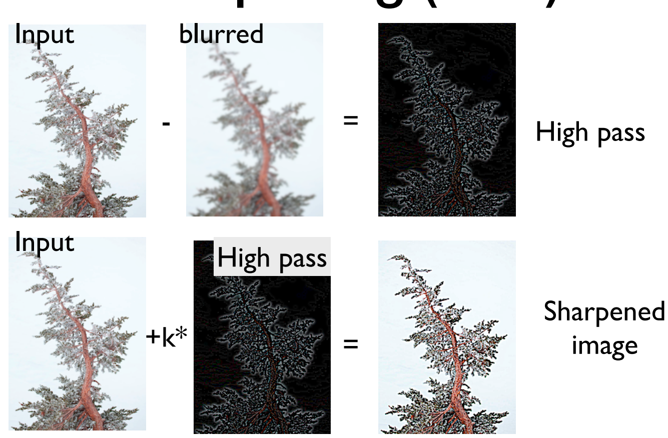


(a) Source image. (b) 17 × 17 Box. (c) 17 × 17 Gaussian.

Figure 6.10. Smoothing examples.

Types of Filters: Sharpening

Sharpening (Idea)



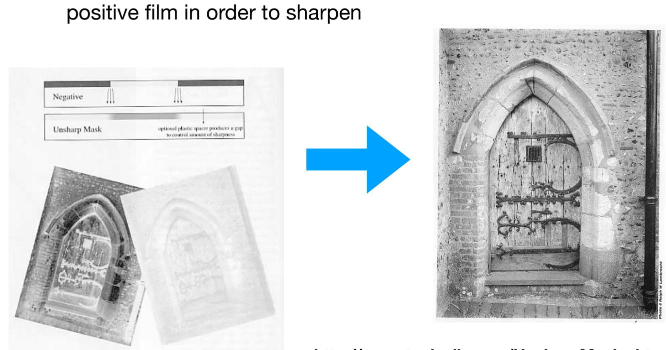
Another example

Original Image, Imaged convolved

Left: difference (only boundaries are non-black)
Right Imaged minus differences convolved

Unsharp Masks

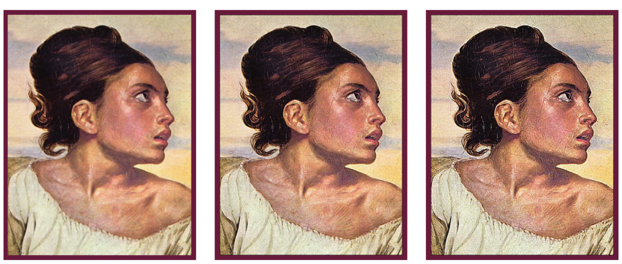
- Sharpening is often called “unsharp mask” because photographers used to sandwich a negative with a blurry positive film in order to sharpen



http://www.tech-diy.com/UnsharpMasks.htm

Edge Enhancement

- The parameter α controls how much of the source image is passed through to the sharpened image.



(a) Source image. (b) $\alpha = .5$. (c) $\alpha = 2.0$.

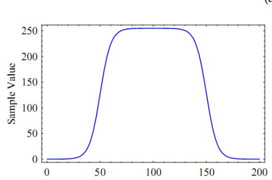
Figure 6.20. Image sharpening.

Defining Edges

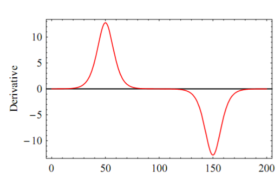
- Sharpening uses negative weights to enhance regions where the image is changing rapidly
  - These rapid transitions between light and dark regions are called **edges**
- Smoothing reduces the strength of edges, sharpening strengthens them.
  - Also called **high-pass** filters
- Idea: smoothing filters are weighted averages, or integrals. Sharpening filters are weighted differences, or derivatives!

Edges

(a)



(b)



(c)

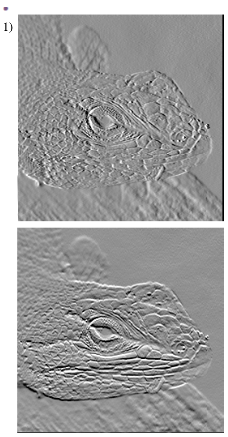
Figure 6.11. (a) A grayscale image with two edges, (b) row profile, and (c) first derivative.

Taking Derivatives with Convolution
(just in case you studied calculus. Not required)

$$\frac{\partial}{\partial x} f(x, y) \approx \frac{1}{2} f(x+1) - f(x+1)$$

$$\begin{matrix} 0 & 0 & 0 \\ -1 & 0 & 1 \\ 0 & 0 & 0 \end{matrix}$$

$$\begin{matrix} 0 & -1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{matrix}$$



Gradients with Finite Differences

(just in case you studied calculus. Not required)

- These partial derivatives approximate the image gradient, ∇I.
- Gradients are the unique direction where the image is changing the most rapidly, like a slope in high dimensions
- We can separate them into components kernels G_x, G_y. $\nabla I = (G_x, G_y)$

$$\nabla I(x,y) = \begin{pmatrix} \delta I(x,y)/\delta x \\ \delta I(x,y)/\delta y \end{pmatrix}.$$

$$G_x = [1, 0, -1] \qquad G_y = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix};$$

$$\nabla I = \begin{pmatrix} \delta I/\delta x \\ \delta I/\delta y \end{pmatrix} \simeq \begin{pmatrix} I \otimes G_x \\ I \otimes G_y \end{pmatrix}.$$

Figure 6.12. Image gradient (partial).

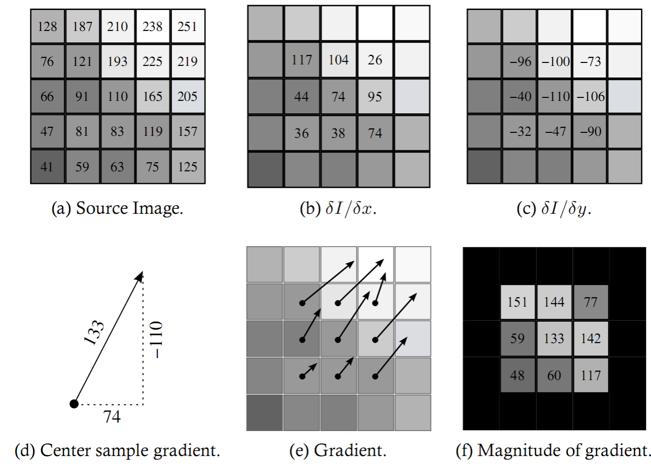
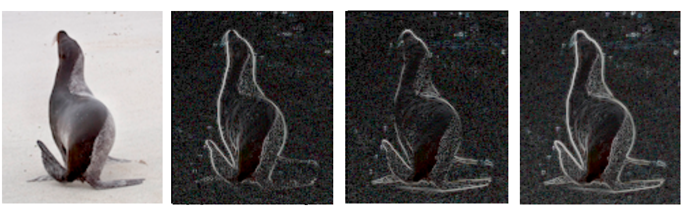


Figure 6.14. Numeric example of an image gradient.

Gradients G_x, G_y



$|G| = \sqrt{(G_x^2 + G_y^2)}$

Second Derivatives (Sharpening, almost)

- Partial derivatives in x and y lead to two kernels:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

and, similarly, in the y-direction we have

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

Compare with Sharpening filter: unbalanced counts!

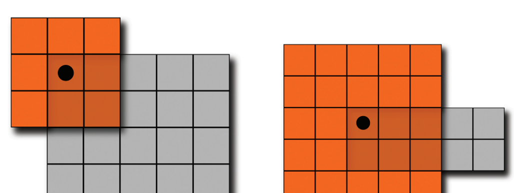
$$\begin{bmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & (9+8\alpha) & -\alpha \\ -\alpha & -\alpha & -\alpha \end{bmatrix}$$

1	1	1
1	-9	1
1	1	1

Boundaries

Handling Image Boundaries

- What should be done if the kernel falls off of the boundary of the source image as shown in the illustrations below?



(a) Kernel at $I(0,0)$. (b) Kernel larger than the source.

Figure 6.4. Illustration of the edge handling problem.

Handling Image Boundaries

- When pixels are near the edge of the image, neighborhoods become tricky to define
- Choices:
  1. Shrink the output image (ignore pixels near the boundary)
  2. Expanding the input image (padding to create values near the boundary which are "meaningful")
  3. Shrink the kernel (skip values that are outside the boundary, and reweigh accordingly)

Boundary Padding

- When one pads, they pretend the image is large and either produce a constant (e.g. zero), or use circular / reflected indexing to tile the image:

(a) (b) (c)

Figure 6.5. (a) Zero padding, (b) circular indexing, and (c) reflected indexing.