CSC 433/533 Computer Graphics

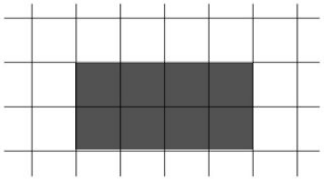
Review

Course Material Source Credits

Transformations in 2D Short version

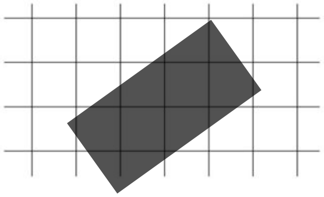
We will discuss transformation in 3D, and with full details, later in the course (will need Matrix Multiplication and Homogenous coordinates)

About hw1
Aliasing and Anti-Aliasing



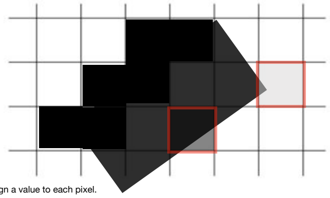
- This about an image where each pixels is fully black or fully white

What if we rotate the rectangle



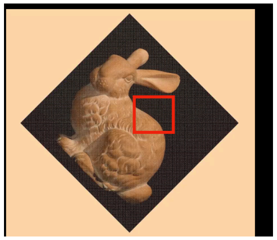
- Some pixels are **partially** covered by the rectangle. Show they be rendered as black, white, or some shade of grey ?

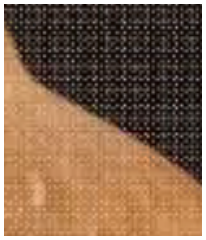
What if we rotate the rectangle



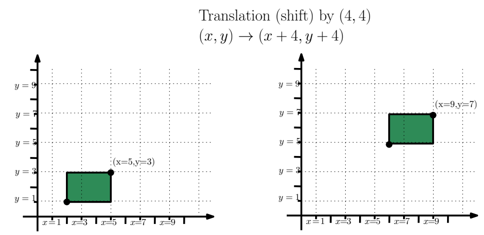
- IWe still need to assign a value to each pixel.
- If we draw each **partially covered** pixel as black, we will obtain a very pixelated shape. This is an example of **aliasing.**
- A possible solution is to render some pixels as gray. For example, based on the portion of its area which is covered. This technique is call antialiasing. Essentially, the color of a pixel might be determined using input from several neighboring pixels.
- We will study much much more about it. Do not worry about it in hw1.
- In hw1, each rendered pixel has the (rgb) value of one (single) input pixel. No averaging or mixing.

Something to be careful about with hw1





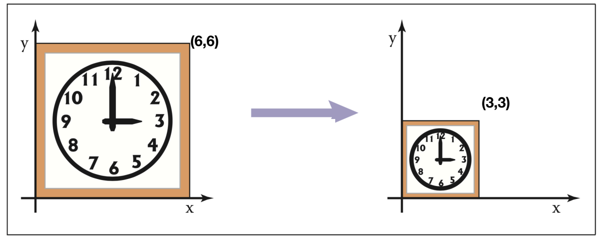
Translations (shift) by (α, β)



- Adding a constant α to the x-coordinate of every point
- Adding a constant β to the y-coordinate of every point
- $(x, y) \to (x + \alpha, y + \beta)$

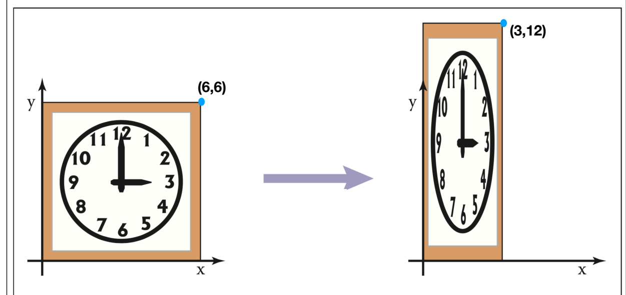
Scaling

- We can use two constants (s_x, s_y) for the x-axis and the y-axis. Then we shift each point (x,y) into the point $(s_x \cdot x,\ \ s_y \cdot y)$
- $(x, y) \to (s_x \cdot x,\ \ s_y \cdot y)$
- Example $(x, y) \to (x/2,\ y/2)$

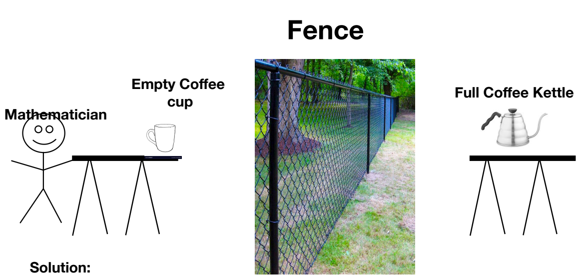


Scaling

- Example: $(x, y) \to (0.5x,\ \ 2y)$

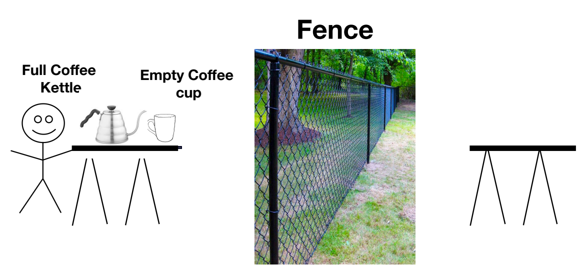


The mathematician and coffee cup non-funny joke Part 1



Solution:
1. **Walk around the fence,**
2. **fetch coffee kettle,**
3. **walk back pure coffee,**
4. **drink**

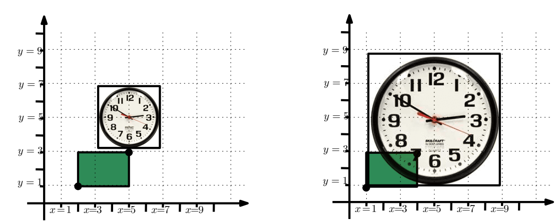
The mathematician and coffee cup non-funny joke Part 2



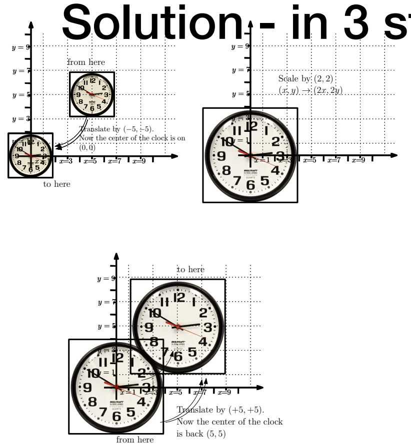
Solution:
1. **Bring the coffee Kettle to the other table, and walk to the left table**
2. **Apply the solution from the previous slide**

Resize the clock, without changing its center

Problem: scale the clock, but without changing its center and without effecting the green rectangle

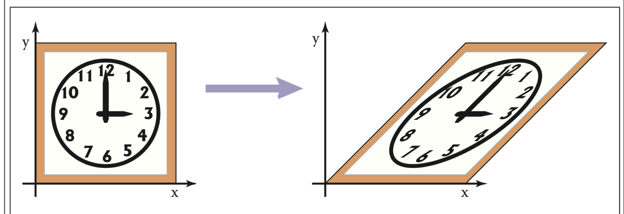


Solution - in 3 steps



Shearing

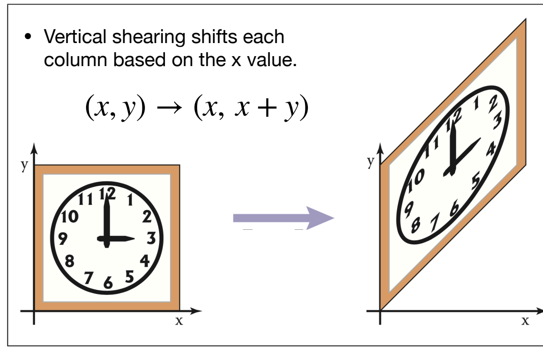
- If we move each point (x,y) into the point $(x,y) \to (x+y,\ y)$



Shearing

- Vertical shearing shifts each column based on the x value.

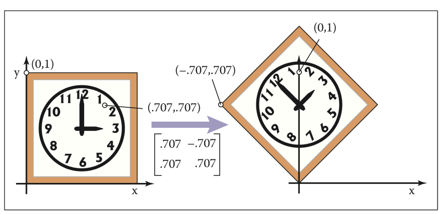
$$(x,y) \to (x,\ x+y)$$



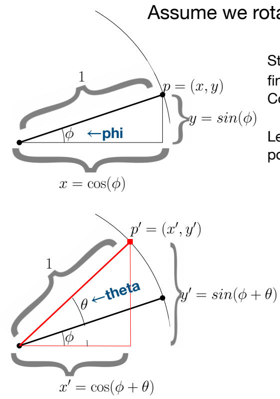
Rotation

- Rotate counterclockwise by an angle ϕ about the origin.

$$(x,y) \to (\underbrace{x\cos\phi - y\sin\phi}_{\text{New } x},\ \underbrace{x\sin\phi + y\cos\phi}_{\text{New } y})$$



Assume we rotate p by an angle θ CCW



Starting from a point p=(x,y), where will this point find itself after rotation by θ in the CounterClockwise direction ?

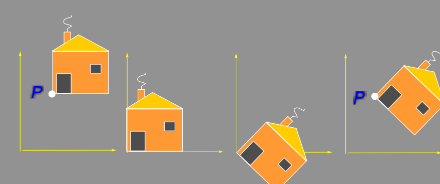
Let $p' = (x', y')$ denote the new location of this point. Lets compute this location:

$$x' = \cos(\phi+\theta) = \underbrace{\cos(\phi)}_{=x}\cos(\theta) - \underbrace{\sin(\phi)}_{=y}\sin(\theta) = x\cos(\theta) - y\sin(\theta)$$

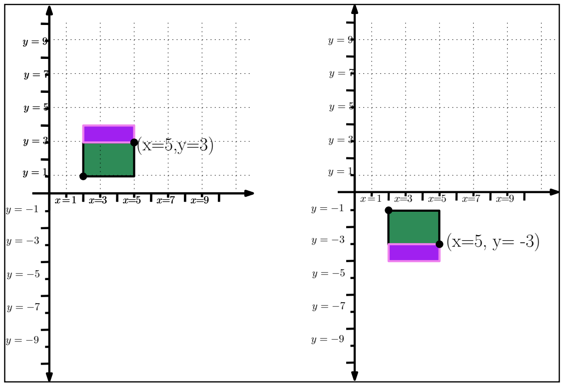
$$y' = \sin(\phi+\theta) = \underbrace{\sin(\phi)}_{=y}\cos(\theta) + \underbrace{\cos(\phi)}_{=x}\sin(\theta) = x\sin(\theta) + y\cos(\theta)$$

Transformation Composition

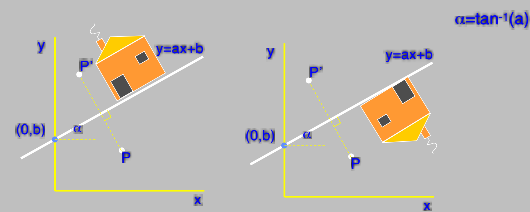
- What operation rotates by θ around $P = (p_x, p_y)$?
  - Translate P to origin
  - Rotate around origin by θ
  - Translate back



Reflection on the x-axes: $(x, y) \rightarrow (x, \ -y)$



Arbitrary Reflection - promo
We will get back to it later in the semester



1. Compute b.
2. Shift by (0,-b)
3. Rotate by –α CCW
4. Reflect through x
5. Rotate by α
6. Shift by (0,b)

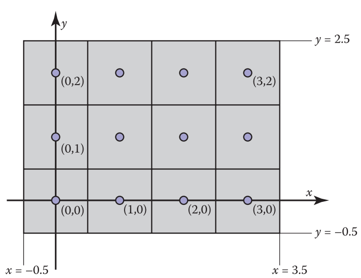
Very scarrrrry….
Unless we represent transformation by matrices
And then it is trivial

A Simple Mathematical Abstraction for Images

- We can abstract an image as a *function*, I
- $I: R \rightarrow V$
  - The **domain**, R, is a some **continuous** rectangular area ($R \subseteq \mathbb{R}^2$) and the **range**,
- V, is a set of possible values.
- Since R is two dimensional, we can use $I(x,y)$ to represent the value of the image at a position $(x,y) \in R$

Raster Images

- We digitize $I(x,y)$ as an array of values, $I[y][x]$, called **pixels**, for **pic**ture **el**ements



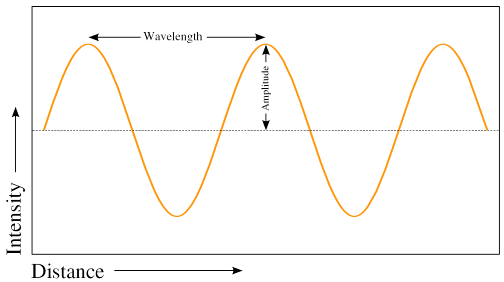
NOTE: (0,0) is often the top left, not the bottom left!

How Do We Acquire Raster Images?

Light

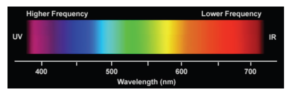
- Is both: (1) particles known as **photons** that (2) act as **waves**.
  - **Amplitude** (height of wave)
  - **Wavelength** (distance of which wave repeats)
    - Frequency is the inverse of wavelength
    - Relationship between wavelength (λ) and frequency (f):
      - $\lambda = c / f$
    - Where c = speed of light = 299,792,458 m / s

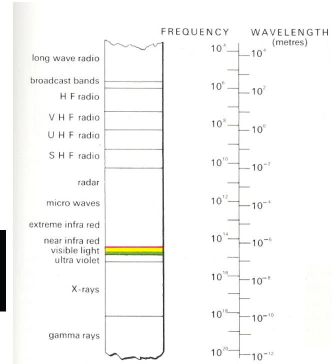
Light



Light is Electromagnetic Radiation

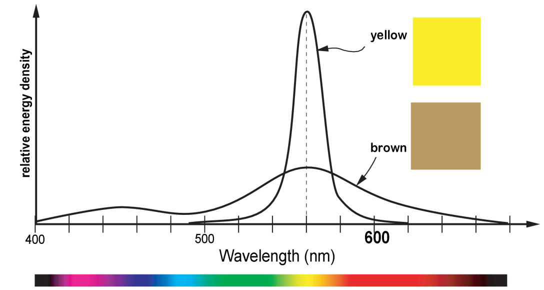
- Visible spectrum is "tiny"
- Wavelength range: 380-740 nm



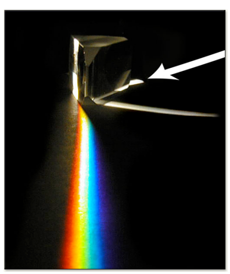


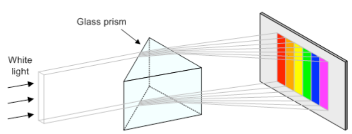
Color != Wavelength

- But rather, a combination of wavelengths and energy



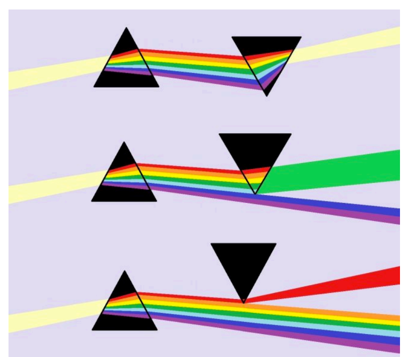
Isaac Newton, 1666





Newton's experiment for splitting white light into a spectrum

http://www.webexhibits.org/colorart/bh.html
https://www.clivemaxfield.com/diycalculator/popup-m-cvision.shtml

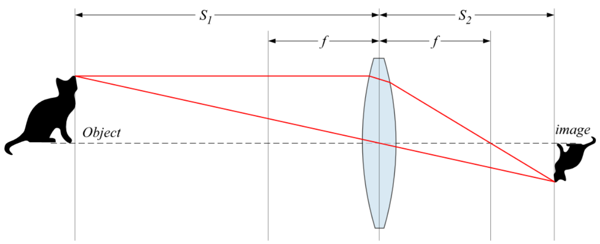


http://www.thestargarden.co.uk/Newtons-theory-of-light.html

Optics: Thin Lenses

- A **lens** is a transparent device that allows light to pass through while causing it to either converge or diverge.
- Given a camera, a target object, and a single converging lens:
  - Let S1 and S2 be the distance from the lens to the target and film
  - The **focal length**, f, is a measure of how strongly a lens converges light
  - The **magnification factor**, m = S2/S1, relates the two distances.

Thin Lens Equation

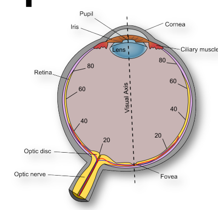


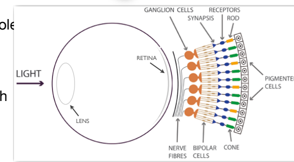
$$m = \frac{\text{image size}}{\text{object size}} = \frac{S_2}{S_1}$$

$$\frac{1}{S_1} + \frac{1}{S_2} = \frac{1}{f}$$

Human Optics

- In human vision, the **cornea** acts as a protective lens that roughly focuses incoming light
- Iris controls the amount of light that enters the eye
- The **lens** sharply focuses incoming light onto the retina
  - Absorbs both infrared and ultraviole light which can damage the lens
- The **retina** is covered by **photoreceptors** (light sensors) which measure light





Photoreceptors

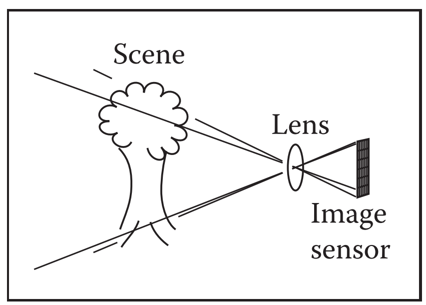
- **Rods** (detect low-light / scoptopic vision)
  - Approximately 100-150 million rods (Non-uniformly distributed across the retina)
  - Sensitive to low-light levels (scotopic vision)
- **Cones** (detect day-light / phototopic vision)
  - Approximately 6-7 million cones.
  - Detect color with 3 different kinds:
    - Red (L cone) : 564-580nm wavelengths (65% of all cones)
    - Green (M cone) : 534-545nm (30% of all cones)
    - Blue (S cone) : 420-440nm (5% of all cones)

From Humans to Machines: Charge-Coupled Devices (CCDs)

- A CCD is an electronic circuit with a grid of small rectangular photocells.
- The optical lens focuses a scene onto the sensors.
- Each photocell measures the amount of light that hits it.
- The collective data of the sensors represents an image when viewed from a distance.

http://en.wikipedia.org/wiki/Charge-coupled_device

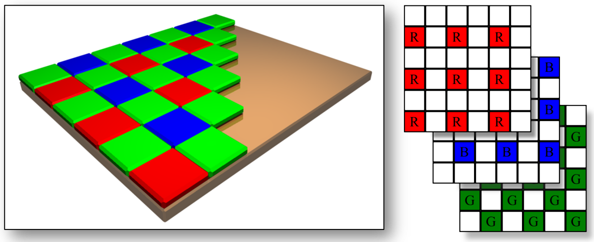
Color Image Acquisition



Color Image Acquisition

- In a single CCD color digital camera each individual photosite of the CCD is filtered to detect either red, green, OR blue light
- Most filters mimic the cone density of the human eye
- The Bayer filter uses 50% green and 25% red and blue sites.
- The 'RAW' data must be **demosaiced** (fill in the gaps) to produce a true-color image.

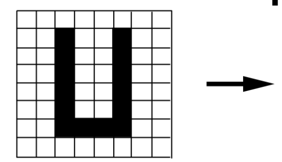
Bayer Filter



- Newer technology allows each photosite is able to discriminate and measure red, green and blue light simultaneously.

How Do We <u>**Encode**</u> Raster Images?

Bitmaps



```
1 1 1 1 1 1 1 1
1 1 0 1 1 0 1 1
1 1 0 1 1 0 1 1
1 1 0 1 1 0 1 1
1 1 0 1 1 0 1 1
1 1 0 1 1 0 1 1
1 1 0 0 0 0 1 1
1 1 1 1 1 1 1 1
```

- **Bitmap**: digital image that is a 2d array of pixels which store one bit.
  - Simplest digital image, a representation of a black and white image.
- Bit: ones/zeros, convention is 0 = black & 1 = white.

Digital Images Linearized

```
1 1 1 1 1 1 1 1
1 1 0 1 1 0 1 1
1 1 0 1 1 0 1 1
1 1 0 1 1 0 1 1
1 1 0 1 1 0 1 1
1 1 0 1 1 0 1 1
1 1 0 0 0 0 1 1
1 1 1 1 1 1 1 1
```

- While we think of images as 2-dimensional, in memory we usually prefer to pack storage so that they are 1-dimensional.
- The same image can be represented in both binary and hexadecimal

11111111	11011011	11011011	11011011	11011011	11011011	11000011	11111111

FF	DB	DB	DB	DB	DB	C3	FF

Greyscale Images - Pixmaps

- We use 0 for black and 1 for white -- what value should we use for grey?
- Could use floating point numbers
- Instead, one convention is to use 8 bits for pixel -- how many different "shades of grey"?
- Can convert to [0.0,1.0] by dividing by 255

Javascript and Arrays

- Standard `Array` type in Javascript is *sparse*:
  - No guarantee of contiguous block of memory, memory is allocated on demand.
  - Can store mixed types
- Javascript `TypedArray` does use a contiguous block of memory
  - But, requires a fixed type. E.g. Byte,

The operation Array.from

Definition and Usage
The Array.from() method returns an Array object from any object with a length property or an iterable object.

Create an Array from a String:

```
var myArr = Array.from("ABCDEFG");
```

`Array.from()` is not supported in Internet Explorer 11 (or earlier).

Definition and Usage
The Array.from() method returns an Array object from any object with a length property or an iterable object.

Syntax

```
B=Array.from(object, mapFunction, thisValue)
```

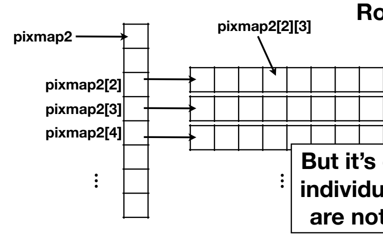
```
B=Array.from([1, 2, 3], x => x + x)
console.log(B);
Output:   Array [2, 4, 6]
```

```
function f() {
  return Array.from(arguments);
}
f(1, 2, 3);
// [ 1, 2, 3 ]
```

Image Allocation

```
let ROWS = 8;
let COLS = 8;
let pixmap2 = Array.from(Array(ROWS), () => Array(COLS));
```

Rows separated in memory!



But it's even worse since individual elements likely are not adjacent either!

Pixmap Declaration In Javascript

```
let ROWS = 8; let COLS = 8;

//pixmap is an array of arrays
let pixmap = [];
for (let r = 0; r < ROWS; r++) {
  pixmap[r] = [];
}

//pixmap2 is an array of arrays, but with lengths specified
 let pixmap2=Array.from(Array(COLS), ()=>new Array(COLS))

//pixmap3 is utilizes a 1-d array for the whole thing
let pixmap3 = Array(ROWS * COLS);

//top left pixel (x,y) = (0,0)
pixmap[0][0]; pixmap2[0][0]; pixmap3[0];

//top right pixel (x,y) = (7,0)
pixmap[0][7]; pixmap2[0][7]; pixmap3[7];

//bottom left pixel, in general [index] = [y*COLS+x]
pixmap[7][0]; pixmap2[7][0]; pixmap3[56];
```

Pixmap Declaration with TypedArrays

```
let ROWS = 8; let COLS = 8;

//pixmap4 is utilizes a 1-d array for the whole thing
let pixmap4 = new Uint8Array(ROWS*COLS);

//Instead of storing as 0..255, can use other types
//e.g. Uint8ClampedArray, Float32Array, etc.
// Clamped here means "automatic and reasonable rounding:
  // -3.9-> 0;      266.5->255 etc"

//Can we do multidimensional TypedArray
//using Arrays of arrays?
let pixmap4 = new Uint8Array(ROWS);
pixmap4[0] = new Uint8Array(COLS);    //incorrect!


//for TypedArrays, we must use 1-d indexing
//e.g. [index] = [y*COLS+x]
```

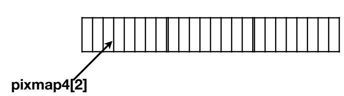
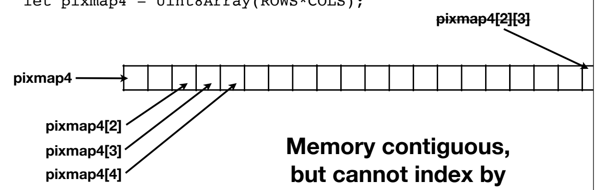


Image Allocation

```
let ROWS = 8;
let COLS = 8;
let pixmap4 = Uint8Array(ROWS*COLS);
```

We don't have this



Memory contiguous, but cannot index by row / column

Encoding Color Images

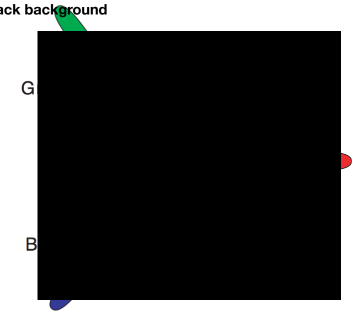
- Could encode 256 colors with an unsigned char. But what convention to use?
- One of the most common is to use 3 **channels** or bands
- Red-Green-Blue or RGB color is the most common -- based on how color is represented by lights.
- Coincidentally, this just happens to be related to how our eyes work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

NOTE: There are many schemes to represent color, most use 3 channels. We'll come back to this next lecture

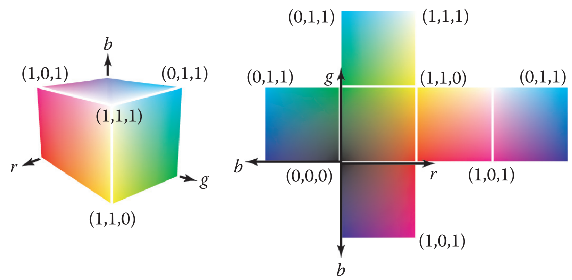
RGB Colors

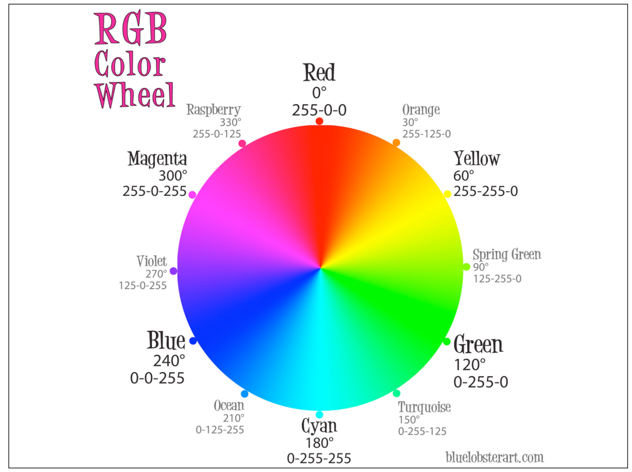
Think about black background

- Additive Mixing of 3 Lights



RGB Color Cube





Encoding Color Channels

- Could use 8-bits, spread across all 3 channels (a bit ugly…)

$$59_{16} = \underset{\text{R}}{\boxed{010}}\underset{\text{G}}{\boxed{110}}\underset{\text{B}}{\boxed{01}} = (2/7, 6/7, 1/3) = (0.286, 0.757, 0.333)$$

- The textbook outlines a collection of other methods. The most common? 8-bit RGB images (24-bits per pixel)

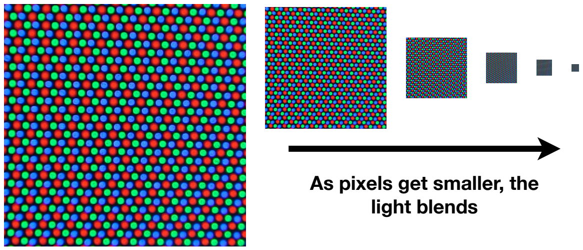
```
//separate channel encoding
let red_pixmap = new Uint8Array(ROWS*COLS);
let green_pixmap = new Uint8Array(ROWS*COLS);
let blue_pixmap = new Uint8Array(ROWS*COLS);

//all together, could use an 32-bit uint,
//by standard convention we have 4 channels
let rgb_pixmap = new Uint8Array(4*ROWS*COLS);

//access colors
let index = COLS*r + c;
rgb_pixmap[4*index + 0];  //red
rgb_pixmap[4*index + 1];  //green
rgb_pixmap[4*index + 2];  //blue
     Alpha channel is skipped
```

RGB Pixmap Encoding Options

How Do We **Display** Raster Images?

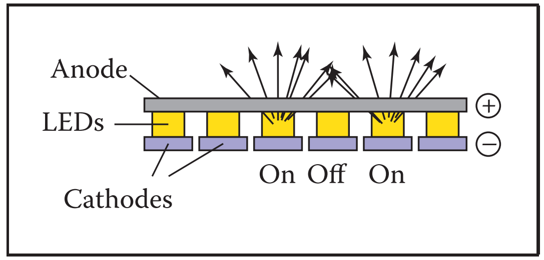
Optical Mixing

- To make (almost) any color, we combine light from three channels, Red, Green, Blue

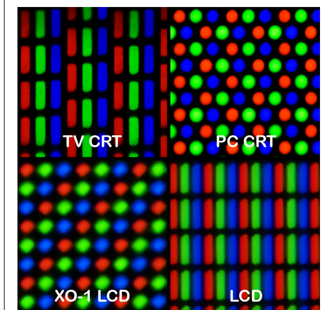


As pixels get smaller, the light blends

LEDs (Light-Emitting Diodes)



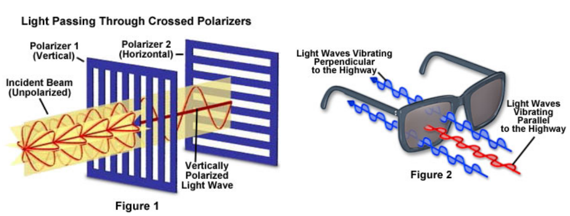
LCD Technology



- LCDs or **L**iquid **C**rystal **D**isplays produce color by selectively blocking light through different filters
- Pixels are organized in various units of 3

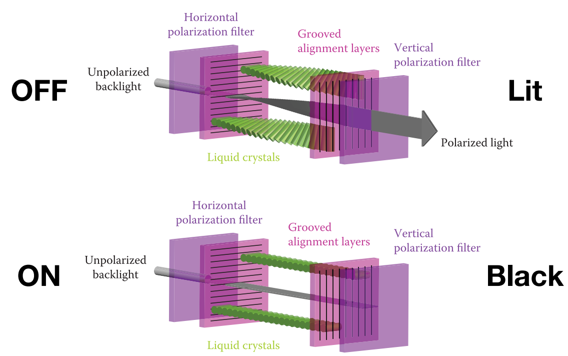
Polarized Light

- Light oscillates as a wave in all directions perpendicular to its path. Polarizers selectively block certain oscillations



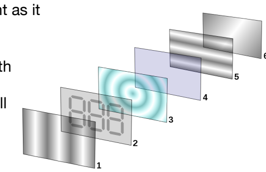
http://www.olympusmicro.com/primer/lightandcolor/polarization.html

Twisted Nematics



LCD Technology

- Four basic layers (in twisted nematics displays):
- (1) Vertical filter film to polarize the light as it enters/exits.
- (2 – 4) Glass substrate sandwiched with electrodes. The activation of these electrodes will determine what light will penetrate via twisted nematic LCDS
- (5) Horizontal polarizer to filter light.
- (6) Reflective surface or light source to send light back to viewer.



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

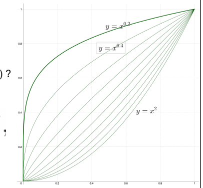
Sidebar: Framebuffers are used to prepare image data for the screen

- A **framebuffer** is an array of memory, large enough to store an image on the screen. Often implemented in hardware.
- A **lookup table** or LUT converts information from memory to actual color responses on the display.
- Uses:
  - Color correction, since display may not respond at the same levels as how the data represents it.
  - Simple example: Gamma corrections, brightness/contrast adjustment, etc.

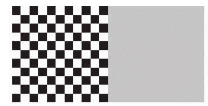
gamma-Correction

- 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 $(a')^{1/\gamma}$

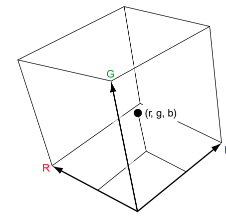


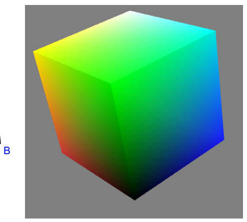
Chessboard of black/white pixels

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

RGB Color Space

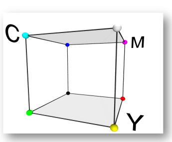
- Additive, useful for computer monitors
- Not perceptually uniform
  - For example, more "greens" than "yellows"



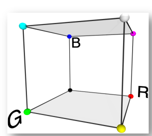


Converting from RGB to CMY

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



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



Converting from CMY to CMYK

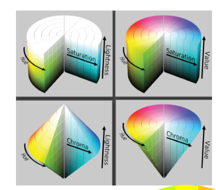
- Assuming CMY values are normalized (all channels between [0,1]), the exact same color in CMYK is

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

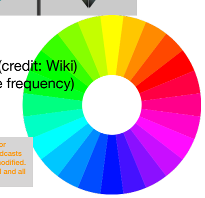
- K is a measure of the 'blackness' of the color and essentially serves as an offset after which the remaining amounts of cyan, magenta and yellow are 'added'

(H,C/S,L/B/V) 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
  - 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.

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, $I: R \rightarrow V$
- The domain is:
  - R, is some rectangular area ($R \subseteq \mathbb{R}^2$)
- The range is:
  - A set of possible values.
  - …in the space of color values we're encoding

Operations on Images

Slides inspired from Fredo Durand

- Point (Range) Operations:


- Affect only the range of the image (e.g. brightness)
- Each pixel is processed separately, only depending on the color

Operations on Images

Slides inspired from Fredo Durand

- Domain Operations:


- Only move the pixels around

Operations on Images

Slides inspired from Fredo Durand

- Neighborhood operations:

- Combine domain and range
- Each pixel evaluated by working with other pixels nearby

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.

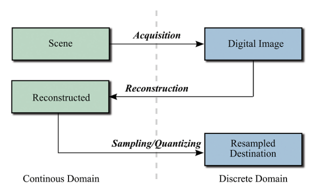


Figure 7.7. Resampling.

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];
   }
}
```

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?

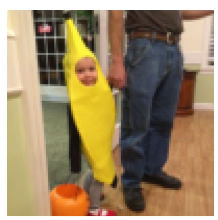
Naive Image Rescaling Code

```
//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

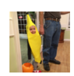
```
//scale factor
let k = 4;

//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

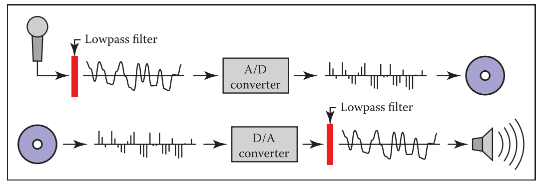
What's the Problem?

- 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**

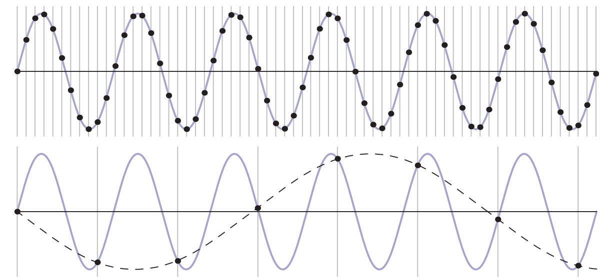
Sampling Artifacts / Aliasing

Motivation: Digital Audio

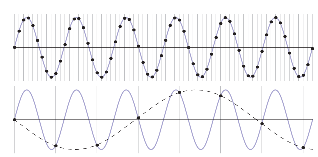
- Acquisition of images takes a continuous object and converts this signal to something digital
- Two types of artifacts:
  - **Undersampling** artifacts: on acquisition side
  - **Reconstruction** artifacts: when the samples are interpreted



Undersampling Artifacts



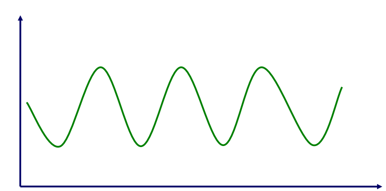
Shannon-Nyquist Theorem
(not needed for the exam)



- 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

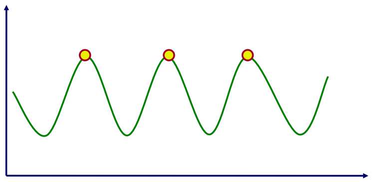
How many samples are enough to avoid aliasing?
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

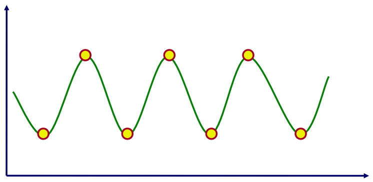
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

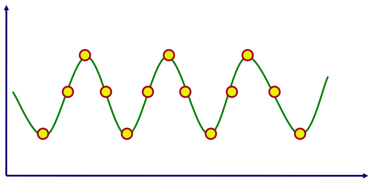
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



S-N Theorem Illustrated

How many samples are enough to avoid aliasing?

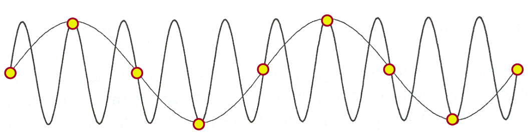
- How many samples are required to represent a given signal without loss of information?
- What signals can be reconstructed without loss for a given sampling rate?



Shannon-Nyquist Theorem

A signal can be reconstructed from its samples, iff the original signal has no content >= 1/2 the sampling frequency - Shannon

Aliasing will occur if the signal is under-sampled

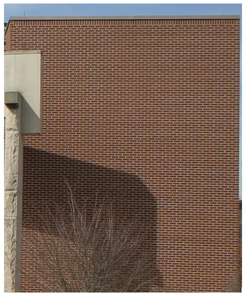


Aliasing in images

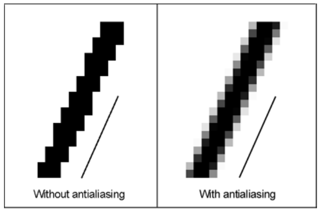
Two outcomes of under-sampling

1) Moire Pattern
2) Rasterization

Moire Patterns




Aliasing for edges

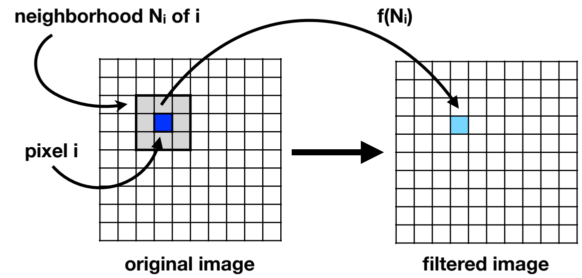


Each pixel is effected by nearby pixels
For example, even though the input image image is black/white,
We allow grey values for output pixels.

Convolution

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

Neighborhood Filtering (Schematic)



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, and set

$$f(N_j) = \frac{1}{|N_j|} \sum_{p_k \in N_j} C_k$$

- Where the N_i is a square, we call these **box** filters
- Think about it as a weighted average:

$$f(N_i) = \sum_{\text{pixel j in the rergion N i}} w_j C_j$$

- The weights $w_1 \ldots w_k$ are convex combination. Meaning that they are all positive, and $w_1 + w_2 + \ldots w_k = 1$. For example, $w_1 = w_2 = w_3 = \frac{1}{3}$

Box Filtering

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

$$1/9 * \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

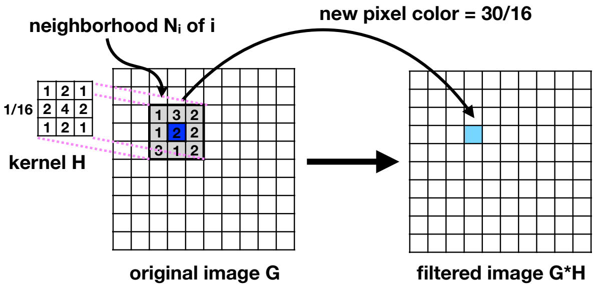
Box Filtering

$$1/9 * \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

$$1/25 * \begin{matrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{matrix}$$

Convolution

- This process of adding up pixels multiplied by various weights is called **convolution**



Kernels

- Convolution employs a rectangular grid of coefficients, 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

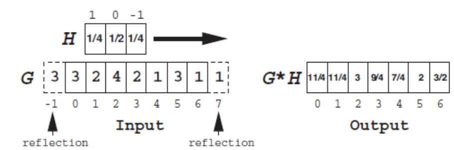
$$\frac{1}{9}\begin{pmatrix}1&1&1\\1&1&1\\1&1&1\end{pmatrix} \quad \frac{1}{13}\begin{pmatrix}1&1&1\\1&5&1\\1&1&1\end{pmatrix} \quad \frac{1}{37}\begin{pmatrix}1&1&1&1&1\\1&2&2&2&1\\1&2&5&2&1\\1&2&2&2&1\\1&1&1&1&1\end{pmatrix}$$

One-dimensional Convolution

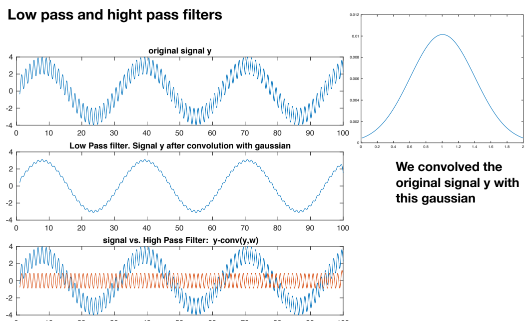
- Can be expressed by the following equation, which takes a filter H and convolves it with G:

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

- Equivalent to sliding a window

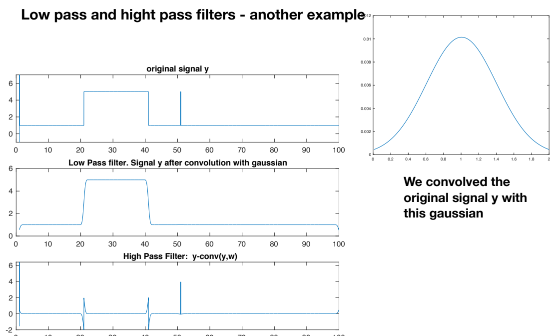


Low pass and hight pass filters



We convolved the original signal y with this gaussian

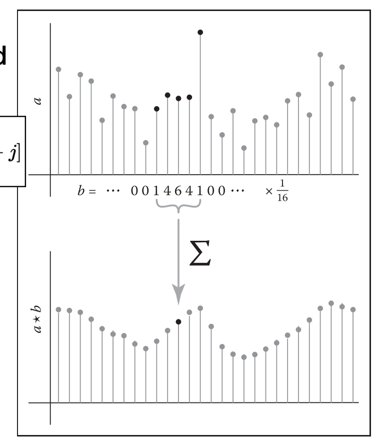
Low pass and hight pass filters - another example



We convolved the original signal y with this gaussian

Convolution is a Moving, Weighted Average

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



- Mathematically, this is equivalent to integrating the product of a and b with a shift in the domain
- Compare a to a*b on the right

2-Dimensional Version

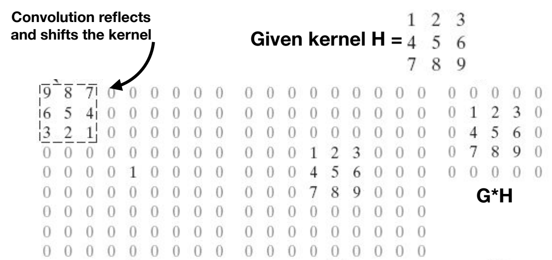
- Given an image a and a kernel b with (2r+1)² values, the convolution of a with b is given below as a*b:

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

- 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!!



An Illustration

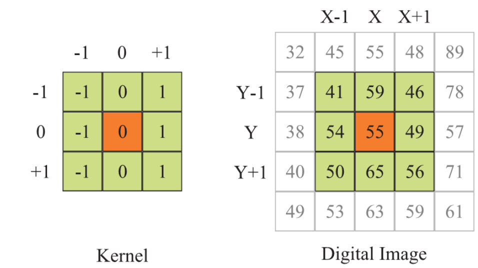


Figure 6.2. A 3 × 3 kernel is centered over sample $I(x, y)$.

An Illustration

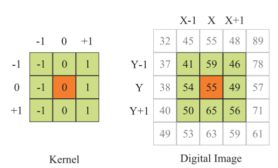


Figure 6.2. A 3 × 3 kernel is centered over sample $I(x, y)$.

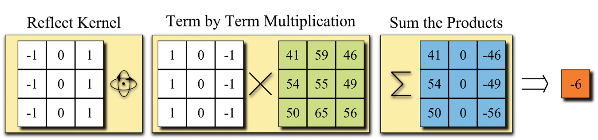
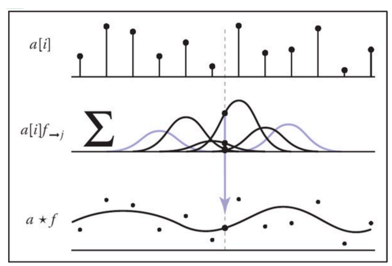


Figure 6.3. Convolution steps.

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



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

Filtering helps to reconstruct the signal better when rescaling

Inverse Rescaling

Reconstructed w/ Discrete-to-Continuous

Discrete-Continuous 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 x = col/k;
      let y = row/k;
      let index = row*k*W + col;
      output[index] = reconstruct(input,x,y);
   }
}
```

Types of Filters: Smoothing

Smoothing Spatial Filters

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

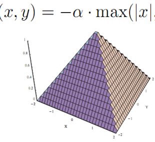
$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}\qquad\frac{1}{16}\times\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix}$$

- 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

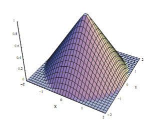
Smoothing Kernels

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

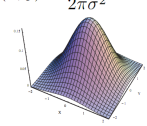
$$G(x,y) = \frac{1}{2\pi\sigma^2}e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$



(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

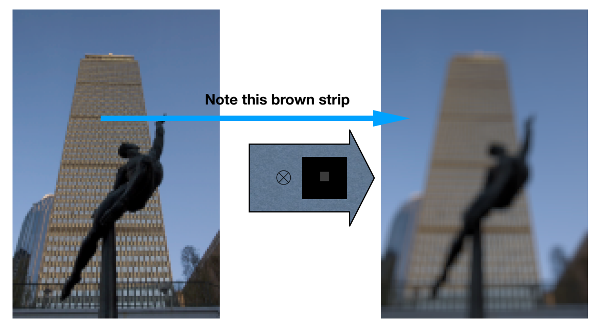
(b) Cone.

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

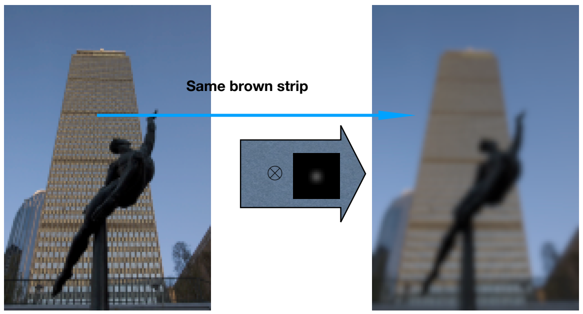
(c) Gaussian.

Table 6.1. Discretized kernels.

Box Filter



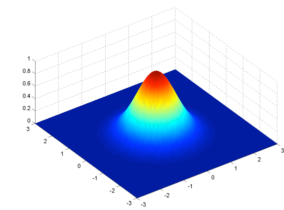
Gaussian Filter



Gaussians

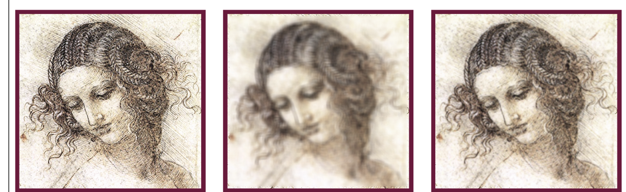
$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{-(x^2+y^2)}{2\sigma^2}}$$

- 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

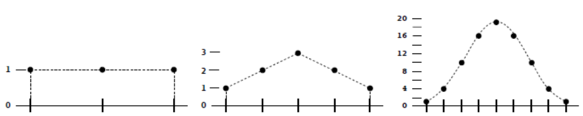


(a) Source image. (b) 17 × 17 Box. (c) 17 × 17 Gaussian.

Figure 6.10. Smoothing examples.

Smoothing the Smoothing Filters

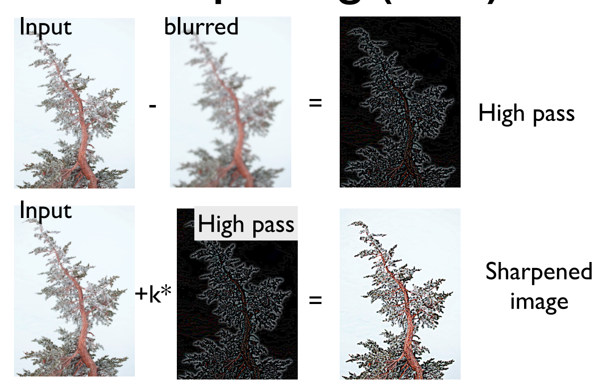
- Box * Box = Tent (Pyramid)
- Tent * Tent = Bell



a) box filter b) tent filter c) bell filter

Types of Filters: Sharpening

Sharpening (Idea)



Another example

Original Image, Imaged convolved

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

Sharpening is a Convolution

- This procedure can then expressed as a single kernel
- Assume that $I = I*d$ and $I_{low} = I*f_{g,\sigma}$.
  - d is the discrete identify function (kernel with 1 in center, 0 elsewhere)
  - $f_{g,\sigma}$ is a smoothing filter (e.g. Gaussian of width σ).
- This leads to:

$$\begin{aligned} I_{\text{sharp}} &= (1+\alpha)I - \alpha(I \star f_{g,\sigma}) \\ &= I \star ((1+\alpha)d - \alpha f_{g,\sigma}) \\ &= I \star f_{\text{sharp}}(\sigma, \alpha), \end{aligned}$$

Sharpening is a Convolution

$$\begin{aligned} I_{\text{sharp}} &= (1+\alpha)I - \alpha(I \star f_{g,\sigma}) \\ &= I \star ((1+\alpha)d - \alpha f_{g,\sigma}) \\ &= I \star f_{\text{sharp}}(\sigma, \alpha), \end{aligned}$$

Note: could also define d as $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

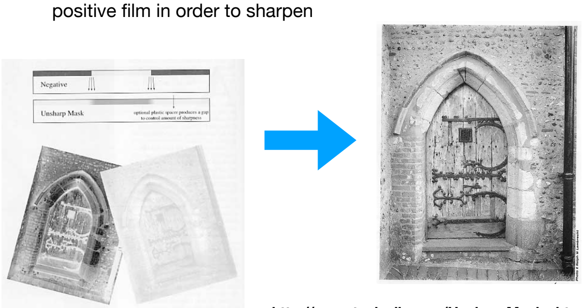
$$d = \frac{1}{9} \times \begin{bmatrix} 0 & 0 & 0 \\ 0 & 9 & 0 \\ 0 & 0 & 0 \end{bmatrix},$$

$$f_{g,\sigma} = \frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix},$$

$$((1+\alpha)d - \alpha f_{g,\sigma}) = \frac{1}{9} \times \begin{bmatrix} -\alpha & -\alpha & -\alpha \\ -\alpha & (9+8\alpha) & -\alpha \\ -\alpha & -\alpha & -\alpha \end{bmatrix}$$

Unsharp Masks

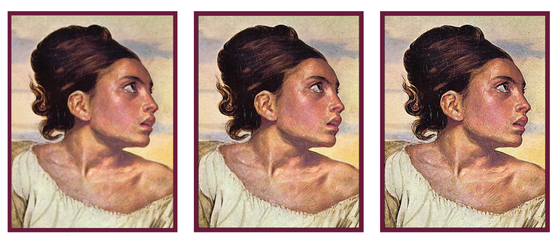
- Sharpening is often called "unsharp mask" because photographers used to sandwich a negative with a blurry positive film in order to sharpen



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

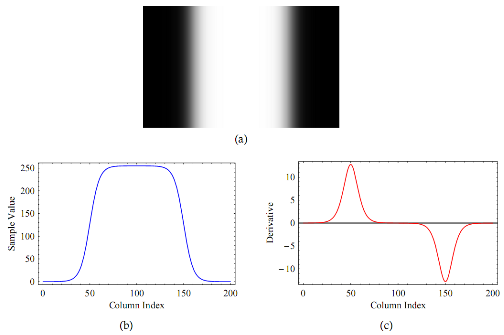


Figure 6.11. (a) A grayscale image with two edges, (b) row profile, and (c) first derivative.

(Review?) Derivatives via Finite Differences

- We can approximate the derivative with a kernel w:

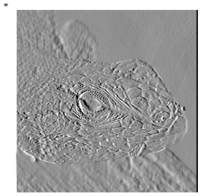
$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+h,y)-f(x-h,y)}{2h} \approx \frac{f(x+1,y)-f(x-1,y)}{2}$$

$$\frac{\partial f}{\partial x} \approx w_{dx} \circ f \qquad w_{dx} = \begin{array}{|c|c|c|}\hline -\frac{1}{2} & 0 & \frac{1}{2} \\ \hline\end{array}$$

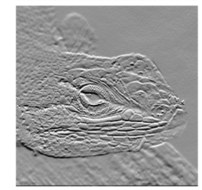
$$\frac{\partial f}{\partial y} \approx w_{dy} \circ f \qquad w_{dy} = \begin{array}{|c|}\hline -\frac{1}{2} \\ \hline 0 \\ \hline \frac{1}{2} \\ \hline\end{array}$$

Taking Derivatives with Convolution

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



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



Gradients with Finite Differences

- 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. $\nabla I = (G_x, G_y)$

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

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

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

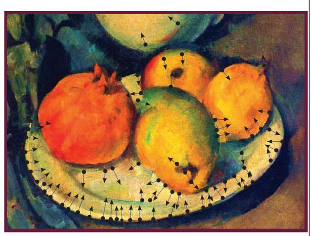


Figure 6.12. Image gradient (partial).

128	187	210	238	251
76	121	193	225	219
66	91	110	165	205
47	81	83	119	157
41	59	63	75	125

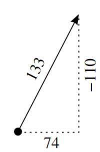
(a) Source Image.

	117	104	26	
	44	74	95	
	36	38	74	

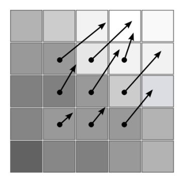
(b) $\delta I/\delta x$.

	−96	−100	−73	
	−40	−110	−106	
	−32	−47	−90	

(c) $\delta I/\delta y$.



(d) Center sample gradient.



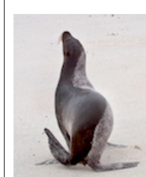
(e) Gradient.

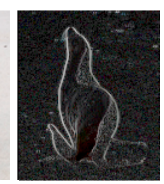
	151	144	77	
	59	133	142	
	48	60	117	

(f) Magnitude of gradient.

Figure 6.14. Numeric example of an image gradient.

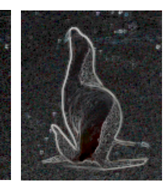
Gradients G_x, G_y





$|G_x|$

$|G_y|$



$|G|$

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

Effects of Rescaling

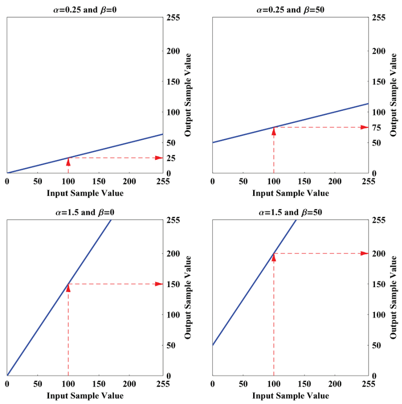


Figure 5.2. Graph of the linear scaling function with various gain and bias settings.

Why Use Both α, β?

- Consider two rescaled source samples of S rescaled to S'.

$$\begin{aligned} S'_1 &= \alpha S_1 + \beta, \\ S'_2 &= \alpha S_2 + \beta. \end{aligned}$$

- Calculate the **contrast** (the absolute difference) between the source and destination, called **Δ**S and **Δ**S'.

$$\begin{aligned} \triangle S' &= |S'_1 - S'_2|, \\ \triangle S &= |S_1 - S_2|. \end{aligned}$$

- Now consider the relative change in contrast between the source and destination.

$$\frac{\triangle S'}{\triangle S} = \frac{|S'_1 - S'_2|}{|S_1 - S_2|}.$$

Why Use Both α, β?

- The relative change in contrast can be simplified as

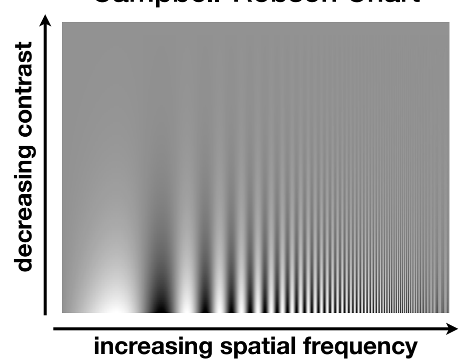
$$\begin{aligned} \frac{\triangle S'}{\triangle S} &= \frac{|(\alpha S_1 + \beta) - (\alpha S_2 + \beta)|}{|S_1 - S_2|} \\ &= \frac{|\alpha| \cdot |S_1 - S_2|}{|S_1 - S_2|} \\ &= |\alpha|. \end{aligned}$$

- Thus, gain (**α**) controls the change in contrast.
- Whereas bias (**β**) *does not* affect the contrast
- Bias, however, controls the final **brightness** of the rescaled image. Negative bias darkens and positive bias brightens the image

Sidebar: Relating Contrast Sensitivities to Signal Processing

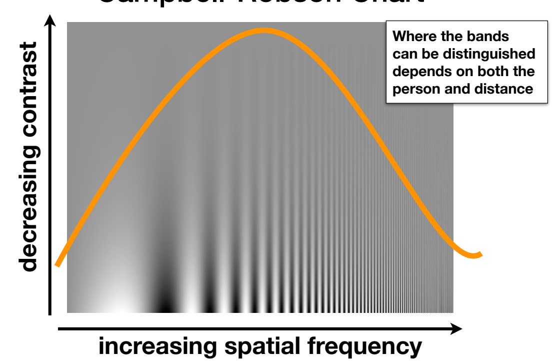
Contrast Sensitivity Function

Campbell-Robson Chart



Contrast Sensitivity Function

Campbell-Robson Chart



Contrast Sensitivities Vary by Channel

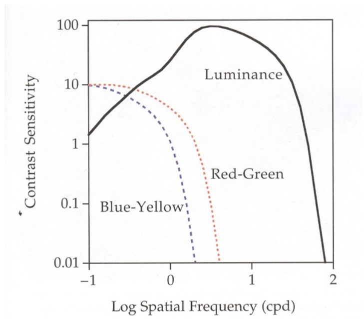
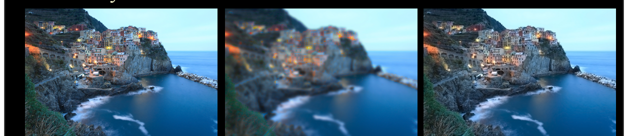


Figure 1-18. Spatial contrast sensitivity functions for luminance and chromatic contrast.

Photoshop demo

- Image > Mode > Lab color
- Go to channel panel, select Lightness
- Filter > Blur > Gaussian Blur , e.g. 4 pixel radius
  - very noticeable
- Undo, then select a & b channels
- Filter > Blur > Gaussian Blur , same radius
  - hardly visible effect



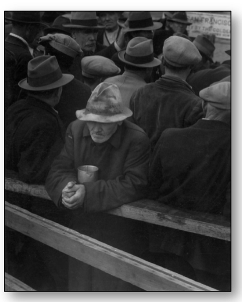
Original Blur Lightness Blur a & b

Important: Clamping

- Rescaling may produce samples that lie outside of the output images (e.g. below 0 or above 255 in 8-bit images)
- **Clamping** the output values ensures that the output samples are truncated to the 8-bit dynamic range limit
- Note that clamping does 'lose' information, since it truncates.

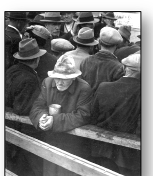
$$clamp(x, min, max) = \begin{cases} \text{min} & \text{if } \lfloor x \rceil \leq \text{min}, \\ \text{max} & \text{if } \lfloor x \rceil \geq \text{max}, \\ \lfloor x \rceil & \text{otherwise.} \end{cases}$$

Rescaling Examples



gain = 1, bias = 55

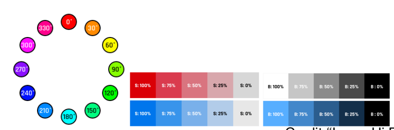
gain = 1, bias = -55



gain = 2, bias=0

gain = .5, bias=0

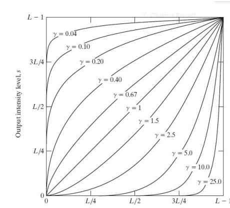
Rescaling Color Images

- Often, it is desirable to apply different gain and bias values to each channel of a color image separately
  - Example: A color image that utilizes the HSB (Hue-Saturation-Brightness) color model.
  - 

    Credit "Learn Ui Design Blog"
  - Since all color information is contained in the H and S channels, it may be useful to adjust ONLY the brightness, encoded in channel B, without altering the color of the image in any way.
- Rescaling the channels of a color image in a non-uniform manner is also possible rescaling each color channel separately.

Example: Gamma Correction

$$s = r^{\gamma}$$



a b
c d

FIGURE 3.9 (a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 3.0, 4.0,$ and 5.0, respectively. (Original image for this example courtesy of NASA.)

Putting it all together: Gain, Bias, and Gamma

- $C_{out} = (\alpha C_{in} + \beta)^{\gamma}$
- α is known as **gain** (exposure)
- β is known as **bias** (offset)
- γ maps to a non-linear curve (**gamma** correction)

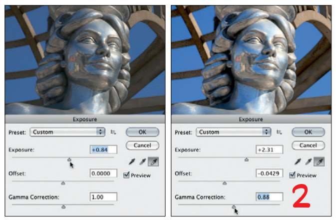
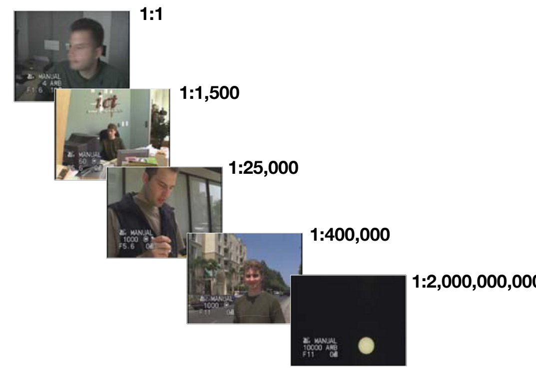


Image of Photoshop from Christian Bloch - The HDRI Handbook 2.0

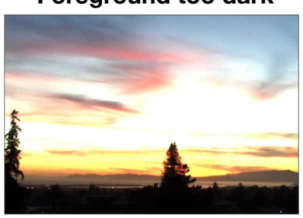
Dynamic Range

The World is a High Dynamic Range (HDR)

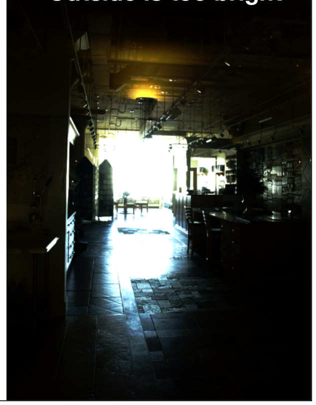


Examples

Sun overexposed
Foreground too dark



Inside is too dark
Outside is too bright

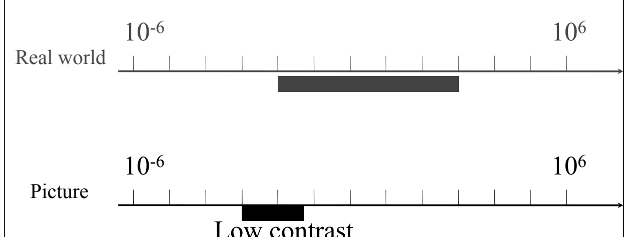


Dynamic Range in Displays?

- Range of pure black vs. pure white?

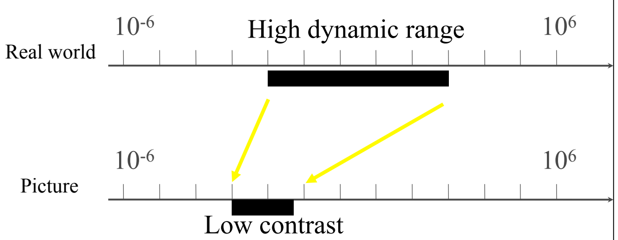
Dynamic Range in Displays?

- **Typically 1: 20 or 1:50**
  - –Black is ~ 50x darker than white
- **Max 1:500**



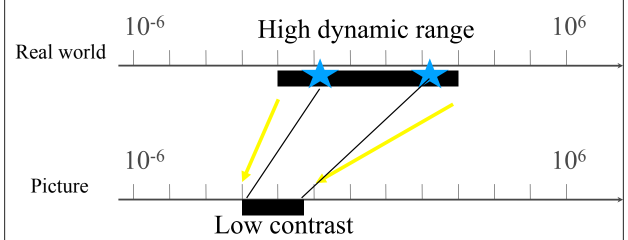
Problem: Displaying the Information

- Problem: How should we map scene radiances (up to 1:100,000) to display radiances (only around 1:100) to produce a satisfactory image?
- Goal: match limited contrast of the display medium while preserving details
- Solution: **Tone Mapping**

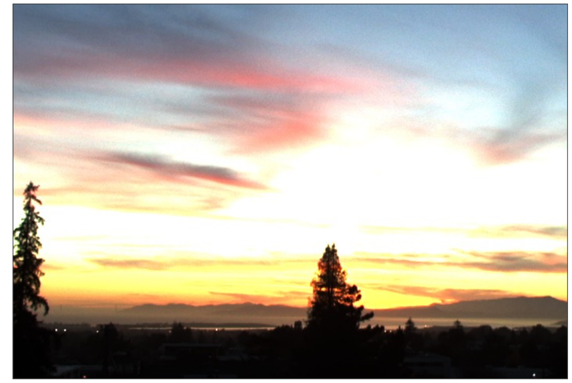


First solution: Linear mapping

- We will find the pixels with min and max intensity in the input image.
- Map them to the min and max intensities of the display
- Everything in between is mapped linearly.



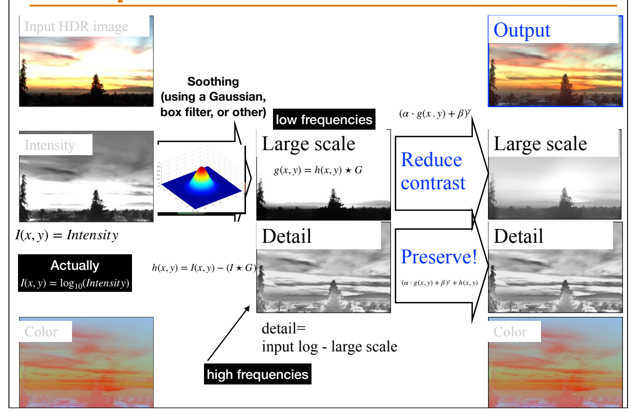
Without HDR + Tone Mapping



From Durand and Dorsey. No single global exposure can preserve both the colors of the sky and the details of the landscape, as shown on the rightmost images.

With HDR + Tone Mapping

Recap



Tone Mapping

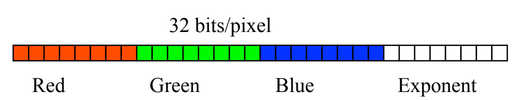
Question: But why do we need more than 100 levels of intensity (luminance) if in the input file we only have 256 values of intensities (RGB) ?

Answer: Not all file format has so few levels.

Even PPM could have 2 bytes per channel, so 256^2=65536 levels per channel.

Other formats gives much wider range:

Radiance RGBE Format (.hdr)



(145, 215, 87, 149) =

(145, 215, 87) * 2^(149-128) =

1190000 1760000 713000

(145, 215, 87, 103) =

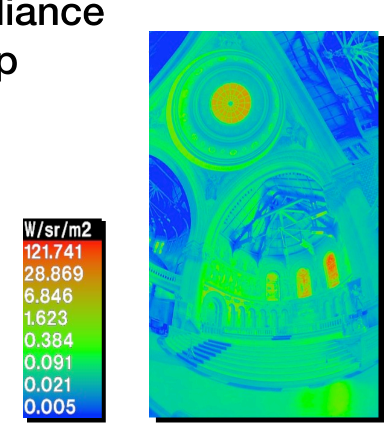
(145, 215, 87) * 2^(103-128) =

0.00000432 0.00000641 0.00000259

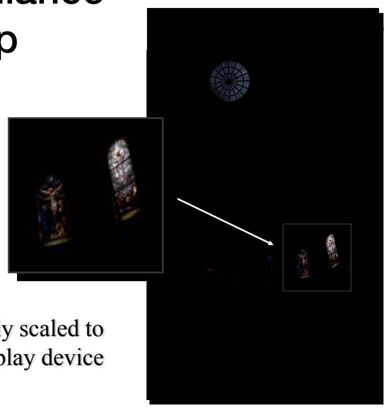
Ward, Greg. "Real Pixels," in Graphics Gems IV, edited by James Arvo, Academic Press, 1994

The Radiance Map

Radiance Definition: much of the power (in watts) is emitted by a cm^2 surface will be received by an optical system looking at that surface from a specified angle of view.

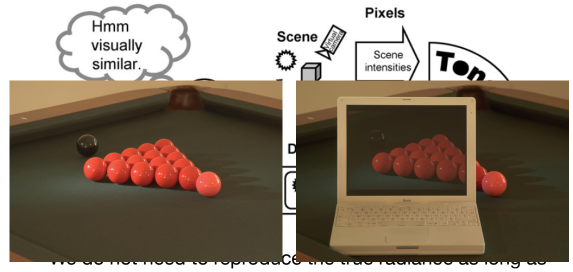


The Radiance Map



Linearly scaled to display device

Approach: Visual Matching



it gives us a visual match.

Eyes and Dynamic Range

- We're sensitive to change (multiplicative)
  - A ratio of 1:2 is perceived as the same contrast as a ratio of 100 to 200
  - Use the log domain as much as possible
- But, eyes are **not** photometers
  - Dynamic adaptation (very local in retina)
  - Different sensitivity to spatial frequencies

Headlights are ON in both photos

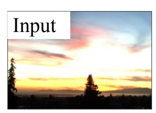
Can we just scale? Maybe!

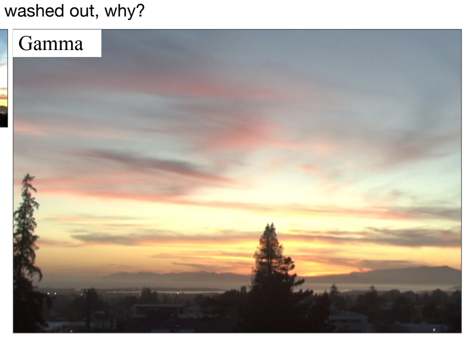
- For a color image, try to convert the input (world) luminance $\mathbf{L_w}$ to a target display luminance $\mathbf{L_d}$
- This type of scaling works (sometimes). In particular, it works best in the log and/or exponential domains
- $log_{10}(x)=1+log_{10}(y)$ means $x=10y$
- The base of the log is not important, as long as we are consistent in the mapping

$$\begin{bmatrix} R_d \\ G_d \\ B_d \end{bmatrix} = \begin{bmatrix} L_d \dfrac{R_w}{L_w} \\ L_d \dfrac{G_w}{L_w} \\ L_d \dfrac{B_w}{L_w} \end{bmatrix}$$

What scale value to use? How about Gamma compression

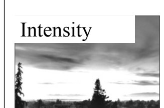
- $C_{out} = C_{in}^{\gamma}$, where $0 < \gamma < 1$ applied to each R,G,B channel
- Colors are washed out, why?

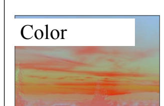


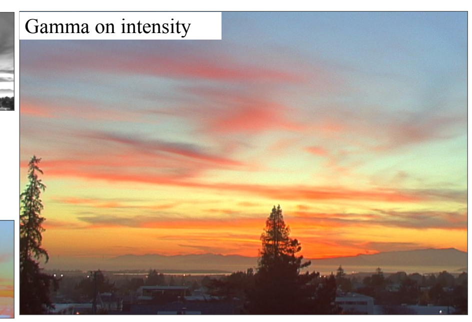


Gamma compression on Intensity

- Colors ok, but details in intensity are blurry





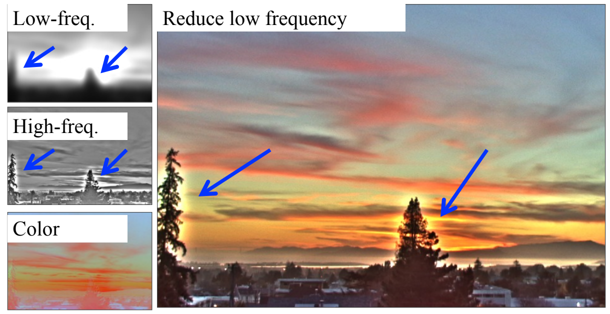


Oppenheim 1968, Chiu et al. 1993

- Reduce contrast of low-frequencies
- Keep mid and high frequencies

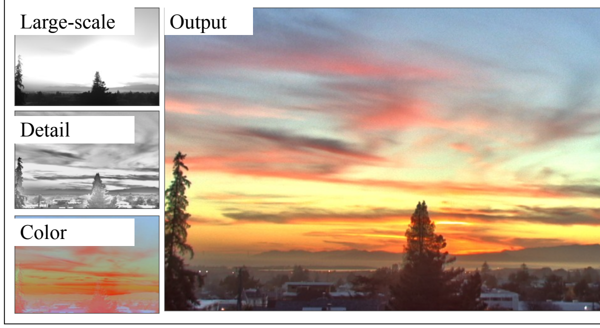
The halo nightmare

- For strong edges
- Because they contain high frequency



Our approach

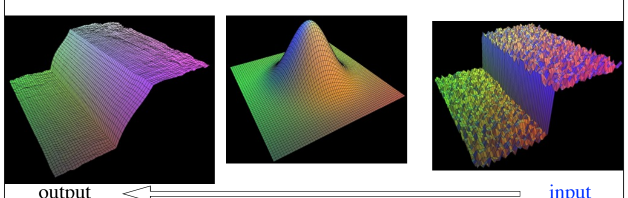
- Do not blur across edges
- Non-linear filtering



Start with Gaussian filtering

- Here, input is a step function + noise

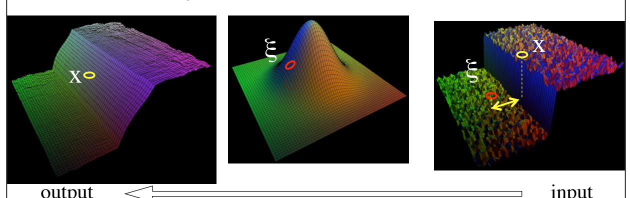
$$J \; = \; f \otimes I$$



Gaussian filter as weighted average

- Weight of ξ depends on distance to x

$$J(x) = \sum_{\xi} f(x,\xi) \; I(\xi)$$



The importance of convex combinations

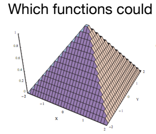
$$J(\mathbf{x}) = \sum \overbrace{f(\mathbf{x}, \xi)}^{\text{weight function}} \underbrace{I(\xi)}_{\text{Intensity}}$$

x is the point where we need the answer
ξ is nearby point

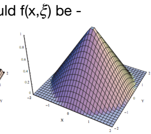
When we smooth, or interpolate we usually use wighted average.

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

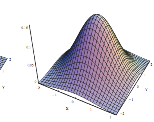
Which functions could f(x,ξ) be -



(a) Pyramid.



(b) Cone.



(c) Gaussian.

$$f(\mathbf{x}, \xi) = \max\left\{0, \frac{3}{\alpha^2} - \frac{3}{\alpha^3}\max(|x-\xi.x|, |y-\xi.y|)\right\}$$

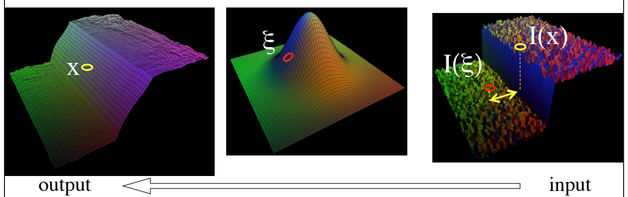
α is the **width** of the base of the pyramid. So in Fig(a), α=4

We will try to make sure that sum of weights =1 (this is called **convex combination**)
See example of bilinear interpolation on the whiteboard

The problem of edges

- **Here, $I(\xi)$ "pollutes" our estimate J(x)**
- **It is too different**
  - To correct it, we will have to change the averaging.
  - Remember that during the smoothing, we will first sum $\sum I(\xi)$ for all point ξ near ***x.***
  - To resolve the **halo problem**, We will avoid summing $I(\xi)$ if $I(\xi)$ is very far from $I(x)$

$$J(x) = \sum_{\xi} \quad f(x,\xi) \quad I(\xi)$$



Principle of Bilateral filtering

[Tomasi and Manduchi 1998]

- **Penalty g on the intensity difference**

Remember that the sum of weights must be 1.
What to do if we skip some terms ?
We will divide the total sum by k(x)

$$J(x) = \frac{1}{k(x)} \sum_{\xi} f(x,\xi) \quad g(I(\xi) - I(x)) \quad I(\xi)$$

$g(|a-b|) = 1$ if a is close to b, and zero otherwise

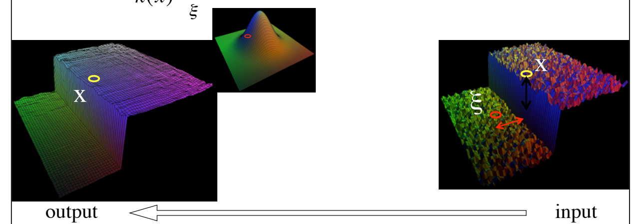
Bilateral filtering

[Tomasi and Manduchi 1998]

- **Spatial Gaussian f**

Remember that the sum of weights must be 1.
What to do if we skip some terms ?
We will divide the total sum by k(x)

$$J(x) = \frac{1}{k(x)} \sum_{\xi} f(x,\xi) g(I(\xi) - I(x)) \quad I(\xi)$$

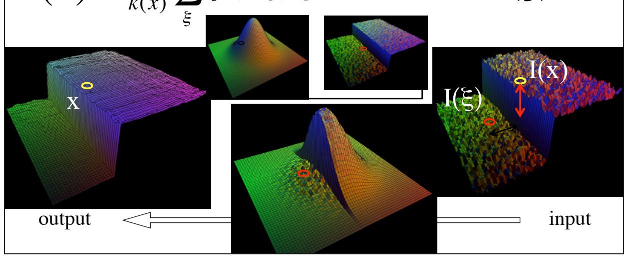


Bilateral filtering

[Tomasi and Manduchi 1998]

- **Spatial Gaussian f**
- **Gaussian g on the intensity difference**

$$J(x) = \frac{1}{k(x)} \sum_{\xi} f(x,\xi) \; g(I(\xi) - I(x)) I(\xi)$$



Normalization factor

[Tomasi and Manduchi 1998]

- **k(x)=** $\sum_{\xi} f(x,\xi) \quad g(I(\xi) - I(x))$

$$J(x) = \frac{1}{k(x)} \sum_{\xi} f(x,\xi) \quad g(I(\xi) - I(x)) \quad I(\xi)$$