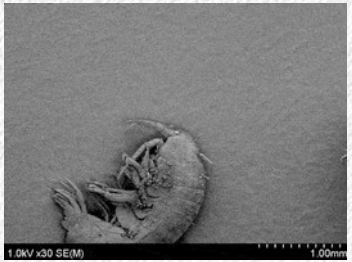
Representing an analog world on a digital computer

Computers think different

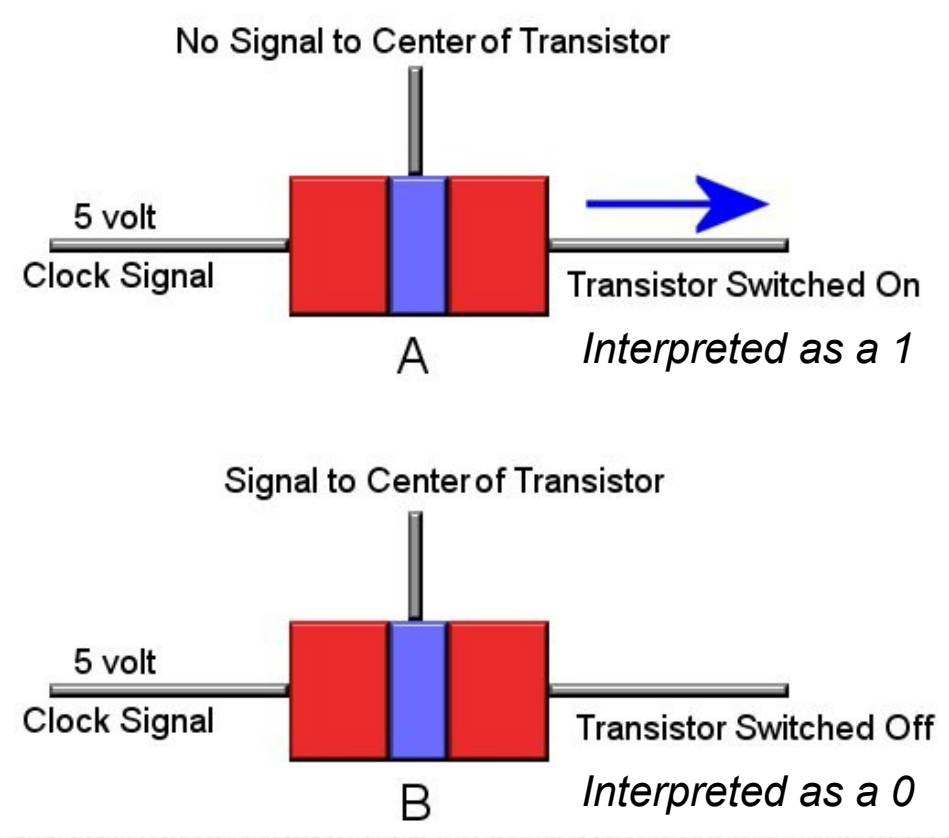
Computers are digital, the world is analog



- Analog = continuous quantities
  - Infinite resolution
- Digital = discrete values
  - Fixed resolution
- We live in an analog world
  - Continuous variables for physical dimensions
  - Shades of gray, gradients of color
  - Continuous gradations in pitch and intensity of sound
- To represent these analog traits on a computer, we need to convert them to a digital representation
- This can have important consequences to the accuracy of your data!

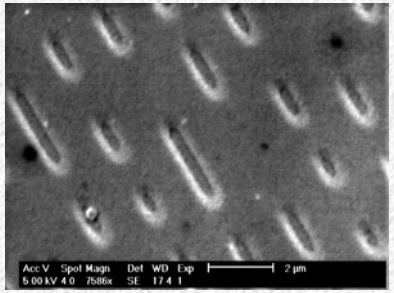
Computers only understand 0 and 1

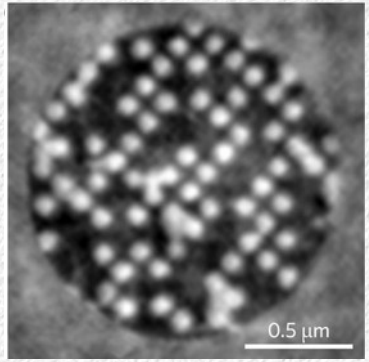
- Microprocessors are made up of many (i.e. billion +) transistors, which are like tiny switches
- Two possible states
  - On = interpreted as a 1
  - Off = interpreted as a 0
- A single transistor, set to a 0 or a 1, is called a "**bit**"
- Everything on a computer has to be represented by a series of bits



Encoding bits for storage

- Can't rely on constant electrical power for storage
- Pits in a surface – optical storage
  - CD-ROM
  - DVD
- Orientation of magnetic crystals – magnetic media
  - Hard drives
  - Magnetic tape
  - Flash media





Some analog traits

- Integer numbers (base 10)
- Continuous numbers
- Time
- Color
- Sound

- How do we represent these things in a computer that only understands 0 and 1?

Decimal integers (base 10)

- We're used to decimal numbering
  - Digits from 0 to 9
  - To represent numbers bigger than 9, we string digits together
  - Positions in a number are exponents of 10
  - Consider 1,324

1000's place	100's place	10's place	1's place
10^3	10^2	10^1	10^0
1 ,	3	2	4

  - 1,324 means we have 1 @ 10^3, 3 @ 10^2, 2 @ 10^1 and 4 @ 10^0
  - Add these together to get the decimal number

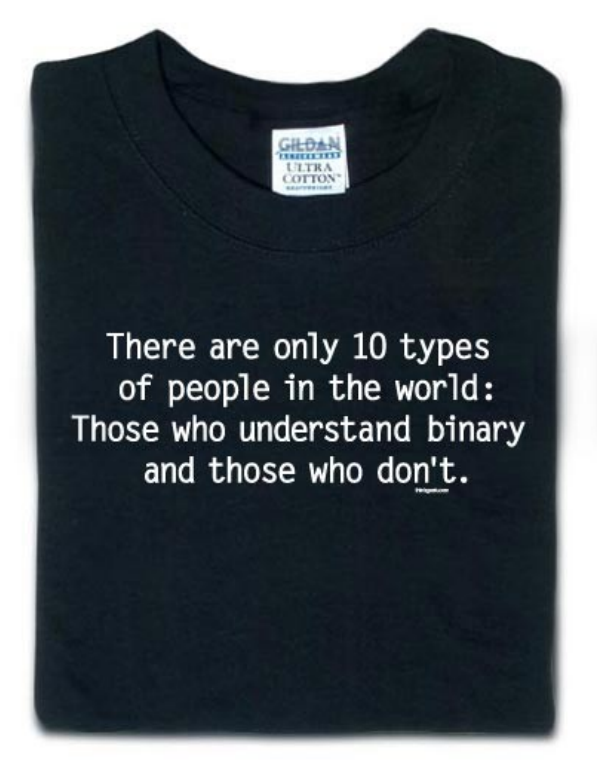
Binary numbers

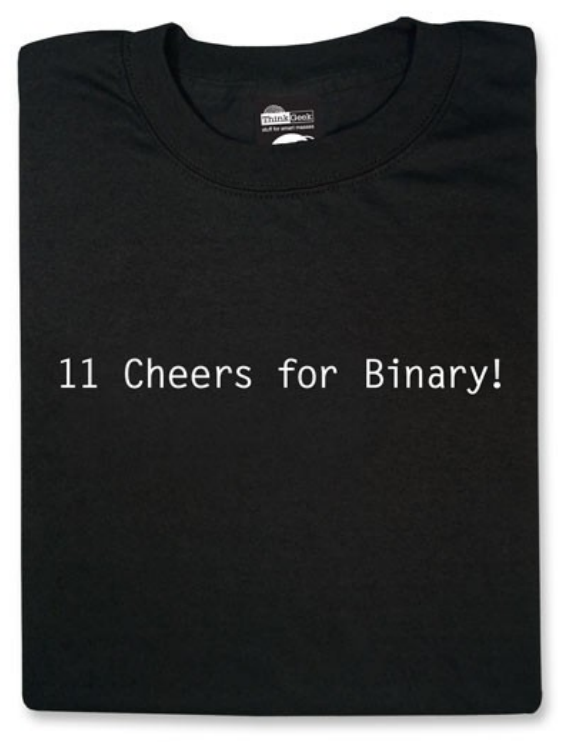
- Only two digits possible – 0 or 1
- To represent numbers bigger than 1, need to string digits together (just like with decimal numbers)
- With only 2 possible digits, each position is a power of 2
- What would 1000 be in binary?

8's place	4's place	2's place	1's place
2^3	2^2	2^1	2^0
1 ,	0	0	0

Plickers...

Convert a binary number to decimal





Plickers...

Largest decimal number you can represent with:

8 bits = 1 byte

1 byte can represent numbers from
0 to 255

We can devote one bit to the sign (+ or -), then we can represent numbers from
-128 to +127

Bits	Largest binary number	Equivalent decimal number
1	1	1 = 1
2	11	2+1 = 3
3	111	4+2+1 = 7
4	1111	8+4+2+1 = 15
5	11111	16+8+4+2+1 = 31
6	111111	32+16+8+4+2+1 = 63
7	1111111	64+32+16+8+4+2+1 = 127
8	11111111	128+64+32+16+8+4+2+1 = 255

Avoiding loss of digits

- Certain types of programs (e.g. databases), and programming languages require you to identify your variable type
- Need to pick a variable of the right kind, and with enough bytes, so that your numbers will fit
- If you don't use the right variable type numbers can be truncated (i.e. the answers will be wrong), or the program could crash (i.e. you don't get an answer at all)

Example – adding two 8 bit unsigned numbers

Addition of decimal numbers

$$\begin{array}{r} 63 \\ +255 \\ \hline 318 \end{array}$$

Addition of binary numbers

$$\begin{array}{r} 00111111 \\ +11111111 \\ \hline 100111110 \end{array}$$

Too big for one byte, so the red 1 would be dropped

Without it, the number 00111110 is 62, not 318 – wrong answer!

Short and long integers

- Typically, integers are either:
  - Short = 16 bit
    - From -32,768 to 32,767
  - Long = 32 bit
    - From -2,147,483,648 to 2,147,483,647
- Why not always use long? Why not use 64 bit (numbers into the quintillions)?

Continuous numbers

- Continuous numbers can take any value on the number line – fractions of digits
- Precision of continuous numbers is arbitrary = infinite precision
  - Can record them to whatever level we want
- Have to pick a level to represent the number on a computer
- Fixed precision = pick how many digits before and after a decimal place
  - xx.xxx = two digits before, three after the decimal place
  - Numbers bigger than 99.999 or smaller than 0.001 can’t be represented
- To avoid this, continuous numbers are represented as **floating point**

Floating point numbers

- Easiest to understand in comparison with fixed point decimal numbers
  - Fixed point decimals are explicit about the location of the decimal
  - If there are 5 sig figs, three after the decimal and two before, then a number smaller than 0.001 or larger than 99.999 cannot be represented accurately
- Floating point numbers divide the decimal number into a mantissa (the recorded, significant digits) and an exponent
- Because of this, 12345, 1.2345, 12345000, 0.00012345 can all be represented with the same floating-point structure
  - 12345×10^{0}, 12345×10^{-4}, 12345×10^{3}, 12345×10^{-8}

The structure of floating point numbers

- A number stored as floating point **double precision** has:
  - 1 sign bit (0 for positive, 1 for negative)
  - 11 exponent bits (representing either positive or negative exponents)
  - 52 bits for the mantissa
  - 64 bits total used
- Spreadsheets use double-precision floating point decimals for *all numbers*, and store 15 significant (decimal) digits
  - Numbers that appear to be integers have been rounded for display

Problems converting fractions to decimals

- Converting fractions to decimals can be problematic even in base 10 numbers
  - Some fractions can be represented perfectly with a decimal number, others can't
  - We're used to this with base 10, but it can surprise us in base 2

$$\frac{1}{5} = 0.2$$

$$\frac{1}{6} \neq 0.16666667$$

- To see the problem, we'll convert some numbers from fractions to decimals using a method that works in both base 10 and base 2

Fractions as decimals – base 10

- Places are 10^{-1} (0.1), 10^{-2} (0.01), 10^{-3} (0.001) and so on
- Calculate one digit at a time
- The process to convert a fraction to a decimal in base 10 is:
  - Multiply the fraction by the base
  - Calculate the integer and remainder for result
  - The integer is the digit for that place
  - Repeat the process with the remainder to find the next digit, until remainder is 0, or you run out of significant digits

1/5 as a decimal

First digit is 2

$$\frac{1}{5}\times 10=\frac{10}{5}=2\frac{0}{5}$$

Remainder is 0, so decimal is 0.2
0.2 is *exactly* 1/5

Not all fractions can be exactly represented as decimals

- Fractions with infinitely repeating digits can only be approximated as decimal numbers
- To use the decimal, we have to round it at a given decimal place
- The amount of error due to this depends on the point at which we do the rounding
- No biggie, we're used to this problem

1/6 as a decimal

First digit is 1

$$\frac{1}{6}\times 10=\frac{10}{6}=1\frac{4}{6}$$

Second digit is 6

$$\frac{4}{6}\times 10=\frac{40}{6}=6\frac{4}{6}$$

Third digit is 6

$$\frac{4}{6}\times 10=\frac{40}{6}=6\frac{4}{6}$$

All subsequent digits are 6, decimal is $0.16\overline{6}$

0.167 is *approximately* 1/6
0.1666666666667 is *closer* to 1/6, but
0.167 ≠ 0.1666666666667 ≠ 1/6

Converting a fraction to a binary decimal

- Places are 2^{-1} (0.5), 2^{-2} (0.25), 2^{-3} (0.125) and so on
- Same process as with base 10, but multiply by the base of 2
- 1/5 in binary is a repeating decimal, no longer exact
- This causes some floating point weirdness

1/5 as a "binary decimal"

First digit is 0 $\frac{1}{5}\times 2=\frac{2}{5}=0\frac{2}{5}$

Second digit is 0 $\frac{2}{5}\times 2=\frac{4}{5}=0\frac{4}{5}$

Third digit is 1 $\frac{4}{5}\times 2=\frac{8}{5}=1\frac{3}{5}$

Fourth digit is 1 $\frac{3}{5}\times 2=\frac{6}{5}=1\frac{1}{5}$

Fifth digit is 0 $\frac{1}{5}\times 2=\frac{2}{5}=0\frac{2}{5}$

Repeat

Binary decimal is $0.\overline{0011}$, so 1/5 can only be approximated by a binary decimal (just like 1/6 can only be approximated by a base 10 decimal)

First a quick quiz (no calculators!):

$$1\times\left(0.5-0.4-0.1\right)=?$$

$$\begin{array}{r} 1.0000000000000010 \\ +0.0000000000000001 \\ \hline ? \end{array} \qquad \begin{array}{r} 1.0000000000000010 \\ -0.0000000000000001 \\ \hline ? \end{array}$$

Examples: floating point weirdness

A = 1*(0.5-0.4-0.1)	B = 0
-2.77556E-17	0
Is A = B?	
FALSE	

A =	1.0000000000001000
B =	0.0000000000000100
A+B =	1.0000000000001000
A-B =	1.0000000000001000

Consequence of floating-point representation in a computer

- If you are working at the limits of your computer's precision, beware
  - Best to pick units of measure that will not put you near these limits
- Some numbers that can be exactly expressed as decimals in base 10 aren’t exact in base 2
  - Even simple calculations can give odd results, so watch for problems!

Letters and symbols in binary

- ASCII = American Standard Code for Information Interchange
  - This is sometimes called “plain text” - no formatting instructions included
- Numeric codes are assigned to 128 specified characters
  - First 32 are control characters (things like carriage returns) – used to control the hardware
  - 33 to 47 are punctuation
  - 48 to 57 are the numbers 0 to 9
  - Separate codes for capital and lower case letters
- Programs that understand ASCII interpret these numeric codes as letters, numbers, and punctuation

Some ASCII symbols and numbers

DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
32	040	20	00100000		&#32;		Space
33	041	21	00100001	!	&#33;		Exclamation mark
34	042	22	00100010	"	&#34;	&quot;	Double quotes (or speech marks)
35	043	23	00100011	#	&#35;		Number
36	044	24	00100100	$	&#36;		Dollar
37	045	25	00100101	%	&#37;		Procenttecken
38	046	26	00100110	&	&#38;	&amp;	Ampersand
39	047	27	00100111	'	&#39;		Single quote
40	050	28	00101000	(	&#40;		Open parenthesis (or open bracket)
41	051	29	00101001	)	&#41;		Close parenthesis (or close bracket)
42	052	2A	00101010	*	&#42;		Asterisk
43	053	2B	00101011	+	&#43;		Plus
44	054	2C	00101100	,	&#44;		Comma
45	055	2D	00101101	-	&#45;		Hyphen
48	060	30	00110000	0	&#48;		Zero
49	061	31	00110001	1	&#49;		One
50	062	32	00110010	2	&#50;		Two
51	063	33	00110011	3	&#51;		Three
52	064	34	00110100	4	&#52;		Four
53	065	35	00110101	5	&#53;		Five
54	066	36	00110110	6	&#54;		Six
55	067	37	00110111	7	&#55;		Seven
56	070	38	00111000	8	&#56;		Eight
57	071	39	00111001	9	&#57;		Nine

ASCII upper and lower case letters

DEC	OCT	HEX	BIN	Symbol	HTML Number	HTML Name	Description
32	040	20	00100000				Space
65	101	41	01000001	A	A		Uppercase A
66	102	42	01000010	B	B		Uppercase B
67	103	43	01000011	C	C		Uppercase C
68	104	44	01000100	D	D		Uppercase D
69	105	45	01000101	E	E		Uppercase E
70	106	46	01000110	F	F		Uppercase F
71	107	47	01000111	G	G		Uppercase G
72	110	48	01001000	H	H		Uppercase H
73	111	49	01001001	I	I		Uppercase I
74	112	4A	01001010	J	J		Uppercase J
97	141	61	01100001	a	a		Lowercase a
98	142	62	01100010	b	b		Lowercase b
99	143	63	01100011	c	c		Lowercase c
100	144	64	01100100	d	d		Lowercase d
101	145	65	01100101	e	e		Lowercase e
102	146	66	01100110	f	f		Lowercase f
103	147	67	01100111	g	g		Lowercase g
104	150	68	01101000	h	h		Lowercase h
105	151	69	01101001	i	i		Lowercase i
106	152	6A	01101010	j	j		Lowercase j

Spelling “cat” in ASCII

Letter	C	A	T
ASCII Decimal code:	67	65	84
Binary number:	01000011	01000001	01010100

Letter	C	a	t
ASCII Decimal code:	67	97	116
Binary number:	01000011	01100001	01110100

Letter	c	a	t
ASCII Decimal code:	99	97	116
Binary number:	01100011	01100001	01110100

Representing time in a computer

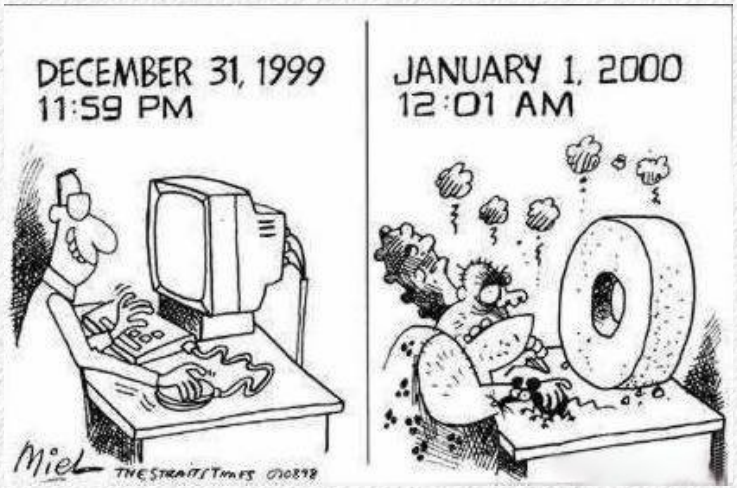
- Time passes continuously
  - We can measure it to an arbitrary level of precision
  - Currently the smallest time unit we can measure is the **attosecond** (10^{-18} s)
  - Very small - an attosecond is to a second what a second is to 31.71 billion years
- All clocks can only measure time to a fixed resolution – computers are no different
  - Computers measure time by counting “ticks”
    - Ticks are 100 nanoseconds = 1×10^{-7} seconds
  - “System time” is the number of ticks that have elapsed since a selected starting time point, called the “epoch”

System time on a computer

- Different computer operating systems use different epochs
  - UNIX-derived systems (Linux, BSD) use 1 January 1970 00:00:00 UT (where UT is time taken at the prime meridian) for the epoch
  - Apple's Mac OS X uses 1 January 2001 00:00:00 UT
  - Windows NT and later Windows OS used 1 January 1601 00:00:00 UT
- Number of ticks before or after the epoch converted to time
  - Positive numbers of ticks are times after the epoch, negative numbers would be times previous to the epoch
- Resolution recorded is usually to the nearest millisecond
- Timekeeping by computers can drift over time, but can synchronize with atomic clock signals, or "time servers" on their network that keep even more accurate time

Problems representing time on computers

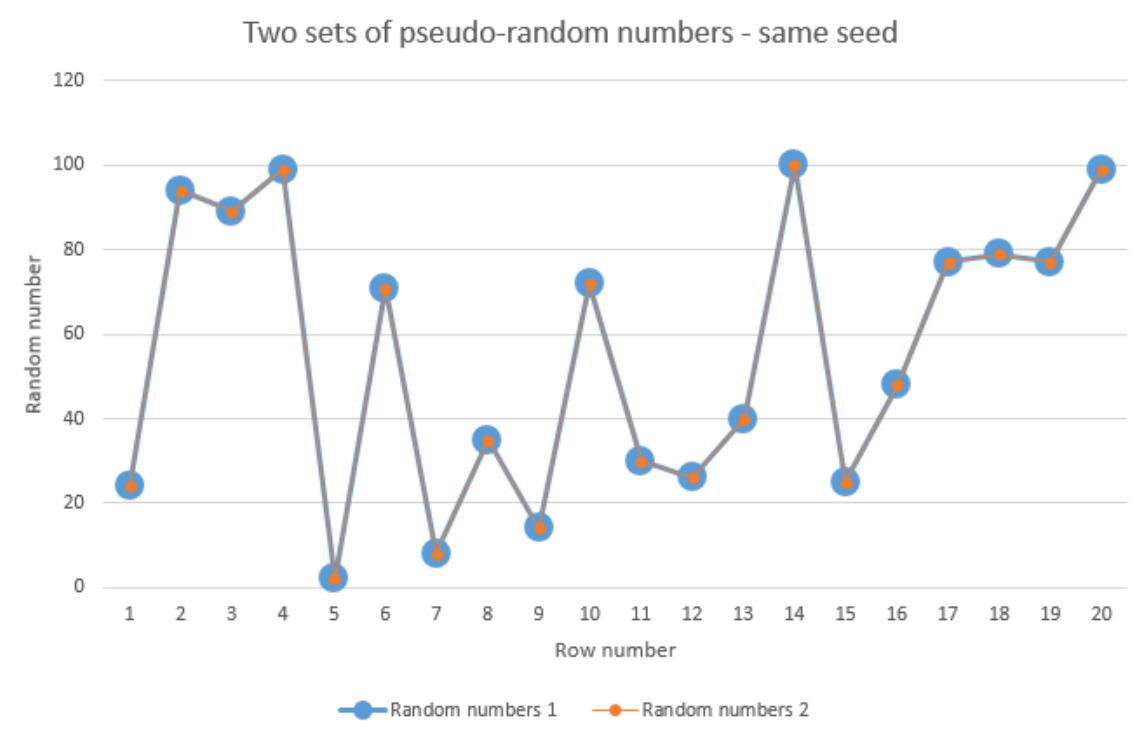
- The Millennium Bug
  - Programs that stored dates with only two numbers, and always expected years to increase, were expected to crash when the year went from 99 to 00
  - Solution – reprogram to use four digit years (what will those year 9,999 people think of us?)
- The Unix 2038 problem
  - The Unix epoch is 1/1/1970 00:00:00 UTC
  - If time is stored as a 32-bit signed integer, the largest number of ticks that can be recorded is $2^{31} - 1$
  - This maximum will be reached in 2038
  - Solution: reprogram to use 64-bit integers for times

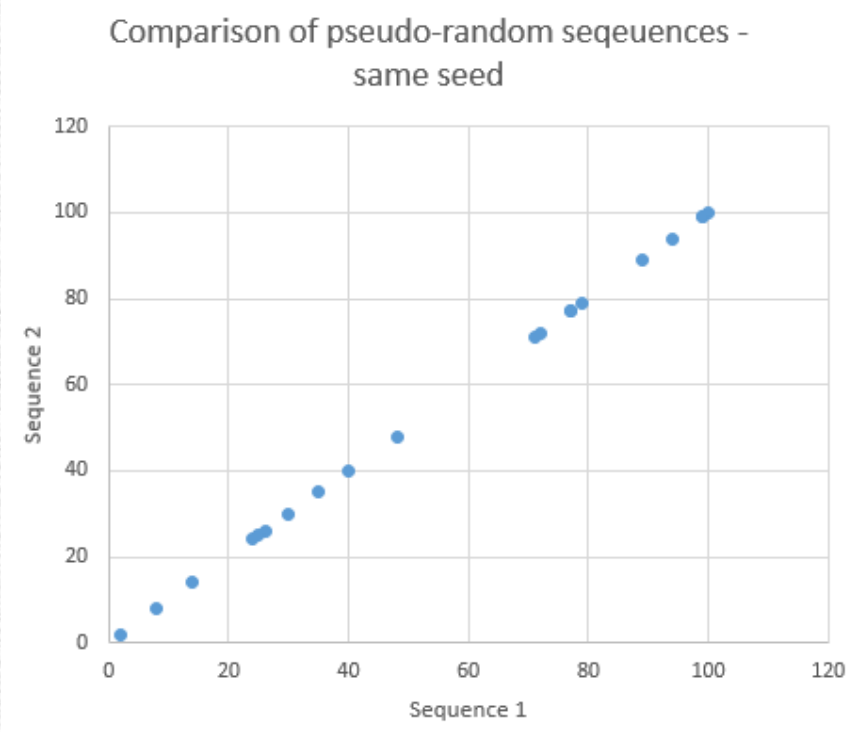


Randomness

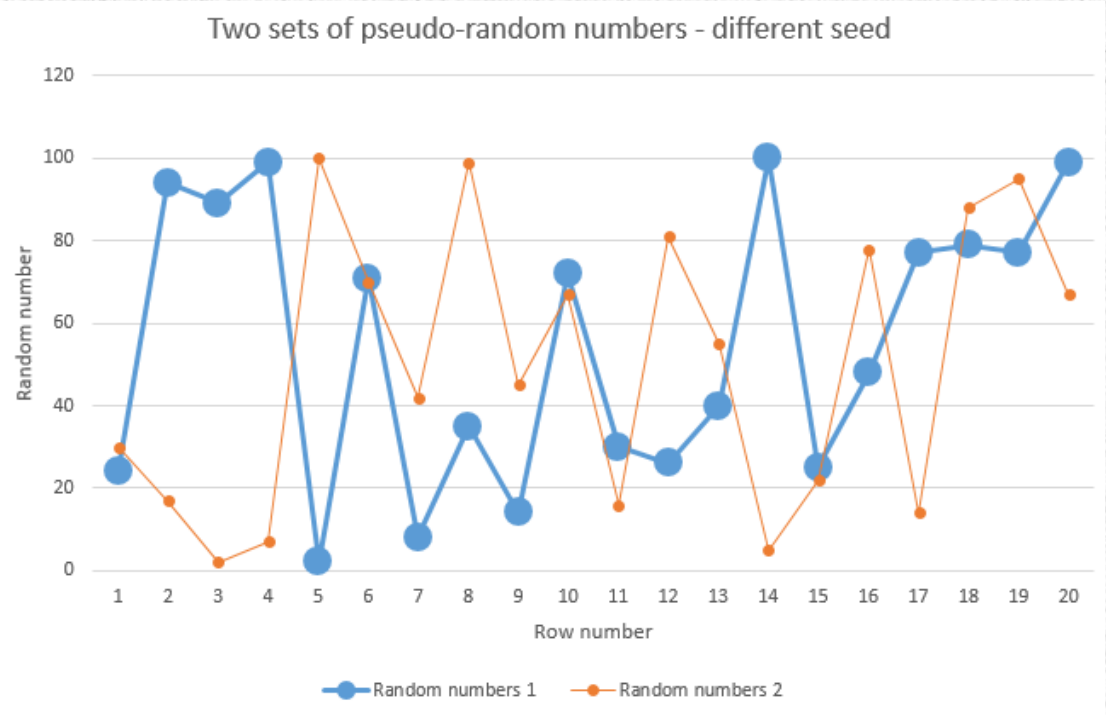
- Unless they break, computers are not capable of being spontaneous
- Can’t generate random numbers – but, can be programmed to produce pseudo-random numbers
  - Algorithms that produce unpatterned sequences of numbers
  - Rely on a seed – a starting number specified by the programmer
  - Given the same seed, the same set of pseudo-random numbers are produced
  - If you don’t know the seed or the algorithm the pseudo-random numbers can be used as though they are random
  - If you change the seed a different set of pseudo-random numbers are produced
  - System time is often used as the seed so that different sets are always produced

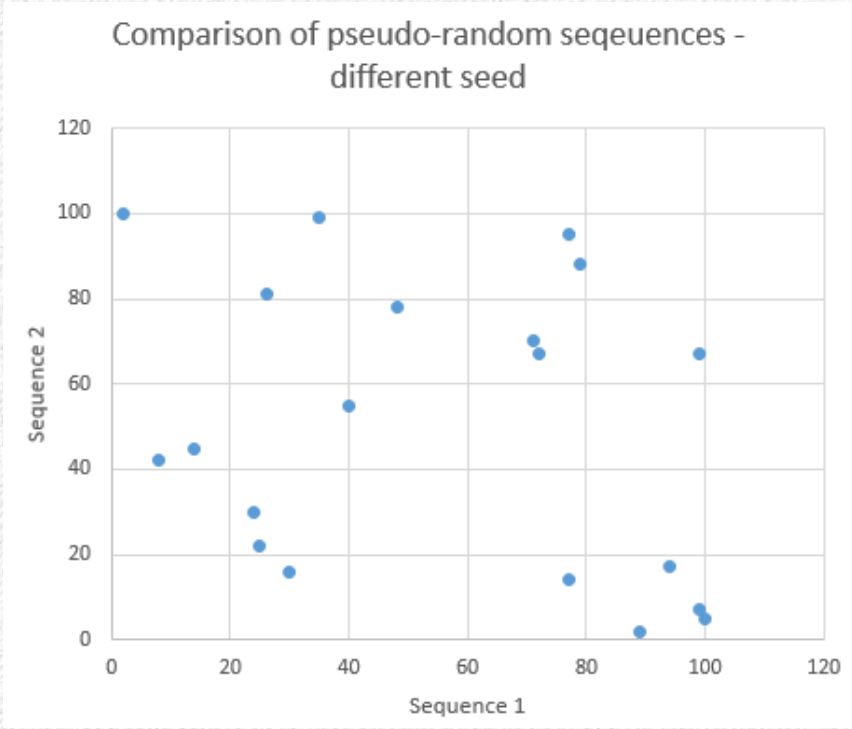
Two pseudo-random sequences with the same seed





Change seed for sequence 2





Images

Consider this picture

What would it look like if we only used 1 bit to represent it?

1 bit image

With 1 bit the only options are 0 (black) and 1 (white)

Anything above a threshold level of lightness will be assigned to 1, and will appear white

Anything below the threshold will be assigned a 0, and will appear black

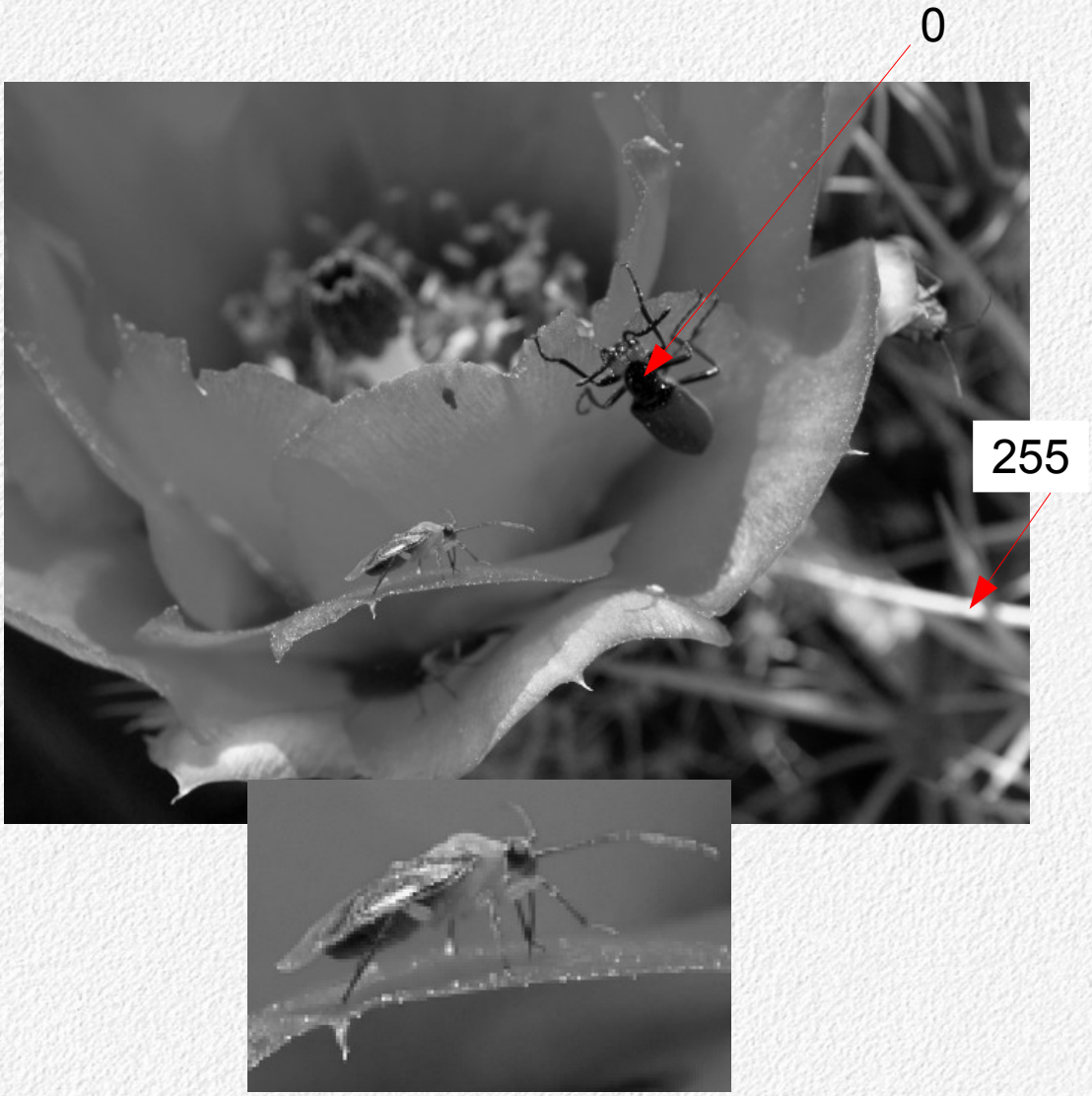
Grayscale:
1 channel with 8 bits

8 bits (0 to 255) for a single channel

Interpreted as 256 shades of gray

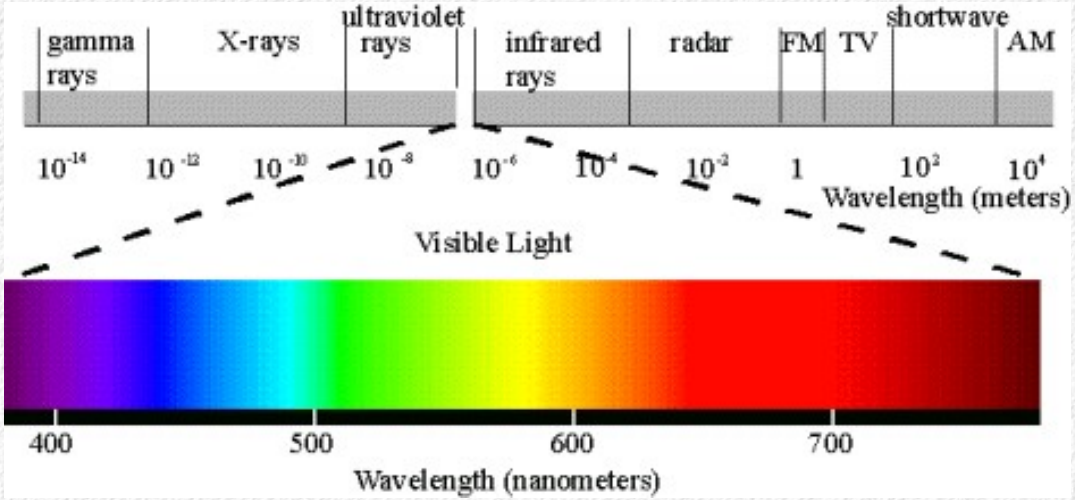
Sufficient number for the transition between shades of gray to appear smooth

But, there are more than 256 possible shades of gray, so some that are very similar get lumped together into one – some loss of detail in the image compared to the actual, analog object



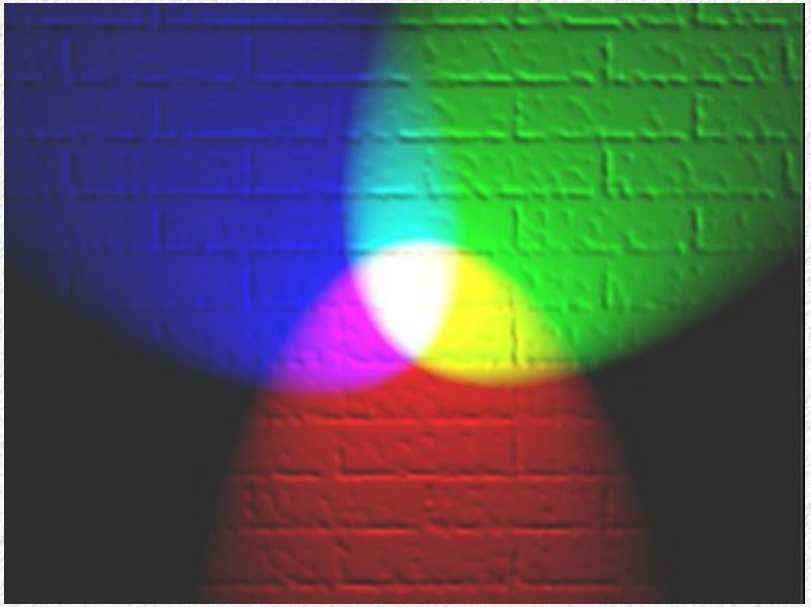
Representing color on a computer

- Color is determined by the wavelength of visible light – continuous, analog quantities
- Colors can be represented in computers using an appropriate color model



The RGB color model

- All colors are represented as combinations of levels of intensity of red, green, and blue light
  - R, G, and B are the primary colors for light
  - Primary colors for pigments are red, yellow, and blue
- Computer displays have a color channel for each primary color
  - Can emit each color channel at different intensities
  - Mixes of intensities of the three primary colors of light produce a large number of different colors on the screen



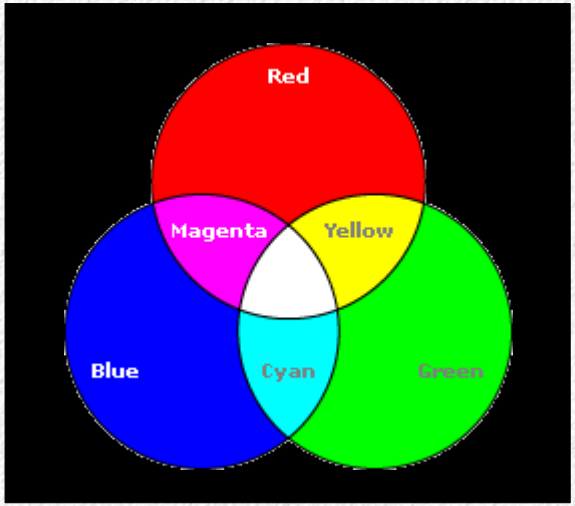
Color mixing

Light

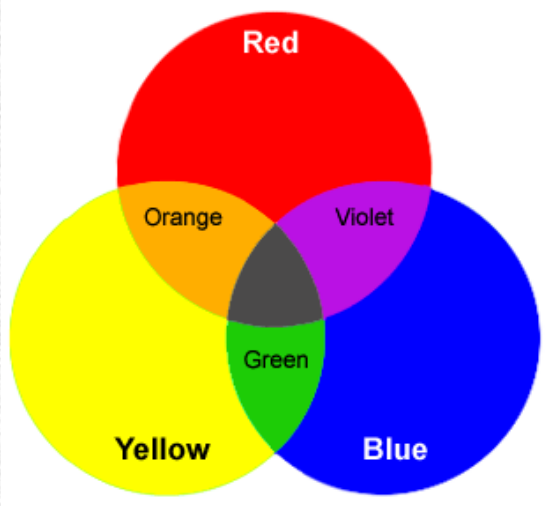
Primary colors of light add together to make other colors

Lack of all colors of light = black

Presence of all colors = white



Pigments



Light is absorbed by pigments, what you see is what is reflected

Lack of all colors of pigment = white
Presence of all colors of pigment = black

Plickers...

24-bit color: an 8-bit channel each for R, G, and B

Red

Blue

8 bit = 255 levels of intensity for each color

Green

Each color channel alone looks like a grayscale image

Plickers...

Mixing the RGB channels gives true color

How many colors with 24 big color?

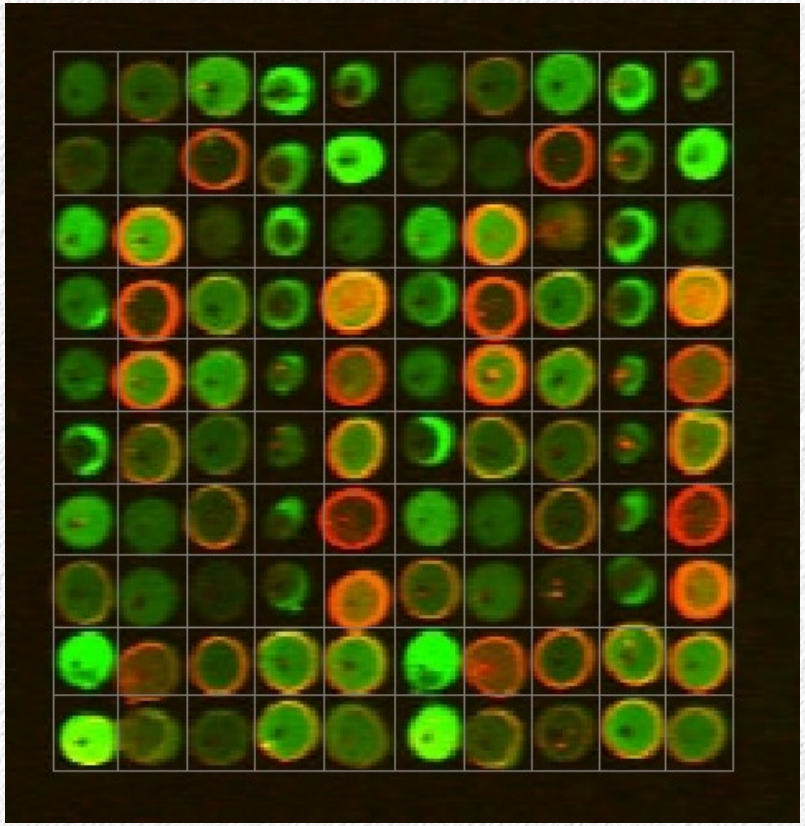
Each channel has 256 levels (0 to 255)

256^3 different RGB combinations → 16,777,216 colors

Fine for human vision – way more than the 1 million colors we see

May not be good enough for some scientific applications (only uses visible light, for example)

Colors as data - microarrays

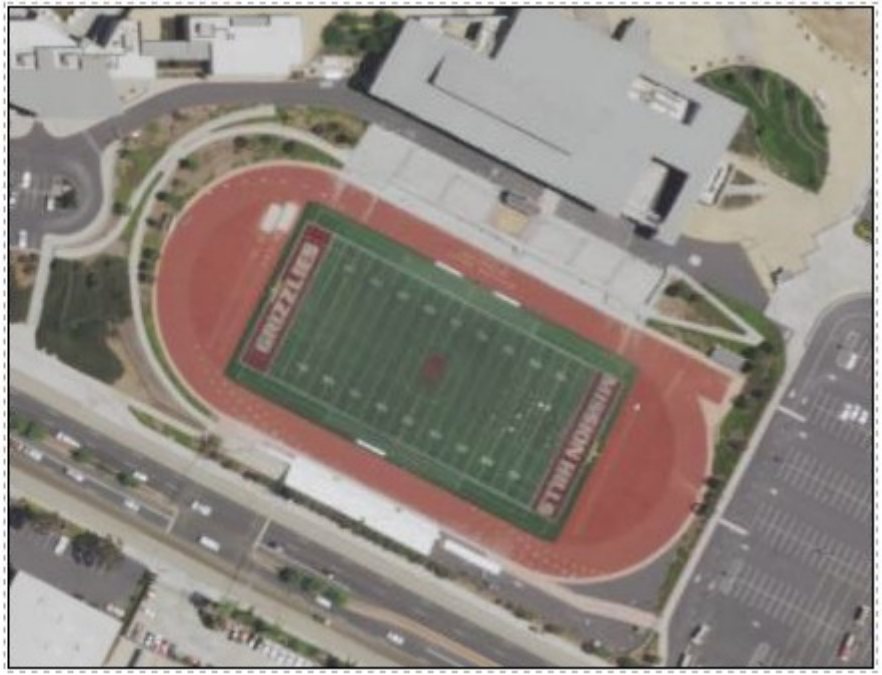


- Measure expression across many genes at once in two groups
- Genes that are expressing one product have a green color, those expressing another are red, expressing both are yellow, neither product are black
- Intensity and shade of color is used to indicate amount of expression

Colors as data – remote sensing

CSUSM

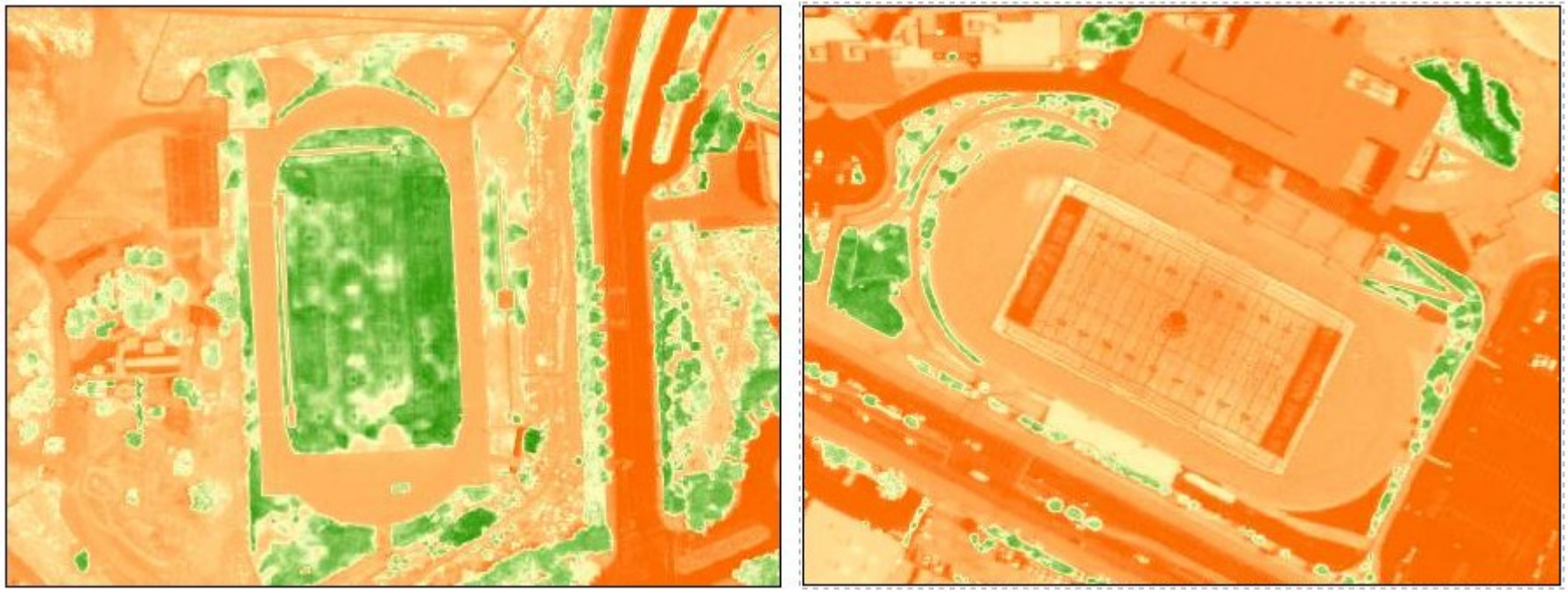
MHHS



Visible light – look the same

NDVI – photosynthetic activity

CSUSM *MHHS*



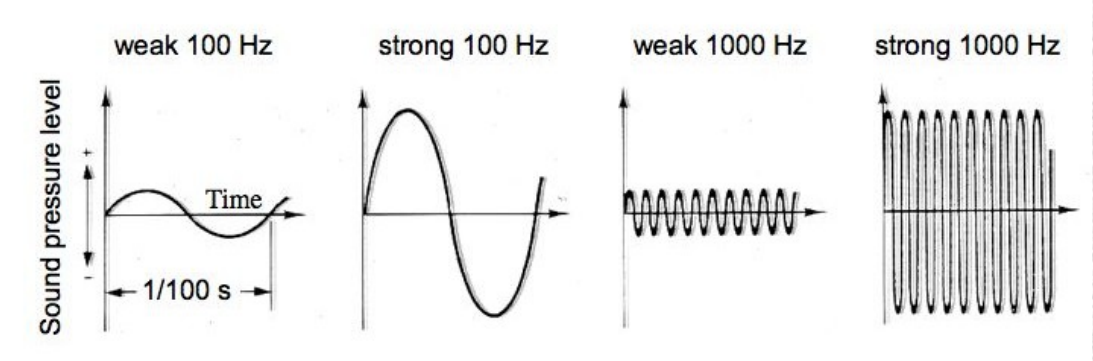
Combination of red and near infrared they look different – why?

Resolution of an image

- Color images will show more detail than grayscale because adjacent pixels with the same shade of gray may be different colors
- The size of the pixels is the spatial resolution of an image (pixels per inch, PPI)
  - To look lifelike, and avoid losing detail, the pixels in an image should not be individually visible
  - Images have fixed numbers of pixels – at a large enough magnification you will see them
- The appropriate spatial resolution depends on how much the image will be magnified

Representing sound on a computer

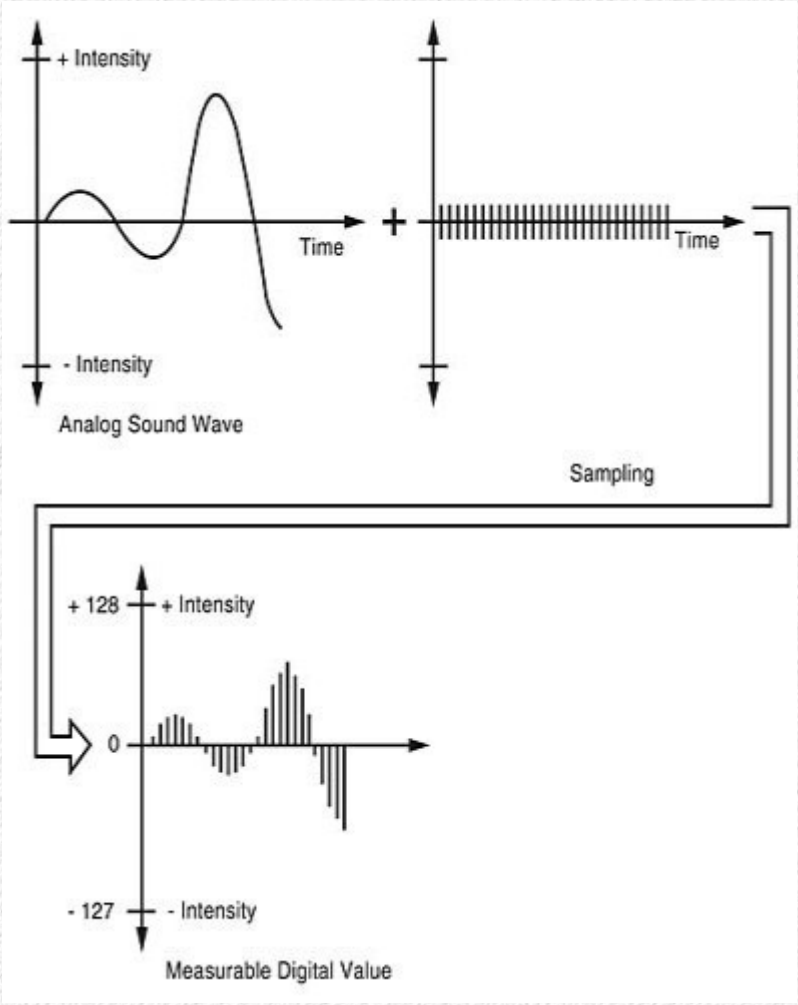
- Sound is a mechanical wave transmitted through the air (or a liquid or solid) – continuous in time
- The tone we hear is determined by the wave form
  - Frequency (cycles per second) determines pitch
  - Amplitude determines volume



- Humans hear sounds between 20 Hz (low pitch) and 20,000 Hz (high pitch), where Hz is cycles per second

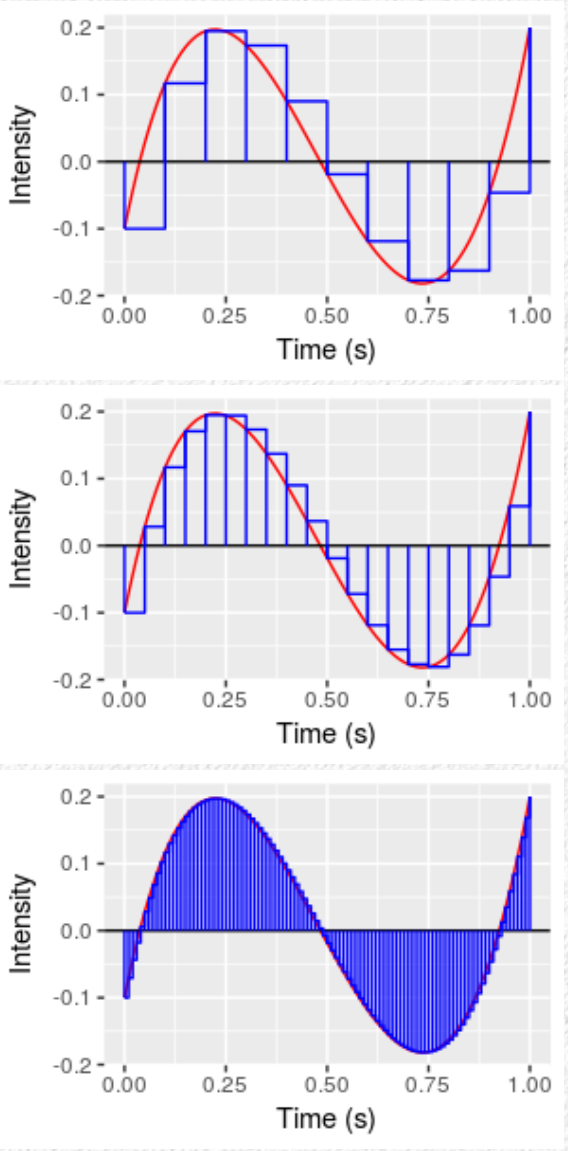
Digital representation of sound

- To digitize the analog sound, samples of intensity are taken over time
- Instead of a continuous wave, digital sound is thus a series of discrete intensities
- These intensities are passed through a digital to analog converter to produce sound



Digital sound fidelity – sampling rate

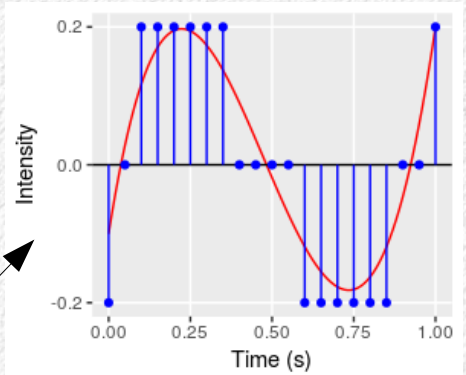
- Sampling rate = how many times per second the sound is sampled
- More times per second allows for closer match to the wave form
- CD's use 44.1 kHz, more than twice the highest audible frequency
- These show the intensity measured exactly at each sample – not really the case

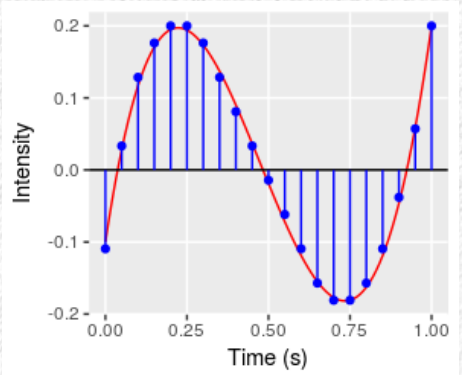


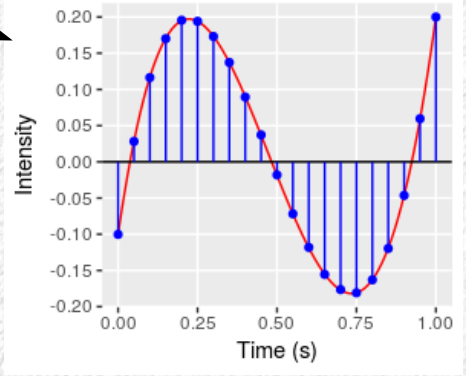
Digital sound fidelity – bit depth

- Bit depth = how many bits are used to store intensity
- Need both + and - intensities, so one bit to the sign
- With 2 bits, can only represent +1,0,-1 as intensities
- With 4 bits, can represent +7 through -8
- With 8 bits, can represent +127 through -128
- With 16 bits (CD quality), can represent from 32767 to -32768

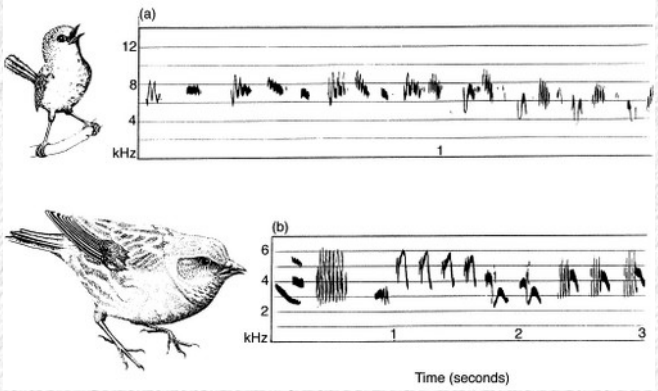
- Best fidelity combines high bit depth and high sampling rate, but this also leads to the largest file sizes

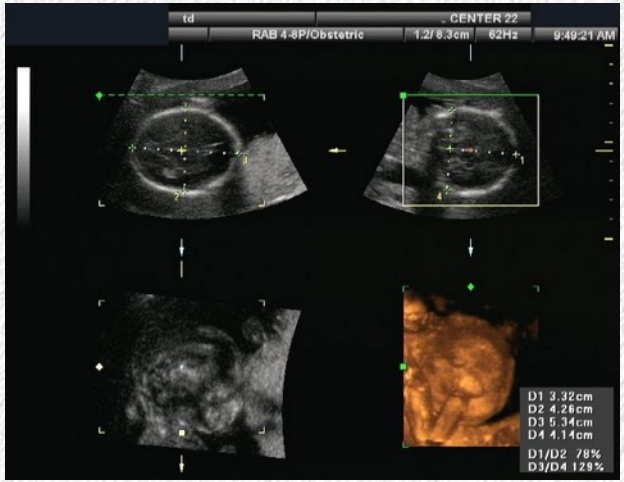






Sound data in biology





- Animal vocalizations
- Ultrasound sonograms
- Sensory physiology

File formats

- Two issues in choosing a file format for your data
  - Openness of standard
  - File size vs. data integrity

Open vs. proprietary

- File formats are created by people, and are a form of intellectual property
  - Can be patented, copyrighted
- Proprietary formats = the specification for the file format is privately owned
  - May not be made public at all
  - Format owner can limit how the file format can be used
- Open standards = file formats with specifications that are made public, without use restrictions

Open file formats are best for scientific work

- Only the owner of a proprietary file format can write software to read the files (well)
  - If the maker of the software goes out of business, your data may become unreadable if it's in a proprietary format
- Open standards are sufficiently well documented that any programmer can write a program to read and write the file
  - Not dependent on one company for access to your own data
- Open standards are best for scientific work – your data will always be available

Example: xls vs. xlsx

- xls is the old standard
  - Proprietary → poorly documented, difficult for third parties to open/write to
  - Binary → need specialized tools to work with
- xlsx is the new standard
  - Open standard → well documented, anyone can write programs to open/write the file format
  - ASCII text-based → no special tools needed to open and work with it
- Given the choice, better to use xlsx

File size issues

- Converting analog to digital representation can result in huge amounts of digital data
  - A single 8 megapixel, 24 bit image file in an uncompressed file format is 24 MB
  - A single audio CD with 80 minutes of uncompressed audio is 700 MB
  - Each second of uncompressed high definition video is 118 MB, each minute is 7.1 GB
- Storage space quickly becomes problematic – various methods are used to compress these files

“Lossy” and “lossless” compression

- There are a couple of major approaches to saving space for audio and images/video:
  - Lossless: Retain all of the recorded information, but represent it in a way that requires fewer binary digits
  - Lossy: Throw out some of the recorded data
- Lossy compression can produce smaller file sizes, but changes the data

A simple lossless compression scheme

- If you have a string of 1's and 0's that looks like this:

  `10000011111111000001110000000111111`

- It's more compact to write it like this:

  `1@10@51@80@51@30@71@6`

- This representation will save space to the extent that 1's and 0's tend to repeat frequently – it's not effective if they alternate a lot

What is lost using lossy formats

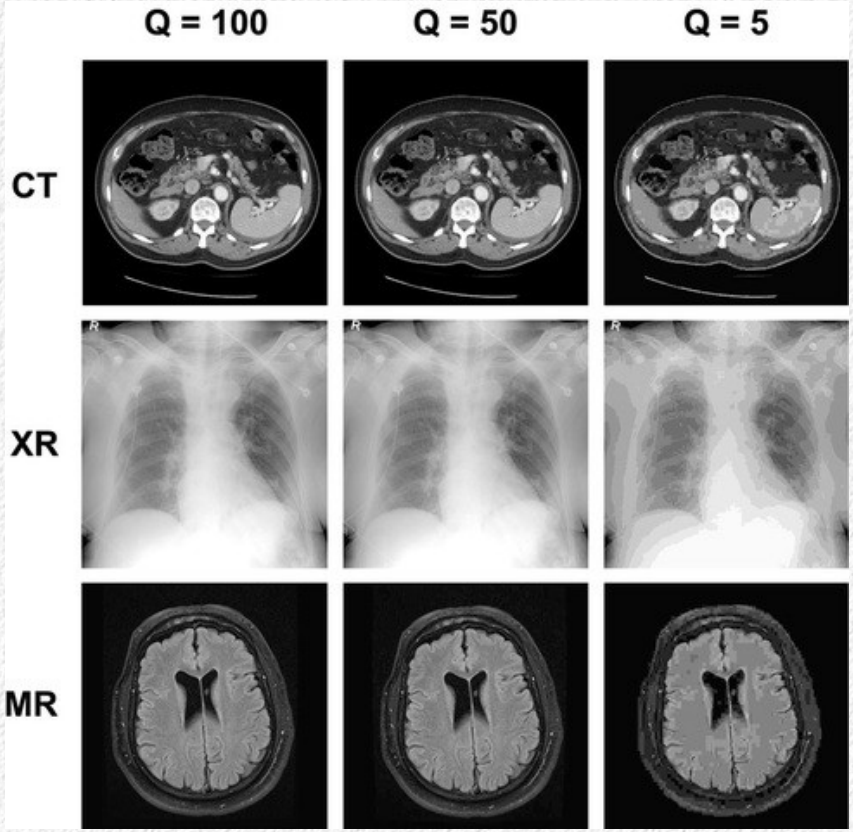
- Good lossy compression only throws out the “unimportant” parts of the data
- What is unimportant depends on the intended use of the file
  - MP3 compressed sound files throw out frequencies that humans can't hear
  - jpeg throws out image data in a way that still looks okay when the image is viewed at its intended magnification
  - In both cases, “important” is defined by human aesthetics
  - “Unimportant” data aesthetically may be very important scientifically

Lossy compression of an image – the jpeg format

- Colors that are similar in a region are assigned to the same color
  - How similar they have to be is determined by the quality setting used
- Pixels are then represented in a condensed way (like the lossless scheme shown before)
- The results can look fine if the image is not enlarged, but the distortions are visible at high magnification

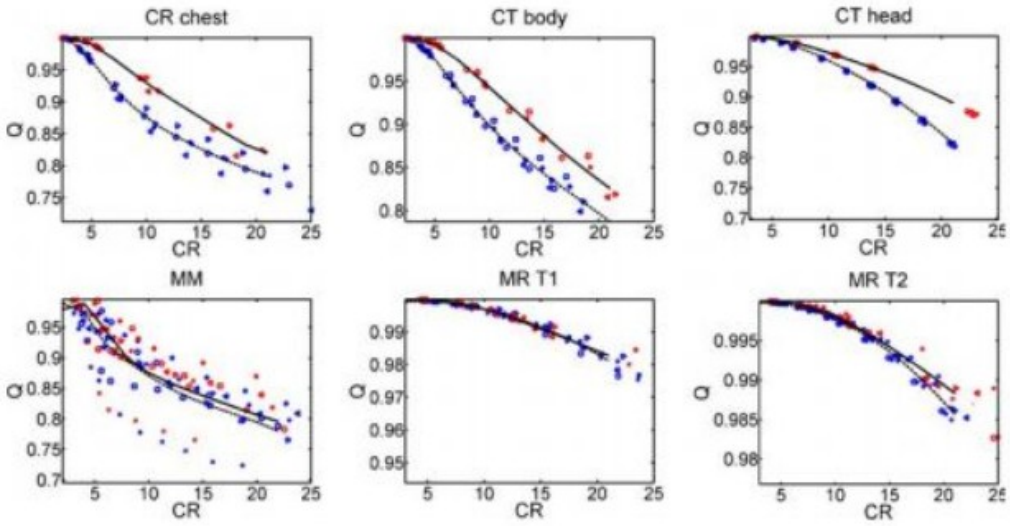
Lossy compression in medical imaging

- Using jpeg compression in medical imaging changes the image
- The ability of a radiologist to diagnose from the image is affected by image quality



More compression → less image quality

- Comparison of quality (Q) vs. compression ratio
- Two different types of compression
  - Standard jpeg (blue)
  - jpeg2000 (red)
- Quality goes down as compression ratio goes up for both



Compression ratio (x-axis) vs. Q index (y-axis) as a measure of image quality in 6 different image modalities. The blue points show JPEG compression, while the red points show JPEG-2000 compression (another compression scheme). The apparent trend is an inverse relationship between image quality and compression ratio. (Shiao, 2007)

Lossy vs. lossless in digital data

- Lossless compression is less effective at saving storage space, but doesn’t modify the original, measured data
  - Should always be okay for scientific work
- Lossy compression may be okay, but you have to know how it changes your data to decide
  - mp3 throws out frequencies out of human audible range, which may be a problem if you’re studying bat vocalization
  - jpeg compression is designed to maintain the appearance of the image on a web site, but for medical imaging or GIS data you want each pixel to be accurate