Data encoding

Lauri Võsandi

Binary data

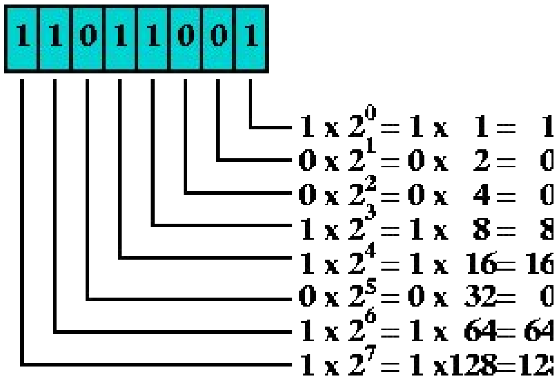
- Binary can represent
  - Letters of alphabet, plain-text files
  - Integers, floating-point numbers (of finite precision)
  - Pixels, images, video
  - Audio samples
- Could be stored in processor registers, RAM, harddisk, transmitted over network etc
- Quantization, quantization error

Binary encoding

<table>
<tr><td colspan="16">Word</td></tr>
<tr><td colspan="8">Byte 1 (High)</td><td colspan="8">Byte 0 (Low)</td></tr>
<tr><td colspan="4">Nibble 3</td><td colspan="4">Nibble 2</td><td colspan="4">Nibble 1</td><td colspan="4">Nibble 0</td></tr>
<tr><td>Bit 15</td><td>Bit 14</td><td>Bit 13</td><td>Bit 12</td><td>Bit 11</td><td>Bit 10</td><td>Bit 9</td><td>Bit 8</td><td>Bit 7</td><td>Bit 6</td><td>Bit 5</td><td>Bit 4</td><td>Bit 3</td><td>Bit 2</td><td>Bit 1</td><td>Bit 0</td></tr>
</table>

Binary to decimal

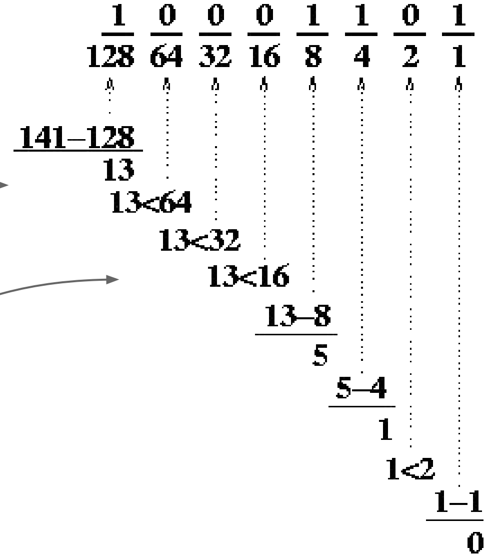
- Bit indexing starts from 0
- Least significant bit is usually on the right
- Each bit has weight of 2^n
- Multiply each bit with it's weight
- Add the multiplications



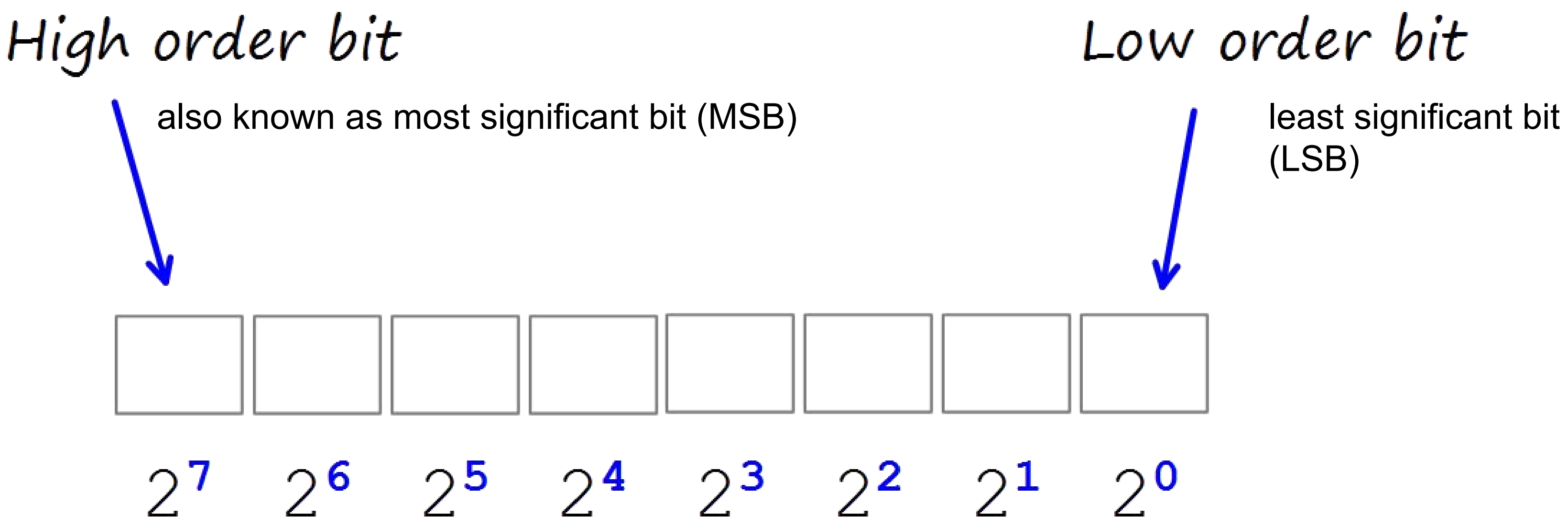
$1 + 8 + 16 + 64 + 128 = 217$

Decimal to binary

- If weight can be subtracted, bit corresponds to one
- If the number is smaller than next weight, bit corresponds to zero



Bit order


High order bit
also known as most significant bit (MSB)
Low order bit
least significant bit (LSB)
2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

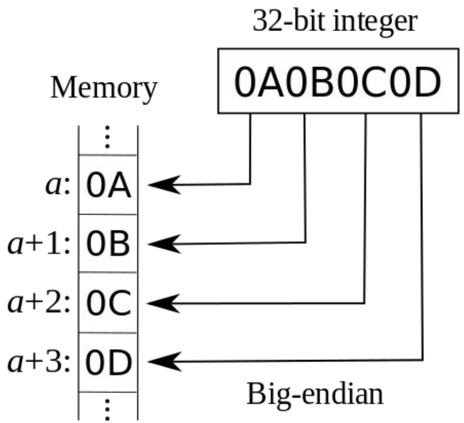
Hexadecimal representation

- Each hexadecimal digit corresponds to nibble (4-bits)
- Hexadecimal retains alignment to binary data opposed to decimal

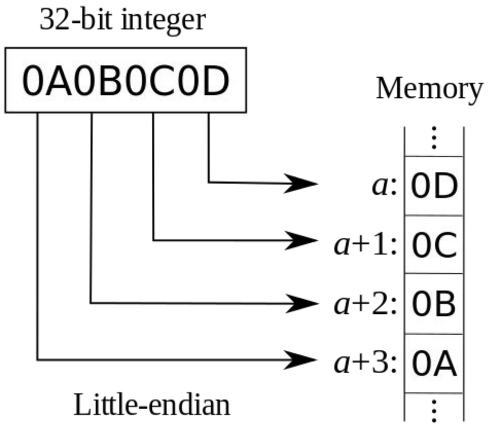
```
binary     hexadecimal     decimal
0000   =>  0           =>  0
0001   =>  1           =>  1
0010   =>  2           =>  2
0011   =>  3           =>  3
0100   =>  4           =>  4
0101   =>  5           =>  5
0110   =>  6           =>  6
0111   =>  7           =>  7

1000   =>  8           =>  8
1001   =>  9           =>  9
1010   =>  A           =>  10
1011   =>  B           =>  11
1100   =>  C           =>  12
1101   =>  D           =>  13
1110   =>  E           =>  14
1111   =>  F           =>  15
```

Endianess

Motorola 68k (Macintosh)



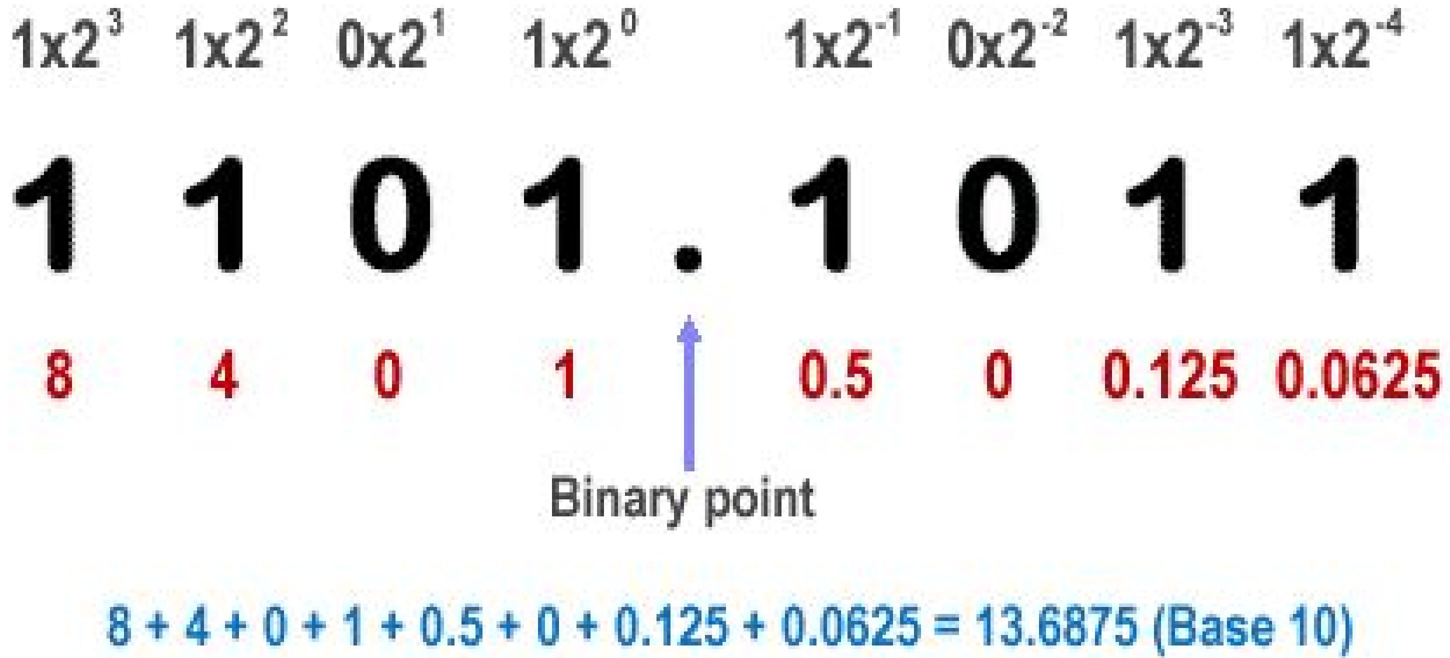
Intel x86 (PC-s)



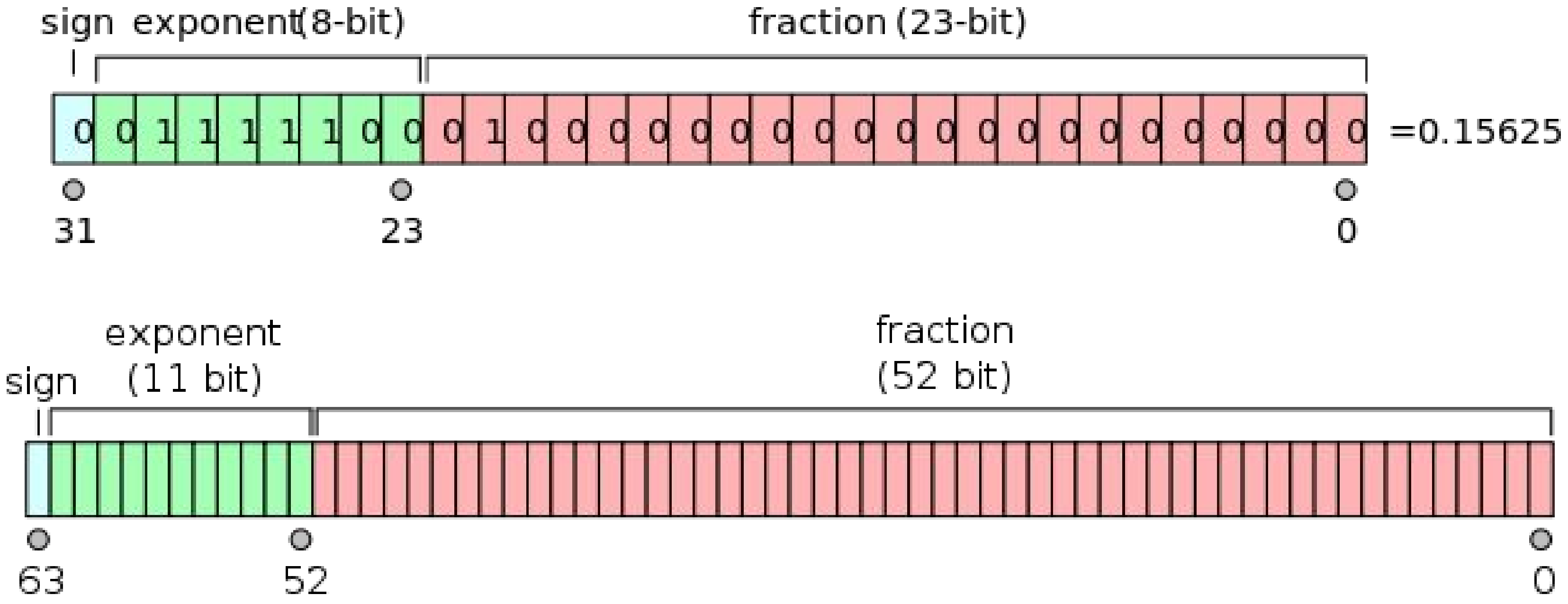
Integer representation

- Number 42 (decimal) could be represented as
  - 0b101010 (binary)
  - 0x2a (hexadecimal)
  - 052 (octal)
  - 0o52 (also octal)
- Check out http://baseconvert.com

Fixed-point numbers



IEEE754 floating point numbers



Text encoding

ASCII, Unicode

The ASCII code

American Standard Code for Information Interchange

www.theasciicode.com.ar

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	Ì	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	

ISO8859-13 (Baltic)

- Portion of extended ASCII replaced with letters from Baltic languages

	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
80	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143
90	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
A0	160	” 161	¢ 162	£ 163	€ 164	„ 165	¦ 166	§ 167	Ø 168	© 169	Ŗ 170	« 171	¬ 172	- 173	® 174	Æ 175
B0	° 176	± 177	² 178	³ 179	“ 180	µ 181	¶ 182	· 183	ø 184	¹ 185	ŗ 186	» 187	¼ 188	½ 189	¾ 190	æ 191
C0	Ą 192	Į 193	Ā 194	Ć 195	Ä 196	Å 197	Ę 198	Ē 199	Č 200	É 201	Ź 202	Ė 203	Ģ 204	Ķ 205	Ī 206	Ļ 207
D0	Š 208	Ń 209	Ņ 210	Ó 211	Ō 212	Õ 213	Ö 214	× 215	Ų 216	Ł 217	Ś 218	Ū 219	Ü 220	Ż 221	Ž 222	ß 223
E0	ą 224	į 225	ā 226	ć 227	ä 228	å 229	ę 230	ē 231	č 232	é 233	ź 234	ė 235	ģ 236	ķ 237	ī 238	ļ 239
F0	š 240	ń 241	ņ 242	ó 243	ō 244	õ 245	ö 246	÷ 247	ų 248	ł 249	ś 250	ū 251	ü 252	ż 253	ž 254	’ 255

Problems

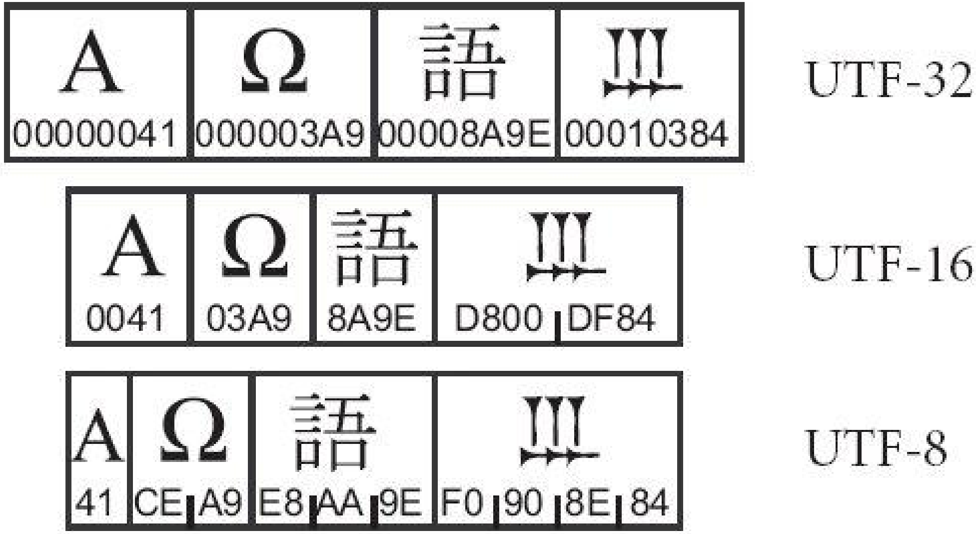
- Impossible to mix documents of different character sets
- 8-bits not enough to describe alphabets of different languages

Unicode

- More than million characters are described
- Unicode code point refers to a index of symbol: 0x00000 to 0x10FFFF
- How it gets mapped to bits is different story:
  - UTF-8 - Variable length coding (1 to 4 bytes)
  - UTF-16 - Also variable-length coding (2 or 4 bytes)
  - UTF-32 - Only fixed-width coding (4 bytes)

Unicode

- ASCII was used for source code,
  text files etc.
- Has been replaced by UTF-8
- In-memory data structures different



Python 2.x str is ASCII

```
>>> type("γεια σας")
<type 'str'>
>>> len("γεια σας")
15
>>> type(u"γεια σας")
<type 'unicode'>
>>> len(u"γεια σας")
8
```

Python 3.x str is Unicode

```
>>> type("γεια σας")
<class 'str'>
>>> len("γεια σας")
8
>>> type(b"γεια σας")
  File "<stdin>", line 1
SyntaxError: bytes can only contain ASCII literal characters.
>>> "γεια σας".encode("utf-8")
b'\xce\xb3\xce\xb5\xce\xb9\xce\xb1 \xcf\x83\xce\xb1\xcf\x82'
>>> type(b"hello world")
<class 'bytes'>
```

Data types in Java

Primitive Types					
Type Name	Wrapper class	Value	Range	Size	Default Value
byte	java.lang.Byte	integer	−128 through +127	8-bit (1-byte)	0
short	java.lang.Short	integer	−32,768 through +32,767	16-bit (2-byte)	0
int	java.lang.Integer	integer	−2,147,483,648 through +2,147,483,647	32-bit (4-byte)	0
long	java.lang.Long	integer	−9,223,372,036,854,775,808 through +9,223,372,036,854,775,807	64-bit (8-byte)	0
float	java.lang.Float	floating point number	±1.401298E−45 through ±3.402823E+38	32-bit (4-byte)	0.0
double	java.lang.Double	floating point number	±4.94065645841246E−324 through ±1.79769313486232E+308	64-bit (8-byte)	0.0
boolean	java.lang.Boolean	Boolean	true or false	8-bit (1-byte)	false
char	java.lang.Character	UTF-16 code unit (BMP character or a part of a surrogate pair)	'\u0000' through '\uFFFF'	16-bit (2-byte)	'\u0000'

Data types in C (x86)

```
sizeof(bool) == 1         #  8-bit boolean
sizeof(char) == 1         #  8-bit ASCII char or byte
sizeof(short) == 2        # 16-bit integer
sizeof(int) == 4          # 32-bit integer
sizeof(long) == 4         # 32-bit integer
sizeof(long long) == 8    # 64-bit integer
sizeof(float) == 4        # 32-bit floating point number
sizeof(double) == 8       # 64-bit floating point number
sizeof(void*) == 4        # 32-bit pointer
```

Data types in C (armhf)

```
sizeof(bool) == 1
sizeof(char) == 1
sizeof(short) == 2
sizeof(int) == 4
sizeof(long) == 4
sizeof(long long) == 8
sizeof(float) == 4
sizeof(double) == 8
sizeof(void*) == 4
```

Data types in C (x86-64)

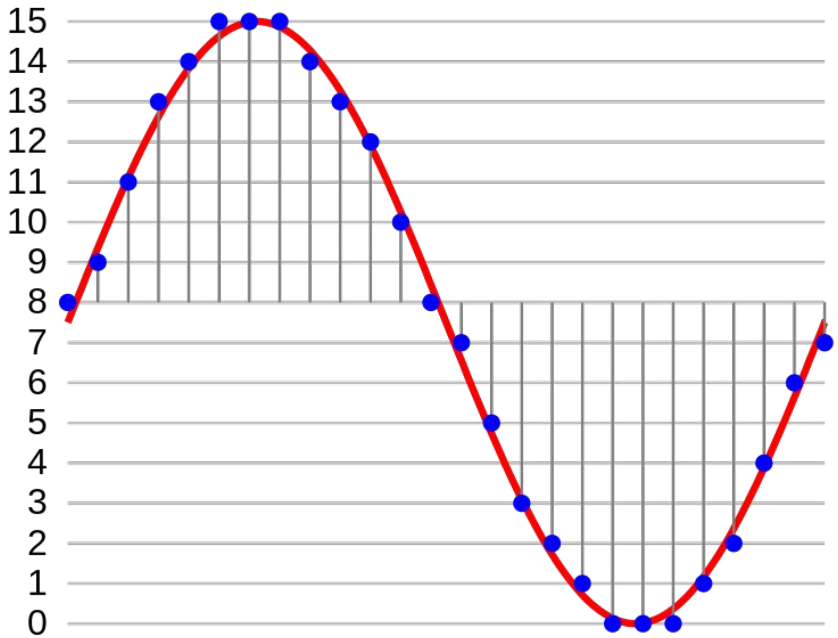
```
sizeof(bool) == 1          #  8-bit boolean
sizeof(char) == 1          #  8-bit ASCII char or byte
sizeof(short) == 2         # 16-bit integer
sizeof(int) == 4           # 32-bit integer
sizeof(long) == 8          # 64-bit integer (!)
sizeof(long long) == 8     # 64-bit integer
sizeof(float) == 4         # 32-bit floating point number
sizeof(double) == 8        # 64-bit floating point number
sizeof(void*) == 8         # 64-bit pointer
```

Audio encoding

Resolution, sampling rate

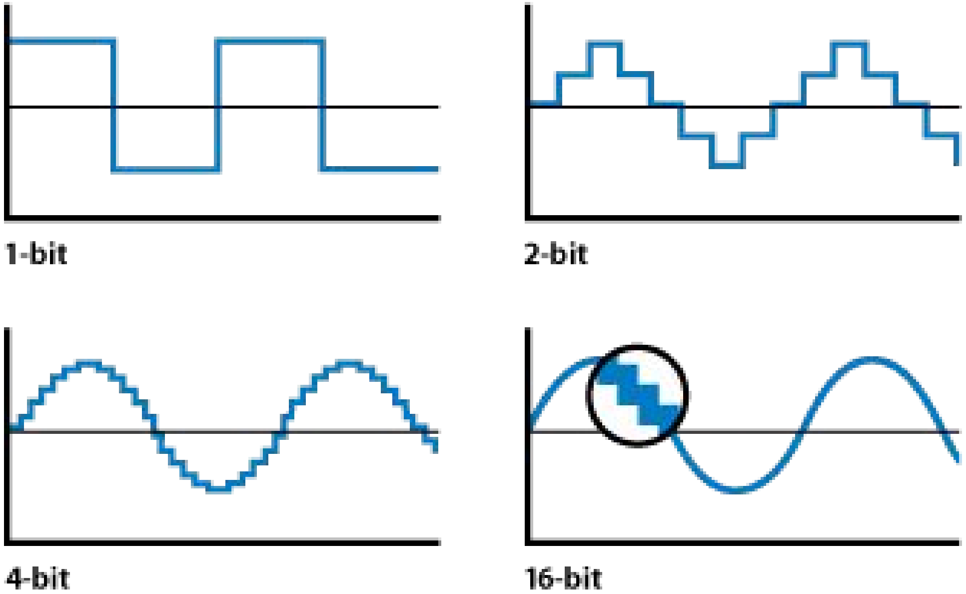
Pulse-coded modulation (PCM)

- Common bit depths are 8, 16 and 24 bits
- Example on the right uses 4 bits per channel



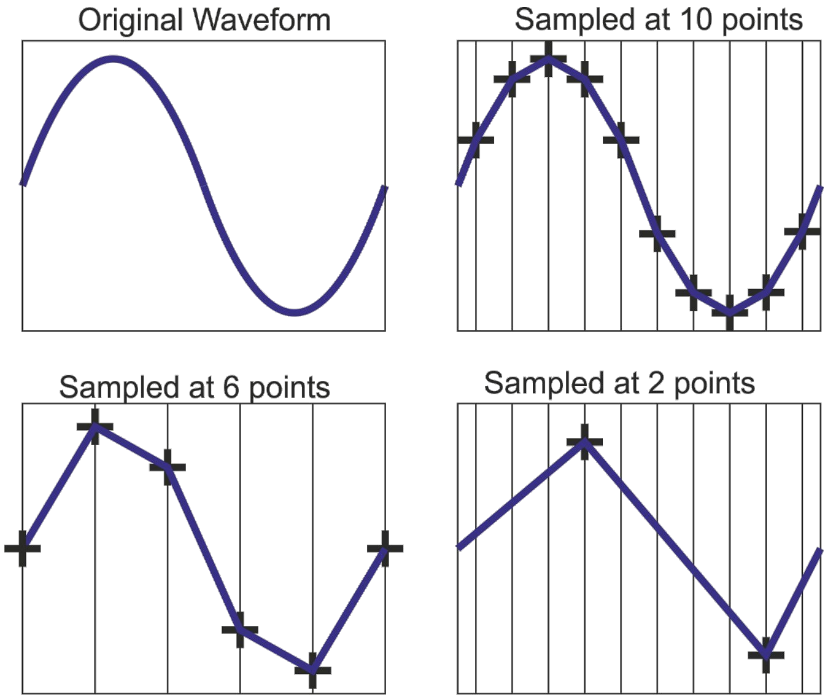
Audio resolution

- How accurately audio signal can be represented
- Speaker cone displacement measuring precision
- Audio CD: 16-bits/ch



Audio sampling rate

- How accurately audio signal can be represented
- Frequently of speaker cone displacement measurement
- Audio CD: 44.1kHz



Digital-to-analog conversion

- Each output bit is connected to bit weight resistor
- Resistances are aggregated
- Op-amp amplifies the final voltage

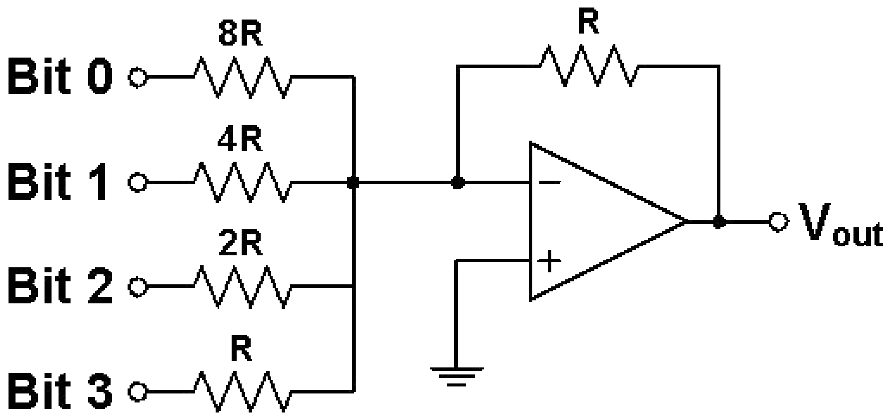
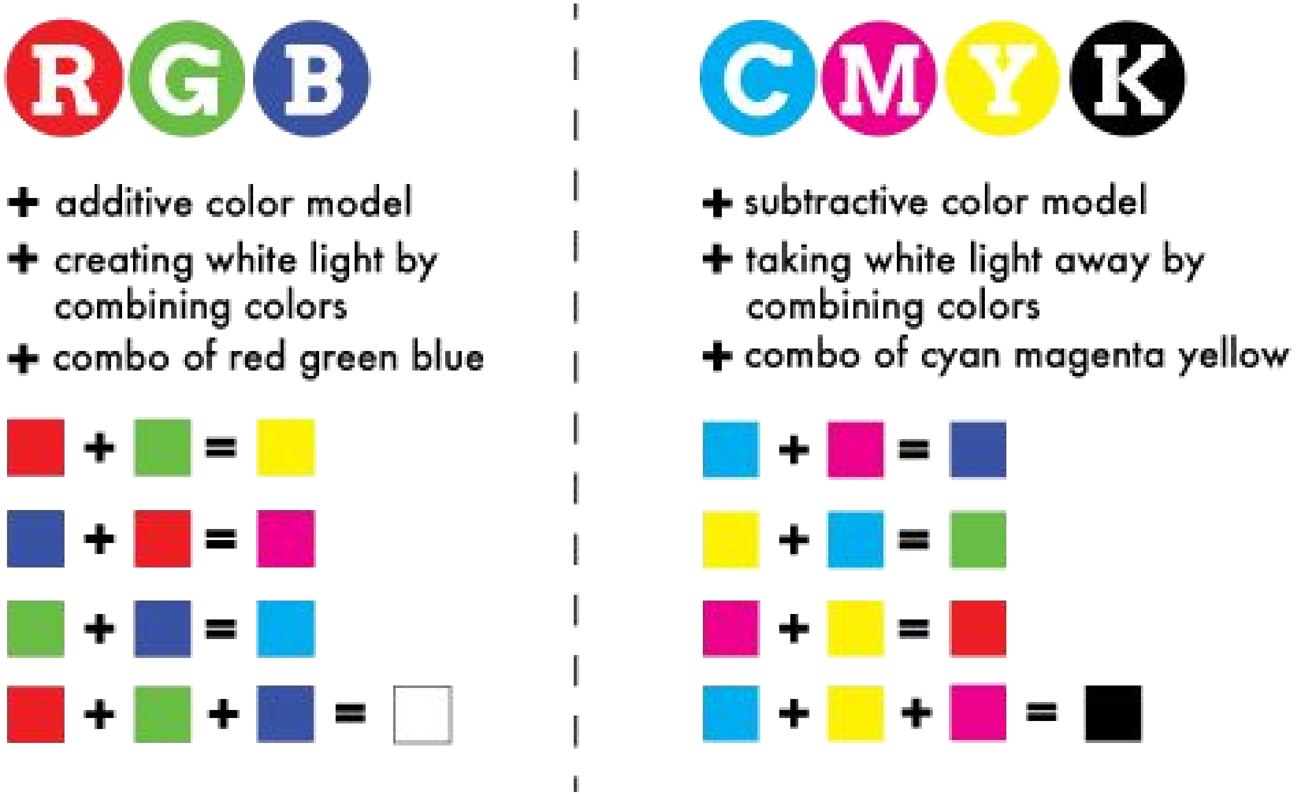


Image encoding

Pixels, color depth, resolution

Color models



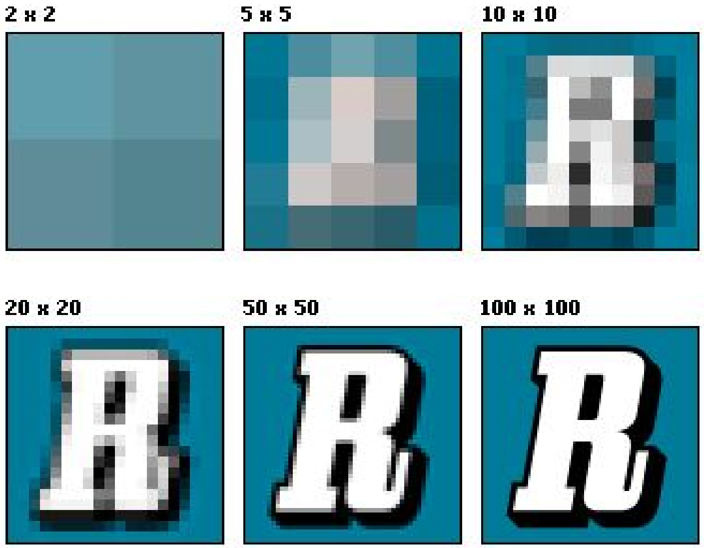
Images

- Picture element usually known as *pixel*
- Red, green, blue channels represent intensity
- Alpha channel represents transparency
- Different modes: RGB, BGR, ARGB, RGBA, ABGR, ...

Sample Length:	8	8	8	8
Channel Membership:	Alpha	Red	Green	Blue
Bit Number:	31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0

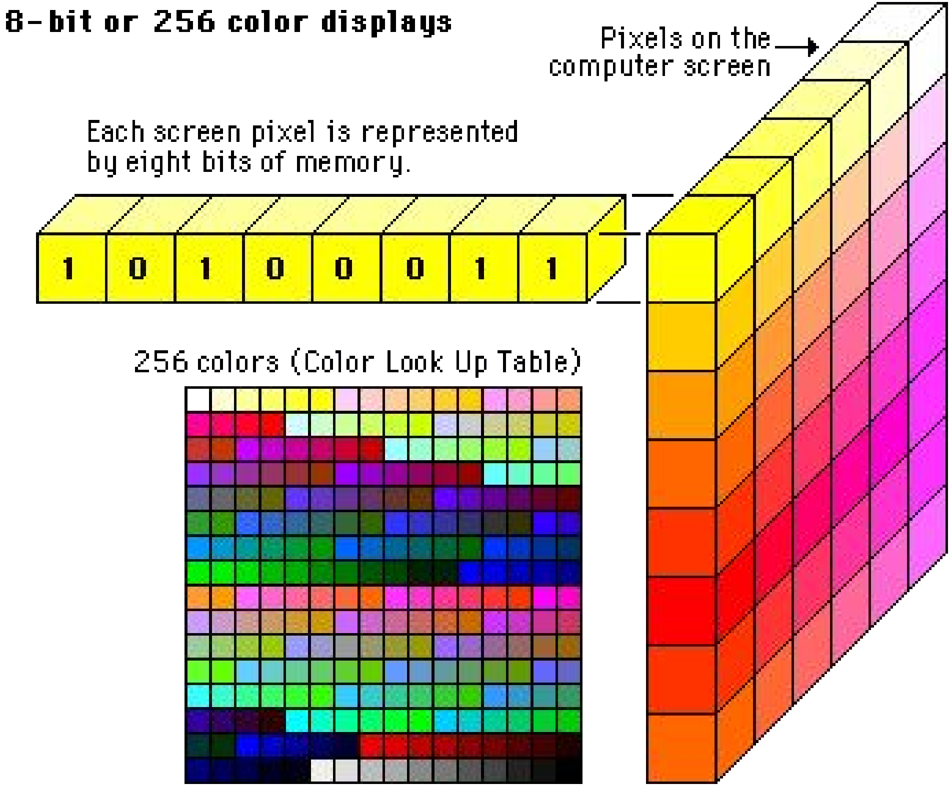
Resolution

- How many pixels
  - Horizontally
  - Vertically
- DPI (dots per inch)
- The more pixels, the better it looks



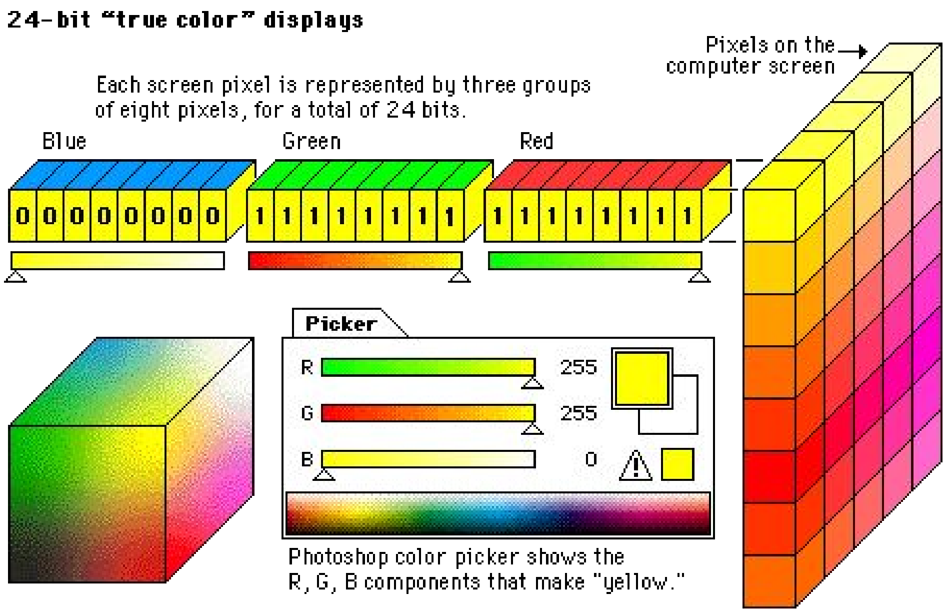
Indexed colors

- Video card contains the look up table
- Each pixel is the index in the lookup table
- RGB values computed on the fly at video output

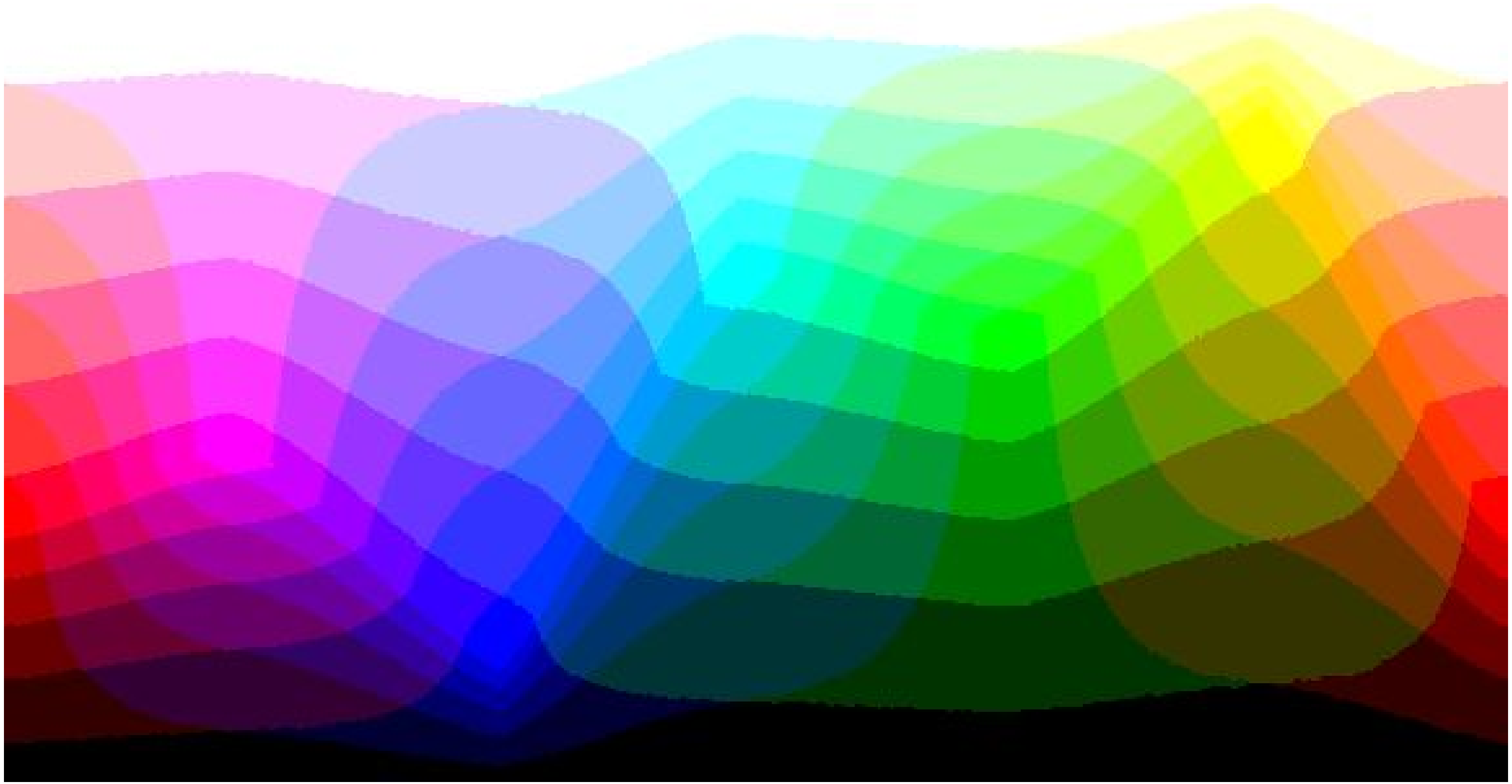


True color

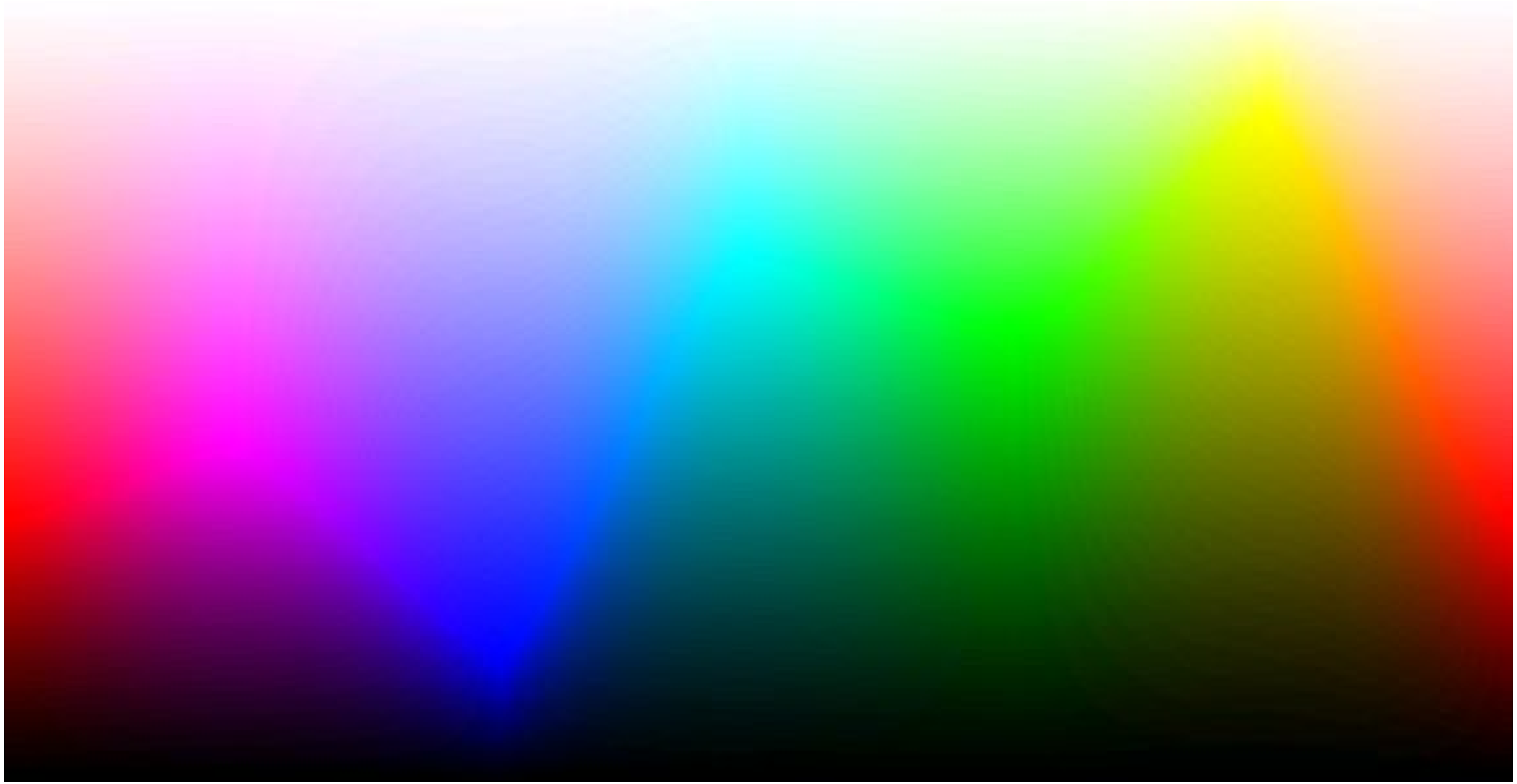
- Each pixel contains actual RGB data
- RGB 8:8:8 corresponds to 2^{24} = 16777216 colors
- RGB 5:6:5 corresponds to 2^{16} = 65536 colors



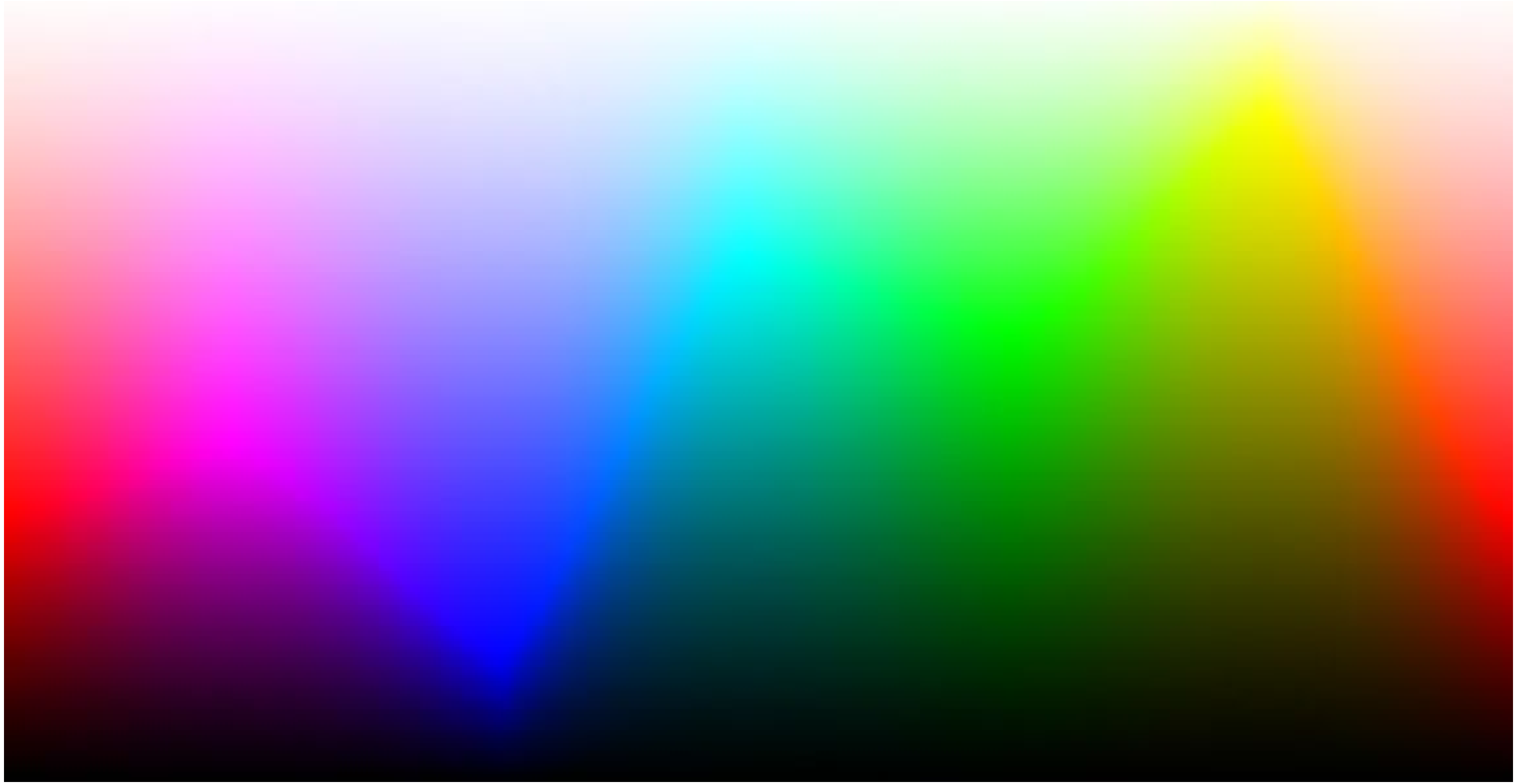
256 colors



16 bits per pixel (RGB 5:6:5)

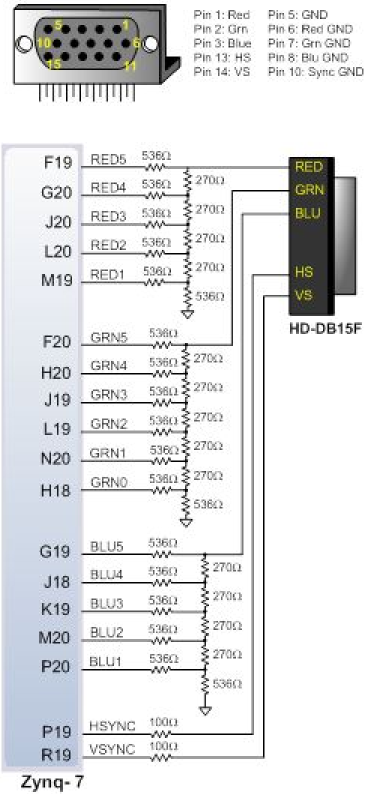


24 bits per pixel (RGB 8:8:8)



Video DAC

- The simplest/ cheapest use resistor ladder similar to audio DAC



Compression

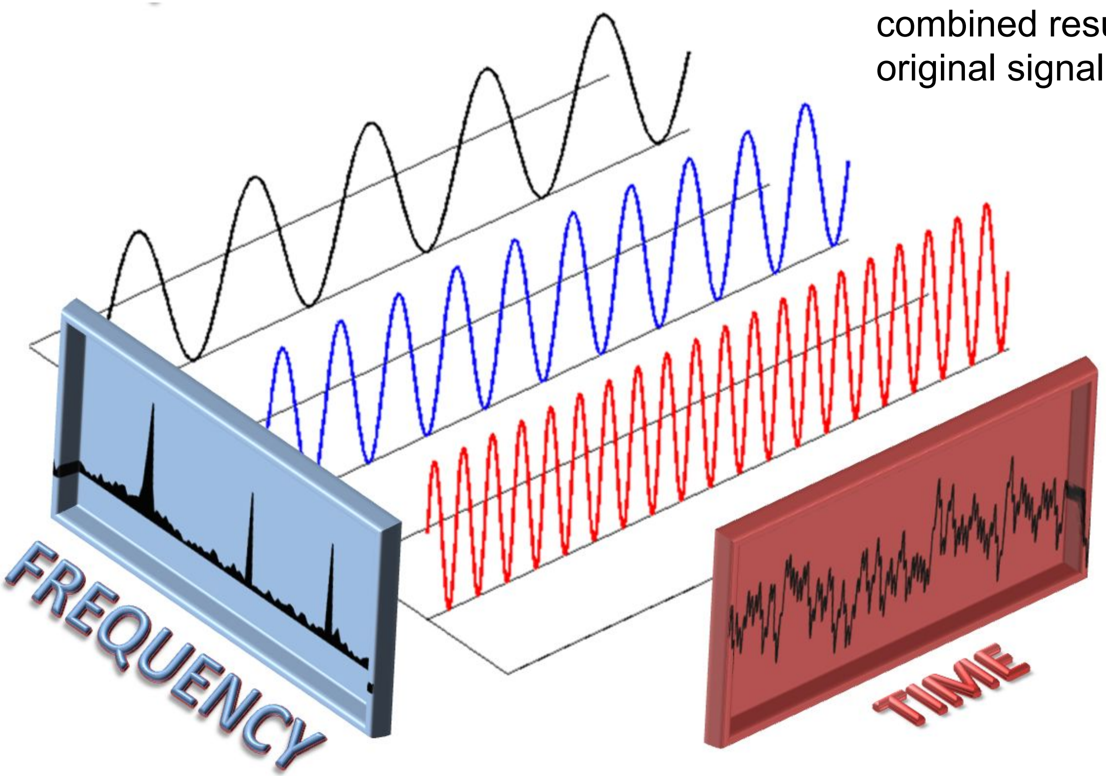
Fourier transform, RLE, Huffman encoding

Audio compression

- Frames (group of audio samples) are converted from time domain to frequency domain
- Frequencies with low energy are discarded
- Peaking frequencies are rounded
- Adjacent peaks are merged
- Phase offset information is lost

Fourier transform

Frequencies that combined result in the original signal



Frequency domain representation (frequencies and their amplitudes)

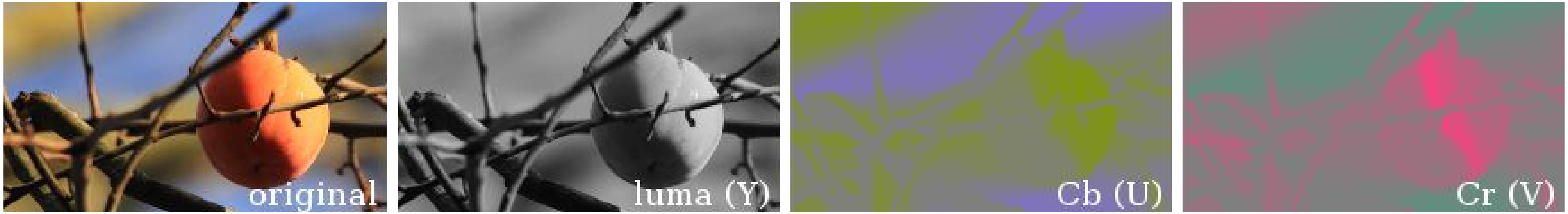
Time domain representation (samples)

Image compression

- Photographs
  - High correlation between RGB channels
  - No independent pixels
  - A lot of gradients
- Computer graphics eg. screenshots
  - Adjacent pixels of same color
  - Some pixels occur more frequently than others

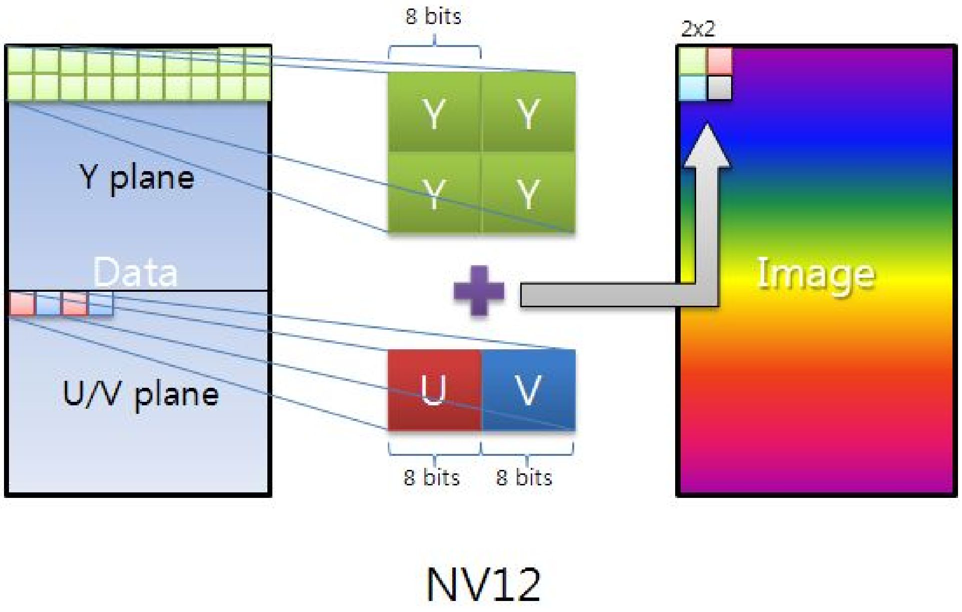
Other colorspaces

- YUV or YCbCr used in image/video
- Luma and chroma information instead of RGB
- Less resolution and bit depth for chroma
- No perceived image quality degradation



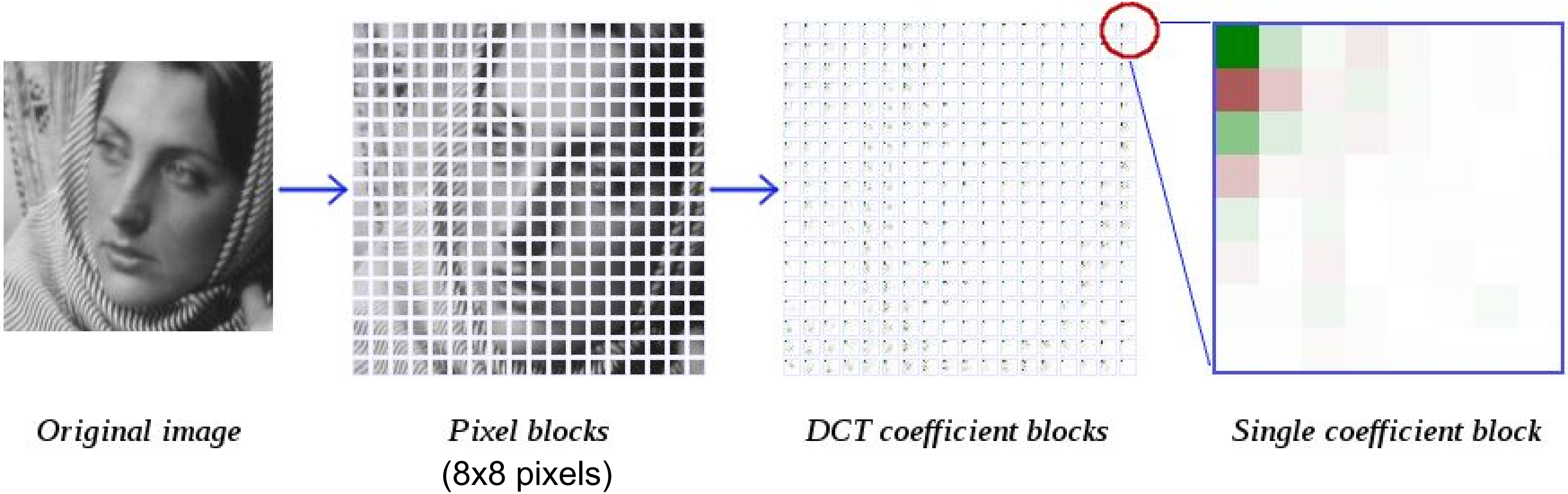
RGB vs YUV

- RGB (8:8:8) representation would result in 12 bytes per 4 pixels
- The representation on right would result in 6 bytes per 4 pixels

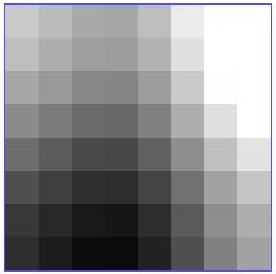


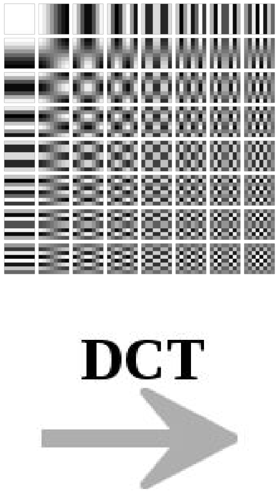
Discrete cosine transform

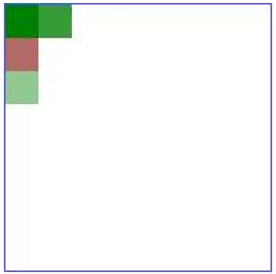
- Used in JPEG, MPEG
- A simplified case of Fourier transform



Discrete cosine transform







Original pixel data

114	108	100	99	109	129	152	166
109	102	95	94	104	124	146	161
99	93	85	84	94	114	137	151
86	80	72	71	82	102	124	138
73	66	58	57	68	88	110	125
60	53	46	45	55	75	97	112
50	43	36	35	45	65	88	102
45	38	31	30	40	60	82	97

DCT coefficient data

700	200	0	0	0	0	0	0
-150	0	0	0	0	0	0	0
110	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Running length encoding

Substitute group of identical numbers:

- How many?
- What number?

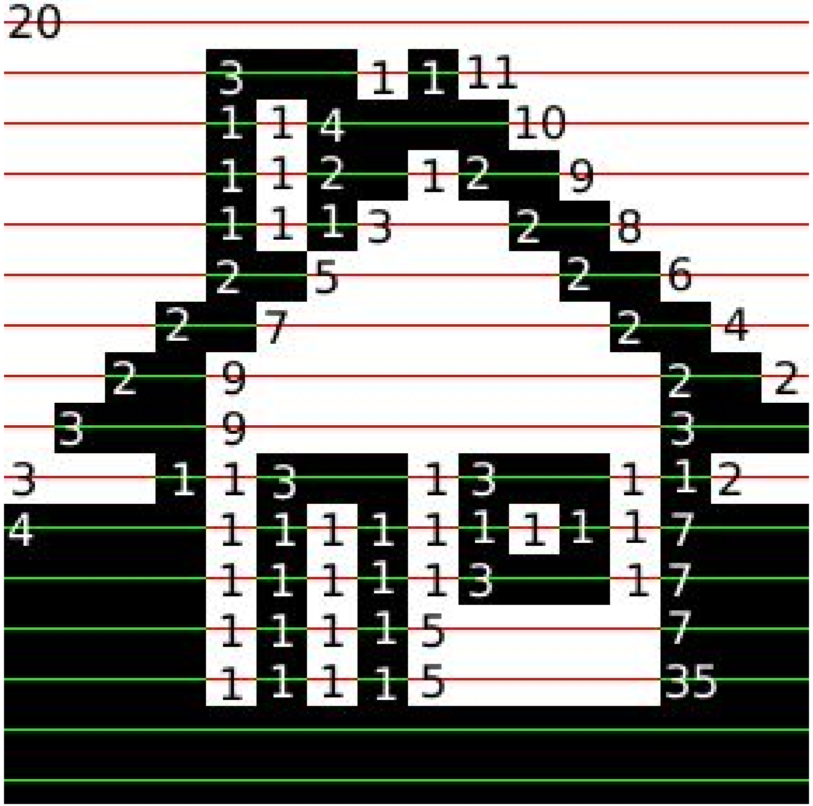


Photo compression with JPEG

- Colorspace transformation from RGB to YCbCr
- Downsampling by discarding chroma bits
- Block splitting usually to 8x8 pixel blocks
- DCT to convert pixels to waves
- Quantization, round off insignificant coefficients
- Running length encoding
- Huffman encoding, use less bits to represent frequently occurring bit sequences

Potential exam questions

- What is 0xFF, 0xFFFF, 0xFFFFFF in decimal?
- What is 0755 in binary?
- How many bits are required do describe integer range -63 to 64?
- What integer range / how may colors can be described using 24 bits?
- What color is 0x88FF8800 (ARGB)?

Potential exam questions

- Describe simplest 8-bit stereo DAC
- Describe RGB (4:4:4) DAC
- What is the minimum audio CD capacity assuming stereo sound at 44.1kHz sampling rate and 16-bits per channel for 80 minute album?
- What is the bitrate for 7.1 sound system sampled at 96kHz and 24-bits per channel?

Potential exam questions

- What is the significance of Fourier transform?
- What is time domain representation?
- What is frequency domain representation?
- What is running length encoding?
- What is Huffman encoding?

Where are we know

- We know how to install and run OS
- We know how to use command-line
- We know how to invoke a program
- We know how to represent in binary
  - Plain text, integers, floating point numbers
  - Audio and images
  - How to store them efficiently

What next?

- How is an actual CPU processing the data?