IERG4190 / IEMS5707
Multimedia Coding and Processing

Chapter 2: Multimedia Coding Basics

Bolei Zhou
Department of Information Engineering, CUHK
Spring 2021

Part of the materials courtesy of Dr William Hui, Prof Tang Xiaoou, Prof. Liu JianZhuang. All images from Internet belong to respective owners.

Announcement

- Assignment 1 will be out this week on Blackboard
- TA Office Hour: Wed 6:00 pm – 7:00 pm
- TA Tutorial: Wed 6:00 pm – 6:45 pm (Week 3 – Week 10). It continues to be Office Hour after the tutorial. ZOOM link is in the Google Doc

Outline

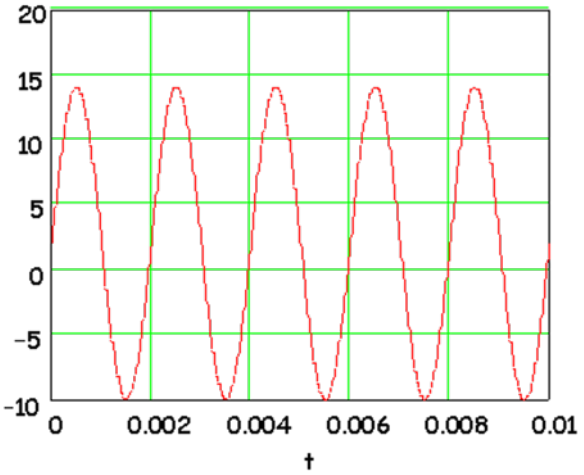
1. Some Basics
2. Redundancy
3. Elements of a Multimedia Coding System
4. Scalar Quantization
5. Vector Quantization
6. Dithering
7. Huffman Coding
8. Multi-Symbol Coding with Static Dictionaries

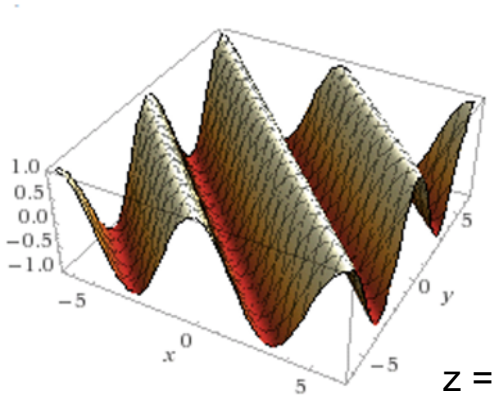
Basic Terminology

- Pixels (of an image, we will investigate in more details later)
- Frames (of a video)
- Encode v.s. Decode
- Compress v.s. Decompress/Reconstruct

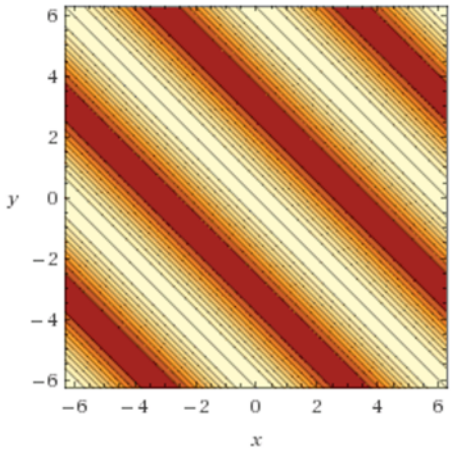
2D Signals

A 2D signal is just 2D. No need to be scared.





z = cos(x+y)



E.g. amplitude of the signal may denote any channel (R,G,B) of a color image

2D Signals

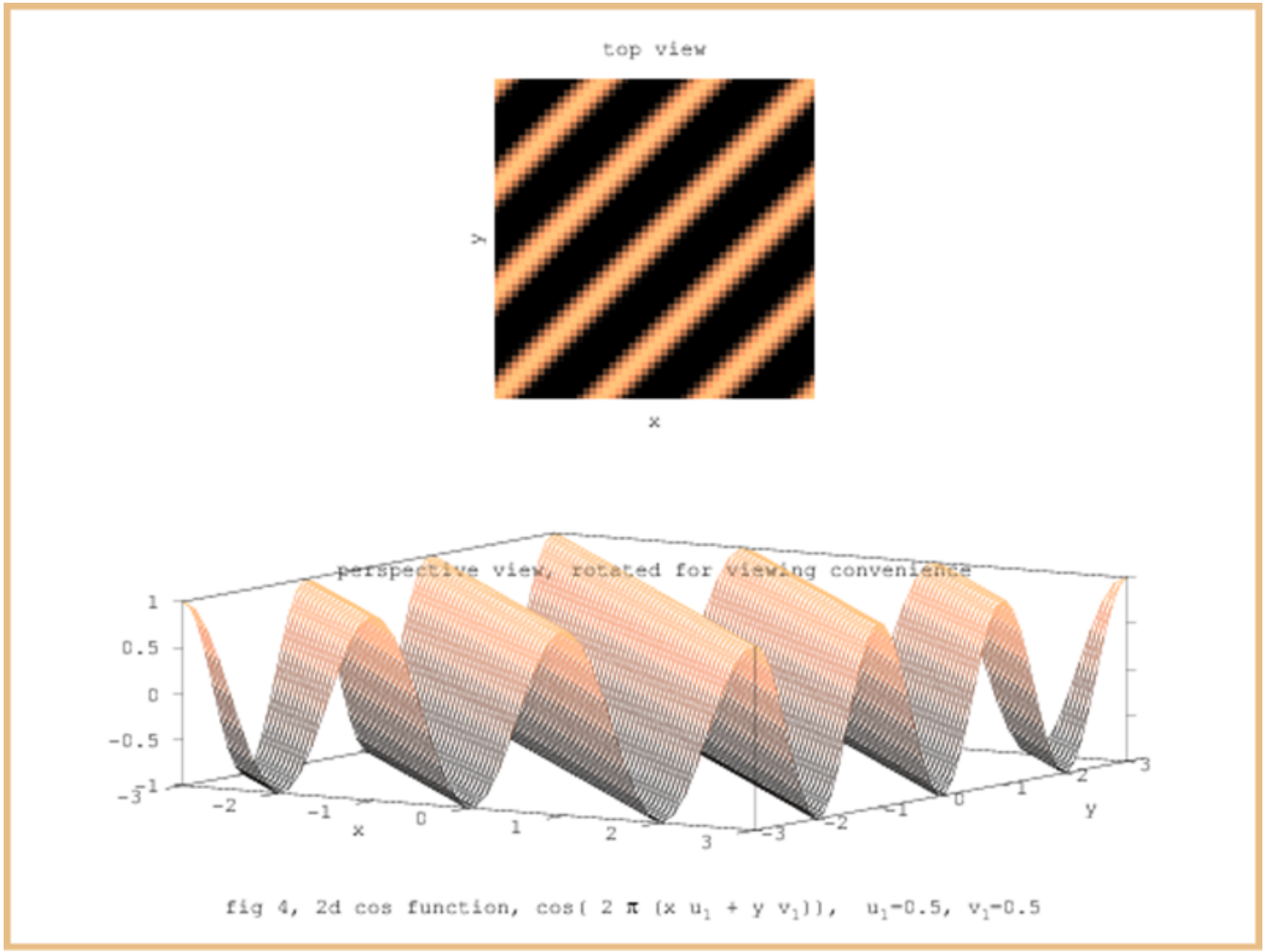


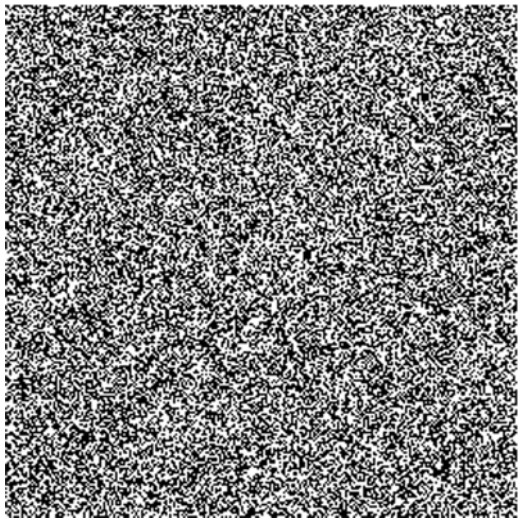
fig 4, 2d cos function, $\cos(2 \pi (x u_1 + y v_1))$, u_1=0.5, v_1=0.5

Multimedia Coding

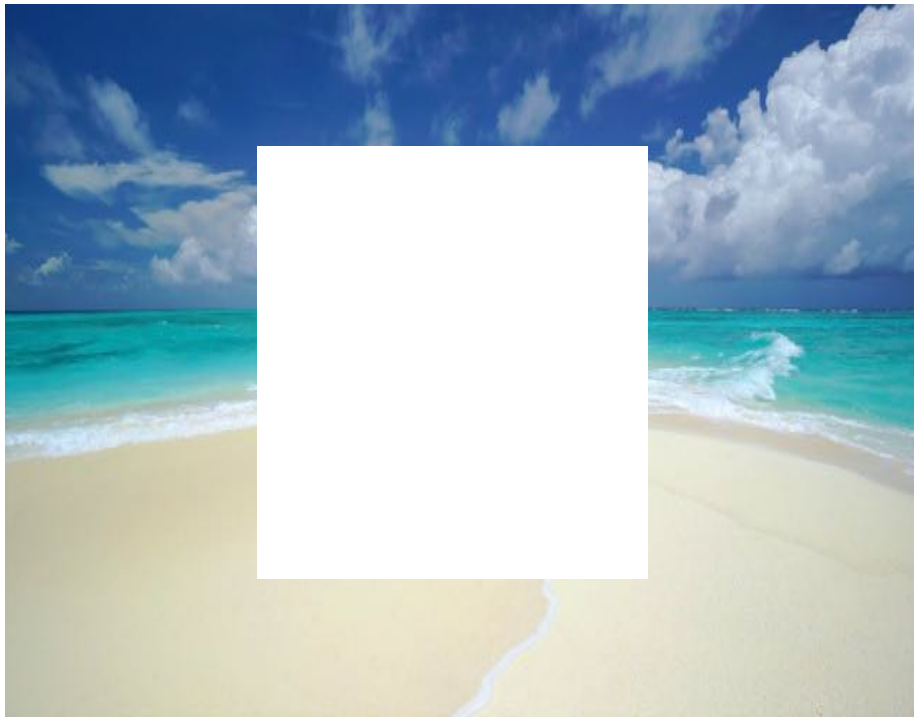
- We want to store and transmit videos, images and sound efficiently
- So rather than storing or transmitting them in its naive representation, we want to compress/decompress them
- How? **Redundancy**
- Redundancy comes in many forms

Statistical Redundancy

- For an image, neighboring pixels are usually similar (spatial redundancy)
- For a video, successive frames are usually similar (temporal redundancy)



Semantic Redundancy



Psychovisual or -acoustical Redundancy

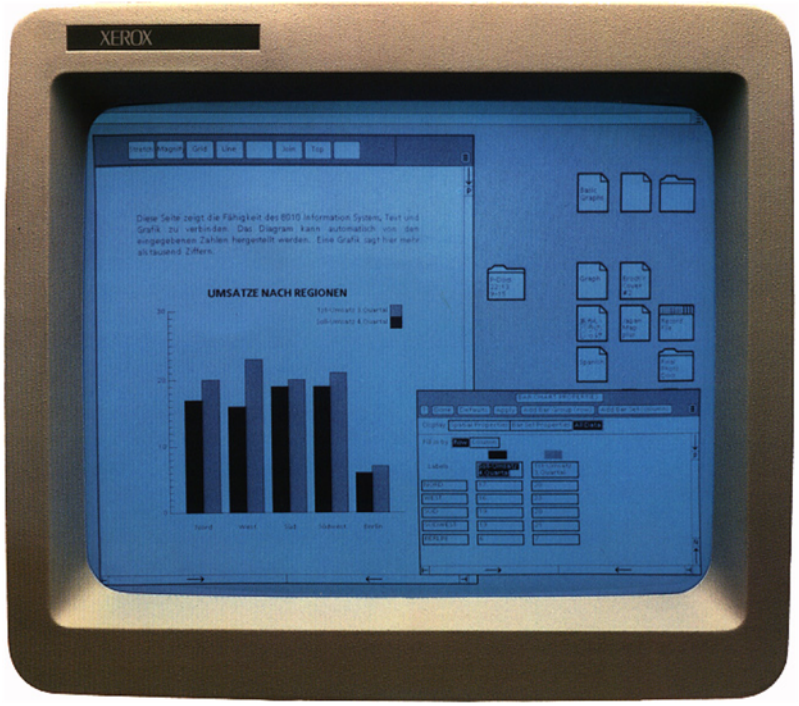
- Our perception is not omnipotent and not perfect
- We may not notice some color differences, or may not hear some part of a tone - in such cases part of our signal could be redundant

Unconvinced? Try the Color Challenge:
https://www.xrite.com/hue-test

Let's Have an Experiment!

Appliance Specific Redundancy

- Our appliance itself has limited capability
  - E.g. for sound, your speaker

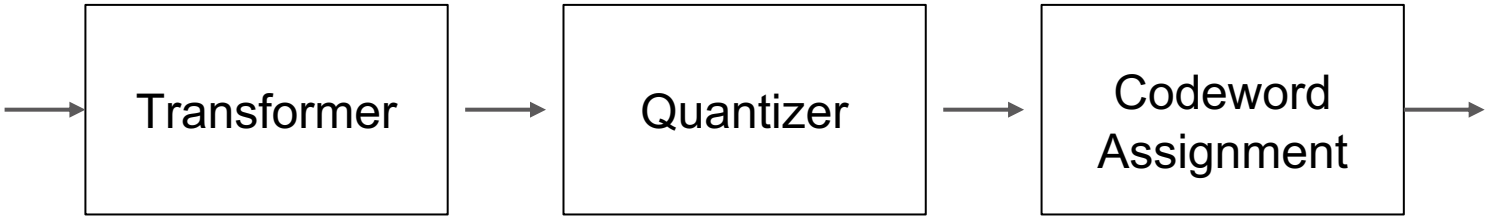


"Look how colorful it is!!"

Lossless v.s. Lossy Compression

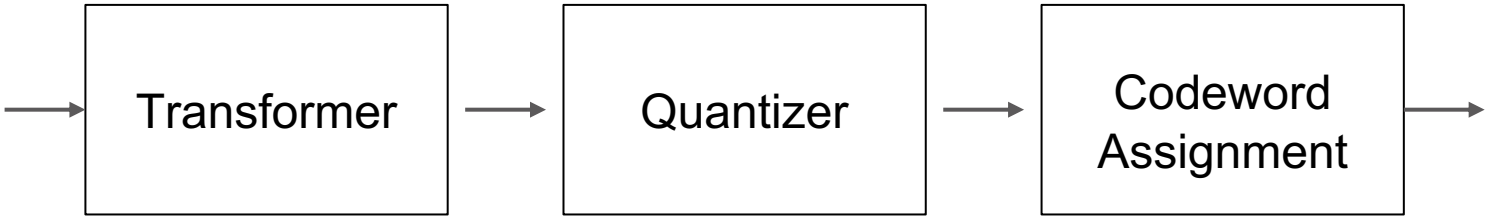
- **Lossless compression:** can only rely on reducing statistical redundancy
- **Lossy compression:** exploit psychovisual or psychoacoustical redundancy
  - Irreversible mapping
  - Exact reconstruction not possible

Elements of a Multimedia Coder



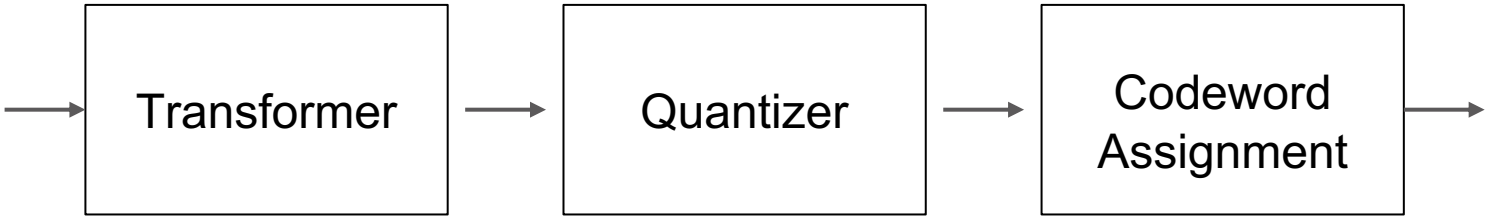
Transformer: transform the input data into a form more amenable to compression
e.g. Discrete Fourier Transforms (DFT), Discrete Cosine Transform (DCT) etc.

Elements of a Multimedia Coder



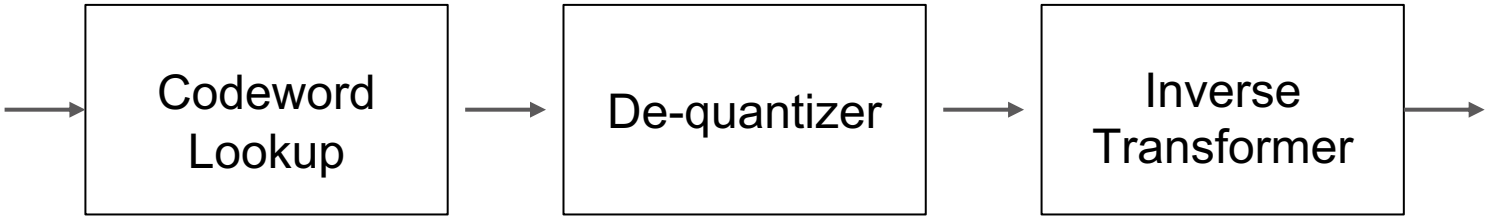
Quantizer: represent transformed signal with a limited number of levels/symbols; an irreversible operation;
E.g. scalar quantization, vector quantization

Elements of a Multimedia Coder



Codeword Assignment: Assign codewords to the quantized output, creating a bit stream
e.g. fixed length coding v.s variable length coding

Elements of a Multimedia Decoder



Codeword Lookup: Decodes bitstream into quantized levels

Dequantizer: Reverse the quantization of quantizer

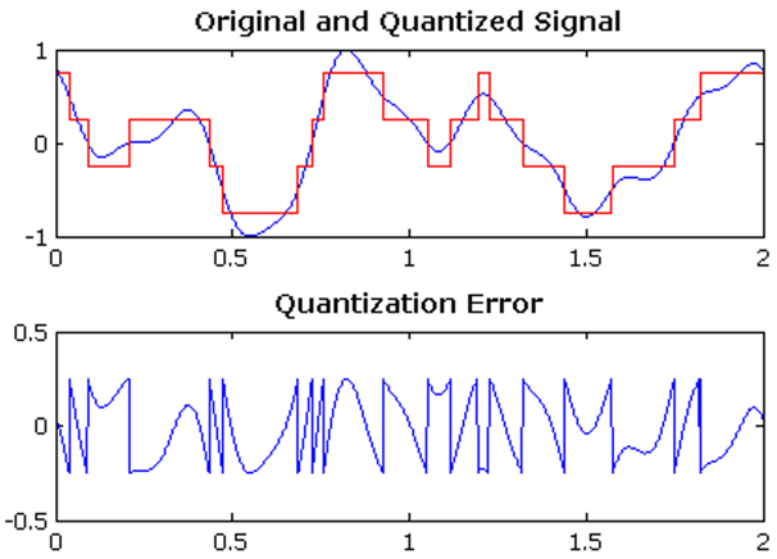
Inverse Transformer: Reverse the transformation for display/playback

Different Encoding/Decoding Methods

- Choices of transformer, quantizer and encoder (and their reconstruction equivalents) result in different multimedia coding methods
- Human psychovisual data used in standards like JPEG, MPEGs and H.26x

Scalar Quantization

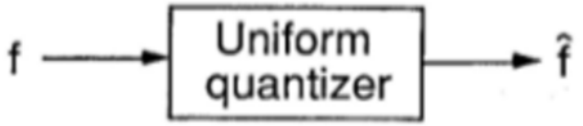
To represent a continuous scalar value *f* with a finite number of bits, only a finite number of quantization levels *L* can be used. If each scalar is quantized independently, the procedure is called scalar quantization.



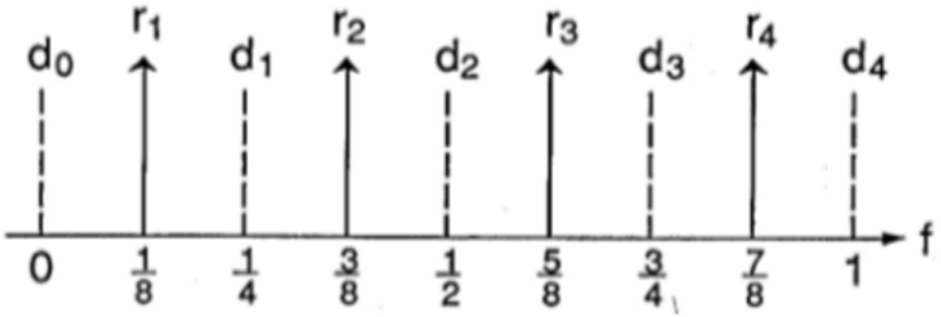
Uniform Quantization

Uniform quantization: equal spacing of reconstruction levels.
Example: image intensity f : 0~1

r_i : reconstruction levels
d_i : decision bounaries



Number of reconstruction levels : 4



Uniform Quantization

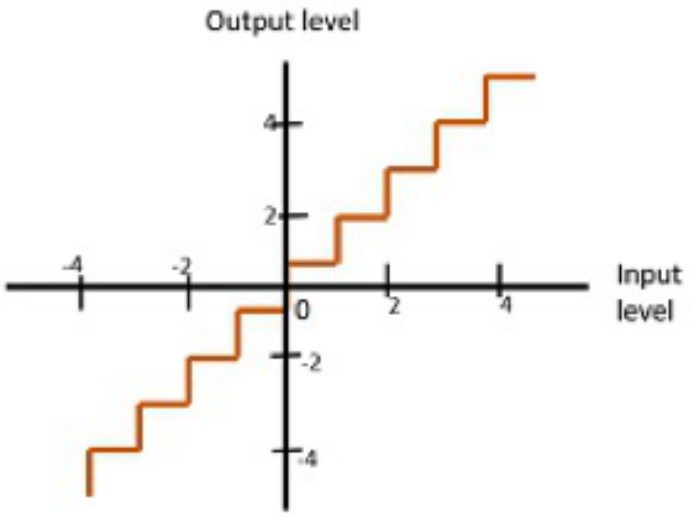


Fig 1 : Mid-Rise type Uniform Quantization

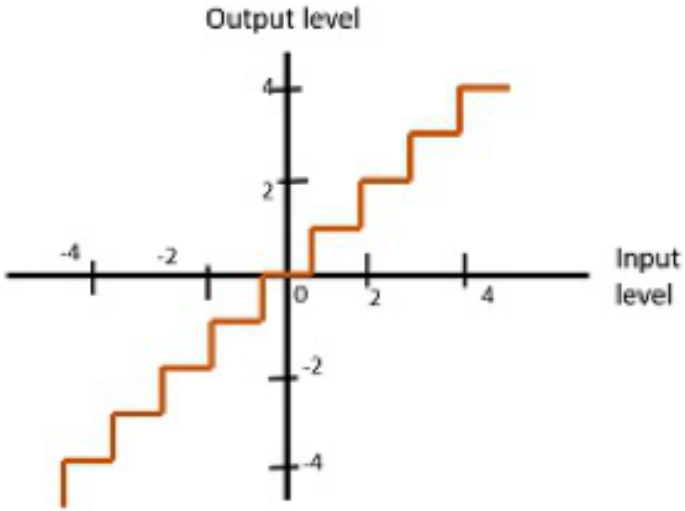
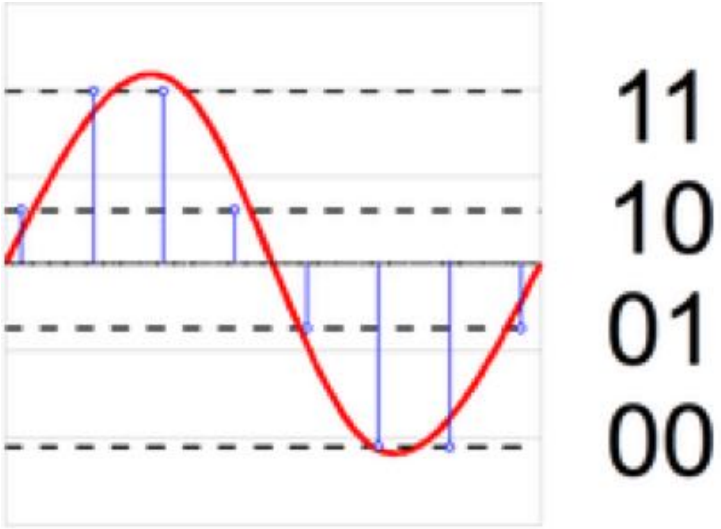
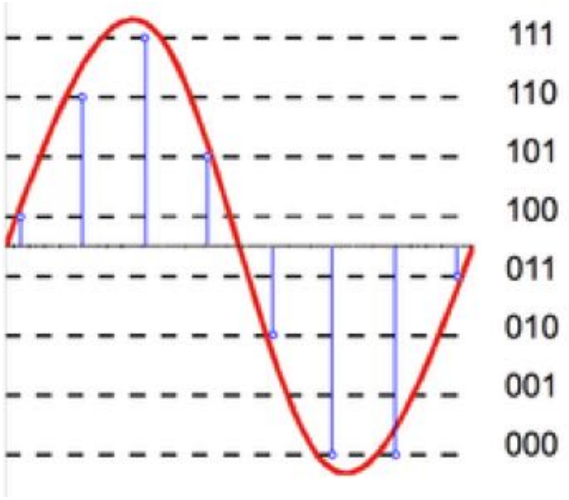


Fig 2 : Mid-Tread type Uniform Quantization

Bits and levels



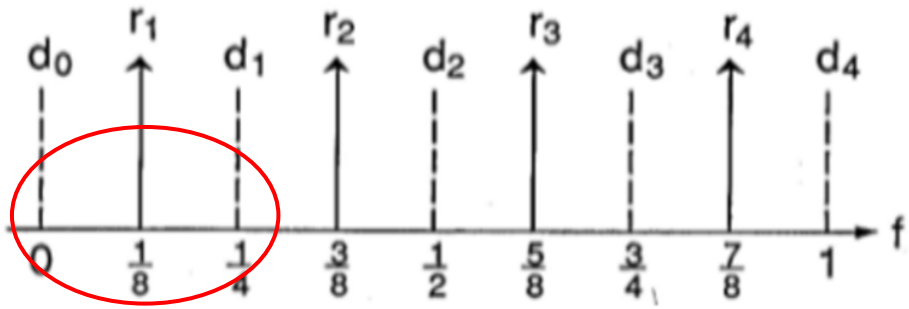
2-bit resolution with four levels



3-bit resolution with eight levels

Non-Uniform Quantization

r_i : reconstruction levels
d_i : decision bounaries



Quantization error/noise: $e_q = \hat{f} - f$

Uniform quantization: straightforward and natural, but not optimal

e.g., if f rarely falls between d_0 and d_1, r_1 is rarely used (thus only 3 levels are used actually).

Rearranging r_{1-4} so that they all lie between d_1 and d_4 makes more sense and reduces total quantization error.

Non-Uniform Quantization

r_i : reconstruction levels
d_i : decision bounaries

Optimally choose r_i and d_i

Assume $f_{MIN} \le f \le f_{MAX}$
L = the number of reconstruction levels
$p_f(f_0)$: probability density function for f

$$\text{Minimize } D = E[(\hat{f} - f)^2] = \int_{f_0=f_{MIN}}^{f_{MAX}} (\hat{f} - f_0)^2 \cdot p_f(f_0) df_0 \ , \ \hat{f} \in \{r_1, r_2, ..., r_L\}$$

Non-Uniform Quantization

We can write *D* as

$$D = \sum_{k=1}^{L} \int_{f_0=d_{k-1}}^{d_k} p_f(f_0)(r_k - f_0)^2 df_0$$

To minimize D, we need,

$$\frac{\partial D}{\partial r_k} = 0, \qquad 1 \le k \le L$$

$$\frac{\partial D}{\partial d_k} = 0, \qquad 1 \le k \le L-1$$

Non-Uniform Quantization

Solving the two sets of equations, we get,

$$\frac{\partial D}{\partial r_k} = 2\int_{f_0=d_{k-1}}^{d_k} p_f(f_0)(r_k - f_0)df_0 = 0$$

$$\frac{\partial D}{\partial d_k} = p_f(d_k)(r_k - d_k)^2 - p_f(d_k)(r_{k+1} - d_k)^2 = 0$$

Therefore,

$$r_k = \frac{\int_{f_0=d_{k-1}}^{d_k} f_0 p_f(f_0)df_0}{\int_{f_0=d_{k-1}}^{d_k} p_f(f_0)df_0}, \qquad 1 \le k \le L$$

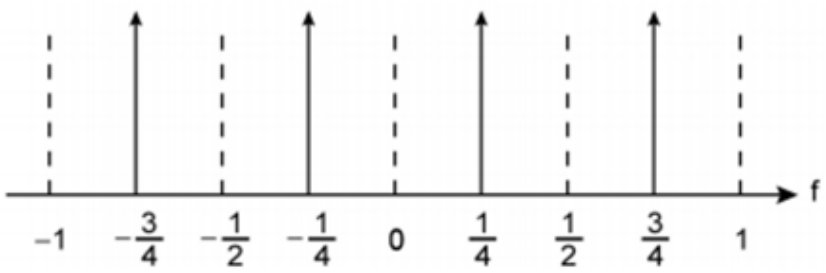
can be solved recursively, given proper initial values

$$d_k = (r_k + r_{k+1})/2, \qquad 1 \le k \le L-1, \qquad (d_0 = d_{\min}, d_L = d_{\max})$$

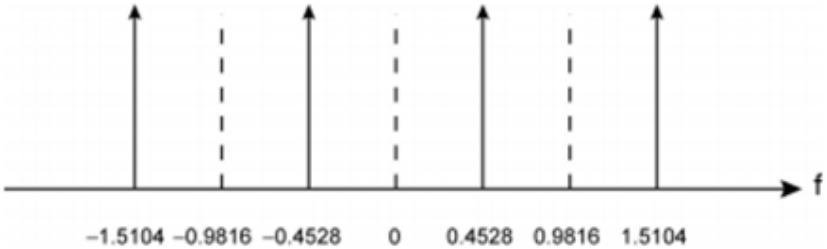
Non-Uniform Quantization

Solutions to optimal problems:

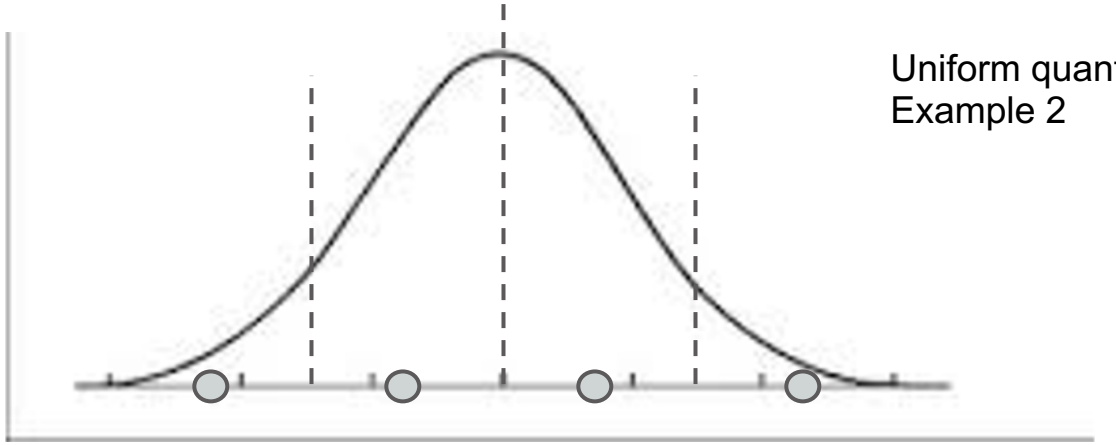
Example 1 $p_f(f_0)$: Uniform density, $-1 \le f_0 \le 1$
2 bits (4 reconstruction levels)



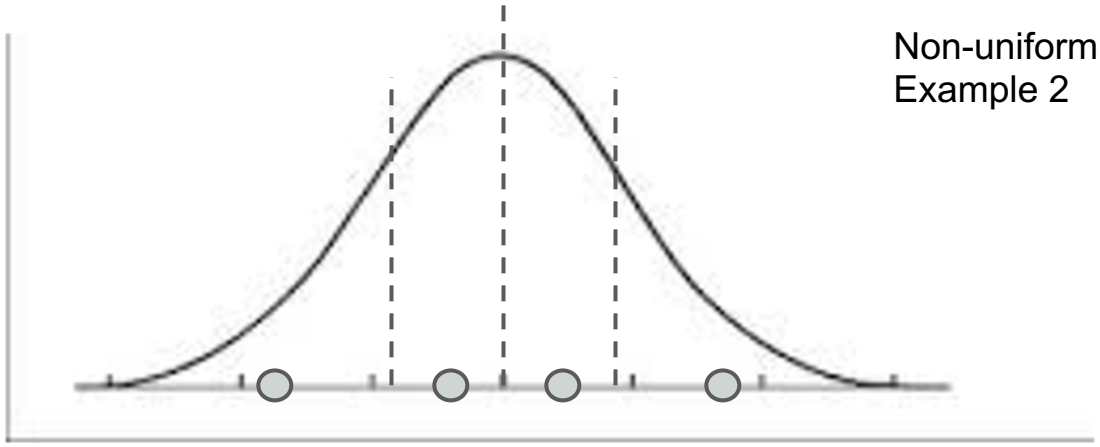
Example 2 $p_f(f_0)$: Gaussian with mean of 0 and variance of 1
2 bits (4 reconstruction levels)



Non-Uniform Quantization



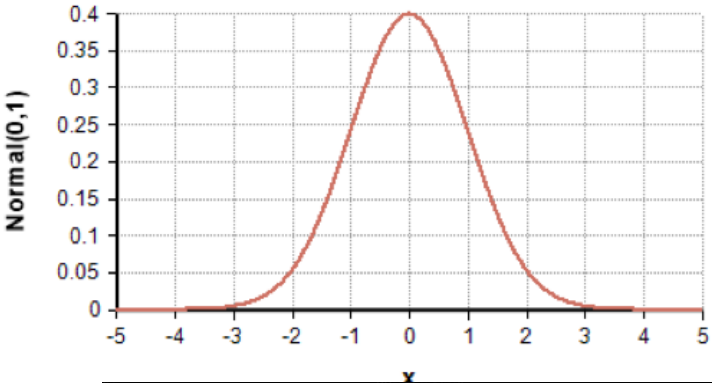
Uniform quantization for Example 2

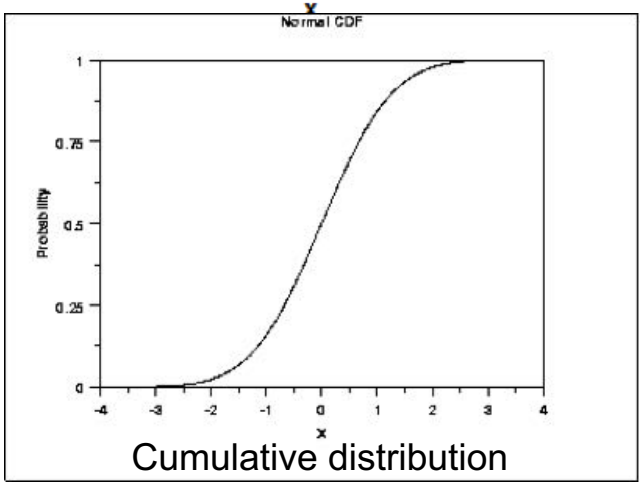


Non-uniform quantization for Example 2

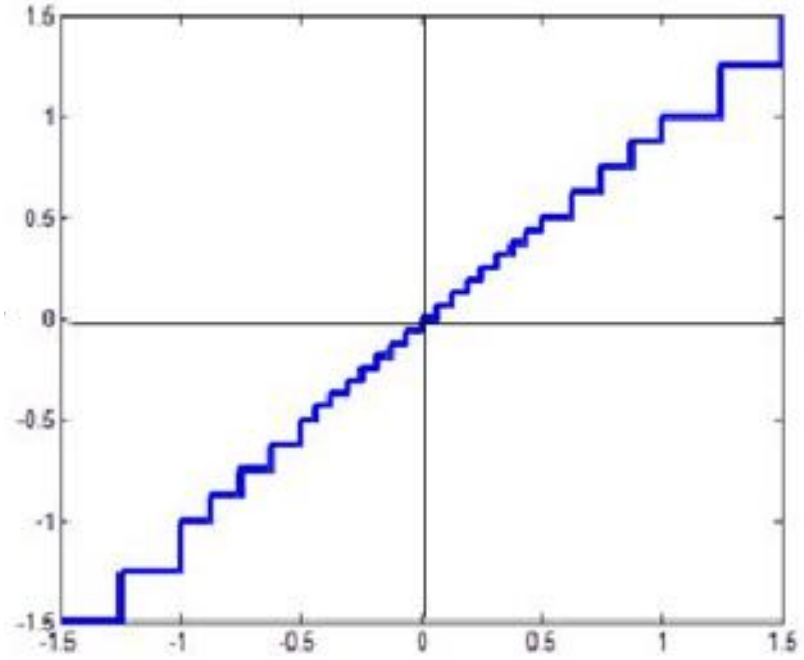
Non-Uniform Quantization

- Input signal follows Gaussian distribution





Cumulative distribution



Vector Quantization

- Sometimes scalar quantization is too limited
- For instance, a pixel of an image may have 3 channels (red, green, blue)

Just In Case...

- Modern images are usually composed of three color channels: Red, Green and Blue
- Each pixel is thus represented by an RGB vector [r,g,b] usually already quantized individually to 256 levels ([0,255])
- How many bits?

The three RGB colors are each **8**-bits (possible values [0.. 255], 2^8 = **256**)

Vector Quantization

- Instead of quantizing each scalar value (red,green,blue intensities) separately, we want to quantize the combined color (a vector)

Some finite levels of blues for the sky

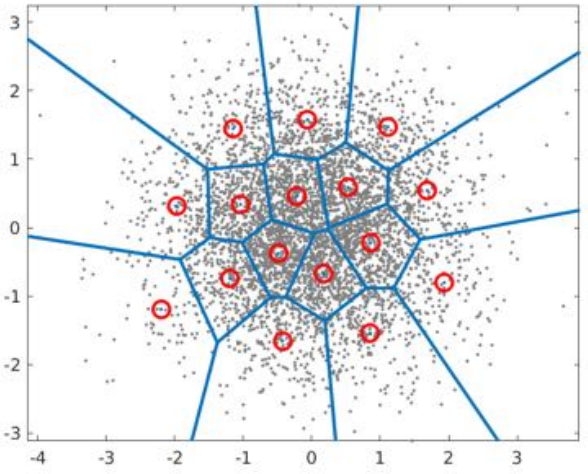
Some finite levels of aqua color for the water

Vector Quantization

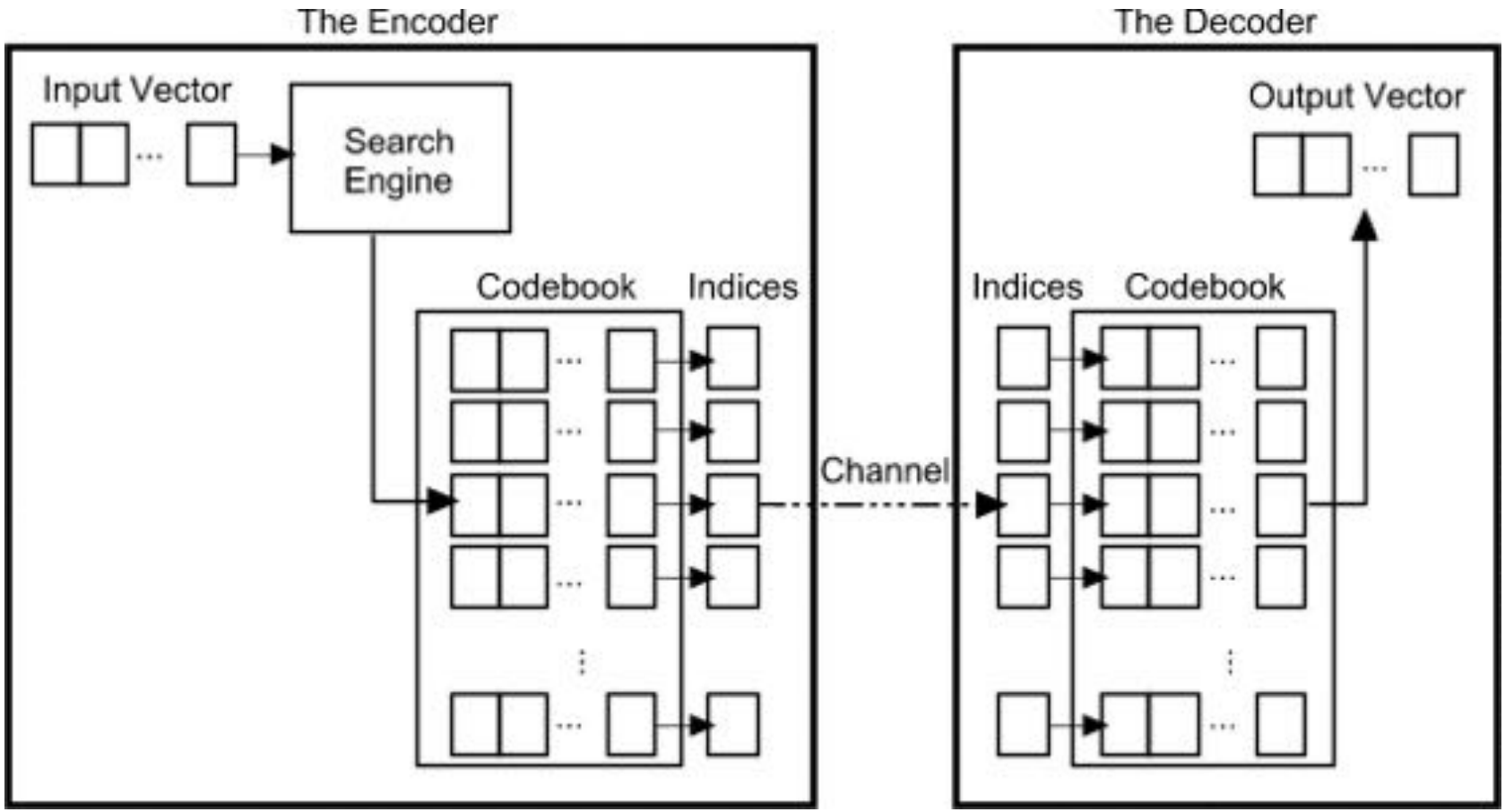
- Similar to scalar quantization, we need to have decision boundaries and reconstruction levels - but they will also now be vectors in the RGB space
- How to set them automatically?
  - Very similar to the non-uniform scalar quantization we have seen just now

Vector Quantization

- In Vector Quantization we call the set of reconstruction levels a **codebook** or **dictionary** and the space with each decision boundary a **cell**
- Using our intuition, the reconstruction levels could be the **center** (or more properly **centroids)** of these cells



Pipeline of Vector Quantization



K-Means Clustering

Necessary conditions for optimality: $\min E[d(\mathbf{f},\mathbf{r}_i)\,|\,\mathbf{f}\in C_i]$

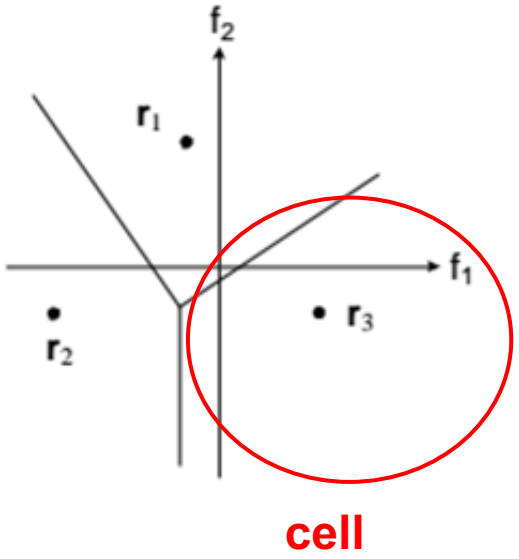
Condition 1:

Given reconstruction levels ($\mathbf{r}_1$, $\mathbf{r}_2$, $\mathbf{r}_3$, …)

Choice of cell boundaries:

For every **f**,

VQ(**f**) = $\mathbf{r}_i$ for smallest d(**f**, $\mathbf{r}_i$)



Condition 2:

Given cell boundaries

Choice of reconstruction levels:

$\mathbf{r}_i$ with minimum average d(**f**, $\mathbf{r}_i$) for **f** within cell boundary

$$\mathbf{r}_i = \arg\min E[d(\mathbf{f},\mathbf{r}_i)\,|\,\mathbf{f}\in C_i]$$

K-Means Clustering

Implications of the two conditions:

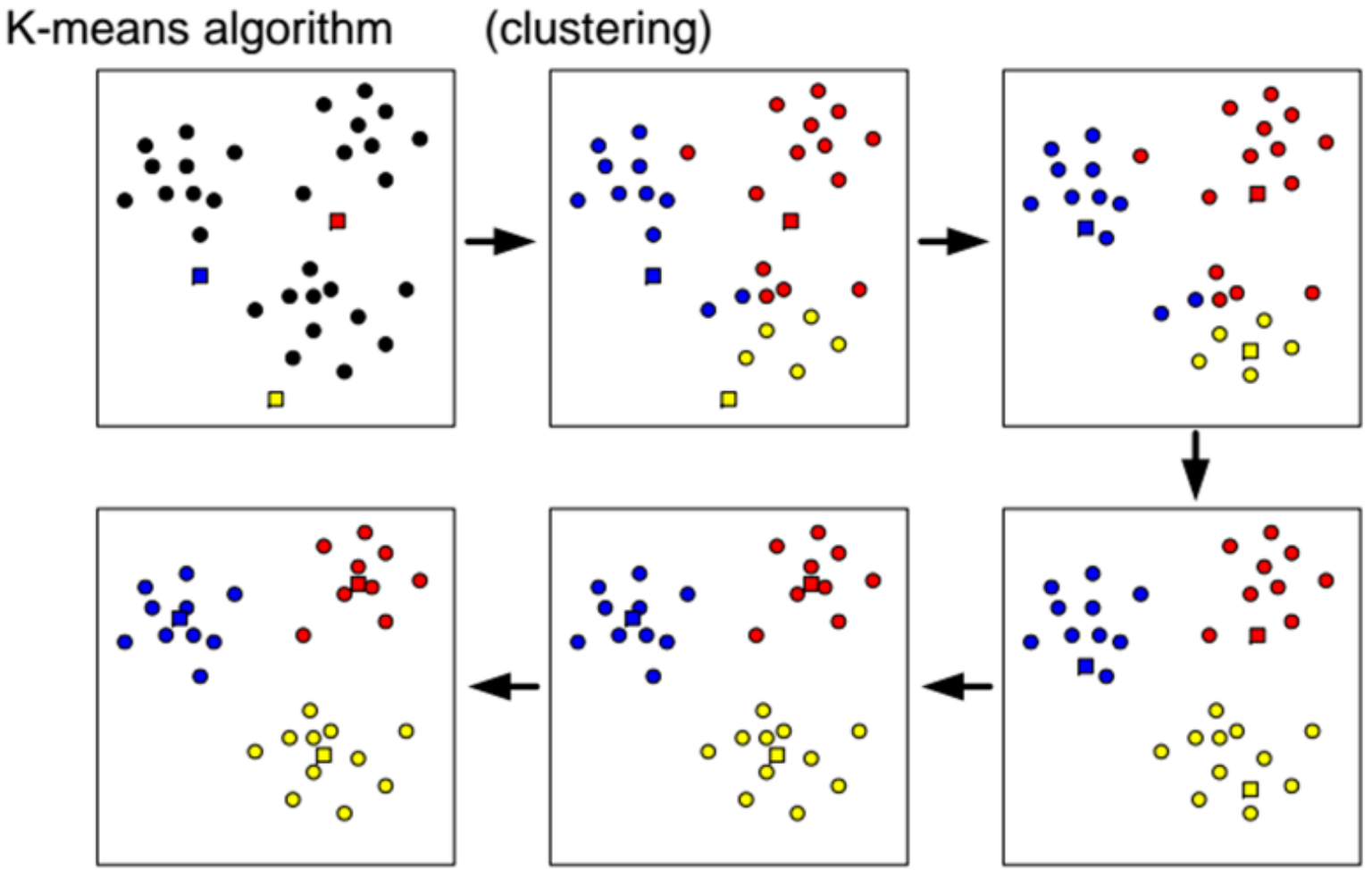
Reconstruction levels and cell boundaries: interdependent
Reconstruction levels $\leftrightarrow$ Cell boundaries
Codebook : Need to store only reconstruction levels

Iterative procedure
Initial reconstruction levels
↓
Cell boundaries
↓
Revised reconstruction levels
↓
Revised cell boundaries
↓
…

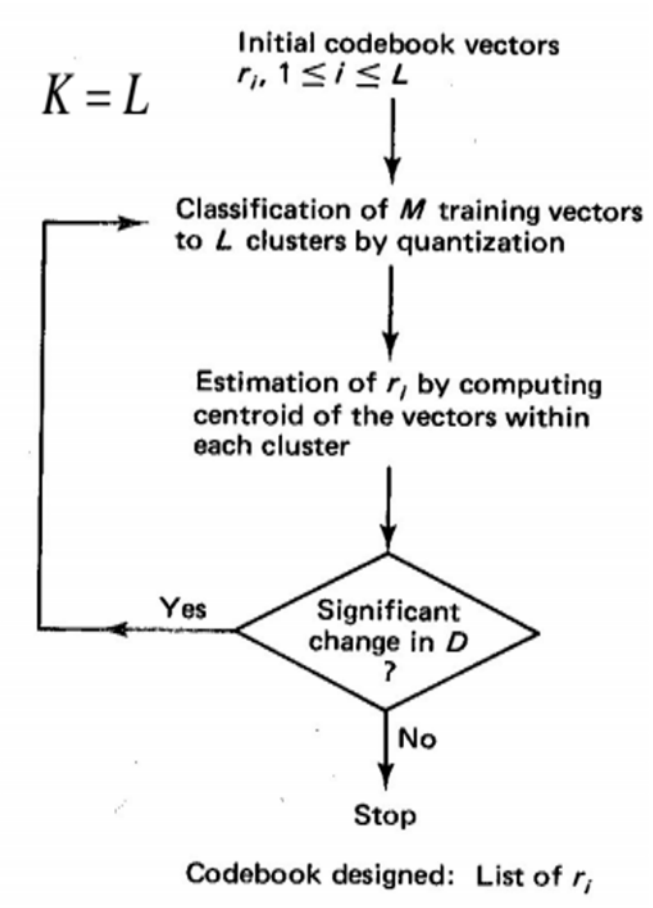
Difficulties:
(1) too many $d(\mathbf{f}, \mathbf{r}_i)$ to compute
(2) $\mathbf{r}_i$ is hard to compute since $p_{\mathbf{f}}(\mathbf{f}')$ is often unknown

Solution:
a set of training samples + K-means algorithm

K-Means Clustering



K-Means Clustering



Use a large set of M training data

Average distortion:

$$D = \frac{1}{M} \sum_{i=1}^{M} d(\mathbf{f}_i, \hat{\mathbf{f}}_i)$$

Interactive Demonstration

http://user.ceng.metu.edu.tr/~akifakkus/courses/ceng574/k-means/

K-Means Clustering

- Having K reconstruction levels means codebook of size K
  - Need only $\log_2(K)$ bits per pixel to store/transmit

Original
8 x 3 bits

K = 24 VQ
~5 bits

K = 64 VQ
6 bits

K-Means Clustering

- Most used in indexed color coding
- Supported in BMP, PCX, GIF and an option in PNG

Original
8 x 3 bits

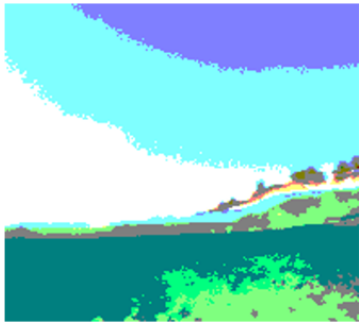
K = 24 VQ
~5 bits

K = 64 VQ
6 bits

Vector v.s. Scalar Quantization

- Clearly VQ is computationally more intensive
  - Requires training data
  - Clustering takes time (for transmitter only)
  - Receiver only needs to look up codebook
- As a trade-off, it offers potentially much better performance than scalar quantization
  - In previous example, scalar quantization could afford only 2-bits per color channel if we intend to keep the quantization output the same size

A simulated non-uniform scalar quantization result, 2 bits per color channel



Can We Improve Scalar Quantization?

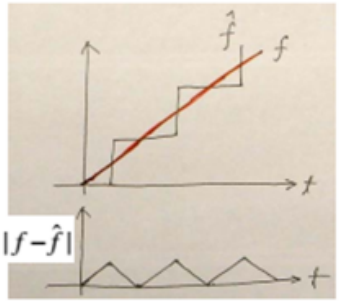
- Sometimes scalar quantization is needed
  - Hardware limitation etc.
- Can be considered a **pulse-code modulation (PCM)** problem (and have been under research long ago)

Original image

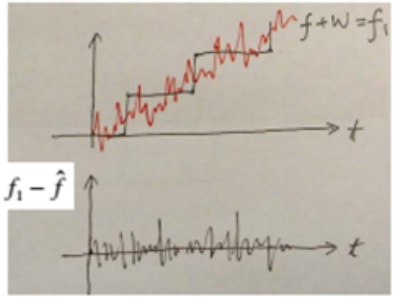
PCM-coded
2 bits/pixel
4 grey levels

False contours
(signal-dependent
quantization noise)

The signal-dependent noise can be periodic and very distracting to the human observer - not desirable



signal-dependent quantization noise



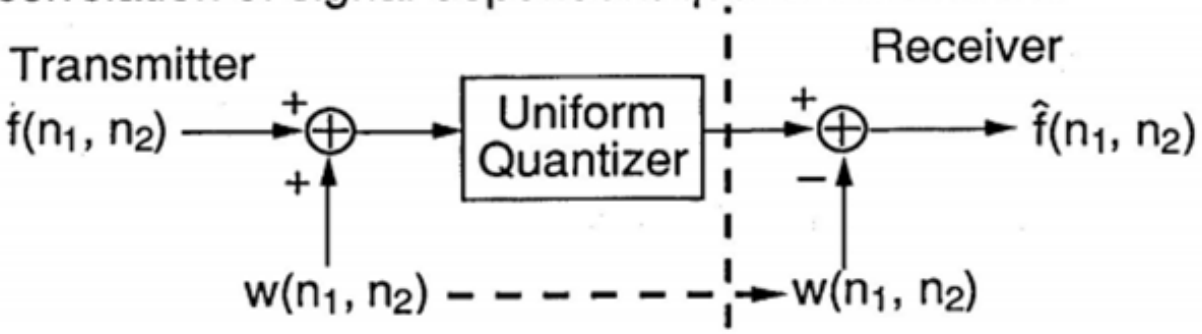
signal-independent quantization noise

Adding Pseudo Noise

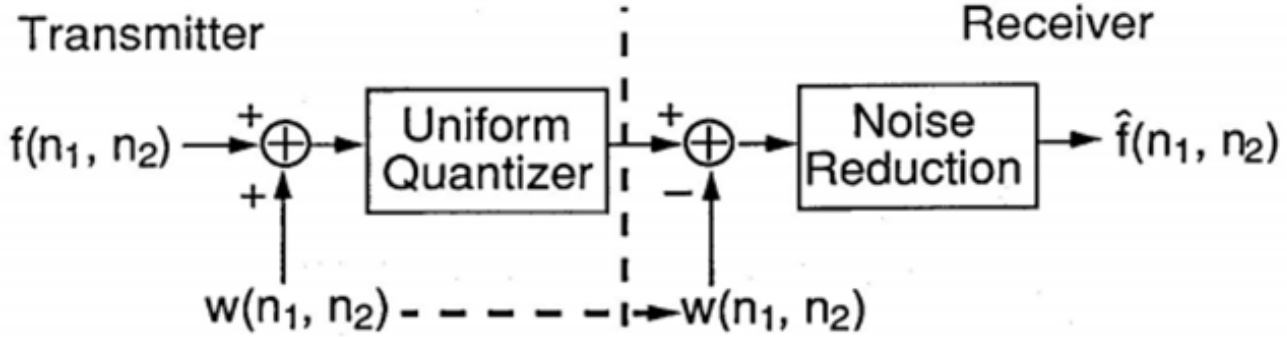
Improvements of PCM

- Roberts's pseudonoise technique

  Decorrelation of signal-dependent quantization noise



- Reduction of quantization noise



Adding Pseudo Noise

Original image

PCM-coded
2 bits/pixel
(4 grey levels)

PCM-coded
2 bits/pixel
(4 grey levels)
with Roberts's
pseudonoise
technique

PCM-coded
2 bits/pixel
(4 grey levels)
with Roberts's
pseudonoise
technique +
noise reduction

Adding Pseudo Noise

- Original paper: http://www.packet.cc/files/pic-code-noise.html
- Gives you a perspective of researchers trying to solve multimedia problems long before use of computers

Dithering

- An well-designed applied noise to randomize quantization error
- In this extreme example we only have 2 quantization levels - black and white
  - "Threshold" is generic scalar quantization and "Random" is adding pseudo noise
  - Both not particularly acceptable

Effect of Color Banding

8-bit gradient

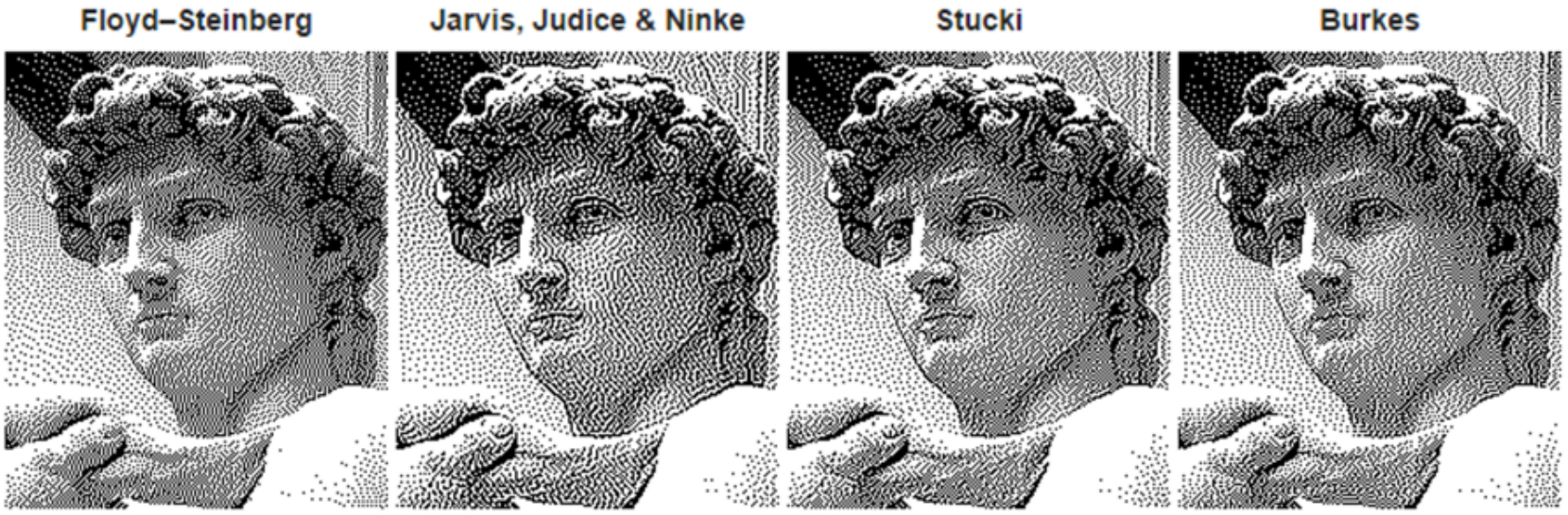
8-bit gradient, dithered

Effect of Color Banding

original picture | no dithering | Floyd–Steinberg dithering

Dithering

- Concept of error diffusion
  - Quantization residual is distributed to neighboring pixels that have not yet been processed
  - Invented for fax machines and black-and-white copiers where very coarse quantization is a must



Sierra

Two-row Sierra

Sierra Lite

Atkinson

How Dithering works

Error diffusion: push the residual quantization error of a pixel onto its neighboring pixels

Convolution kernel

$$\begin{bmatrix} & * & \frac{7}{16} & \cdots \\ \cdots & \frac{3}{16} & \frac{5}{16} & \frac{1}{16} & \cdots \end{bmatrix}$$

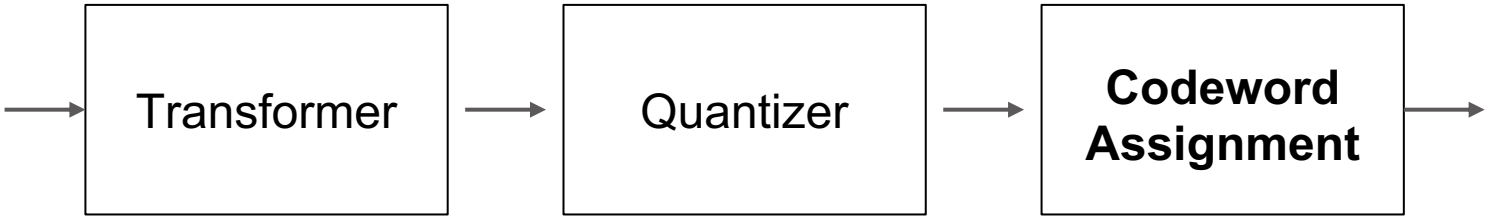
```
for each y from top to bottom do
    for each x from left to right do
        oldpixel := pixel[x][y]
        newpixel := find_closest_palette_color(oldpixel)
        pixel[x][y] := newpixel
        quant_error := oldpixel - newpixel
        pixel[x + 1][y    ] := pixel[x + 1][y    ] + quant_error × 7 / 16
        pixel[x - 1][y + 1] := pixel[x - 1][y + 1] + quant_error × 3 / 16
        pixel[x    ][y + 1] := pixel[x    ][y + 1] + quant_error × 5 / 16
        pixel[x + 1][y + 1] := pixel[x + 1][y + 1] + quant_error × 1 / 16
```

https://en.wikipedia.org/wiki/Floyd%E2%80%93Steinberg_dithering

Dithering in Modern Times

- A cool video game with dithering effect
  - https://dukope.itch.io/return-of-the-obra-dinn

Codeword Assignment



- We have seen some simple design choices we have for **quantization**
- We will now move on to **codeword assignment**
  - Fixed-length coding (FLC)
  - Variable-length coding (VLC)

Fixed-Length Coding

- We code a quantized symbol into bits of a certain length
- Least trouble: fixed length coding
  - 8 symbols (levels of quantization)
  - 3 bits!

L = 8	3 bits
r_0	0 0 0
r_1	0 0 1
r_2	0 1 0
r_3	0 1 1
r_4	1 0 0
r_5	1 0 1
r_6	1 1 0
r_7	1 1 1

0010101010000011111110

↓

001 010 101 000 001 111 110

↓

r1 r2 r5 r0 r1 r7 r6

Variable-Length Coding

- We code a quantized symbol into bits of a certain length
- Least trouble: check the total number of symbols and determine a code length
  - e.g. 64 symbols -> 6 bits minimum
- Can we use less bits?
  - Intuition: if we use more bits for more rare symbols and less bits for more frequent symbols, will we be using less bits, as a whole?
  - e.g. use 1 bit for most frequent color, and many bits for less used color

Uniquely Decodable

- To use variable-length coding we need to make sure code is uniquely decodable
  - Codeword are assumed to be received sequentially on the receiver side

L=4 (uniquely decodable)		L=4 (not uniquely decodable)		
r_0	0 0	r_0	0	
r_1	0 1	r_1	1	
r_2	1 0	r_2	0 0	r_2 or two r_0?
r_3	1 1	r_3	0 1	

Entropy

From Information Theory, the entropy of a discrete random variable *X* with possible values $\{x_1,\ldots,x_n\}$ and probability mass function *P(X)* is written as:

$$H(X) = \sum_i P(x_i)\, I(x_i) = -\sum_i P(x_i) \log_b P(x_i)$$

Here *I* is called the information content of *X*

Entropy

Message possibility	Probability	
a_1	$p_1 = 1$	$H =$?
a_2	$p_2 = 0$	

Message possibility	Probability	
a_1	$p_1 = \frac{1}{2}$	$H =$?
a_2	$p_2 = \frac{1}{2}$	

What is the entropy in the above 2 cases?

Entropy

First case is trivial (a zero in both terms)

Second case:

$$\begin{aligned}\mathrm{H}(X) &= -\sum_{i=1}^{n} \mathrm{P}(x_i) \log_b \mathrm{P}(x_i) \\ &= -\sum_{i=1}^{2} \frac{1}{2} \log_2 \frac{1}{2} \\ &= -\sum_{i=1}^{2} \frac{1}{2} \cdot (-1) = 1\end{aligned}$$

Entropy

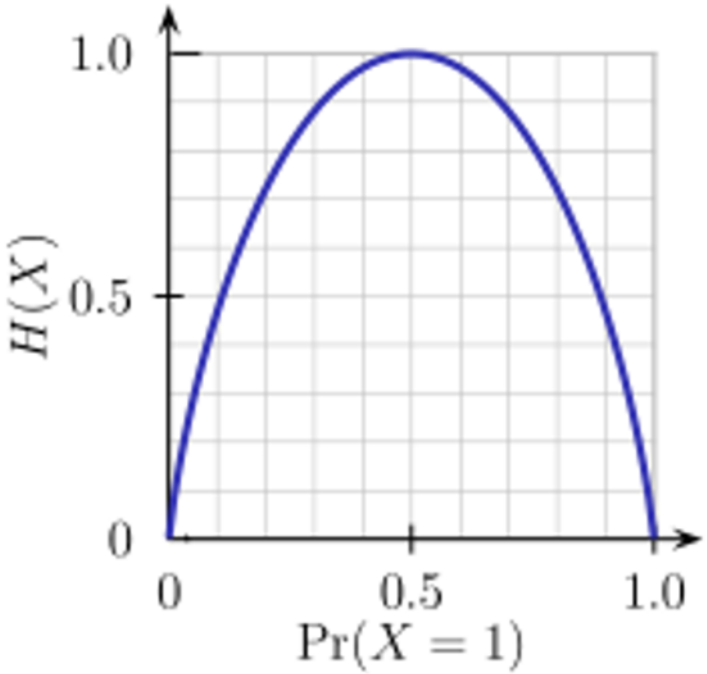
Message possibility	Probability	
a_1	$p_1 = 1$	$H = 0$
a_2	$p_2 = 0$	

Message possibility	Probability	
a_1	$p_1 = \frac{1}{2}$	$H = 1$
a_2	$p_2 = \frac{1}{2}$	(maximum entropy for L=2)

The more "uncertainty" a message resolves, the more information it contains.

Entropy

Plotted results for different coin probabilities: A binary entropy function



Does it fit your intuition?

Entropy and Coding

In information theory, *H* is the theoretically minimum possible average bit rate required to code a message.

Definition of entropy does not specify a method to design codewords, but is useful in practice, since if the average bit rate is close to *H*, the coder is considered efficient.

Example:

Message possibility	Probability	Codeword
a_1	$p_1 = 1/4$	0 0
a_2	$p_2 = 1/4$	0 1
a_3	$p_3 = 1/4$	1 0
a_4	$p_4 = 1/4$	1 1

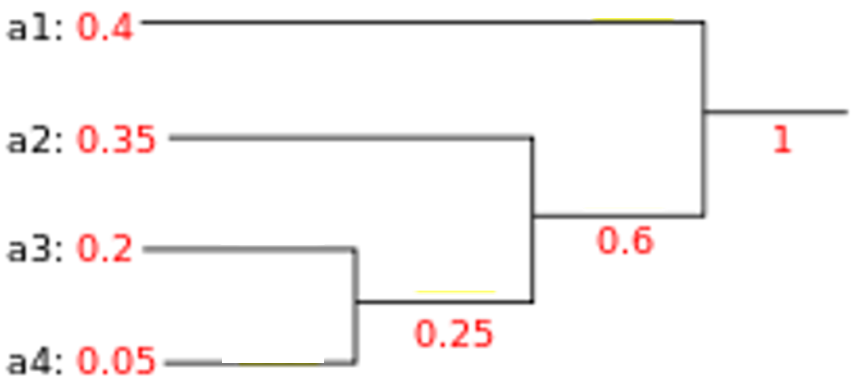
Entropy $H = 2$, average bit rate = 2 bits/message

Huffman Coding

- A simple optimal prefix code
  - **Prefix code** means no valid code word is a prefix of any other valid code word
  - This implies code word can be uniquely decoded in our case
  - Let's look at an example

Huffman Coding

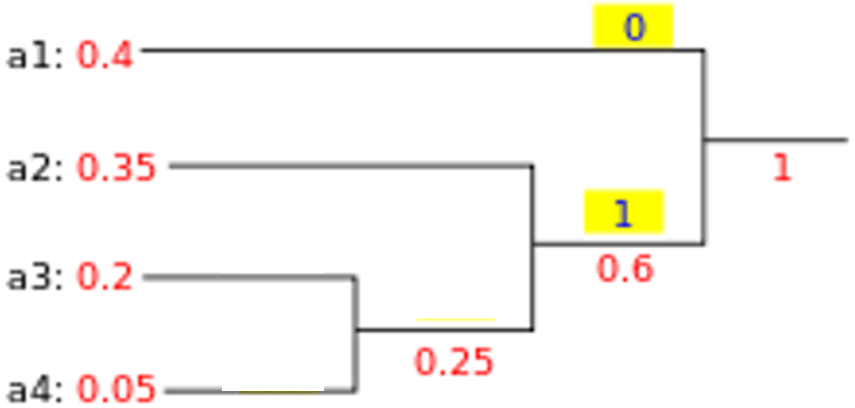
- Step 1: Create a binary tree of nodes for each symbol
  - Start with the least probable, combine the lowest probability pair to form a new node
  - the new node has a probability equal to sum of the probability of its children



Source: en.wikipedia.org

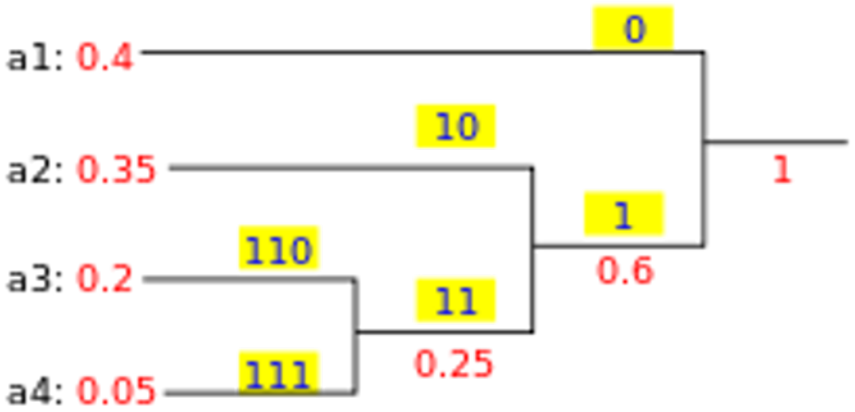
Huffman Coding

- Step 2: After tree is complete (with root node), start assigning code word from the right
  - Give 0 to top/left child and 1 to bottom/right child
  - Go along the tree, increase code length by 1 each time



Huffman Coding

- Step 3: Final Huffman Code would be the code assigned to each leaf node
  - Note that they are all unique decodable
  - The rarest symbols would have the longest code words



Symbol	Final Code
a1	0
a2	10
a3	110
a4	111

Huffman Coding

- Near optimal coding
  - In the above example: 1.85 bits/symbol versus 1.74 bits/symbol theoretical limit
  - Linear computational time
- Example on right: English alphabet

Symbol	Frequency	Huffman Code
[space]	67962112	111
e	37907119	010
t	28691274	1101
a	24373121	1011
o	23215532	1001
i	21820970	1000
n	21402466	0111
s	19059775	0011
h	18058207	0010
r	17897352	0001
l	11730498	10101
d	10805580	01101
c	8982417	00001
u	8022379	00000
f	7486889	110011
m	7391366	110010
w	6505294	110001
y	5910495	101001
p	5719422	101000
g	5143059	011001
b	4762938	011000
v	2835696	1100000
k	1720909	11000011
x	562732	110000100
j	474021	1100001011
q	297237	11000010101
z	93172	11000010100

Huffman Coding

- Let's try! Let's code the following in Fixed coding v.s. Huffman coding
  - "Hello"
  - "Henry"
  - "Qin Zhou"

Symbol	Frequency	Huffman Code
[space]	67962112	111
e	37907119	010
t	28691274	1101
a	24373121	1011
o	23215532	1001
i	21820970	1000
n	21402466	0111
s	19059775	0011
h	18058207	0010
r	17897352	0001
l	11730498	10101
d	10805580	01101
c	8982417	00001
u	8022379	00000
f	7486889	110011
m	7391366	110010
w	6505294	110001
y	5910495	101001
p	5719422	101000
g	5143059	011001
b	4762938	011000
v	2835696	1100000
k	1720909	11000011
x	562732	110000100
j	474021	1100001011
q	297237	11000010101
z	93172	11000010100

Limits of Huffman Coding

- Huffman coding requires probability distribution to be known in prior
- Can only code symbol by symbol
  - Base on an assumption that the symbols are independent and identically distributed
- Is there a codeword assignment method...
  - that is adaptive to changing input statistics, and
  - can combine symbols for even more efficient coding? (e.g. 'ist' often come together in English, or equivalent patterns in images and sound)

Multiple-Symbols Encoding

For simplicity, let's use **text as example** - intuitive steps would be:

- reading in input symbols
- looking for groups of symbols that appear in a dictionary
- outputting a pointer or index to the phrase dictionary (instead of the codes for single symbols) if a string match is found.

The longer the match is, the better the compression ratio. The method takes advantage of the correlation within the input.

Two types:

- static dictionary
- adaptive dictionary

Static Dictionary Case

Digram coding:

The dictionary consists of all letters of the source alphabet followed by as many pairs of letters, called *digrams*, as can be accommodated by the dictionary.

Example: Suppose we have a source with five letter alphabet $\{a,b,c,d,r\}$. Based on knowledge about the source, we build the dictionary:

Code	Entry	Code	Entry
000	a	100	r
001	b	101	ab
010	c	110	ac
011	d	111	ad

Static Dictionary Case

To encode the sequence *abracadabra*, we have:

ab:	101
ra:	not in the dictionary
r:	100
ac:	110,
Output:	101100110111101100000

Code	Entry	Code	Entry
000	a	100	r
001	b	101	ab
010	c	110	ac
011	d	111	ad

More efficient? Yes, because 21 bits < 11 x 3 = 33 bits.

Static Dictionary Example

TABLE 5.2 **Thirty most frequently occurring pairs of characters in a 41,364-character-long LaTeX document.**

Pair	Count	Pair	Count
e␣	1128	*ar*	314
␣t	838	*at*	313
␣␣	823	*␣w*	309
th	817	*te*	296
he	712	*␣s*	295
in	512	*d␣*	272
s␣	494	*␣o*	266
er	433	*io*	257
␣a	425	*co*	256
t␣	401	*re*	247
en	392	*␣$*	246
on	385	*r␣*	239
n␣	353	*di*	230
ti	322	*ic*	229
␣i	317	*ct*	226

␣: a space

Static Dictionary Example

TABLE 5.3 **Thirty most frequently occurring pairs of characters in a collection of C programs containing 64,983 characters.**

Pair	Count	Pair	Count
␣␣	5728	st	442
nl␣	1471	le	440
;nl	1133	ut	440
in	985	f(	416
nt	739	ar	381
=␣	687	or	374
␣i	662	r␣	373
t␣	615	en	371
␣=	612	er	358
);	558	ri	357
,␣	554	at	352
nlnl	506	pr	351
␣f	505	te	349
e␣	500	an	348
␣*	444	lo	347

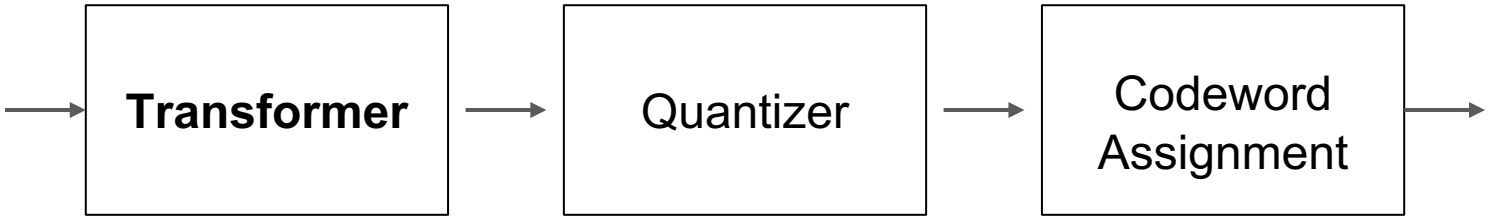
␣: a space; *nl*: a new line

Limitations of Static Dictionary

- Limitations:
  - A dictionary made for one type of document is good for that type of document only
  - Dictionary has to be **transmitted** along with the coded multimedia - high overhead!
- If we want to be better than Huffman Coding we need to be more **adaptive**

Next Chapter…

- LZ77, LZ78 and LZW - codeword assignment with **adaptive dictionary**
  - Used in GIF and widely used in text compression
- Also the missing step 1…



?

Scalar quantization
- Uniform
- Non-uniform
- With pseudo-noise

Vector quantization
- K-Means

Fixed length coding

Variable length-coding
- Huffman Coding

Multiple symbols coding
- Static Dictionary
- **Adaptive Dictionary**