IERG4190/IEMS5707 Multimedia Coding and Processing

Chapter 3: Multimedia Coding (2)

Bolei Zhou
Department of Information Engineering, CUHK
Spring 2021

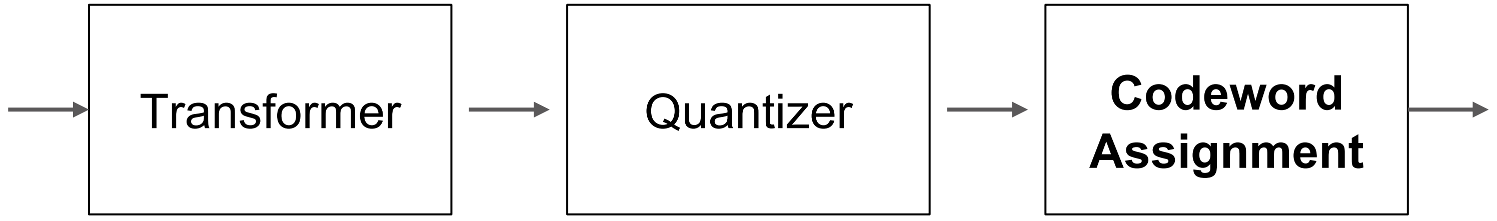
Announcement

- Assignment 1 is out
- Due 18:00 pm Feb 6, 2021
- On quantization, Huffman coding, LZ77/LZW

Outline

1. LZ77
2. LZ78
3. LZW
4. LTI Systems
5. Review on Fourier Analysis
6. LSI Systems
7. Discrete Cosine Transform (DCT)

Last Time...



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**

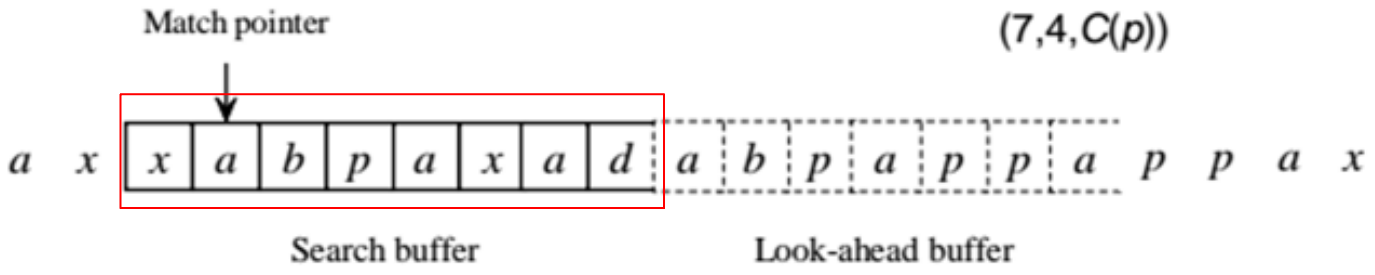
Multi-symbol Coding with Adaptive Dictionary

Instead of having a completely defined dictionary when compression begins:

- Start with no dictionary or a default baseline dictionary
- As compression proceeds, new phrases are added to dictionary
- Used in LZ77 and LZ78
    - The 77 means 1977….

LZ77

In the LZ77 approach, the dictionary is simply a portion of the previously encoded sequence. The encoder examines the input sequence through a sliding window:



Search buffer: contains a portion of the recently encoded sequence, with usually several thousand characters.
Look-ahead buffer: contains the next portion of the sequence to be encoded, with usually ten to one hundred characters.

LZ77

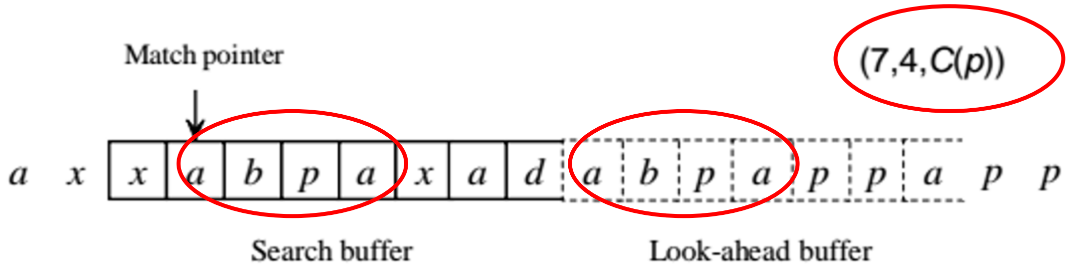
To encode the sequence in the look-ahead buffer, the encoder moves a search pointer back through the search buffer until it encounters a match to the first symbol in the look-ahead buffer.

Offset: the distance of the pointer from the look-ahead buffer.

Length-of-match: the number of consecutive symbols in the search buffer that match consecutive symbols in the look-ahead buffer, starting with the first symbol.

Once the longest match is found, the encoder encodes a triple:

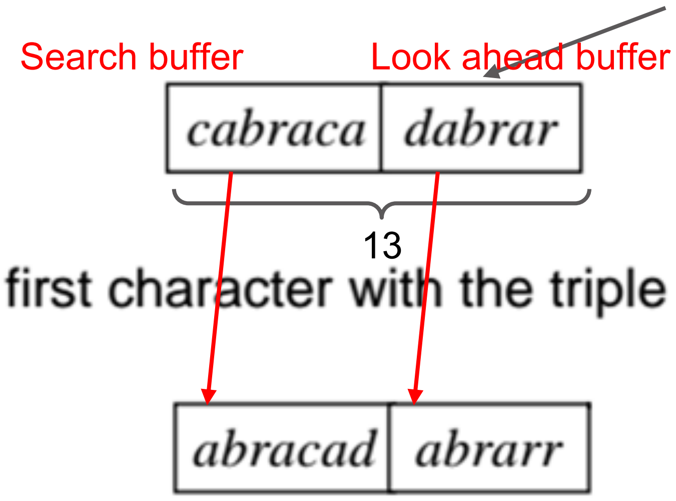
<the offset, the length of match, the symbol following the match>



C(p) means the code for character p, e.g. in ascii, p's code is 112

LZ77 - Example

To code the sequence, …*cabracadabrarrarrad*…, we use a window of length 13, and the look-ahead buffer size is 6.



First symbol in look-ahead buffer is 'd';
There's no 'd' in search buffer;
So match is at 0 point with 0 length...

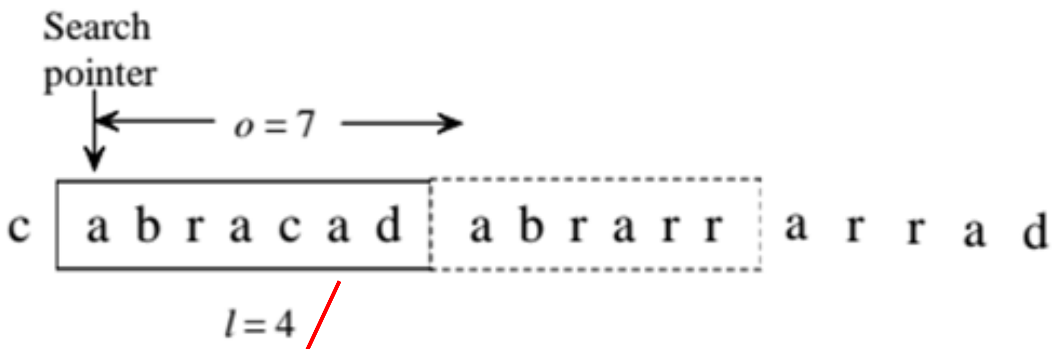
After coding the first character with the triple $\langle 0, 0, C(d) \rangle$, we have,

Moved forward by 1 character

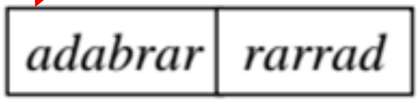
The symbol 'd' is encoded in the triple, so we start with the character after 'd'… i.e. search pointer + 1

Then, search again

LZ77 - Example



We then code the string *abra* with the triple $\langle 7, 4, C(r) \rangle$. The window now has:



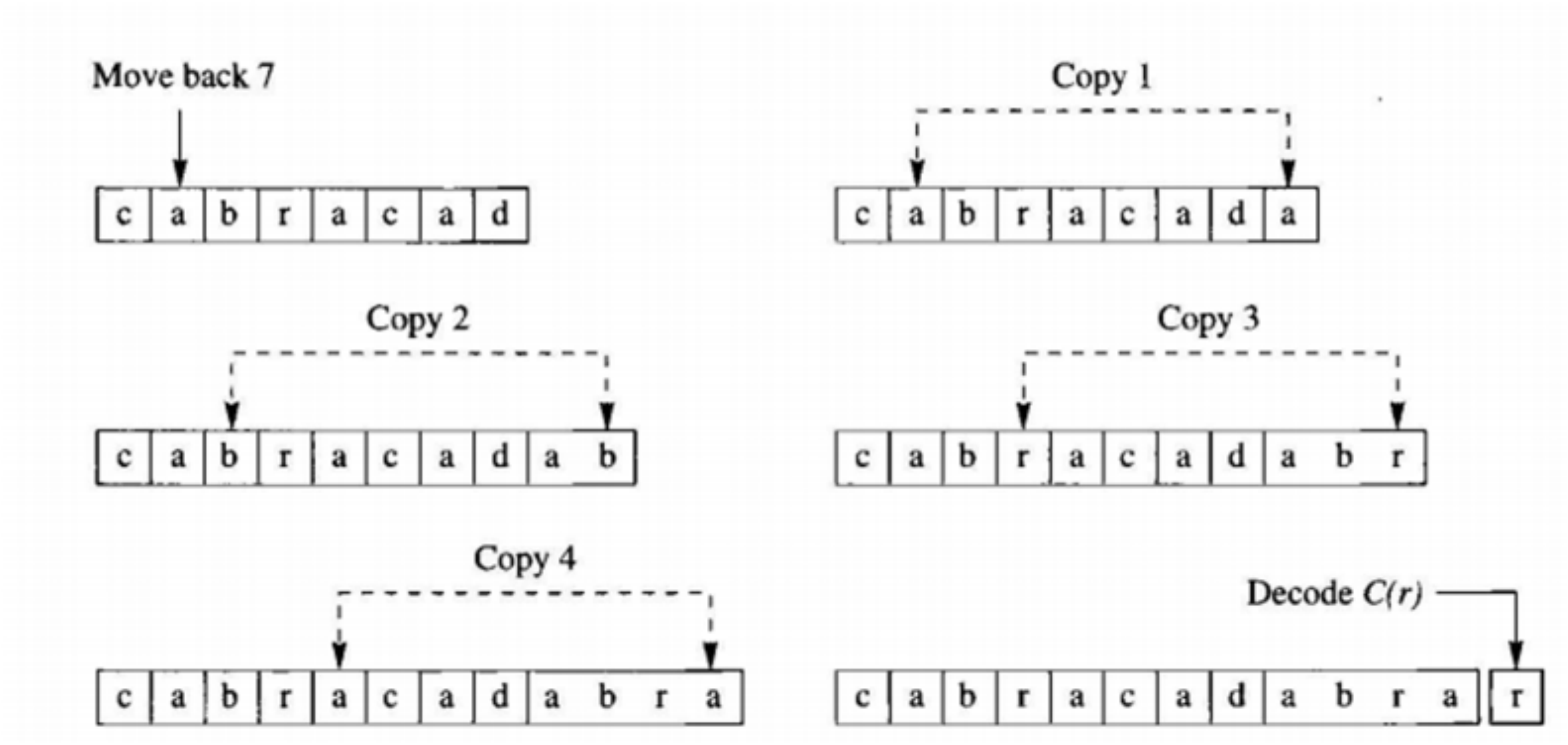
So 'abra' and 'r' is encoded...let's move forward by 5...

We continue to code the next string *rarra* as $\langle 3, 5, C(d) \rangle$.

Yes, you can go beyond the search buffer boundary

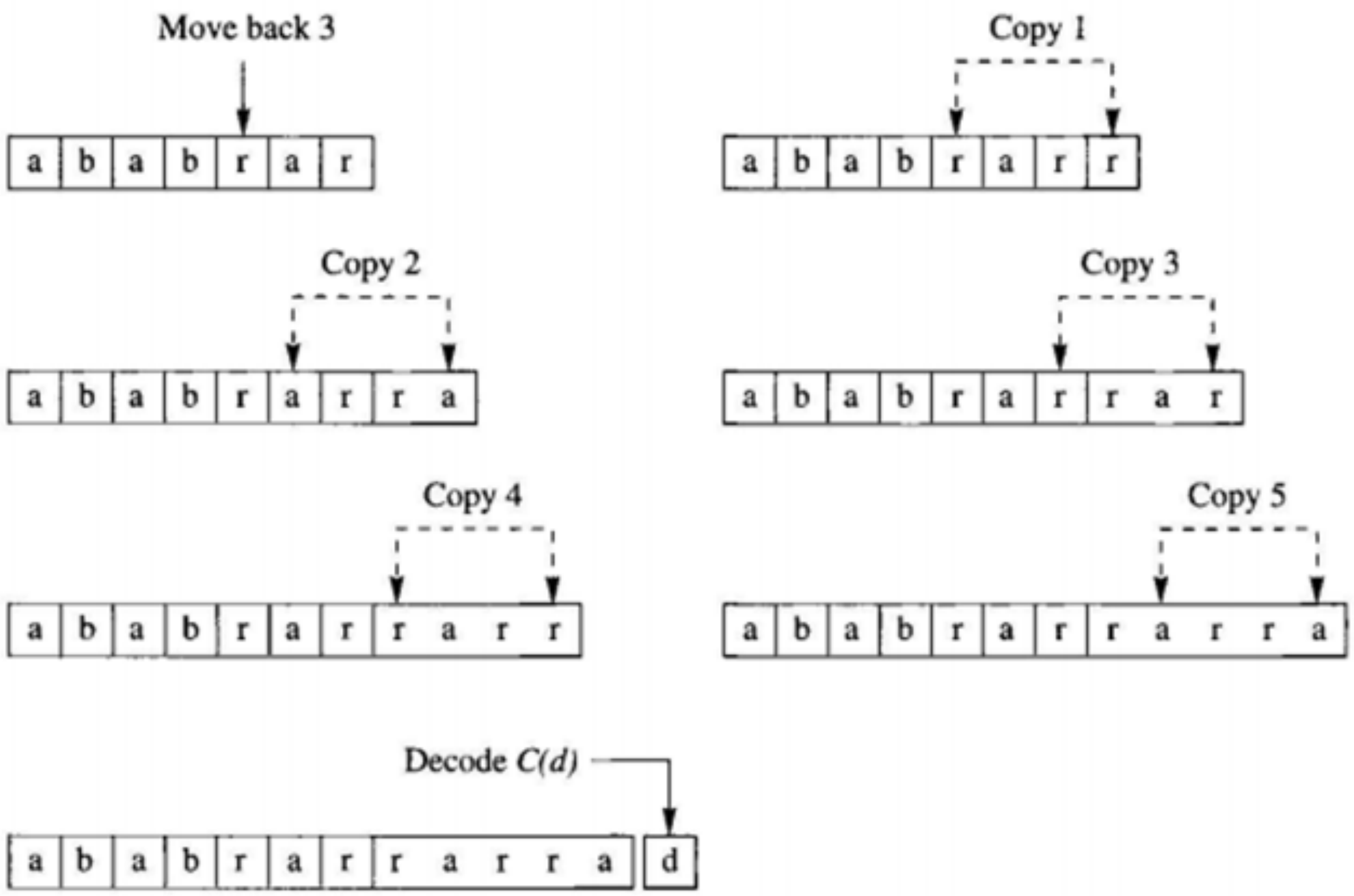
LZ77 - Decode Example

Decoding of the triple $\langle 7, 4, C(r) \rangle$



LZ77 – Decode Example

Decoding of the triple $\langle 3,5,C(d)\rangle$



LZ77 - Another Example

Search Buffer (6)	Look-Ahead Buffer (4)	Remaining	Output
	abab	cbabab	0,0,C(a)
a	babc	babab	0,0,C(b)
ab	abcb	abab	2,2,C(c)
ababc	baba	b	4,3,C(a)
bcbaba	b		2,1,end

LZ77 - Limitations

Very simple adaptive scheme that requires no any prior knowledge of the source. The dictionary is the search window.

Problems with LZ77:

The algorithm uses only a small window into previously seen text, which means it continuously throws away valuable phrases because they slide out of the dictionary.

The limited lengths of the two buffers limit the size of a phrase that can be matched.

Worst case situation is that the sequence to be encoded is periodic with a period longer than the search buffer.

e. g. $\boxed{\text{abcdefghij}}\text{kabcdefghijk}$

What's wrong with this?!

LZ78

LZ77:

The dictionary of phrases is defined by a fixed window of previously seen text.

It outputs a series of tokens (triples):

<the offset, the length of match, the single symbol following the match>

LZ78:

The dictionary is a potentially unlimited list of previously seen phrases. It is built at both the encoder and decoder in an identical manner.

It outputs a series of tokens (doubles):

<the index selecting a given phrase, the single symbol following the phrase>

i.e. dictionary index

LZ78

LZ78 codes the input as a double <*i*,*c*> :

i : an index corresponding to the dictionary entry that is the longest match to the input. It's 0 in the case of no match.

c : the code for the character in the input following the matched portion of the input.

This double then becomes the newest entry in the dictionary.

Each new entry into the dictionary is one new symbol concatenated with an existing dictionary entry.

LZ78 - Example

As an example, we encode the following sequence using LZ78,

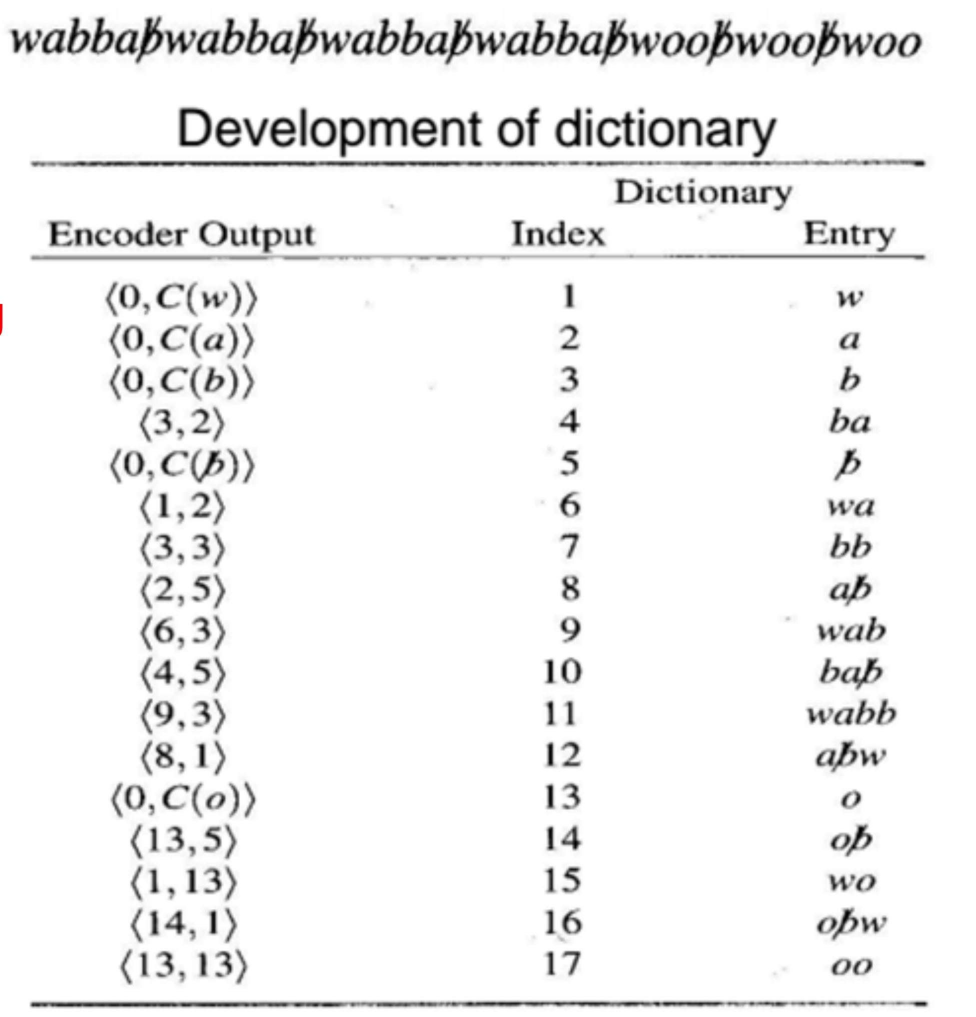
$$wabba\not{b}wabba\not{b}wabba\not{b}wabba\not{b}woo\not{b}woo\not{b}woo$$

The first three encoder outputs are $\langle 0, C(w)\rangle$, $\langle 0, C(a)\rangle$, and $\langle 0, C(b)\rangle$, which form the initial dictionary:

Index	Entry
1	*w*
2	*a*
3	*b*

LZ78 - Example

The dictionary is being built up as we encode (and decode)



wabbaƀwabbaƀwabbaƀwabbaƀwooƀwooƀwoo

Development of dictionary

Encoder Output	Dictionary Index	Dictionary Entry
$\langle 0, C(w)\rangle$	1	*w*
$\langle 0, C(a)\rangle$	2	*a*
$\langle 0, C(b)\rangle$	3	*b*
$\langle 3, 2\rangle$	4	*ba*
$\langle 0, C(ƀ)\rangle$	5	*ƀ*
$\langle 1, 2\rangle$	6	*wa*
$\langle 3, 3\rangle$	7	*bb*
$\langle 2, 5\rangle$	8	*aƀ*
$\langle 6, 3\rangle$	9	*wab*
$\langle 4, 5\rangle$	10	*baƀ*
$\langle 9, 3\rangle$	11	*wabb*
$\langle 8, 1\rangle$	12	*aƀw*
$\langle 0, C(o)\rangle$	13	*o*
$\langle 13, 5\rangle$	14	*oƀ*
$\langle 1, 13\rangle$	15	*wo*
$\langle 14, 1\rangle$	16	*oƀw*
$\langle 13, 13\rangle$	17	*oo*

Notes on LZ77 & LZ78

- Both encoder and decoder will come up with same dictionary
  - No dictionary has to be sent
- LZ77 is not necessarily inferior
  - Used with Huffman Coding in DEFLATE
  - DEFLATE is used in many applications, including originally in PKZip (general purpose) and eventually become a compression standard ("zip")
  - Also in a very modern standard, PNG (image compression)

LZW

LZW is the most well-known improvement to LZ78, proposed by Terry Welch.

Instead of coding the double $\langle i, c \rangle$, the LZW encodes only the index to the dictionary. In order to do this, an initial dictionary containing all the letters of the source alphabet has to be built before hand.

If a pattern p is contained in the dictionary, while the addition of the follow up letter a results in a pattern $p * a$ ($*$ denotes concatenation) that is not in the dictionary, then the index of p is transmitted to the receiver, and the pattern $p * a$ is added to the dictionary.

We then start another pattern with the letter a.

Why? Because unlike in LZ78, we haven't yet encoded the 'a'!

LZW - Example

Again, we look at the example to encode the sequence,

wabbaƀwabbaƀwabbaƀwabbaƀwooƀwooƀwoo

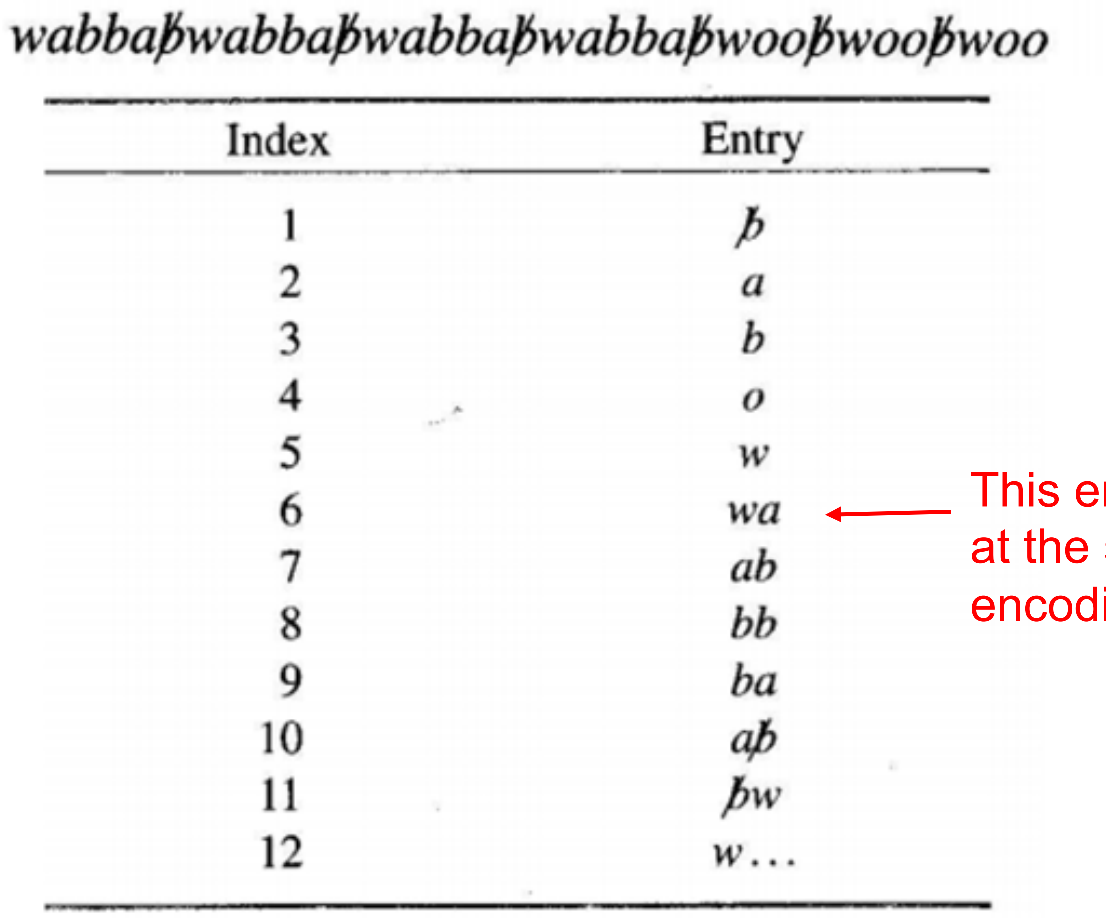
Initial LZW dictionary

This dictionary is premade; no combination of symbols

Index	Entry
1	*ƀ*
2	*a*
3	*b*
4	*o*
5	*w*

LZW - Example

Building the LZW dictionary

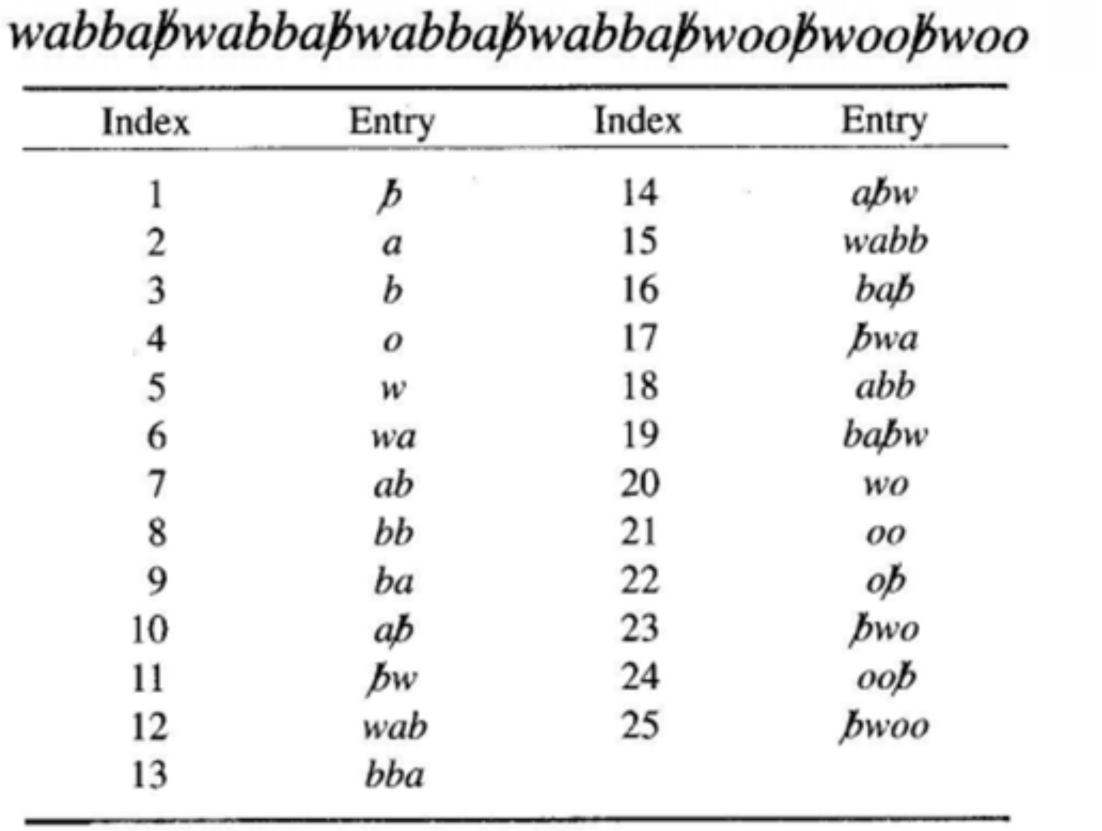


wabbaƀwabbaƀwabbaƀwabbaƀwooƀwooƀwoo

Index	Entry
1	*ƀ*
2	*a*
3	*b*
4	*o*
5	*w*
6	*wa*
7	*ab*
8	*bb*
9	*ba*
10	*aƀ*
11	*ƀw*
12	*w . . .*

This entry is created at the start of encoding/decoding

LZW – Example



wabbaƀwabbaƀwabbaƀwabbaƀwooƀwooƀwoo

Index	Entry	Index	Entry
1	*ƀ*	14	*aƀw*
2	*a*	15	*wabb*
3	*b*	16	*baƀ*
4	*o*	17	*ƀwa*
5	*w*	18	*abb*
6	*wa*	19	*baƀw*
7	*ab*	20	*wo*
8	*bb*	21	*oo*
9	*ba*	22	*oƀ*
10	*aƀ*	23	*ƀwo*
11	*ƀw*	24	*ooƀ*
12	*wab*	25	*ƀwoo*
13	*bba*		

This is the completed LZW dictionary for the example.
The output: 5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4.
More efficient than LZ78? 17x2=34 symbols in LZ78 > 21 symbols in LZW

Note on LZW

- You get longer dictionary entries of length k only when the k-1 prefix is already present in dictionary
- Keep in mind that the encoding always lag behind creation of dictionary entry in LZW
    - e.g. in the previous example, 'wab' is created in dictionary while we encode 'wa' using entry 6
    - So it's natural not all dictionary entries are used!
    - Why must it designed to be so?
    - Do you foresee any problems?

LZW - Decode Example

LZW decoding:

Starting with the same initial dictionary, the decoder builds the dictionary in the same way as the encoder, while decoding each input index. For the example, 5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4, we have

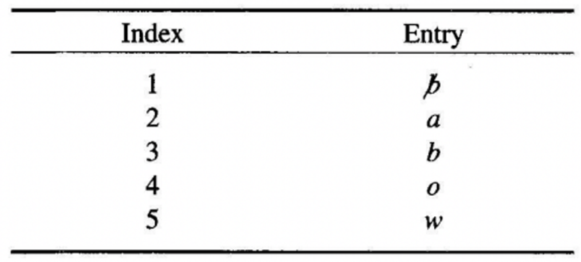
5 - w; 2 - a; (start with w to build dictionary, wa -> 6);

3 - b; (start with a, ab -> 7);

3 - b; (start with b, bb -> 8);

2 - a; (start with b, ba -> 9);

… …



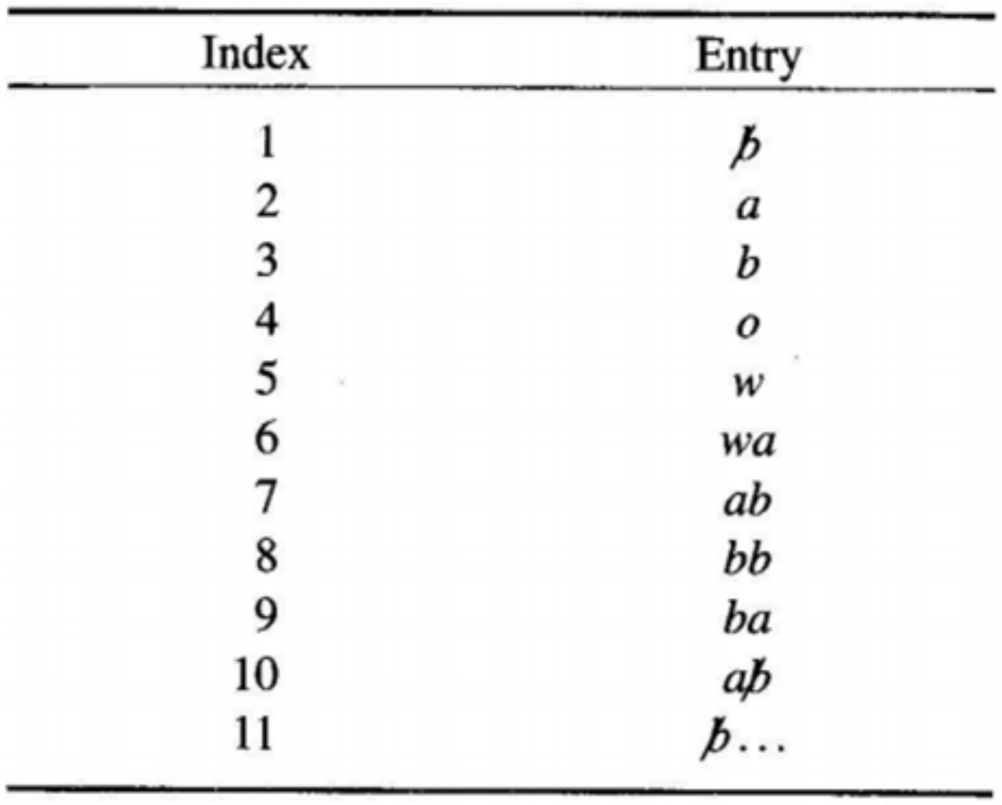
Index	Entry
1	*b̸*
2	*a*
3	*b*
4	*o*
5	*w*

Understanding LWZ Decoding

- One way of thinking it:
    - You receive a certain index, and you look up the symbol e.g. "wab"
    - You know that the encoder has added a new dictionary entry "wab?"
    - You receive the next index that looks up to symbol "ba"
    - You know that the encoder should have created the entry "wabb", so you add the entry too
- Or you can simply say you add an **entry composed of previous symbol + first character of current symbol**

LZW - Decode Example

Constructing the dictionary while decoding with
5 2 3 3 2 1 6 8 10 12 9 11 7 16 5 4 4 11 21 23 4



Index	Entry
1	*ƀ*
2	*a*
3	*b*
4	*o*
5	*w*
6	*wa*
7	*ab*
8	*bb*
9	*ba*
10	*aƀ*
11	*ƀ*. . .

LZW Impact

- An IEEE Milestone
  - Means very important
  - http://en.wikipedia.org/wiki/List_of_IEEE_milestones
  - Along with LCD, ARPANET
- Sadly, being patented caused much trouble
  - Most notedly used in GIF
  - 1999 Unisys, the owner of patent, tries to enforce licensing fee for GIF on the whole Internet!!
  - Widespread condemnation
    - Development of PNG (ironically DEFLATE is also patented but people worked around)
  - Expired in 2003

LZW - Trouble Case

To code the sequence *abababab....*, we have the dictionary:

Sequence:

abababa

a - transmit 1,
adds ab to 3;
b - transmit 2,
adds ba to 4;
ab - transmit 3,
adds aba -> 5
aba - transmit 5,
adds abab to 6....

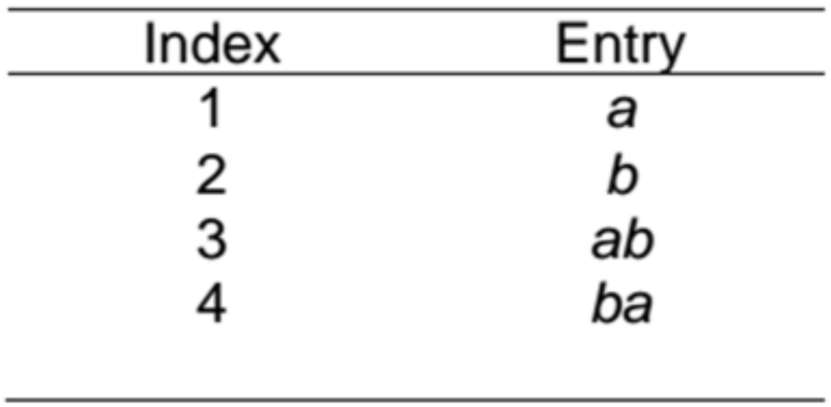
Index	Entry
1	*a*
2	*b*
3	*ab*
4	*ba*
5	*aba*
6	*abab*
7	*b...*

LZW - Trouble Case

The transmitted sequence is 1, 2, 3, 5, ….

Constructing the decoding dictionary:

Sequence:
1 - decode as a
2 - decode as b
adds ab to 3,
3 - decode as ab
adds ba to 4
5 - ????

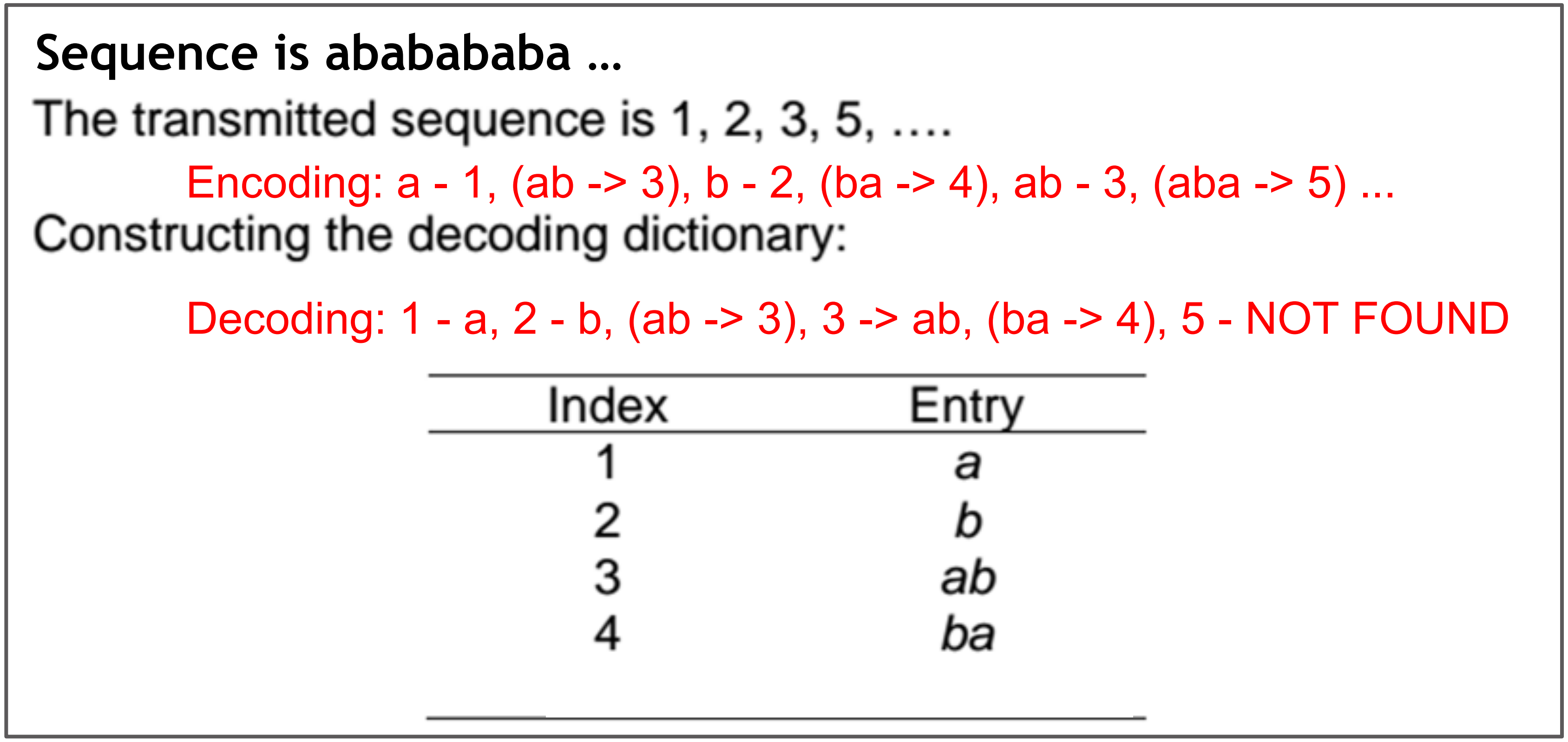


Index	Entry
1	*a*
2	*b*
3	*ab*
4	*ba*

Trouble when we have encoding that doesn't lag behind dictionary creation - not by design but when there is duplicating sequences in succession!

In-Class Question

In-class exercise: how to solve the above trouble case



Sequence is ababababa …

The transmitted sequence is 1, 2, 3, 5, ….

Encoding: a - 1, (ab -> 3), b - 2, (ba -> 4), ab - 3, (aba -> 5) ...

Constructing the decoding dictionary:

Decoding: 1 - a, 2 - b, (ab -> 3), 3 -> ab, (ba -> 4), 5 - NOT FOUND

Index	Entry
1	*a*
2	*b*
3	*ab*
4	*ba*

Solution

- In LWZ, when we create the new dictionary entry, we use the just-encoded pattern plus the new letter. So the entry must begin with the last decoded entry, "ab" - so it's "ab*" where "*" is the unknown next character. But we also know "ab*" is exactly the next decoded message, so "*" is exactly the next character, so it's also "a". Hence "aba".

ab*
a b ab
ab*

1	2	3	5
a	b	ab	ab*

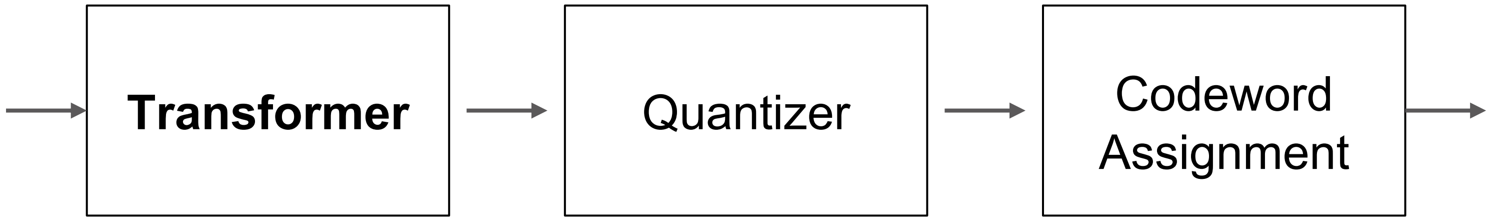
5 = aba

- Can you come up with a general solution?

LZ77 / LZ78 / LZW

- It's a bit tough to understand a compression algorithm thoroughly without going through it yourself
- You will experience it again in tutorial and assignment

Transformer



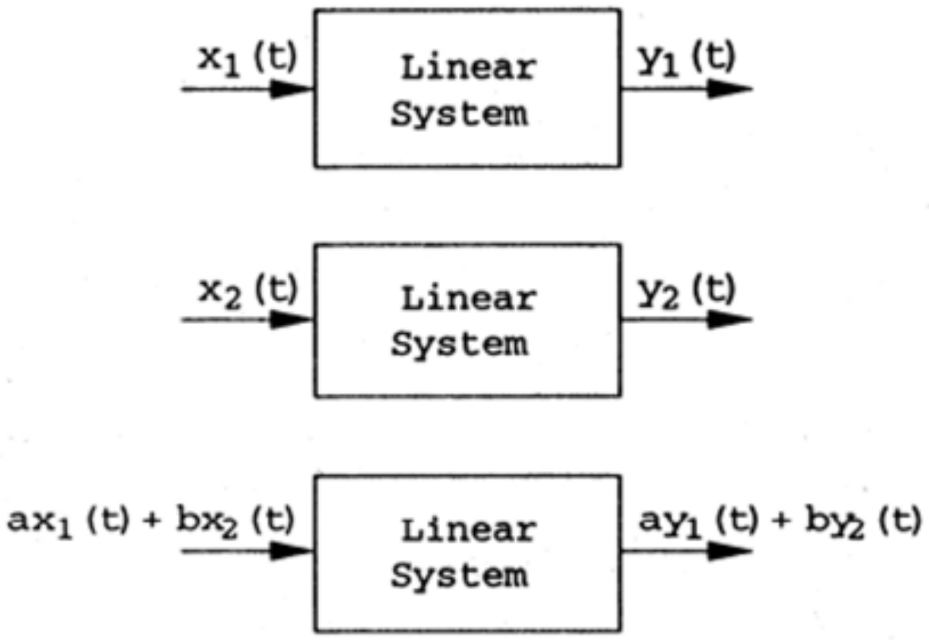
- Transform coding
  - Map signal from raw representation (e.g. pixels for images) to a set of linear transform coefficients
- Success criteria
  - Small number of transform coefficients should carry most signal energy
  - => lossy compression

Review on LTI System Theory

- The theory of linear time-invariant systems has been of tremendous value
  - Theoretical basis for most transform coding
- Two defining properties:
  - Linearity
  - Time invariance
- For those without signal and systems background - conceptual understanding is *sufficient* in the following session

Linearity

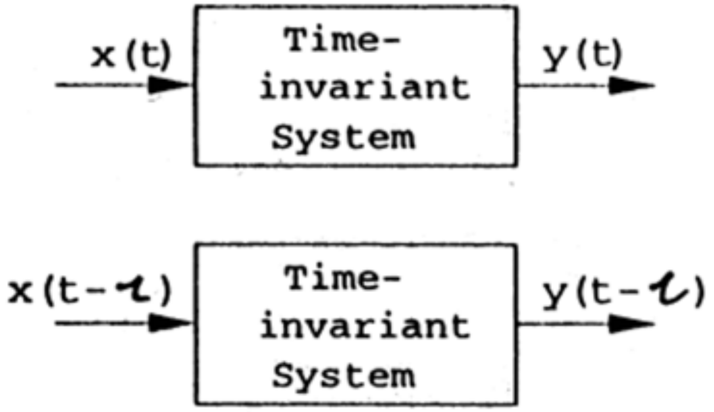
Linear systems



for all $x_1(t), x_2(t), a$ and b.

Time-Invariance

Time—invariant systems



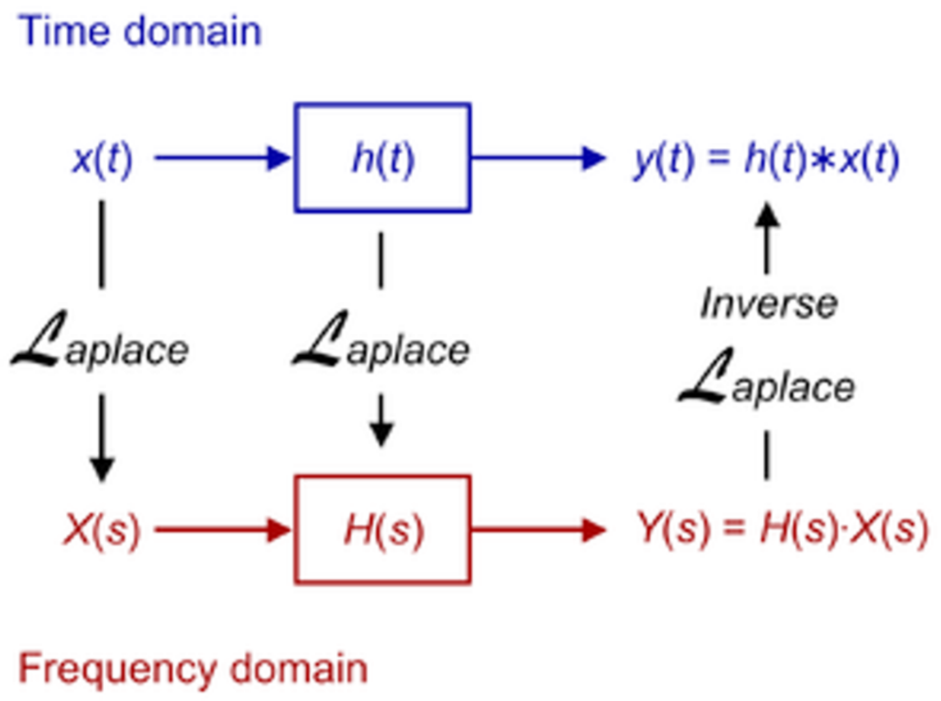
for all $x(t)$ and τ.

- Indicating that system output not dependent on time

LTI System

- Any LTI system can be characterized by its **impulse response**
  - Output of an LTI system is simply the **convolution** of the input with the impulse response
- This behavior in **time domain** is mirrored by its behavior in the **frequency domain**
  - The LTI system is characterized by its **transfer function** in the frequency domain
  - Output of an LTI system in frequency domain is the product of the transform of the input with the system's transfer function

In Picture...



h(t): (unit) impulse response

H(s): transfer function

Just in case...

For a discrete time LTI system

$$y[n] = \sum_{m=-\infty}^{\infty} x[m]h[n-m]$$

where $x[n]$ is an arbitrary input, $h[n]$ is the unit impulse response $y[n]$ is the output

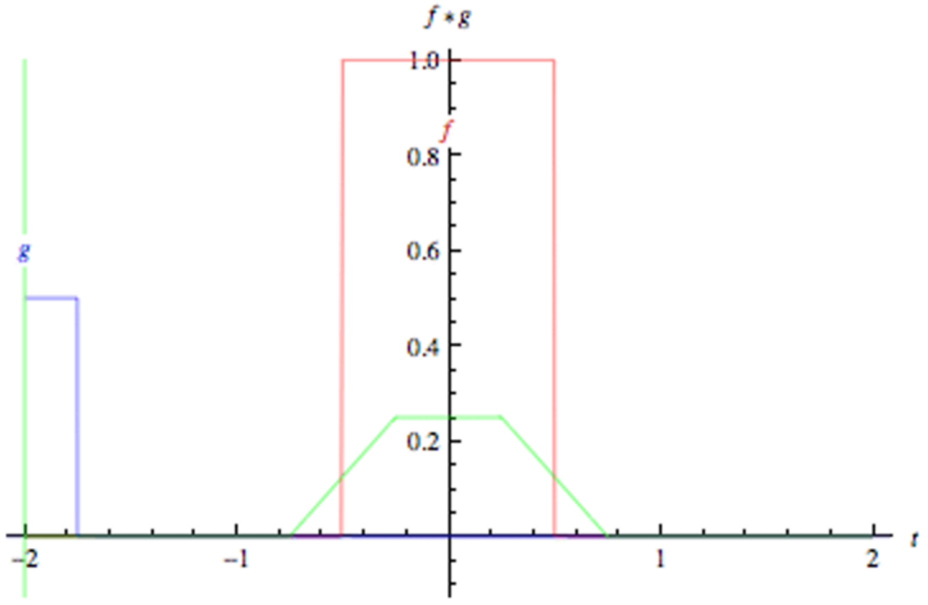
Similarly, for a continuous time system

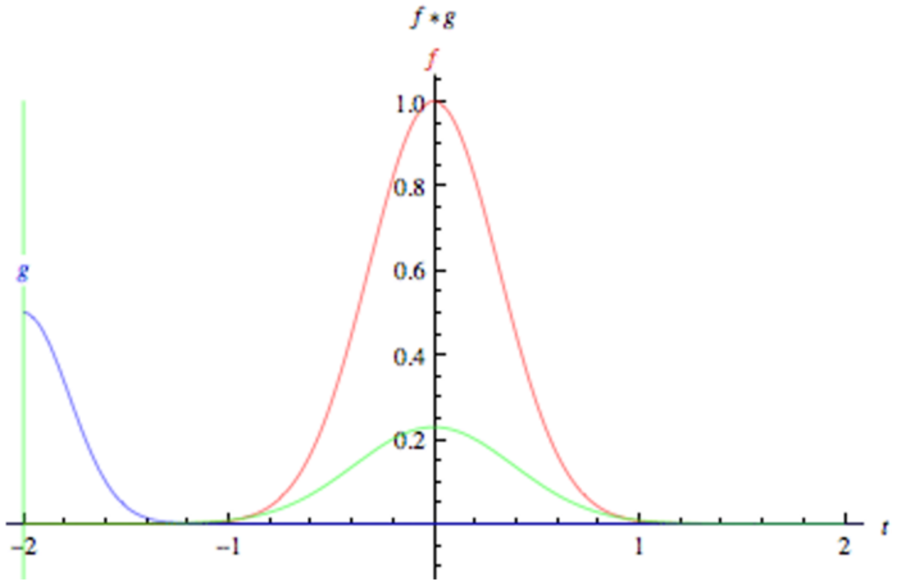
$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau$$

where $x(t)$ is an arbitrary input, $h(t)$is the unit impulse response $y(t)$ is the output

In Picture...

"Convolution is an integral that expresses the amount of overlap of one function as it is shifted over another function" - Wolfram MathWorld





green = convolution of the blue and red

Eigenfunction of an LTI System

Eigenfunction of a system:

A signal for which the system output is just a constant times the input is referred to as an eigenfunction of the system, and the amplitude factor is referred to as the eigenvalue.

Complex exponential functions are eigenfunctions of LTI systems:

In signal analysis tradition we use j instead of i to denote imaginary number

$$e^{j\omega t} \to H(\omega)e^{j\omega t}$$

$$(e^{j\omega t} = \cos(\omega t) + j\sin(\omega t))$$

Eigenfunction of an LTI System

- If we input $Ae^{j\omega t}$ as input to the system, we will get some $Be^{j\omega t}$ as output, where A, B are complex amplitudes
  - The ratio of *B/A* for different values of *s* gives us the transfer function *H(ω)*
  - Implication: you won't input a signal to an LTI with some frequency and get another with different frequency

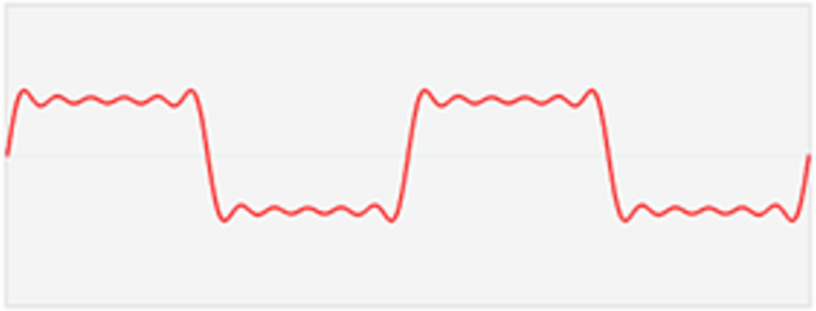
Fourier Transform

- Sinusoids are sums of complex exponentials
  - From Euler's Formula
- For multimedia coding, the Fourier transform, which deals with sinusoids, is particularly of interest to us

$$H(\omega) = \int_{-\infty}^{\infty} h(\tau) e^{-j\omega\tau} d\tau$$

$H(\omega)$ is called the system frequency response, or the Fourier transform of $h(t)$.

In Picture...



In Fourier Analysis, a signal can be decomposed into a series of sinusoidal (sine and cosine) signals with different frequencies, forming a frequency domain representation

* image from en.wikipedia.org

Quick Revision

- Let's start with Fourier Series of a periodic signal:

Definition

Sine and cosine waves of increasing freq.

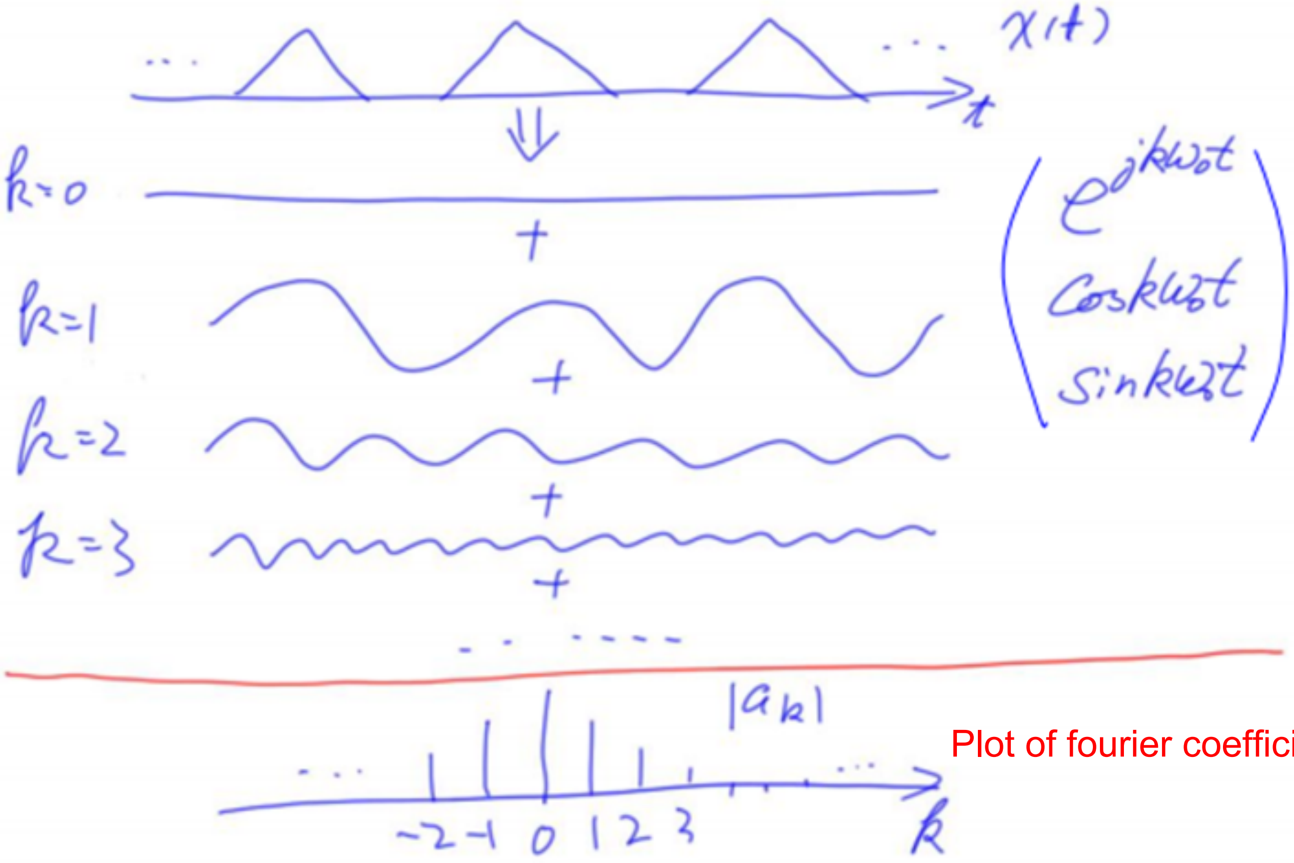
$$x(t) = \sum_{k=-\infty}^{\infty} a_k e^{jk\omega_0 t}, \quad \omega_0 = \frac{2\pi}{T_0}$$

$$a_k = \frac{1}{T_0} \int_{T_0} x(t) e^{-jk\omega_0 t} dt$$

Fourier coefficients

Note: $x(t)$ is continuous-time periodic, but a_k is discrete, non-periodic.

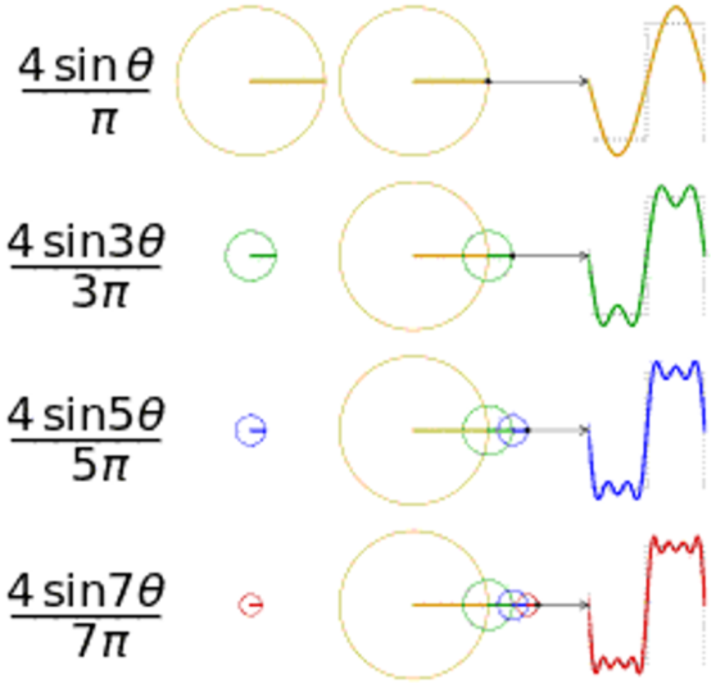
Fourier Series



Plot of fourier coefficients

Fourier Series

- A good visualization of how multiple frequencies add up together…



$\frac{4\sin\theta}{\pi}$
$\frac{4\sin 3\theta}{3\pi}$
$\frac{4\sin 5\theta}{5\pi}$
$\frac{4\sin 7\theta}{7\pi}$

* Image from en.wikipedia.org

Discrete Time Fourier Series

A discrete-time signal $x(n)$ is periodic of N if

$$x(n) = x(n+N)$$

Discrete-time complex exponential signals are periodic with period N

$$\phi_k(n) = e^{jk(2\pi/N)n} = e^{jk\omega_0 n},\ k = 0, \pm 1, \pm 2, \ldots$$

$$\phi_k(n) = \phi_k(n+N)$$

There are only N different signals ϕ_k, because $\longrightarrow$???

$$e^{j(k+rN)\omega_0 n} = e^{jk\omega_0 n} e^{jrN\omega_0 n} = e^{jk\omega_0 n} e^{j2\pi rn} = e^{jk\omega_0 n}$$

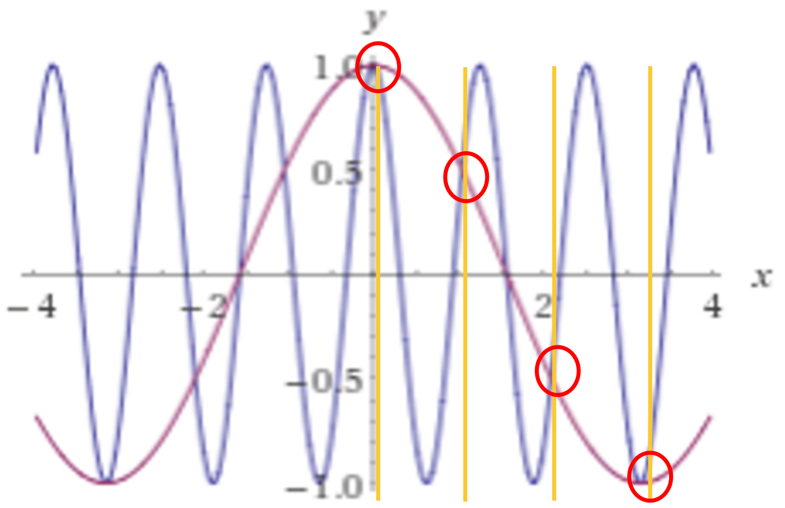
$$\phi_k(n) = \phi_{k+rN}(n)$$

Why Different from Continuous Time?

For all integers *p*,

$$\cos(n\omega) = \cos(n\omega + 2\pi np) = \cos(n(\omega + 2\pi p))$$

Only periodic with *n* (integer) not *t* (continuous)!



A visual understanding: Imagine you increase the frequency of the sinusoid, at some degree, it will end up the same as a lower frequency sinusoid because we are discrete, not continuous….

Discrete Time Fourier Series

We wish to represent periodic sequences in terms of linear combinations of the sequences $\phi_k(n)$. Since $\phi_k(n)$ are distinct only over a range of *N* successive integers, we only need to consider terms over this range.

$$x(n) = \sum_{k=<N>} a_k \phi_k(n) = \sum_{k=<N>} a_k e^{jk(2\pi/N)n}$$

Discrete Time Fourier Series:

$$\boxed{\begin{aligned} x(n) &= \sum_{k=<N>} a_k e^{jk(2\pi/N)n} \\ a_k &= \frac{1}{N} \sum_{n=<N>} x(n) e^{-jk(2\pi/N)n} \end{aligned}}$$

Note: both $x(n)$ and a_k are discrete, periodic with period = *N*.

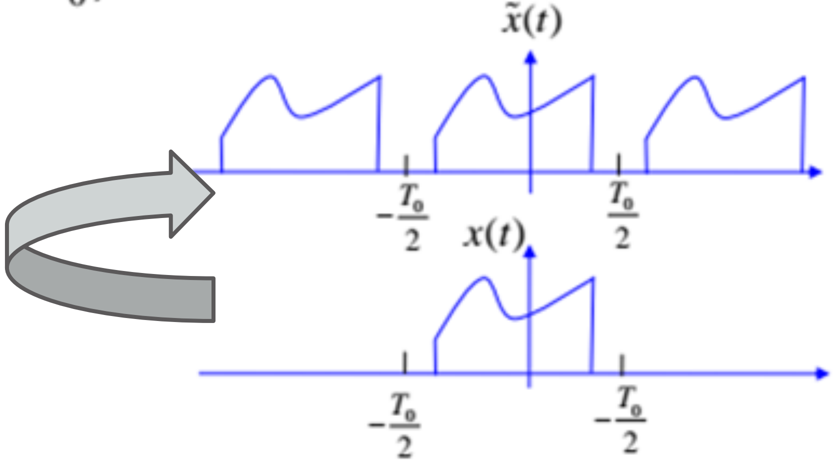
Aperiodic Signals

- Most signals in engineering applications are aperiodic
- Ultimately we want to do Fourier Analysis on aperiodic signal more than anything else
    - Birth of Fourier Transform
    - Trick - extend the aperiodic signal

Continuous Time Fourier Transform

Let $\tilde{x}(t)$ be a periodic signal of period T_0, define

$$x(t)=\begin{cases}\tilde{x}(t), & -\frac{T_0}{2}\le t\le\frac{T_0}{2}\\ 0, & other\end{cases}$$



Fourier series of $\tilde{x}(t)$,

$$\tilde{x}(t)=\sum_{k=-\infty}^{+\infty}a_k e^{jk\omega_0 t}$$

$$a_k=\frac{1}{T_0}\int_{-T_0/2}^{T_0/2}\tilde{x}(t)e^{-jk\omega_0 t}dt$$

We are interested in the **aperiodic** signal *x(t)*

Continuous Time Fourier Transform

$$a_k = \frac{1}{T_0}\int_{-T_0/2}^{T_0/2} \tilde{x}(t)e^{-jk\omega_0 t}dt = \frac{1}{T_0}\int_{-\infty}^{+\infty} x(t)e^{-jk\omega_0 t}dt$$

Let

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}dt$$

Coefficients a_k can be expressed as

$$a_k = \frac{1}{T_0}X(k\omega_0), \quad \omega_0 = \frac{2\pi}{T_0}$$

$$\tilde{x}(t) = \sum_{k=-\infty}^{+\infty} \frac{1}{T_0} X(k\omega_0)e^{jk\omega_0 t} = \frac{1}{2\pi}\sum_{k=-\infty}^{+\infty} X(k\omega_0)e^{jk\omega_0 t}\omega_0$$

Continuous Time Fourier Transform

As $T_0 \to \infty,\ \omega_0 \to d\omega \to 0,\ \tilde{x}(t) \to x(t)$

Now if we allow the period of our periodic signal to approach infinity...

$$\sum_{k=-\infty}^{+\infty} X(k\omega_0)e^{jk\omega_0 t}\omega_0 \to \int_{-\infty}^{\infty} X(\omega)e^{j\omega t}\,d\omega$$

So we have

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{+\infty} X(\omega)e^{j\omega t}\,d\omega$$

$$X(\omega) = \int_{-\infty}^{+\infty} x(t)e^{-j\omega t}\,dt$$

Note: both $x(t)$ and $X(\omega)$ are continuous, aperiodic.

Discrete-Time Fourier Transform

- Using a similar trick, we have Discrete-Time Fourier Transform:

$$x(n) = \frac{1}{2\pi}\int_{2\pi} X(\omega)e^{j\omega n}dw$$

$$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n)e^{-j\omega n}$$

Note: $x(n)$ is discrete & aperiodic, but $X(\omega)$ is continuous & periodic $X(\omega) = X(\omega+2\pi)$.

- Most derivations skipped - go back to your signal textbook if interested

Hope that Refreshed Your Memory...

Hehe~

Still not sure, check this great short video tutorial!

https://www.youtube.com/watch?v=spUNpyF58BY

1768 - 1830

So...?

- Not everyone of you will still be familiar with all the details
- Gladly, having a good sense of each concept is enough

Strategic Break...

Let's see if you can recall the following concepts:

- Impulse response
- Convolution
- Transfer function
- Fourier Series
- Fourier Transform

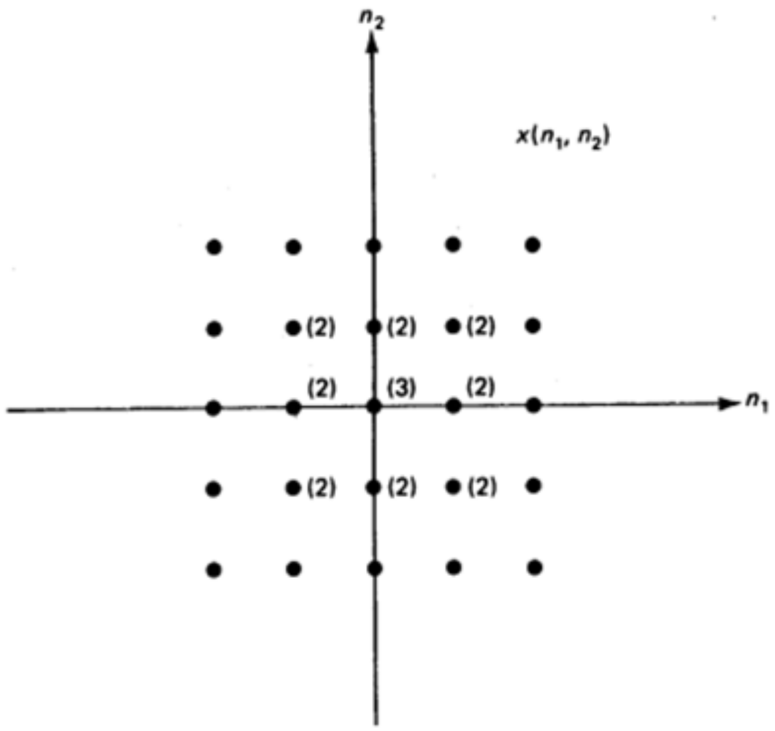
Need a poll?

Why LTI Systems?

- Bridges between frequency domain and time domain
    - Human sensory organs are highly frequency selective
- LTI systems are relatively "easy" to analyse
    - Compared to non-linear, at least
    - The use of complex exponentials is the key

Extending to 2D

A 2-D discrete-space signal (sequence) will be denoted by a function whose two arguments are integers, such as $x(n_1, n_2)$.



Extending to 2D

A digital image can be denoted by a 2-D sequence $x(n_1, n_2)$. The amplitude of a digital image is often quantized to 256 gray levels. Each level is denoted by an integer, with 0 for the darkest level and 255 for the brightest. Each point of $x(n_1, n_2)$ is called a *pixel*.

Linear Shift-Invariant Systems

Linear Shift-Invariant (LSI) Systems

A system T that relates an input $x(n_1,n_2)$ to an output $y(n_1,n_2)$ is represented by

$$y(n_1,n_2)=T[x(n_1,n_2)]$$

Unit impulses: $\delta(n_1,n_2)=\begin{cases}1, & n_1=n_2=0\\ 0, & \textit{otherwise.}\end{cases}$

Linearity of a system:

$$T[ax_1(n_1,n_2)+bx_2(n_1,n_2)]=ay_1(n_1,n_2)+by_2(n_1,n_2)$$

Shift invariance of a system:

$$T[x(n_1-m_1,n_2-m_2)]=y(n_1-m_1,n_2-m_2)$$

Linear Shift-Invariant Systems

From linearity, we have

$$y(n_1,n_2)=T[x(n_1,n_2)]=T\left[\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}x(k_1,k_2)\delta(n_1-k_1,n_2-k_2)\right]$$

$$=\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}x(k_1,k_1)T\left[\delta(n_1-k_1,n_2-k_2)\right]$$

If a system impulse response is: $h(n_1,n_2)=T[\delta(n_1,n_2)]$

Then from shift invariance, we have

$$h(n_1-k_1,n_2-k_2)]=T[\delta(n_1-k_1,n_2-k_2)]$$

So the system output is <u>convolution of input and impulse response</u>:

$$y(n_1,n_2)=T[x(n_1,n_2)]=\sum_{k_1=-\infty}^{\infty}\sum_{k_2=-\infty}^{\infty}x(k_1,k_2)h(n_1-k_1,n_2-k_2)=x(n_1,n_2)*h(n_1,n_2)$$

Linear Shift-Invariant Systems

- The impulse response is sometimes called "kernel"
- Example of 2D convolution
  - http://www.songho.ca/dsp/convolution/convolution2d_example.html

Linear Shift-Invariant System

- Gladly the results from 1D works on 2D in a straight-forward manner
- E.g. concerning transfer function:

$$y(n_1,n_2)=x(n_1,n_2)*h(n_1,n_2)\leftrightarrow Y(\omega_1,\omega_2)=X(\omega_1,\omega_2)H(\omega_1,\omega_2)$$

Linear Shift-Invariant System

Discrete Space Fourier Transform Pair:

$$X(w_1,w_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} x(n_1,n_2) e^{-jw_1n_1} e^{-jw_2n_2}$$

$$x(n_1,n_2) = \frac{1}{(2\pi)^2} \int_{w_1=-\pi}^{\pi} \int_{w_2=-\pi}^{\pi} X(w_1,w_2) e^{jw_1n_1} e^{jw_2n_2} dw_1\, dw_2$$

Note $x(n_1,n_2)$: discrete, aperiodic; $X(\omega_1,\omega_2)$: continuous, periodic with a period = $2\pi \times 2\pi$, i.e., $X(\omega_1,\omega_2) = X(\omega_1+2\pi,\omega_2) = X(\omega_1,\omega_2+2\pi)$.

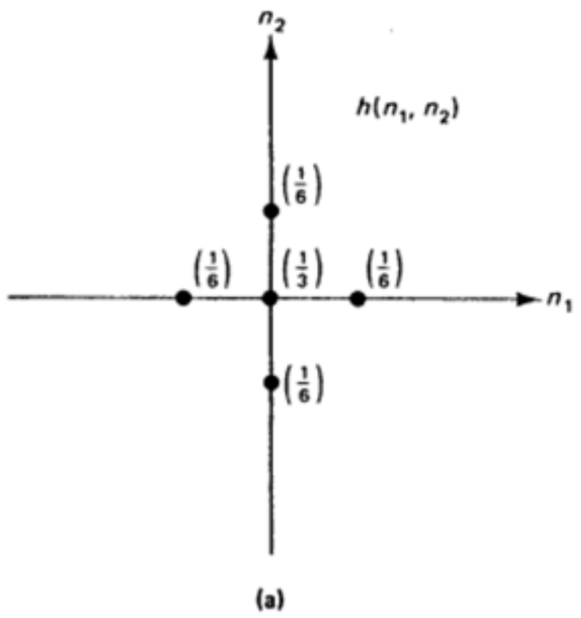
Compare with 1D Discrete Fourier Transform - very intuitive extension

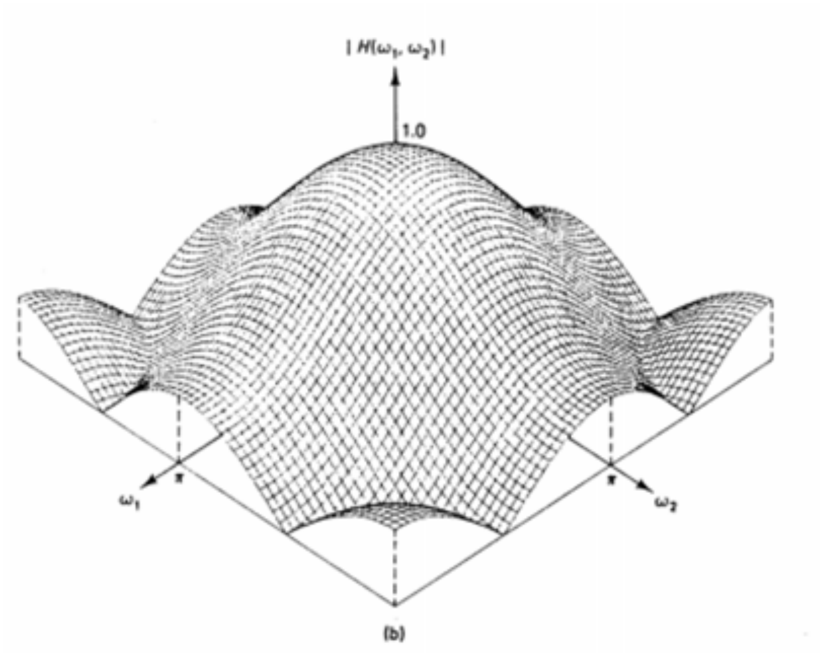
$$x(n) = \frac{1}{2\pi} \int_{2\pi} X(\omega) e^{j\omega n} dw$$

$$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n) e^{-j\omega n}$$

LSI System Example

Example 1:





$h(n_1, n_2)$: impulse response; $H(\omega_1, \omega_2)$: frequency response

$$H(\omega_1, \omega_2) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} h(n_1, n_2) e^{-jw_1 n_1} e^{-jw_2 n_2} = \frac{1}{3} + \frac{1}{3}\cos\omega_1 + \frac{1}{3}\cos\omega_2$$

LSI System Example

Lowpass Filtering

$H(\omega_1, \omega_2)$: a lowpass filter

Input: $x(n_1, n_2) \leftrightarrow X(\omega_1, \omega_2)$

Output :

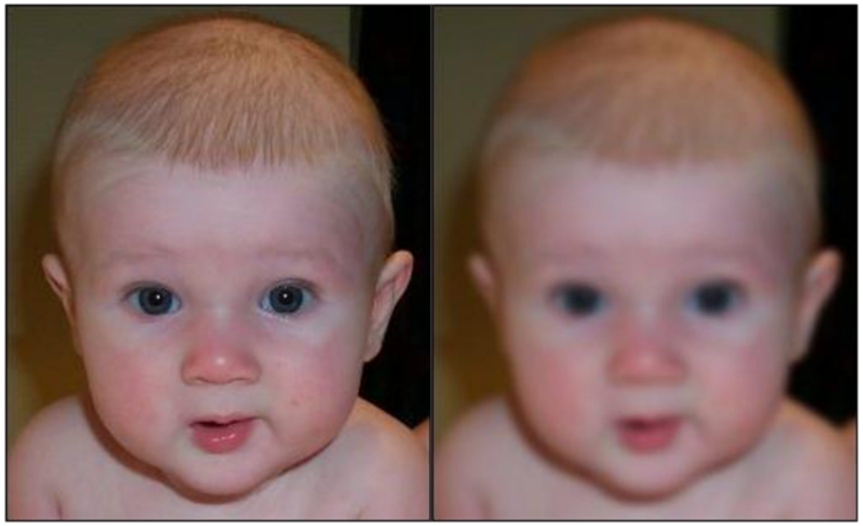
$$Y(\omega_1, \omega_2) = X(\omega_1, \omega_2) H(\omega_1, \omega_2)$$

$$\updownarrow$$

$$y(n_1, n_2) = x(n_1, n_2) * h(n_1, n_2)$$

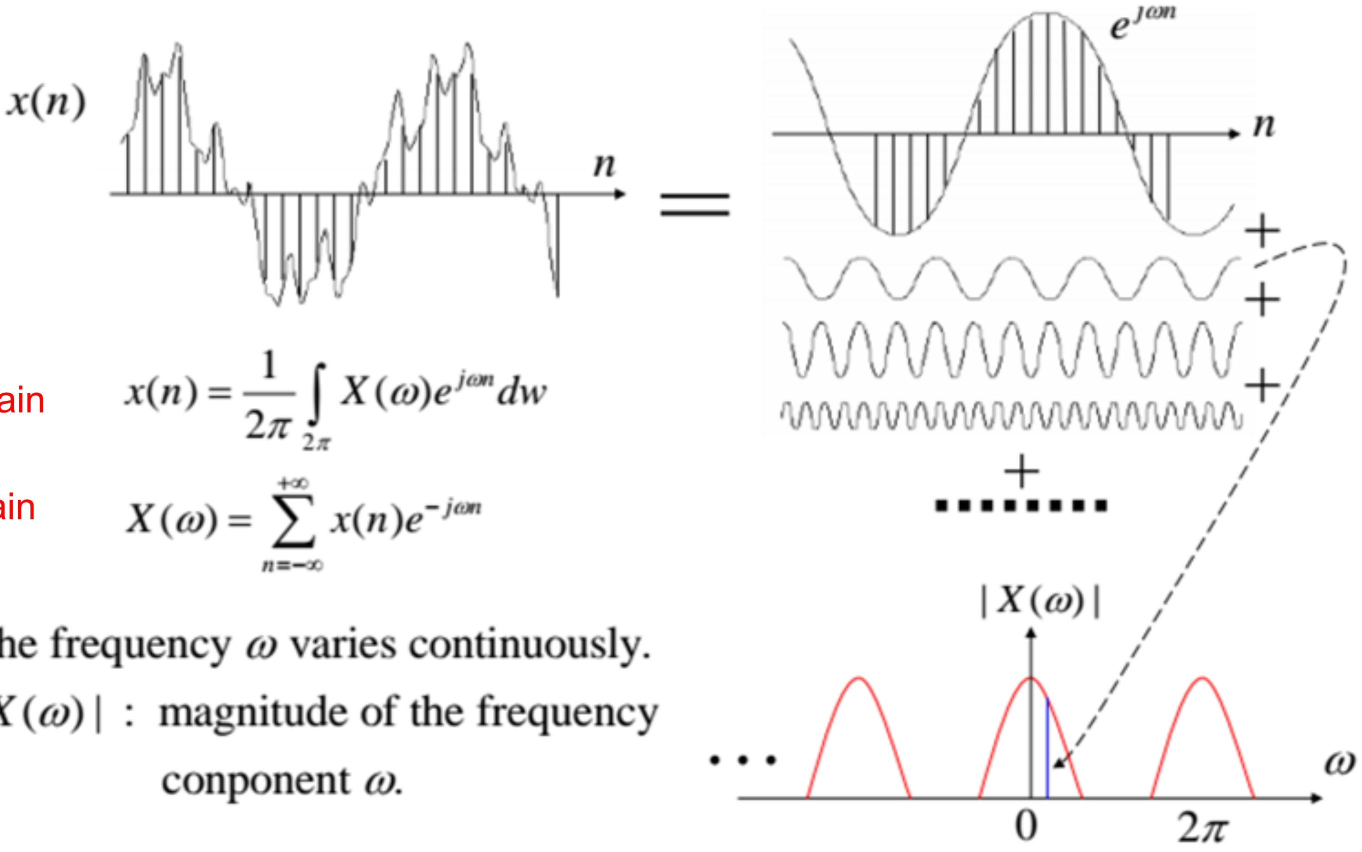
$y(n_1, n_2)$ and $x(n_1, n_2)$ have the same average intensity since $Y(0,0) = X(0,0)$.

$y(n_1, n_2)$ is blurred.



Just In Case...

In 1D, we have...



Time Domain $$x(n) = \frac{1}{2\pi}\int_{2\pi} X(\omega)e^{j\omega n}dw$$

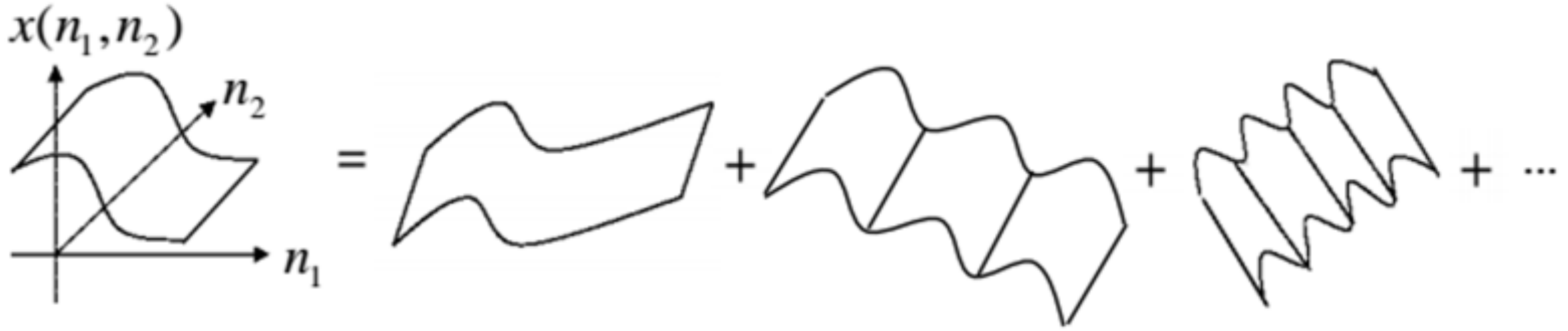
Freq Domain $$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n)e^{-j\omega n}$$

The frequency ω varies continuously.

$|X(\omega)|$: magnitude of the frequency conponent ω.

Just In Case...

In 2D, we have...



$$x(n_1,n_2) = \frac{1}{(2\pi)^2}\int_{w_1=-\pi}^{\pi}\int_{w_2=-\pi}^{\pi} X(\omega_1,\omega_2)e^{j\omega_1 n_1 + j\omega_2 n_2}d\omega_1\, d\omega_2$$

$$X(\omega_1,\omega_2) = \sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty} x(n_1,n_2)e^{j\omega_1 n_1 + j\omega_2 n_2}$$

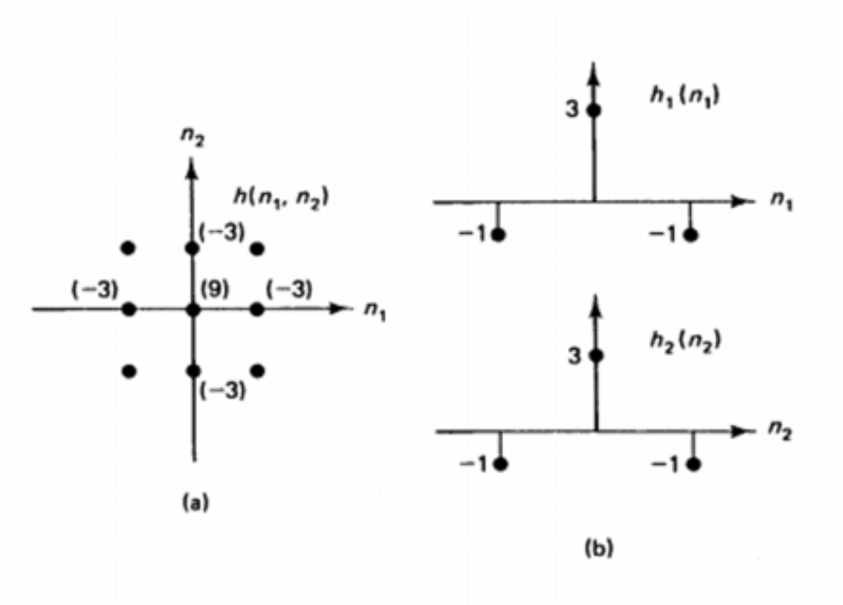
The 2D frequency (ω_1,ω_2) varies continuously.

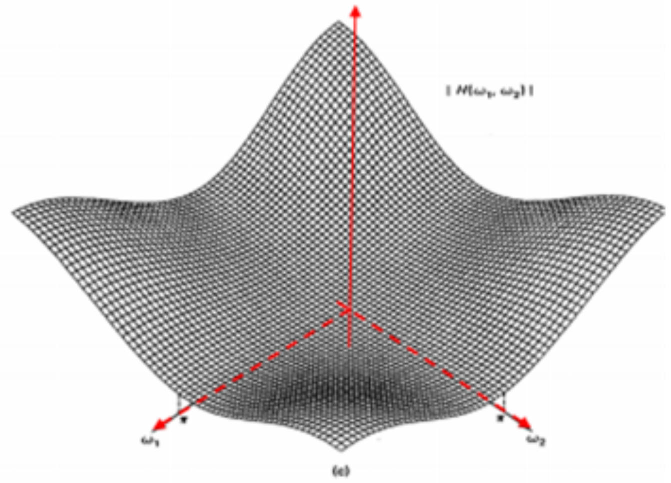
$|X(\omega_1,\omega_2)|$: magnitude of the frequency conponent (ω_1,ω_2).

More LSI System Examples

Example 2:

What kind of filter is this? Can you tell?

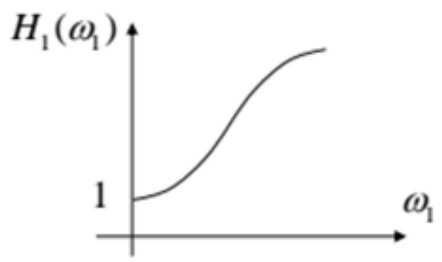




Property: If $h(n_1, n_2) = h_1(n_1)h_2(n_2)$

Then $H(\omega_1, \omega_2) = H_1(\omega_1)H_2(\omega_2)$

$$H(\omega_1, \omega_2) = H_1(\omega_1)H_2(\omega_2)$$
$$= (3 - 2\cos\omega_1)(3 - 2\cos\omega_2)$$



More LSI System Examples

Highpass Filtering:

$H(\omega_1,\omega_2)$: a highpass filter

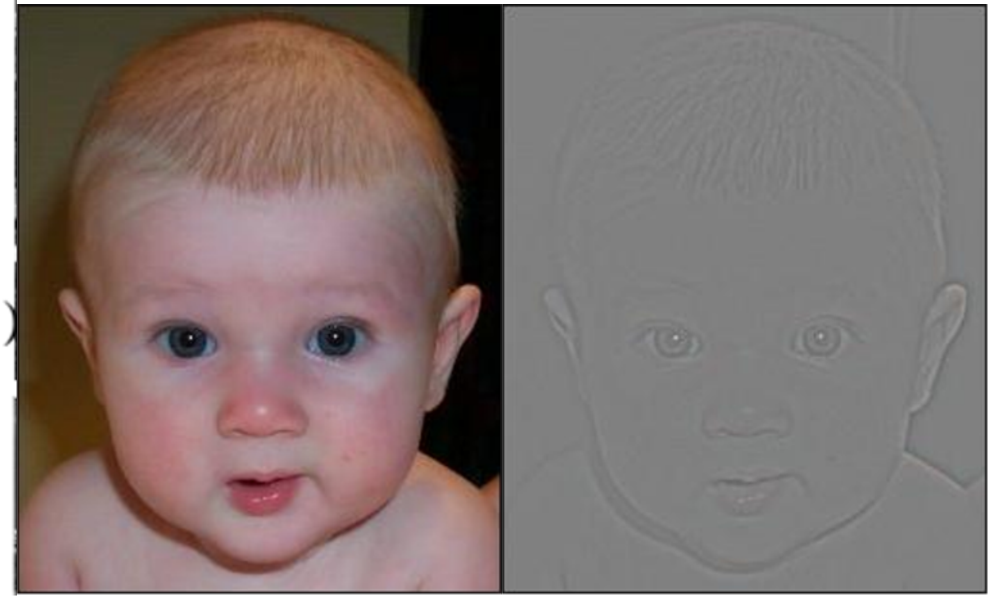
Input: $x(n_1,n_2) \leftrightarrow X(\omega_1,\omega_2)$

Output :

$$Y(\omega_1,\omega_2) = X(\omega_1,\omega_2)H(\omega_1,\omega_2)$$

$$\updownarrow$$

$$y(n_1,n_2) = x(n_1,n_2) * h(n_1,n_2)$$



More LSI System Examples

Example 3

(a) $x(n_1, n_2)$

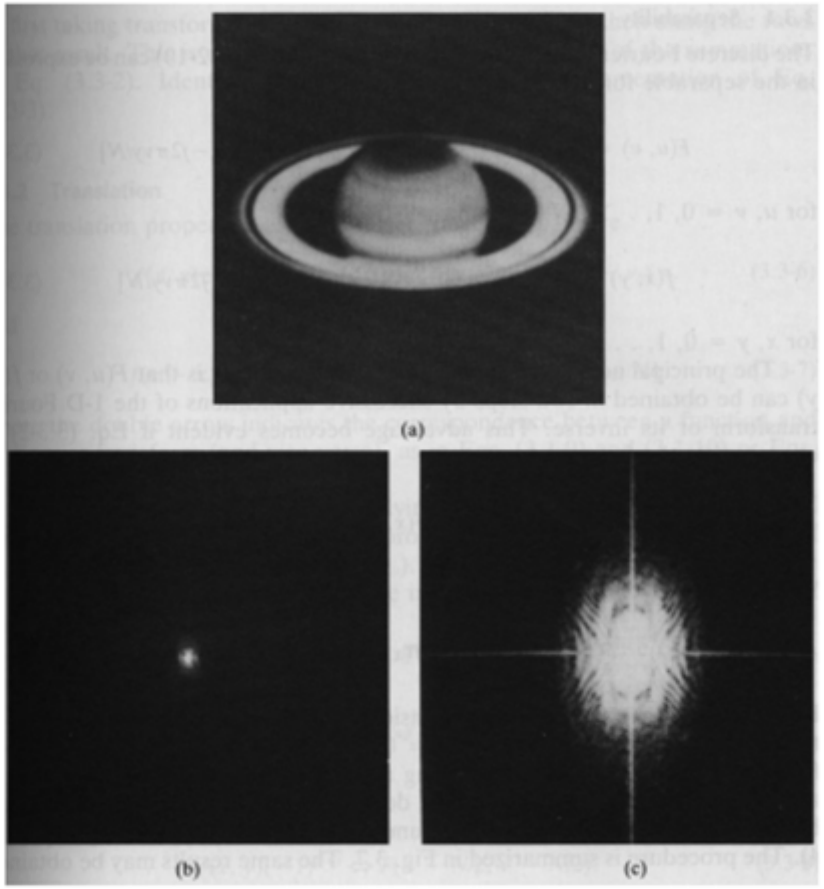
$$x(n_1, n_2) \leftrightarrow X(\omega_1, \omega_2)$$

Show $|X(\omega_1, \omega_2)|$ as an image

(b) $|X(\omega_1, \omega_2)|$

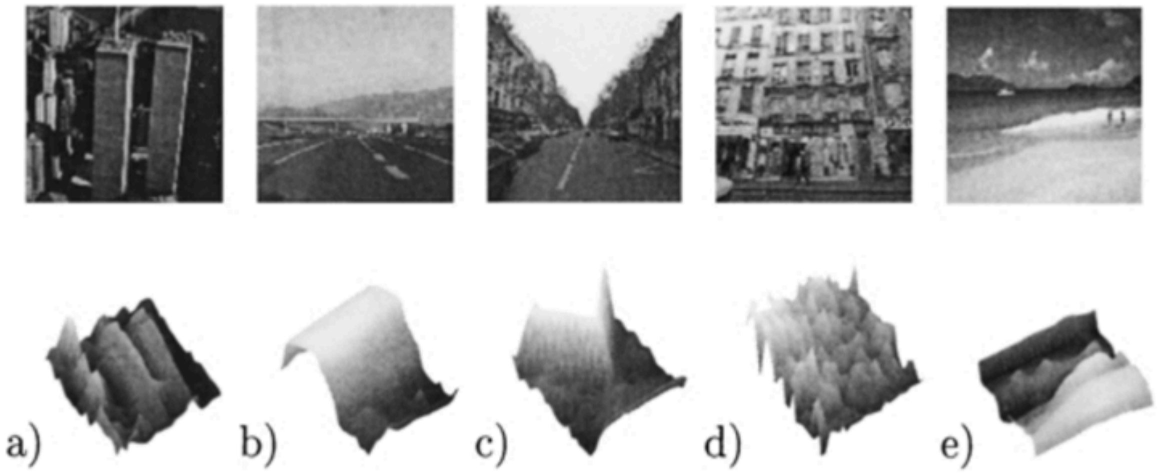
(c) $\log(|X(\omega_1, \omega_2)| + K)$

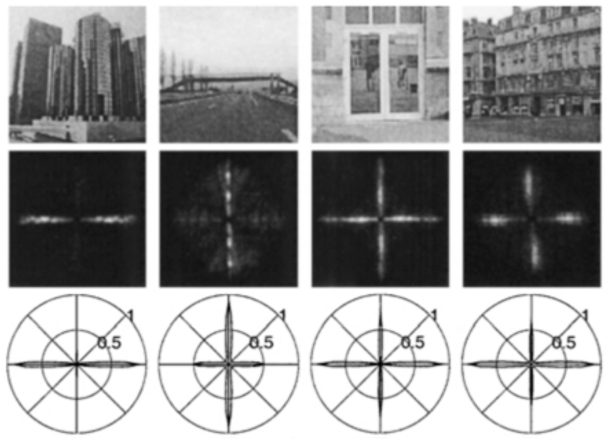
Scaled to [0,255] for display



Fourier transform of a regular image

Modeling the Spatial Envelope of the Scene





$$I(f_x, f_y) = \sum_{x,y=0}^{N-1} i(x, y)h(x, y)e^{-j\,2\pi(f_x x + f_y y)}$$
$$= A(f_x, f_y)\, e^{j\Phi(f_x, f_y)}$$

Oliva and Torralba. http://www.cnbc.cmu.edu/cns/papers/Oliva-Torralba-IJCV-01.pdf

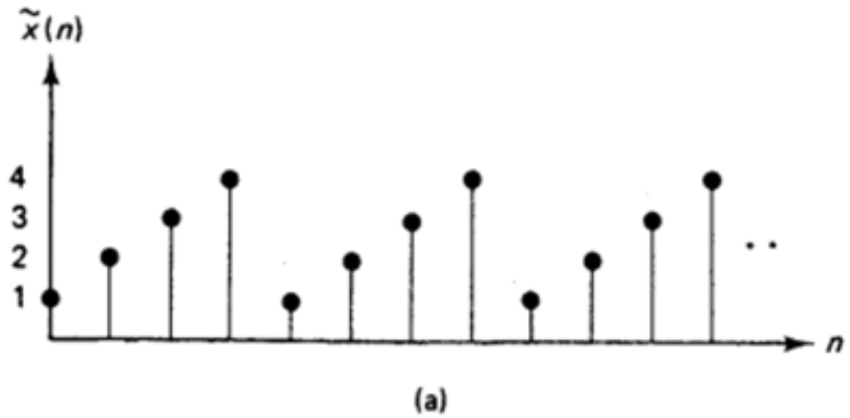
Discrete Cosine Transform (DCT)

- Used as transform coding for many important standards
    - JPEG, MPEG, MP3 etc.
- Uses only cosine functions
    - Sinusoids can include sines and cosines
    - Choose only cosine due to efficiency, which we will see
    - Eight standard DCT variants, four are common (DCT I-VIII)

Why Variants?

- Remember when we do Fourier Transform we extend a signal into a periodic signal
- There are actually quite some choices
  - Odd or even extension
    - e.g. abcd -> abcd*abcd* v.s. abcd*dcba*
  - Point of extension
    - e.g. abcd*cba* v.s. abcd*dcba*
  - Left/right boundary differences
    - e.g. *dcba*abcd*dcba* v.s *abcd*abcd*dcba*
- We are trying to find one with best compression performance

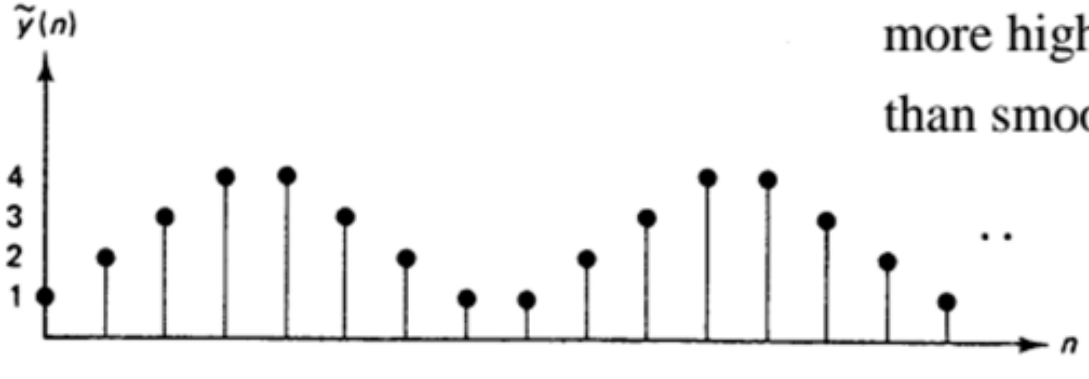
Even/Odd Extension



(a)

$\tilde{x}(n)$ contains more high frequency components than $\tilde{y}(n)$ does

Discontinuous parts contributes more high frequency components than smooth parts in a signal



(b)

$\tilde{y}(n)$ is more desirable

"Small number of transform coefficients carry most of the energy"

DCT (DCT-II)

The DCT is derived through the following relation

$$\overset{N-\text{point}}{x(n)} \;\leftrightarrow\; \overset{2N-\text{point}}{y(n)} \;\overset{DFT}{\leftrightarrow}\; \overset{2N-\text{point}}{Y(k)} \;\leftrightarrow\; \overset{N-\text{point}}{C_x(k)}$$

where y(n) is

$$y(n) = x(n) + x(2N-1-n) = \begin{cases} x(n), & 0 \le n \le N-1 \\ x(2N-1-n), & N \le n \le 2N-1. \end{cases}$$

2N-point DFT Y(k) is

$$Y(k) = \sum_{n=0}^{2N-1} y(n) W_{2N}^{kn} = \sum_{n=0}^{N-1} x(n) W_{2N}^{kn} + \sum_{n=N}^{2N-1} x(2N-1-n) W_{2N}^{kn}, \quad 0 \le k \le 2N-1.$$

$$(W_{2N} = e^{-j(2\pi/2N)})$$

With a change of variables and after some algebra

$$Y(k) = W_{2N}^{-k/2} \sum_{n=0}^{N-1} 2x(n) \cos \frac{\pi}{2N} k(2n+1), \quad 0 \le k \le 2N-1.$$

N terms! :)

DCT (DCT-II)

The N-point DCT of x(n) is then defined as:

$$C_x(k) = \begin{cases} \sum_{n=0}^{N-1} 2x(n)\cos\frac{\pi}{2N}k(2n+1), & 0 \le k \le N-1 \\ 0, & \textit{otherwise.} \end{cases}$$

$$C_x(k) \Rightarrow Y(k) \Rightarrow y(n) \Rightarrow x(n)$$

The inverse DCT can be derived as

$$x(n) = \begin{cases} \frac{1}{N}\left[\frac{C_x(0)}{2} + \sum_{k=1}^{N-1} C_x(k)\cos\frac{\pi}{2N}k(2n+1)\right], & 0 \le n \le N-1 \\ 0, & \textit{otherwise.} \end{cases}$$

DCT (DCT-II)

1D DCT pair:

$$C_x(k) = \begin{cases} \sum_{n=0}^{N-1} 2x(n)\cos\frac{\pi}{2N}k(2n+1), & 0 \le k \le N-1 \\ 0, & otherwise. \end{cases}$$

$$x(n) = \begin{cases} \frac{1}{N}\sum_{k=0}^{N-1} w(k)C_x(k)\cos\frac{\pi}{2N}k(2n+1), & 0 \le n \le N-1 \\ 0, & otherwise. \end{cases}$$

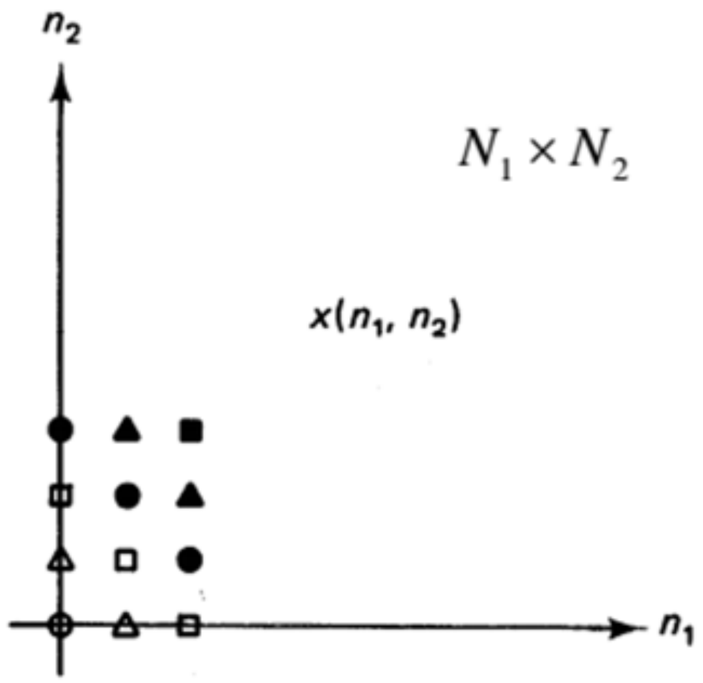
$$w(k) = \begin{cases} 1/2, & k = 0 \\ 1, & 1 \le k \le N-1 \end{cases}$$

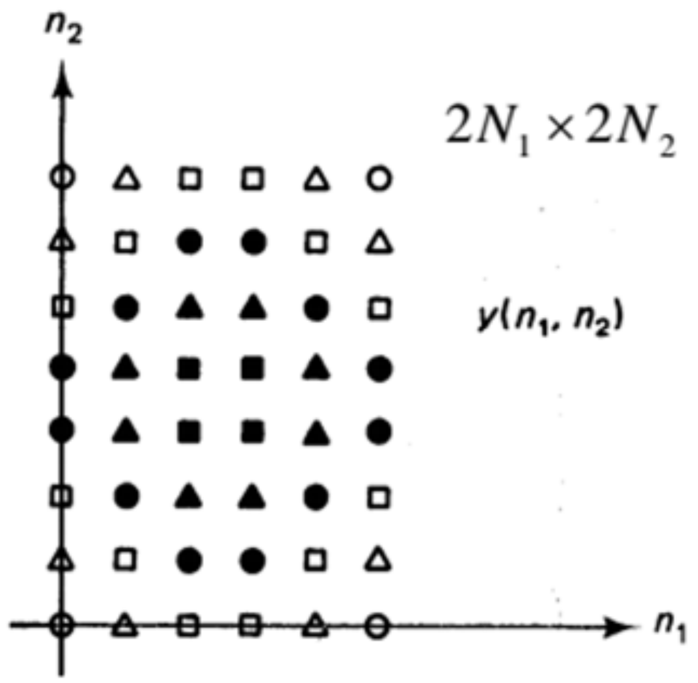
Why DCT?

- Merits:
  - Good compression due to even extension
    - Unlike generic DFT, less high frequency component
  - The DFT step can be computed using Fast Fourier Transform (FFT)
  - Applying to 2D is straight-forward
- Drawbacks:
  - Quite complicated actually
  - Many details beyond scope - consult further references if interested

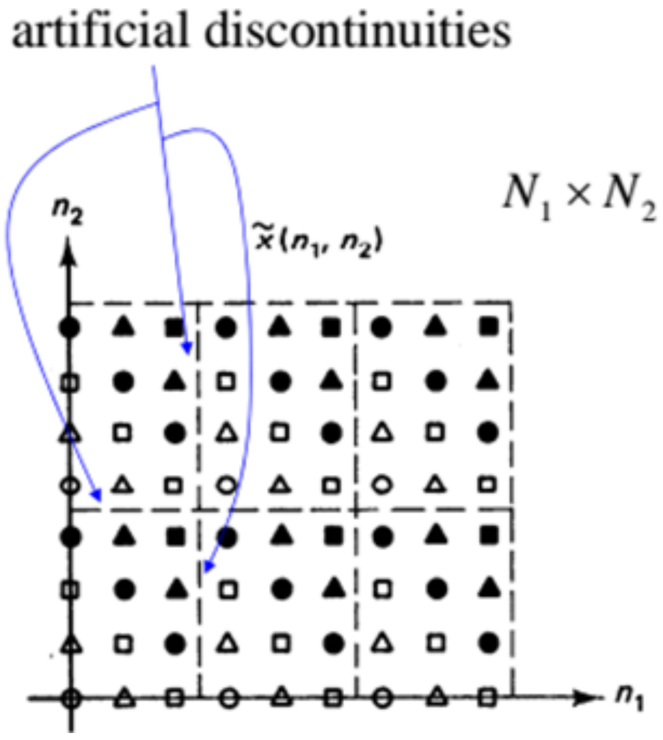
DCT in 2D

2-D DCT

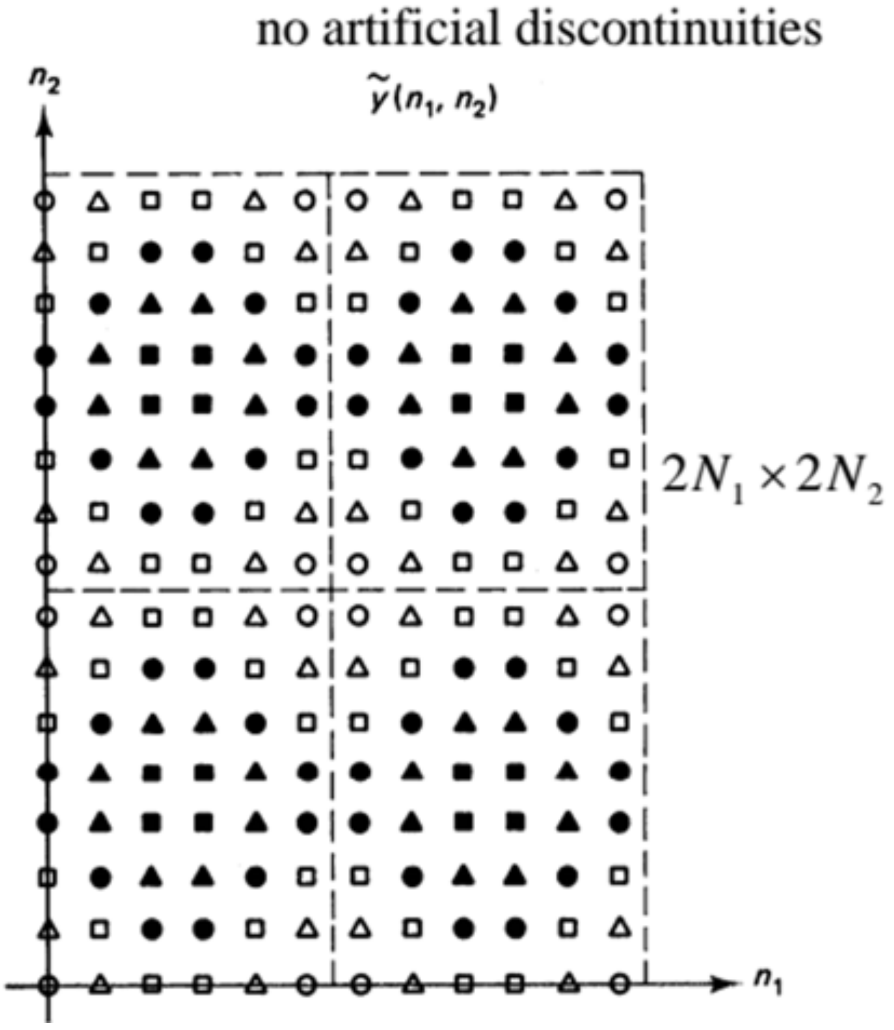




DCT in 2D



artificial discontinuities
n_2
$\tilde{x}(n_1, n_2)$
$N_1 \times N_2$
n_1



no artificial discontinuities
n_2
$\tilde{y}(n_1, n_2)$
$2N_1 \times 2N_2$
n_1

DCT in 2D

Two-Dimensional Discrete Cosine Transform Pair

$$C_x(k_1, k_2) = \begin{cases} \displaystyle\sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} 4x(n_1, n_2) \cos \frac{\pi}{2N_1} k_1(2n_1 + 1) \cos \frac{\pi}{2N_2} k_2(2n_2 + 1), \\ \qquad\qquad \text{for } 0 \le k_1 \le N_1 - 1,\ 0 \le k_2 \le N_2 - 1 \\ 0, \qquad\qquad \text{otherwise.} \end{cases}$$

$$x(n_1, n_2) = \begin{cases} \displaystyle\frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} w_1(k_1) w_2(k_2) C_x(k_1, k_2) \cos \frac{\pi}{2N_1} k_1(2n_1 + 1) \cos \frac{\pi}{2N_2} k_2(2n_2 + 1), \\ \qquad\qquad \text{for } 0 \le n_1 \le N_1 - 1,\ 0 \le n_2 \le N_2 - 1 \\ 0, \qquad\qquad \text{otherwise.} \end{cases}$$

8 x 8 DCT Basis Example

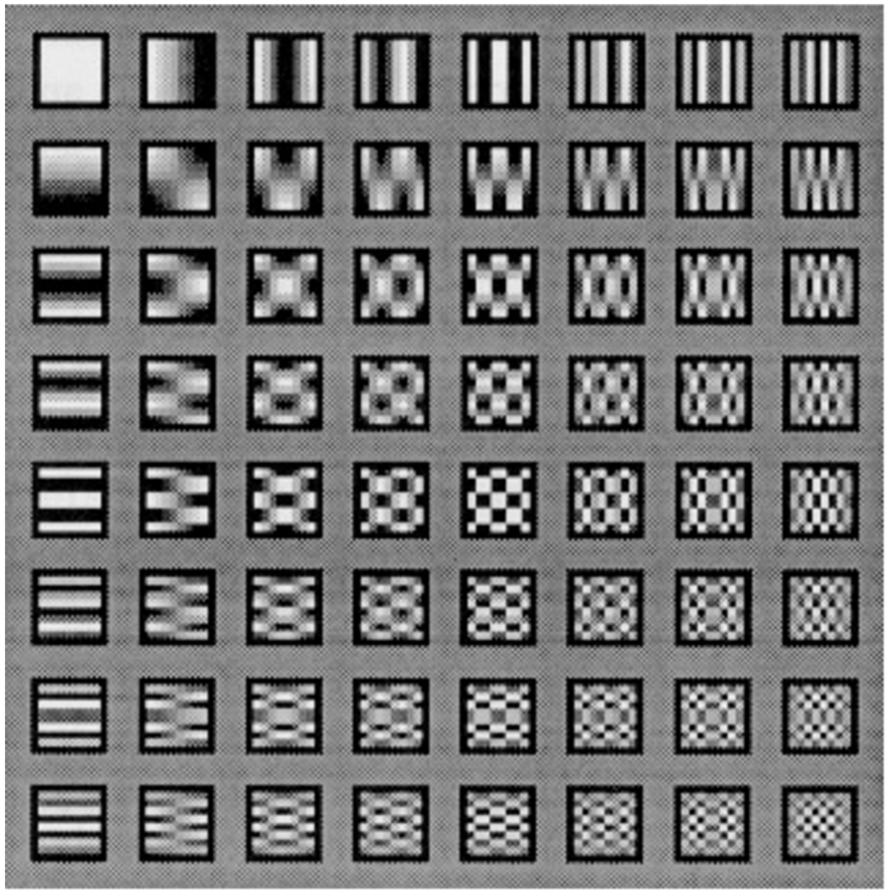
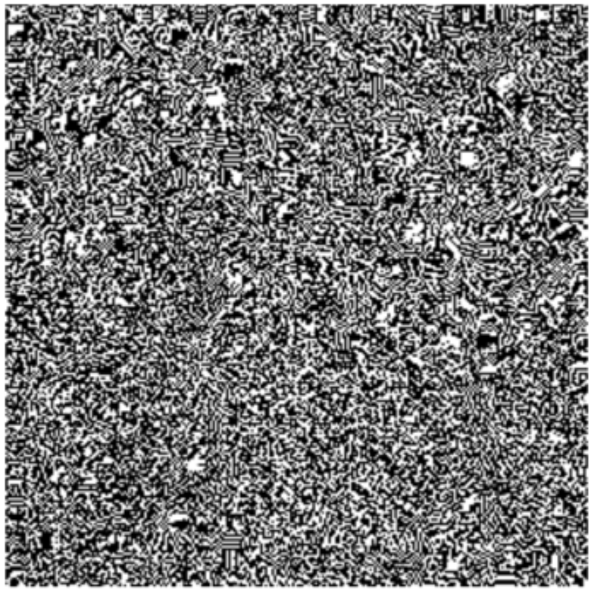
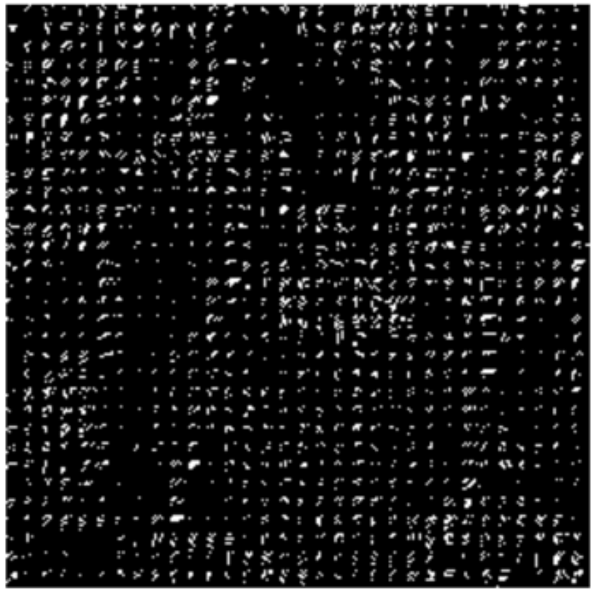


Image from en.wikipedia.org

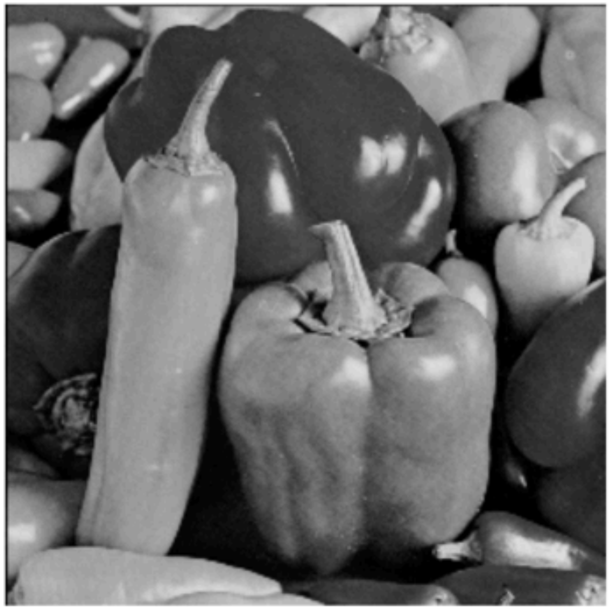
DCT Example

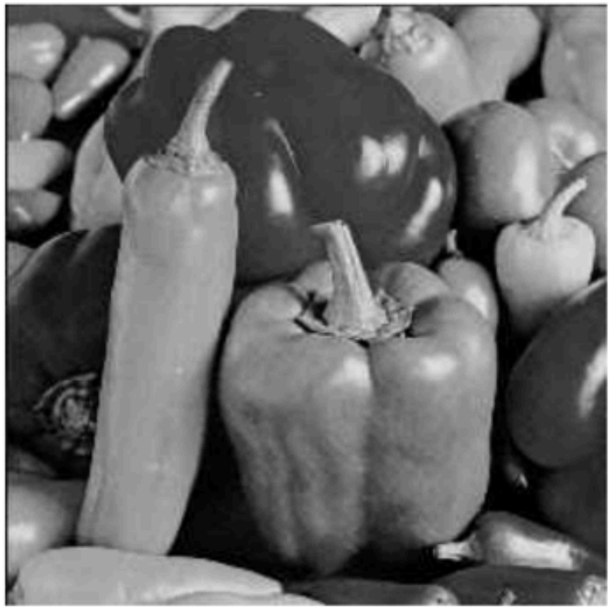


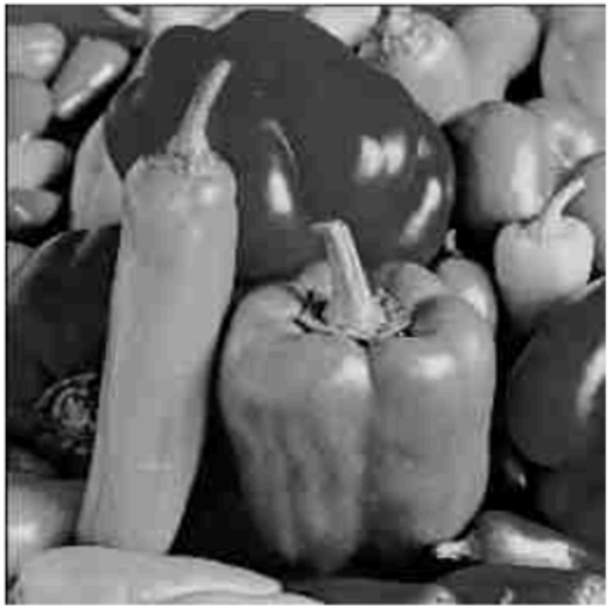
DCT of peppers

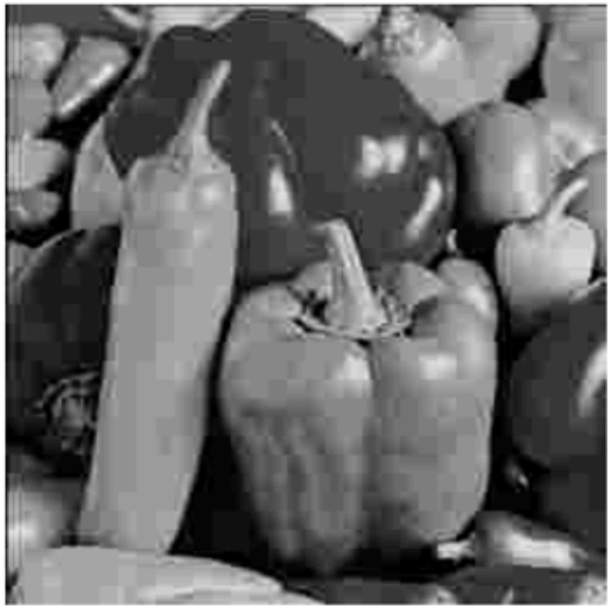


Quantized DCT of peppers


Original

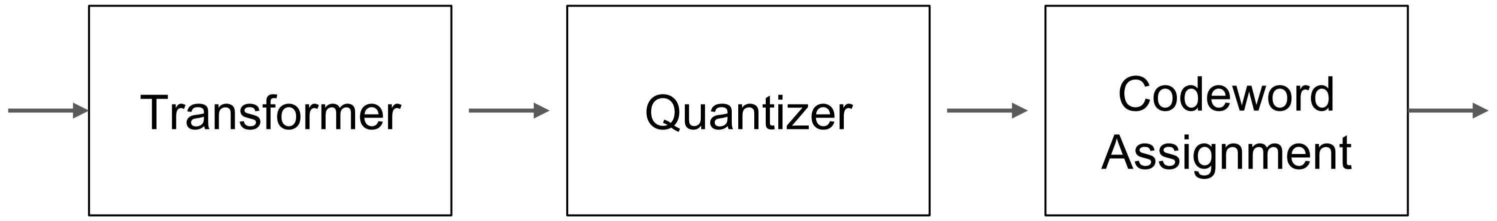

Quality 50 – 84% zeros


Quality 20 – 91% zeros


Quality 10 – 94% zeros

Don't Panic.
Practice!

Next Time...



Discrete Fourier Transform (DFT)
Discrete Cosine Transform (DCT)

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

- How do they fit together?
- How to achieve best compression?
- Relation to human audio and visual system?