T1 – JPEG and JPEG 2000

upf. Universitat Pompeu Fabra Barcelona

JPEG

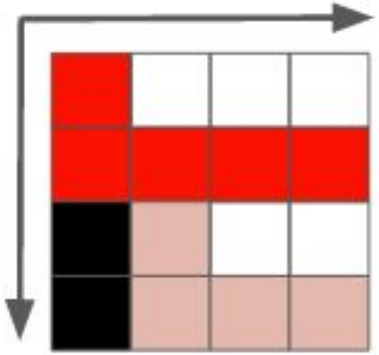
Pixel (picture element)

Represents the intensity (usually a numeric value) of a given color.

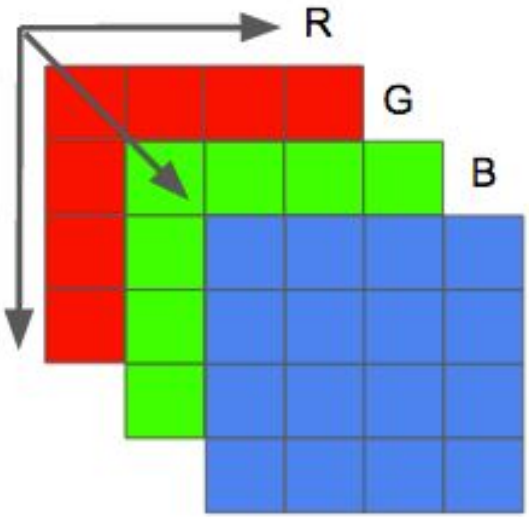
For example:

Red Pixel: 0 of green + 0 of blue + maximum of red

Pink Pixel: 192 of green + 203 of blue + 255 of red



2D

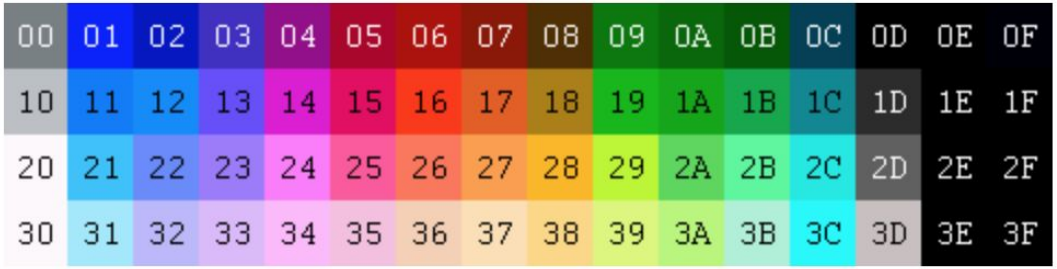


3D

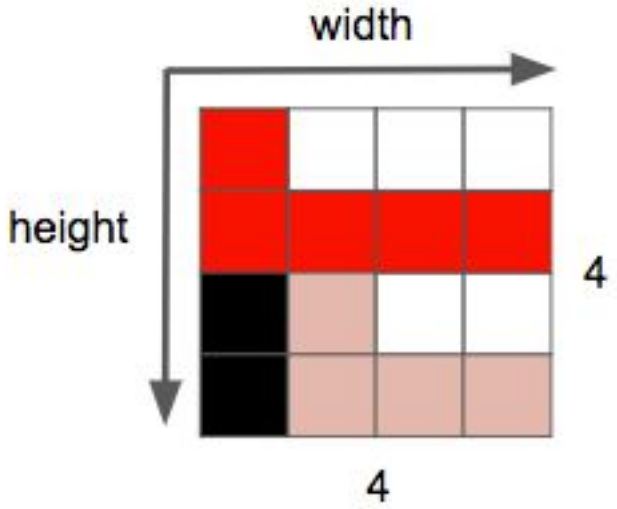
color intensity

Other ways to encode a color image

Many other possible models may be used to represent the colors that make up an image. We could, for instance, use an indexed palette where we'd only need a single byte to represent each pixel instead of the 3 needed when using the RGB model. In such a model we could use a 2D matrix instead of a 3D matrix to represent our color, this would save on memory but yield fewer color options.



Resolution: number of pixels in 1 dimension



Joint Photographic Experts Group

ISO norm created in 1983

Lossy compression method

It also stands for files (.jpg)

Supports true color (24 bits):

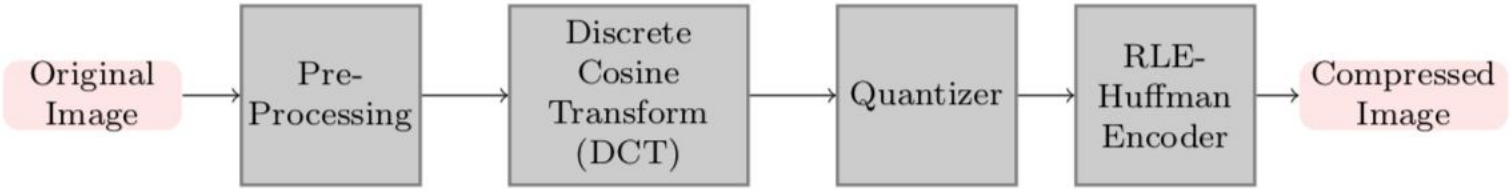
That means 16.777.216 different colors

It's compression is a mathematical model based. This is the formula

$$G_{u,v} = \alpha(u)\alpha(v) \sum_{x=0}^{7} \sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x + \frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y + \frac{1}{2}\right)v\right]$$

$$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

...and this is the encoding diagram



...and of course, the decoding diagram

Our human eye:

- We recognize objects equally well regardless of image size

- Recognition speed doesn’t depend on image size

PROS: nice compression of data.
Extremely used in computing and Internet

upf. Universitat Pompeu Fabra Barcelona

Storage: 83 kilobytes

Storage: 10 kilobytes

about 1/8 the storage and 8 times faster !

CONS: Lossy method

a. Original image

b. With 10:1 compression

c. With 45:1 compression

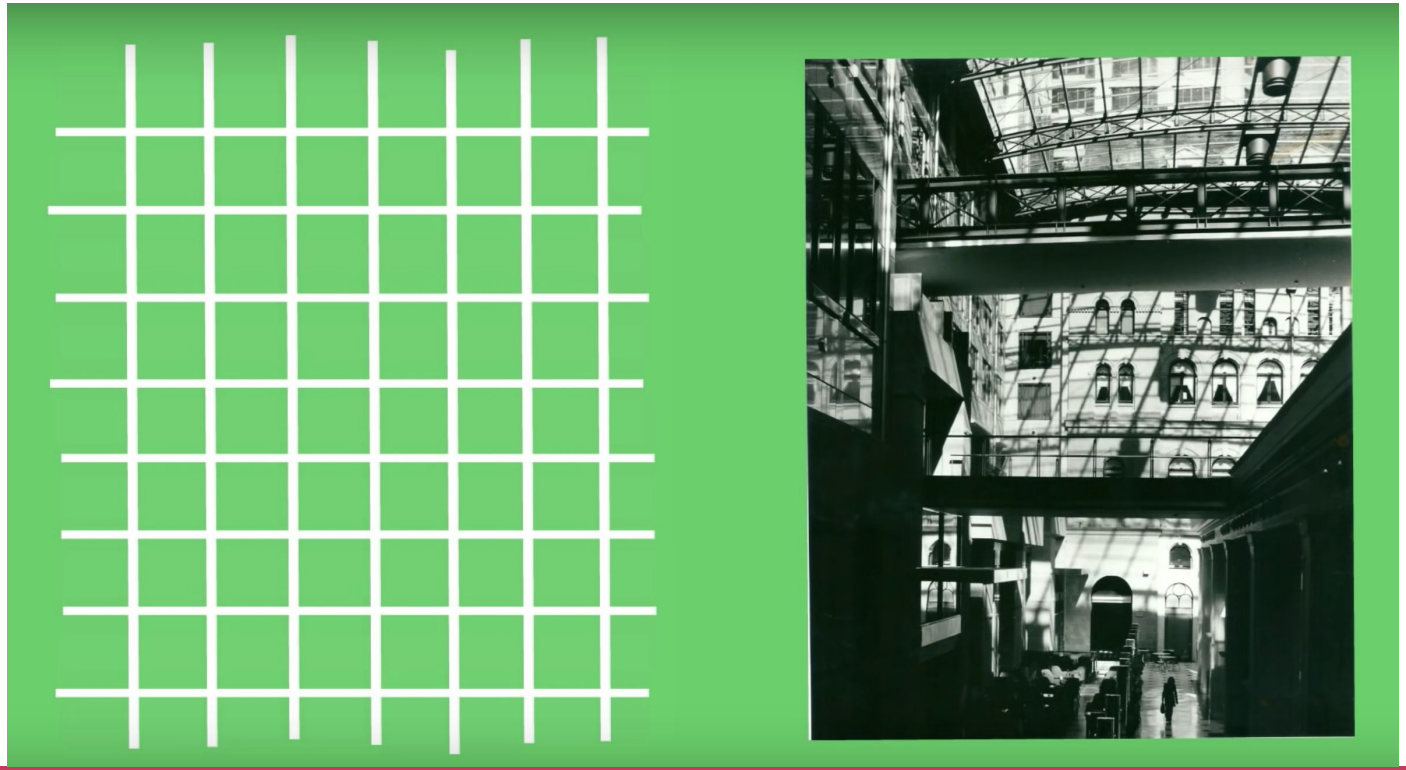
FIGURE 27-15
Example of JPEG distortion. Figure (a) shows the original image, while (b) and (c) shows restored images using compression ratios of 10:1 and 45:1, respectively. The high compression ratio used in (c) results in each 8×8 pixel group being represented by less than 12 bits.

DCT – Discrete Cosine Transform.

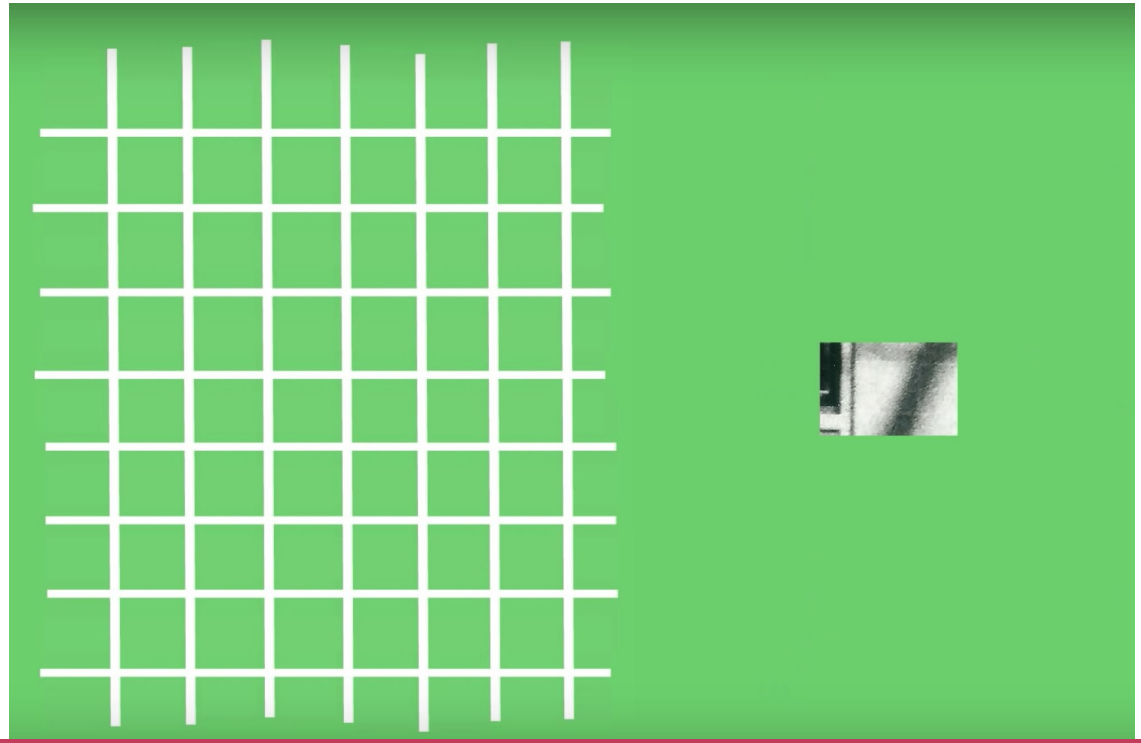
How does it work?

upf. Universitat Pompeu Fabra Barcelona

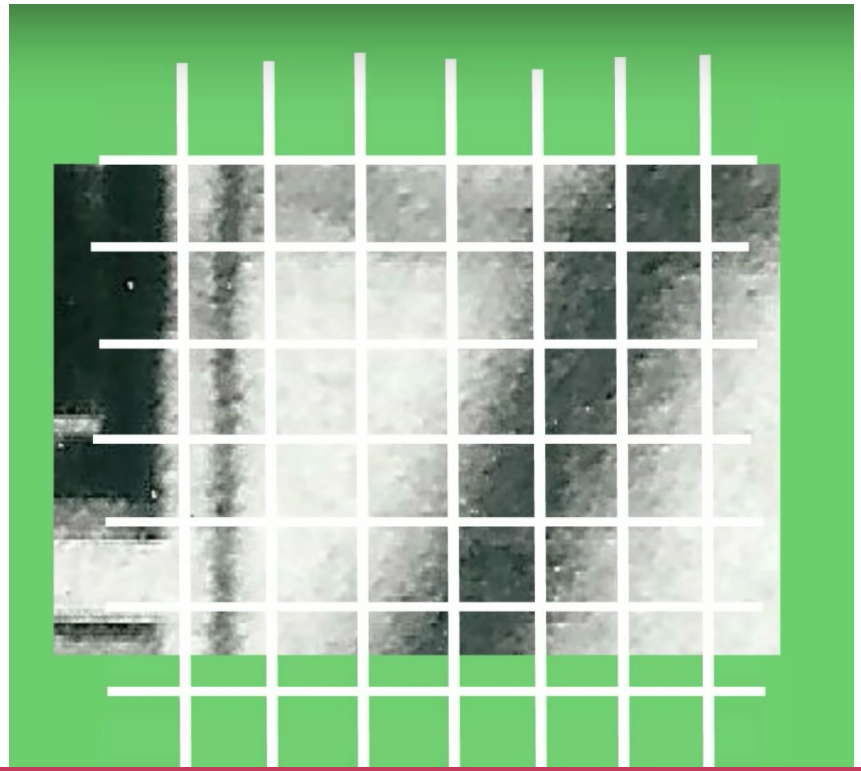
RAW Data: analogic B&W picture



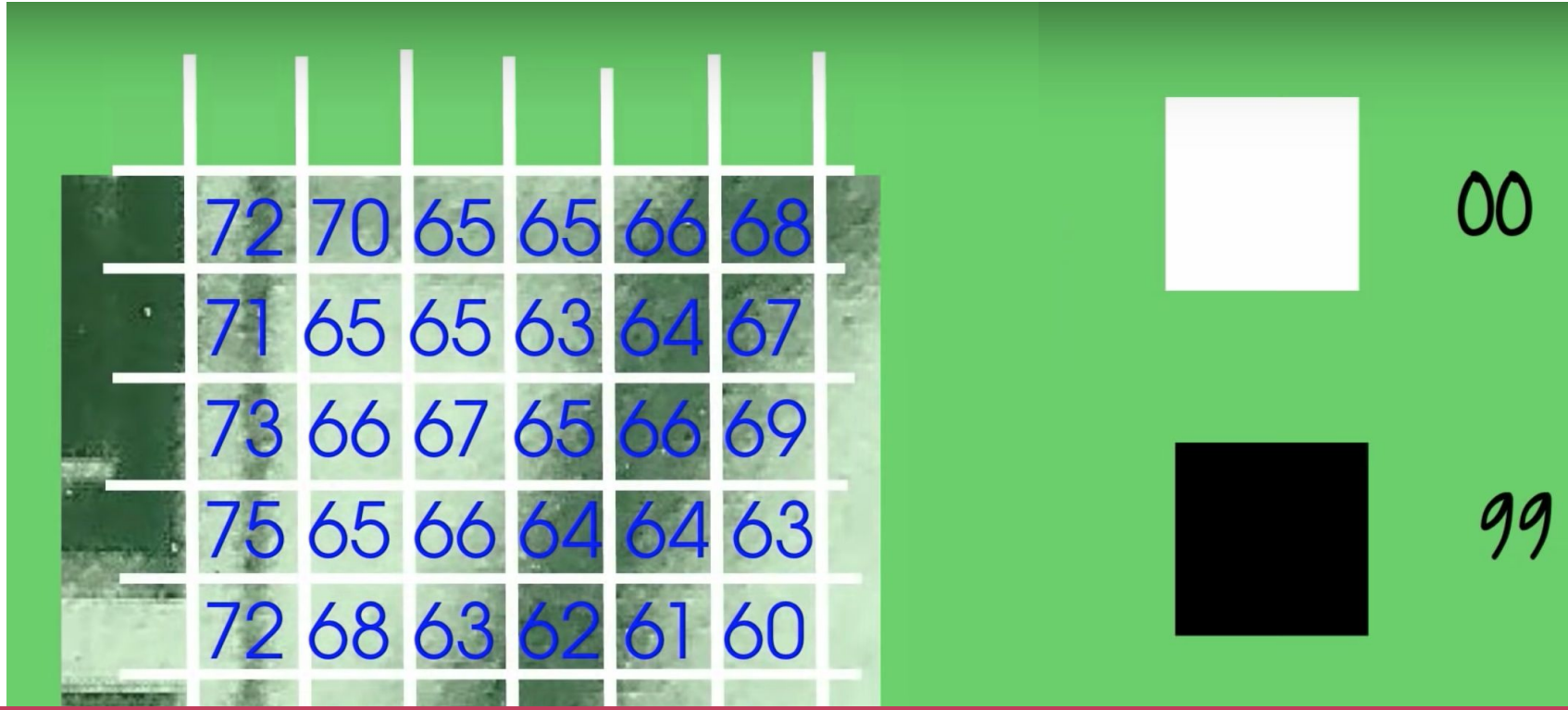
We take a small part into the grid



We take a small part into the grid



We quantize


72 70 65 65 66 68
71 65 65 63 64 67
73 66 67 65 66 69
75 65 66 64 64 63
72 68 63 62 61 60
00
99

upf. Universitat Pompeu Fabra Barcelona

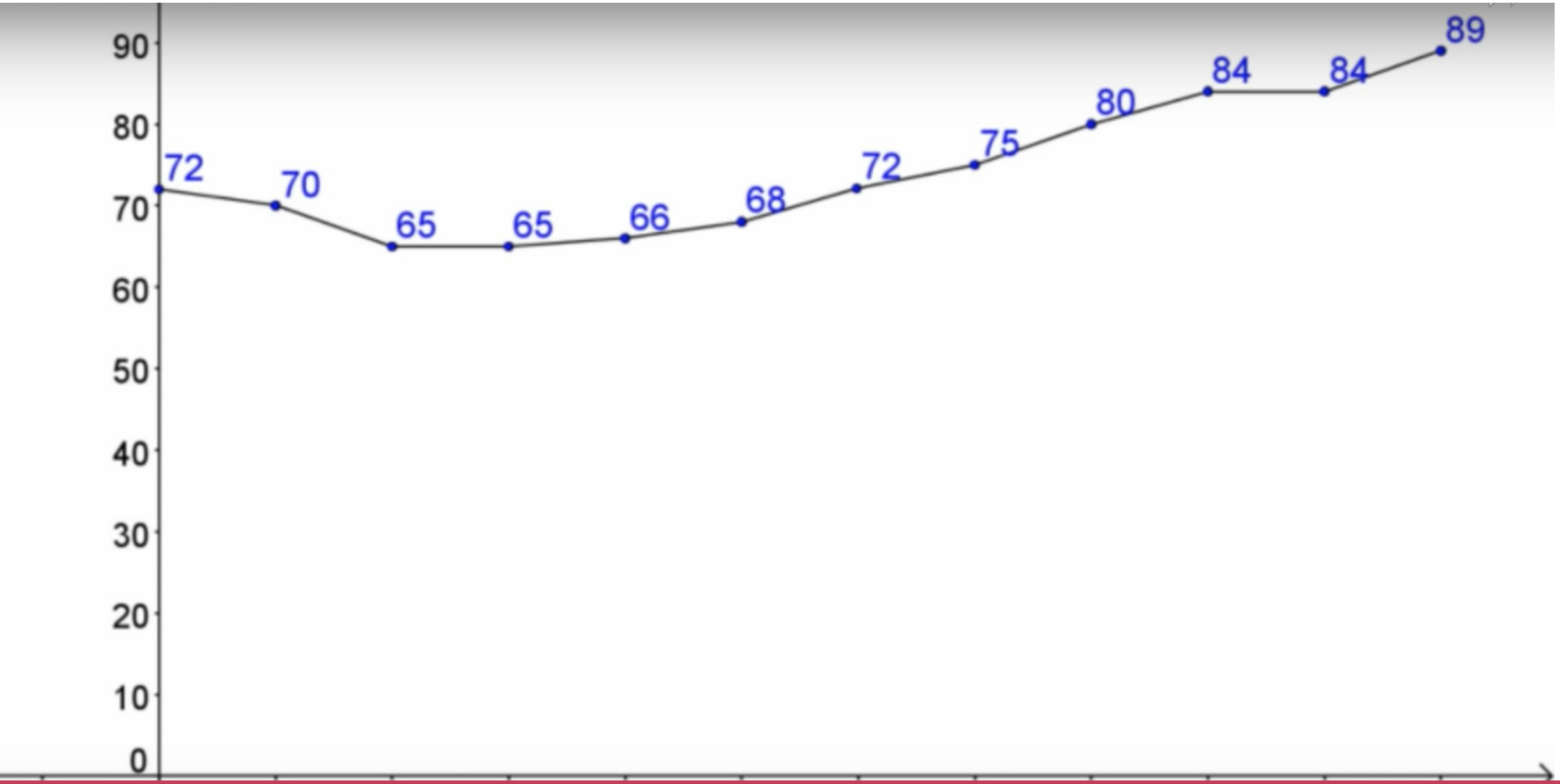
How do we compress?

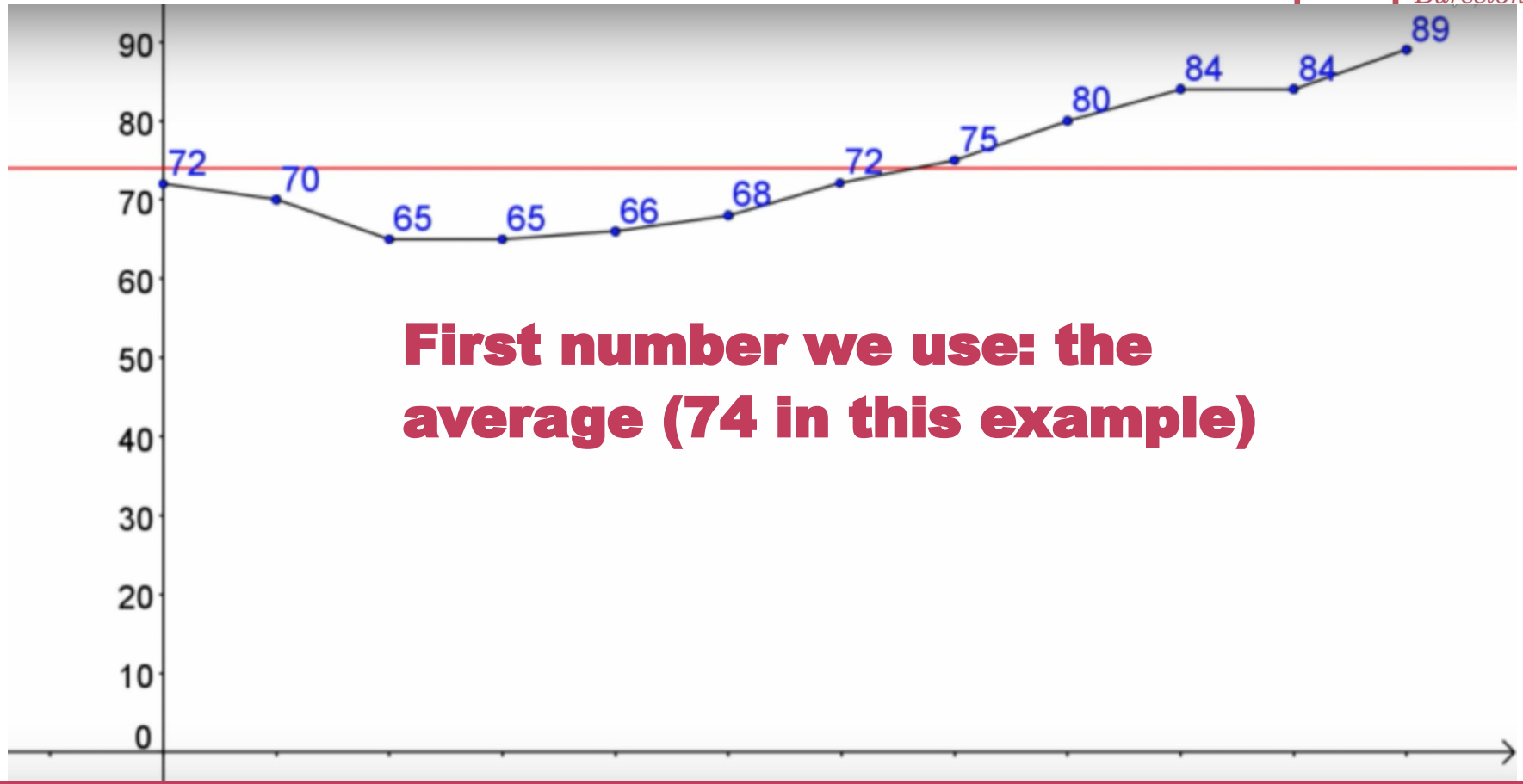
72	70	65	65	66	68
71	65	65	63	64	67
73	66	67	65	66	69
75	65	66	64	64	63
72	68	63	62	61	60

Turns into

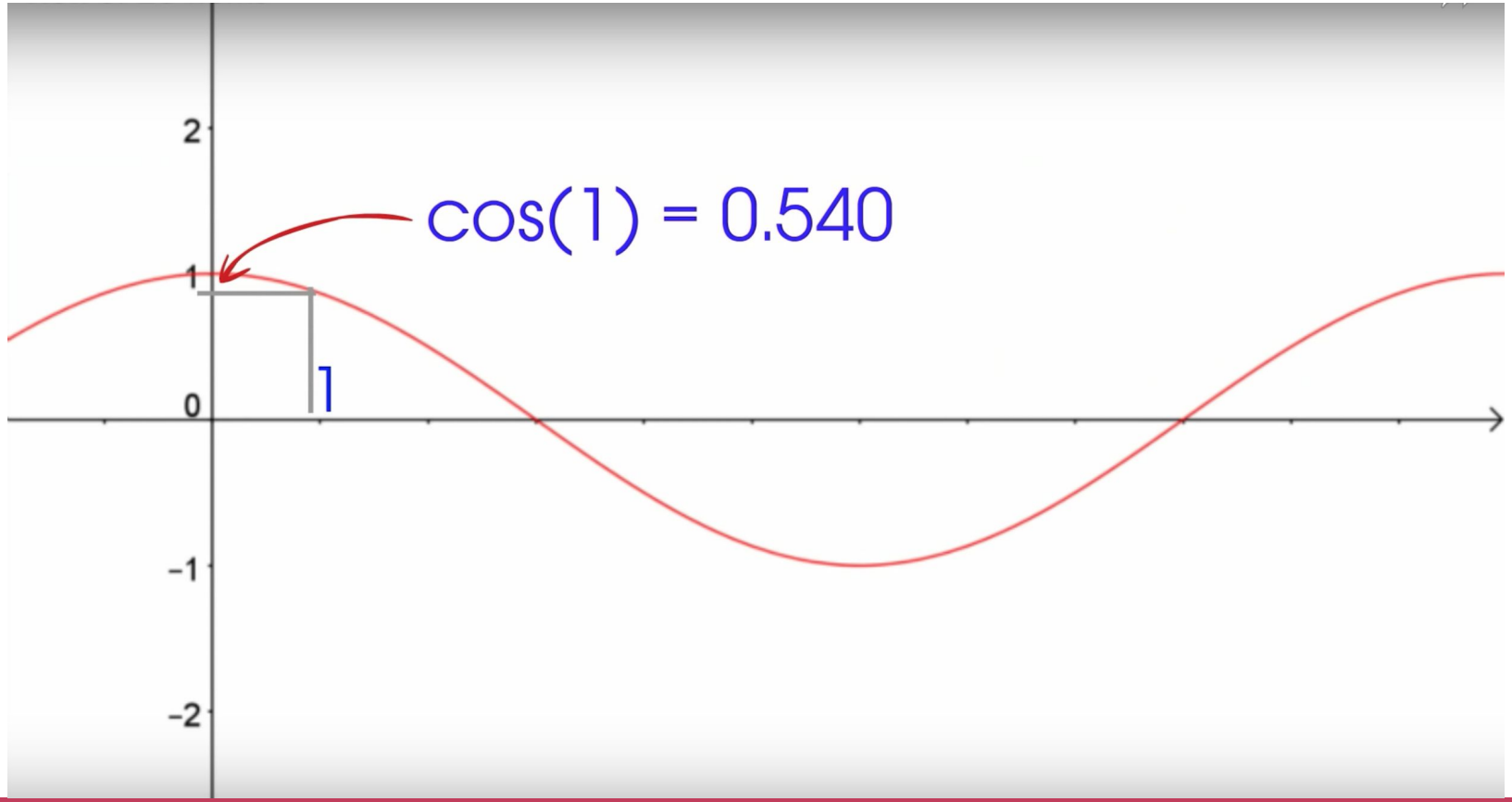
70	12	4
-3	2	
1		

72 70 65 65 66 68 72 75 80 84 84 89


90
80
70
60
50
40
30
20
10
0
72
70
65
65
66
68
72
75
80
84
84
89



First number we use: the average (74 in this example)


cos(1) = 0.540
1
2
1
0
-1
-2

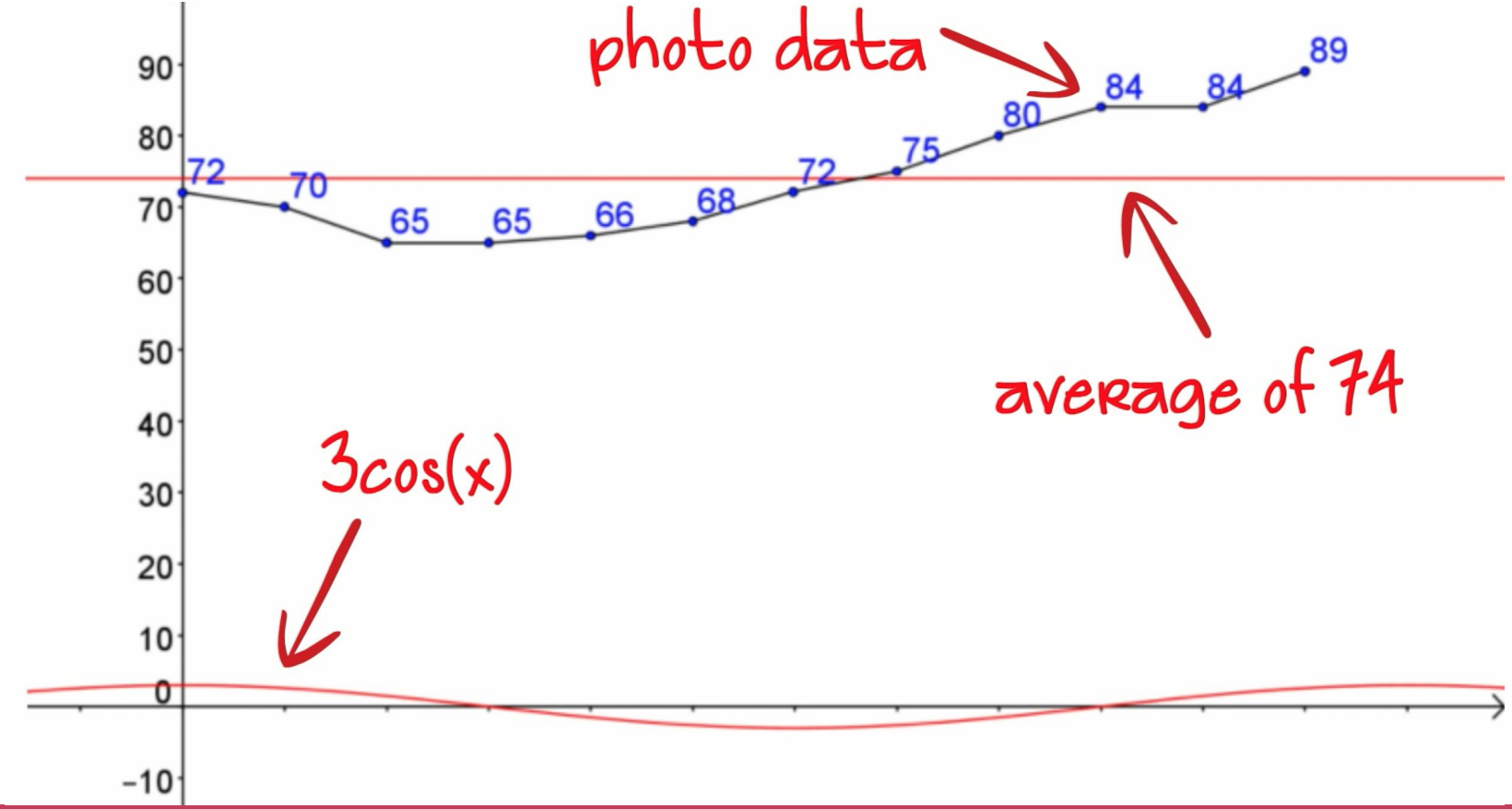
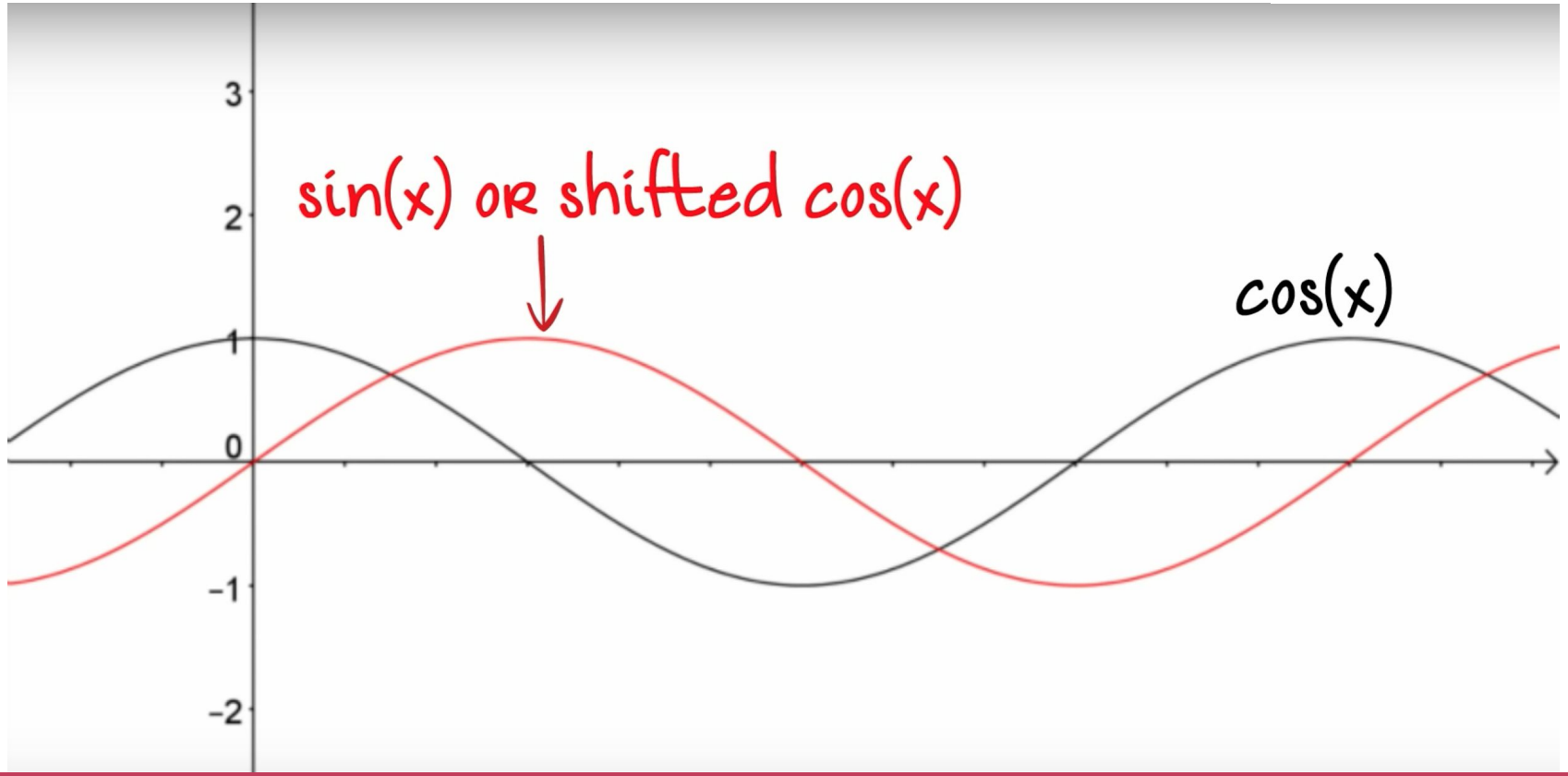
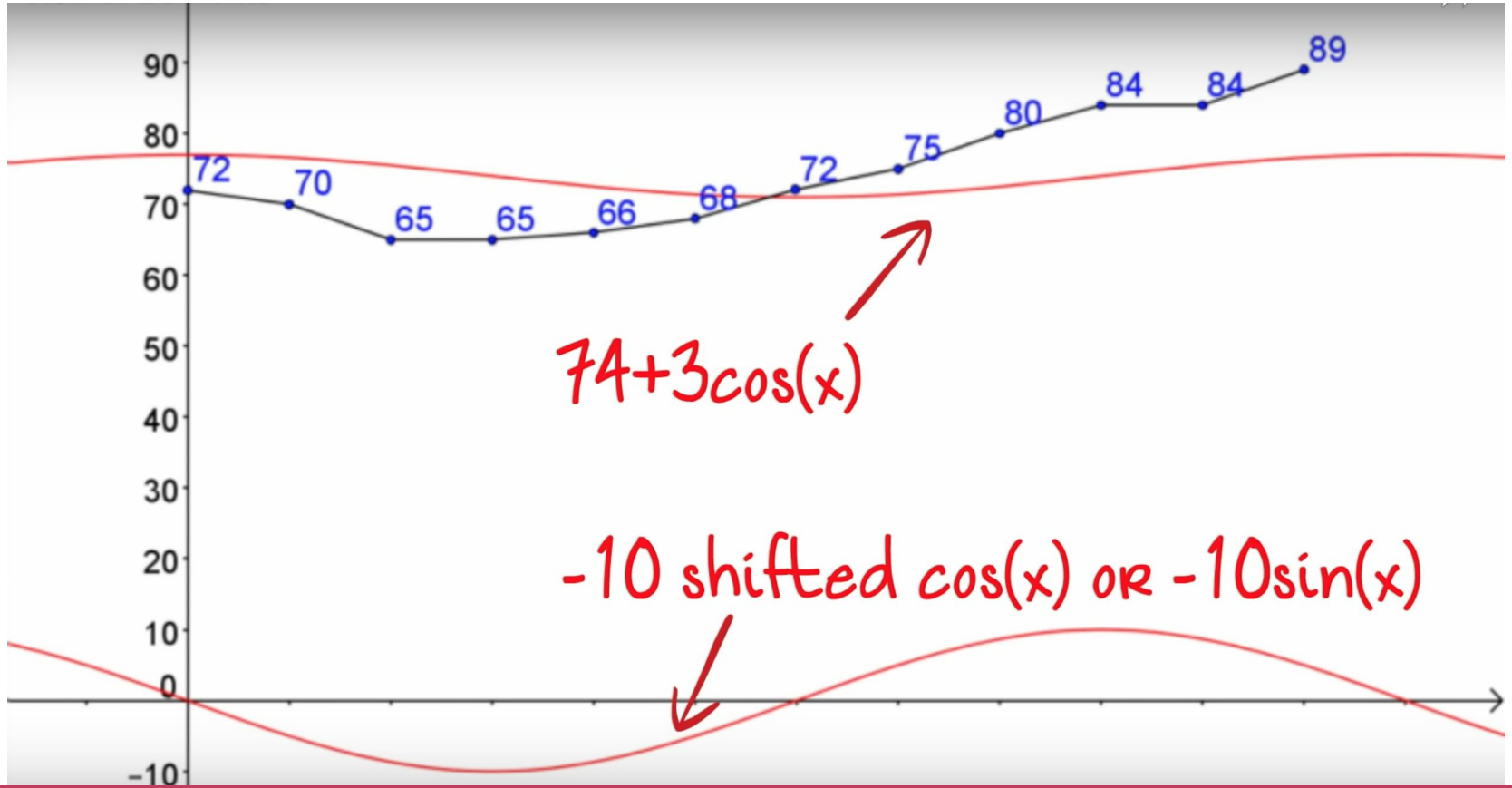


photo data
72
70
65
65
66
68
72
75
80
84
84
89
average of 74
3cos(x)
90
80
70
60
50
40
30
20
10
0
-10


3
2
1
0
-1
-2
sin(x) or shifted cos(x)
cos(x)


90
80
70
60
50
40
30
20
10
0
-10
72
70
65
65
66
68
72
75
80
84
84
89
74+3cos(x)
-10 shifted cos(x) or -10sin(x)


90
80
70
60
50
40
30
20
10
72
70
65
65
66
68
72
75
80
84
84
89
74+3cos(x)-10sin(x)

Remember the formula?

$$G_{u,v} = \alpha(u)\alpha(v) \sum_{x=0}^{7} \sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x + \frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y + \frac{1}{2}\right)v\right]$$

$$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

upf. Universitat Pompeu Fabra Barcelona

Remember the formula?

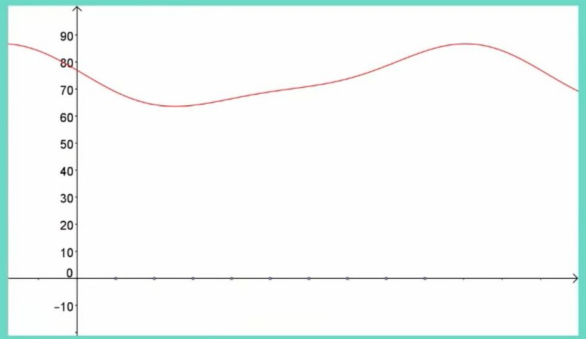
$$G_{u,v} = \alpha(u)\alpha(v) \sum_{x=0}^{7} \sum_{y=0}^{7} g_{x,y} \cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

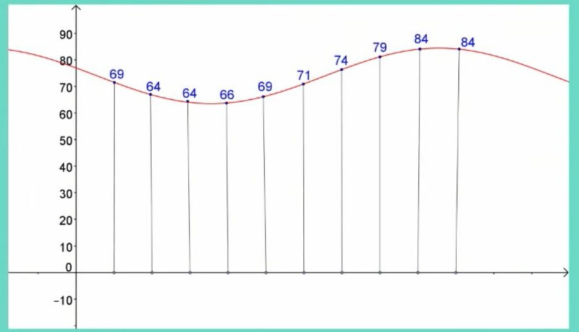
$$\alpha_p(n) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } n = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$$

72 70 65 65 66 72 75 80 84 84 89

74 3 -10

$$74+3\cos(x)-10\sin(x)$$





69 64 64 66 69 71 74 79 84 84

72 70 65 65 66 68 72 75 80 84 84 89

or

74 3 -10

75% less storage space!
4 times faster!

Remember run-length encoding? It's also applied here

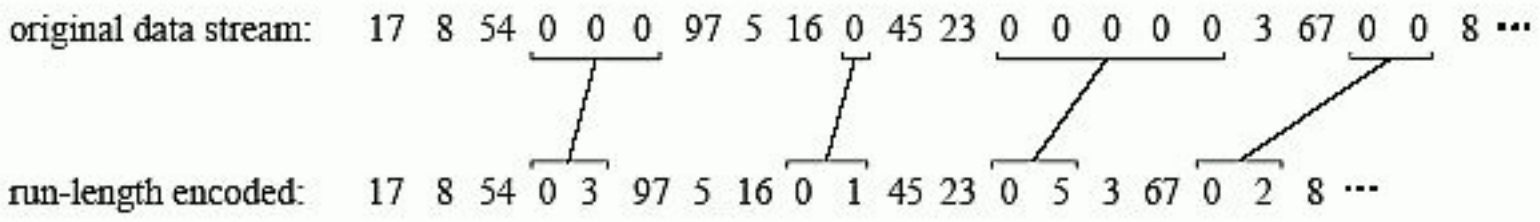
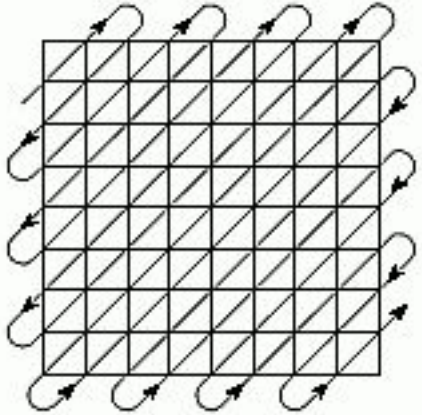
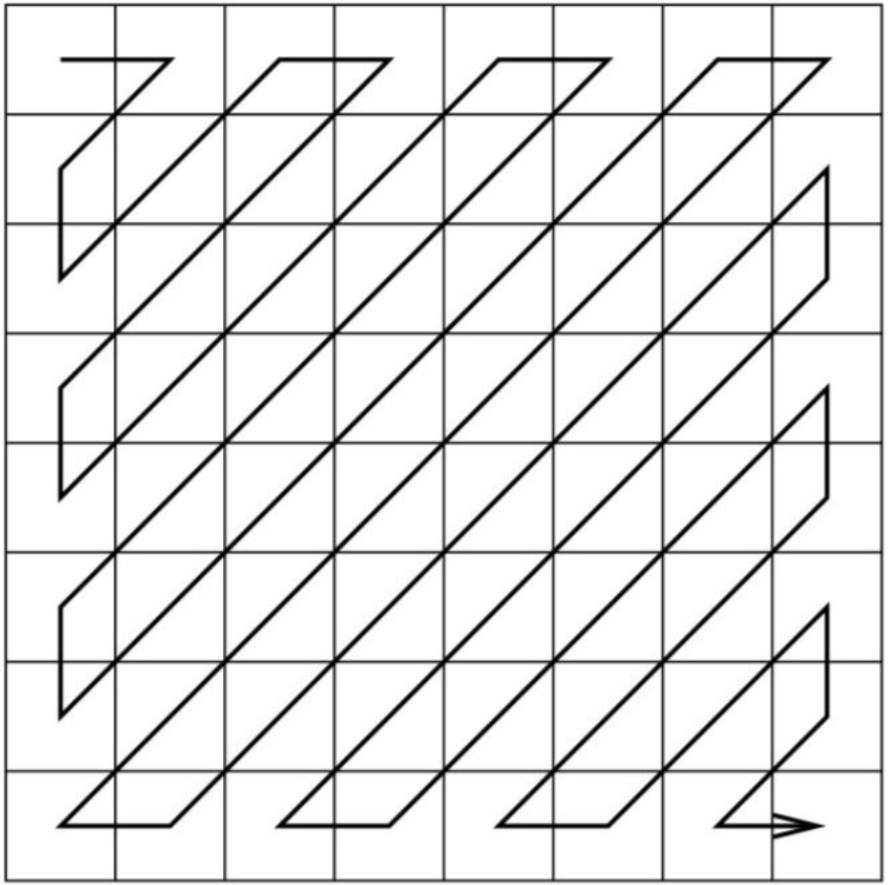


FIGURE 27-1
Example of run-length encoding. Each run of zeros is replaced by two characters in the compressed file: a zero to indicate that compression is occurring, followed by the number of zeros in the run.

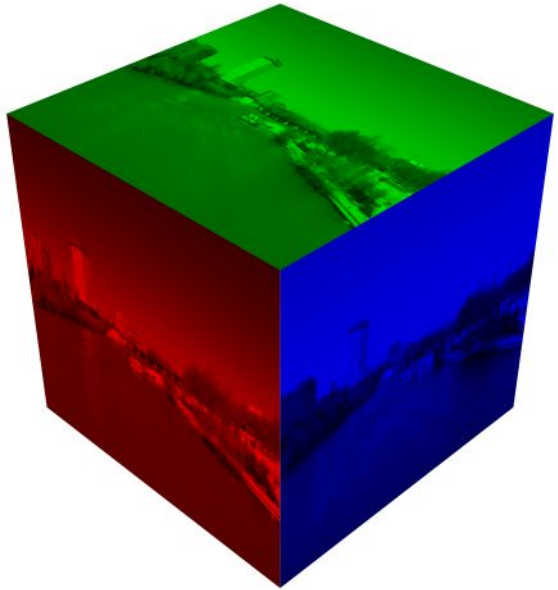
FIGURE 27-14
JPEG serial conversion. A serpentine pattern used to convert the 8×8 DCT spectrum into a linear sequence of 64 values. This places all of the high frequency components together, where the large number of zeros can be efficiently compressed with run-length encoding.





What about color?

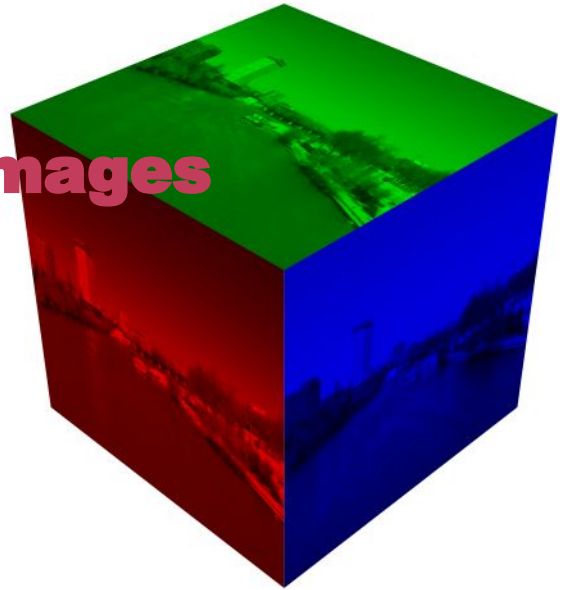
It’s doing x3 the opperation



Used to be

8 bit series for RGB

It’s like compressing 3 different images and sumarizing them...



But JPEG introduced YCbCr model

Y: Luminance

Cb: Chroma blue difference

Cr: Chroma red difference

But JPEG introduced YCbCr model

$$Y' = K_R \cdot R' + K_G \cdot G' + K_B \cdot B'$$

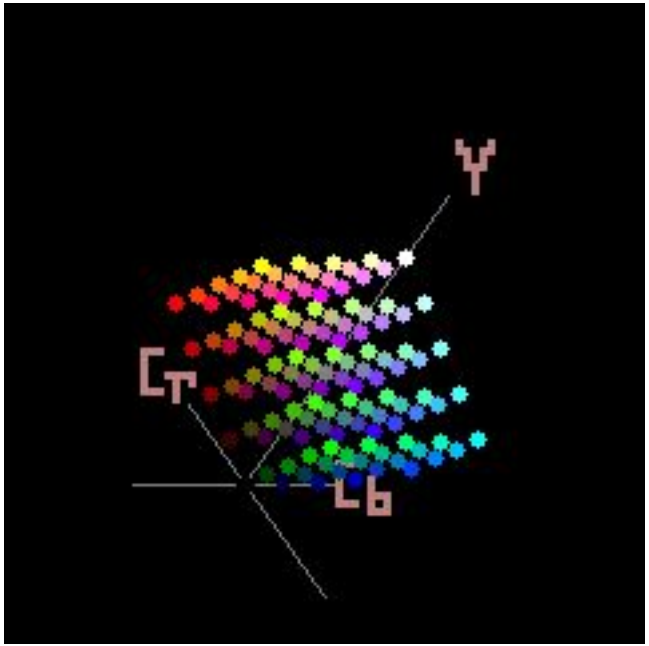
$$P_B = \frac{1}{2} \cdot \frac{B' - Y'}{1 - K_B}$$

$$P_R = \frac{1}{2} \cdot \frac{R' - Y'}{1 - K_R}$$

YCbCr formula from RGB

$$
\begin{aligned}
Y &= 0{,}257 * R + 0{,}504 * G + 0{,}098 * B + 16 \\
Cb = U &= -0{,}148 * R - 0{,}291 * G + 0{,}439 * B + 128 \\
Cr = V &= 0{,}439 * R - 0{,}368 * G - 0{,}071 * B + 128
\end{aligned}
$$

$$
\begin{aligned}
B &= 1{,}164 * (Y - 16) + 2{,}018 * (U - 128) \\
G &= 1{,}164 * (Y - 16) - 0{,}813 * (V - 128) - 0{,}391 * (U - 128) \\
R &= 1{,}164 * (Y - 16) + 1{,}596 * (V - 128)
\end{aligned}
$$


Y

How to solve the quality loss problem?

The JPEG 2000 solution

Joint Photographic Experts Group

JPEG2000:

- New standard presented year 2000 in order to substitute JPEG

- Files extension is .jp2

- Technologically is probably the best possible engineering solution to the problem they had (and that's why we're studying it!)

Cool features

- Superior compression at low bit rates
- Storage of multiple resolutions in a single bitstream
- Precise rate control without re-compression
- Lossy and lossless compression
- Progression by resolution, component, spatial region or quality

JPEG2000:

- However it failed to standarize, as it required the hardware to adapt to this codec (new technology and backwards incompatible with JPEG)

JPEG2000:

- It focuses on the big changes of data

upf. Universitat Pompeu Fabra Barcelona

Remember this slide?

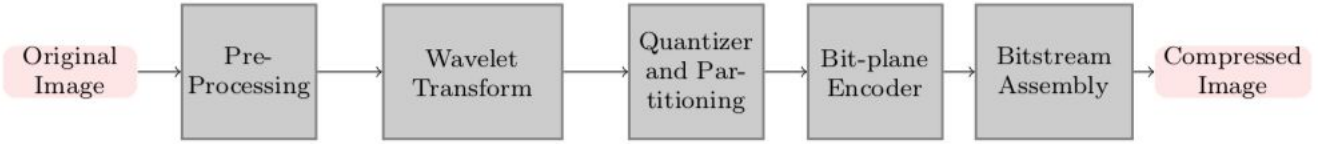
Figure 3.9: Recovered images after JPEG compression with ratios $k = 1, 5, 10, 20$.

SMALL BREAK - the story behind Lenna picture

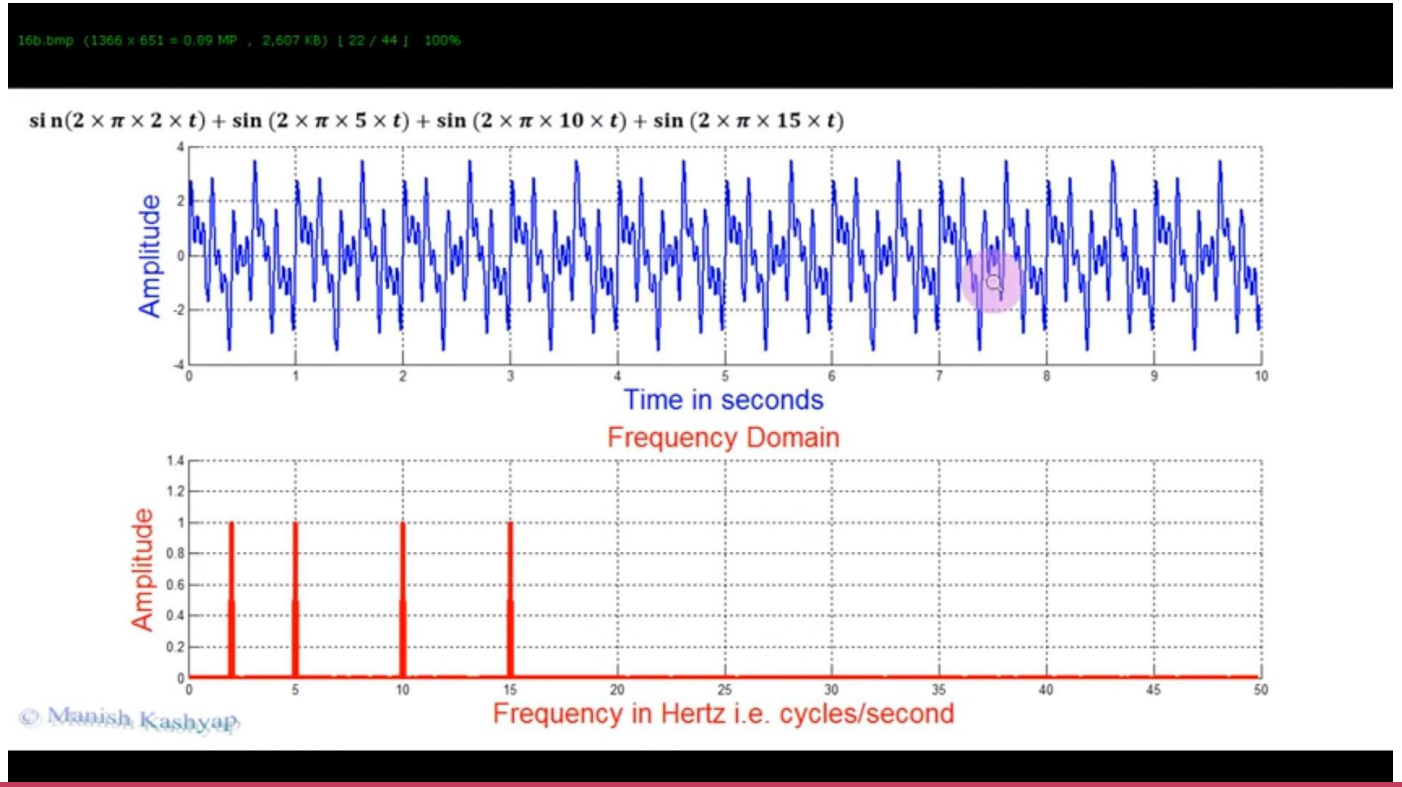
- It’s a sexist image
- I’m using for educational purposes, as most scientifical examples still use it
- If you find alternatives to use it in my slides, please send it to me. You’ll be rewarded for each slide you give me a substitute

(let's continue)

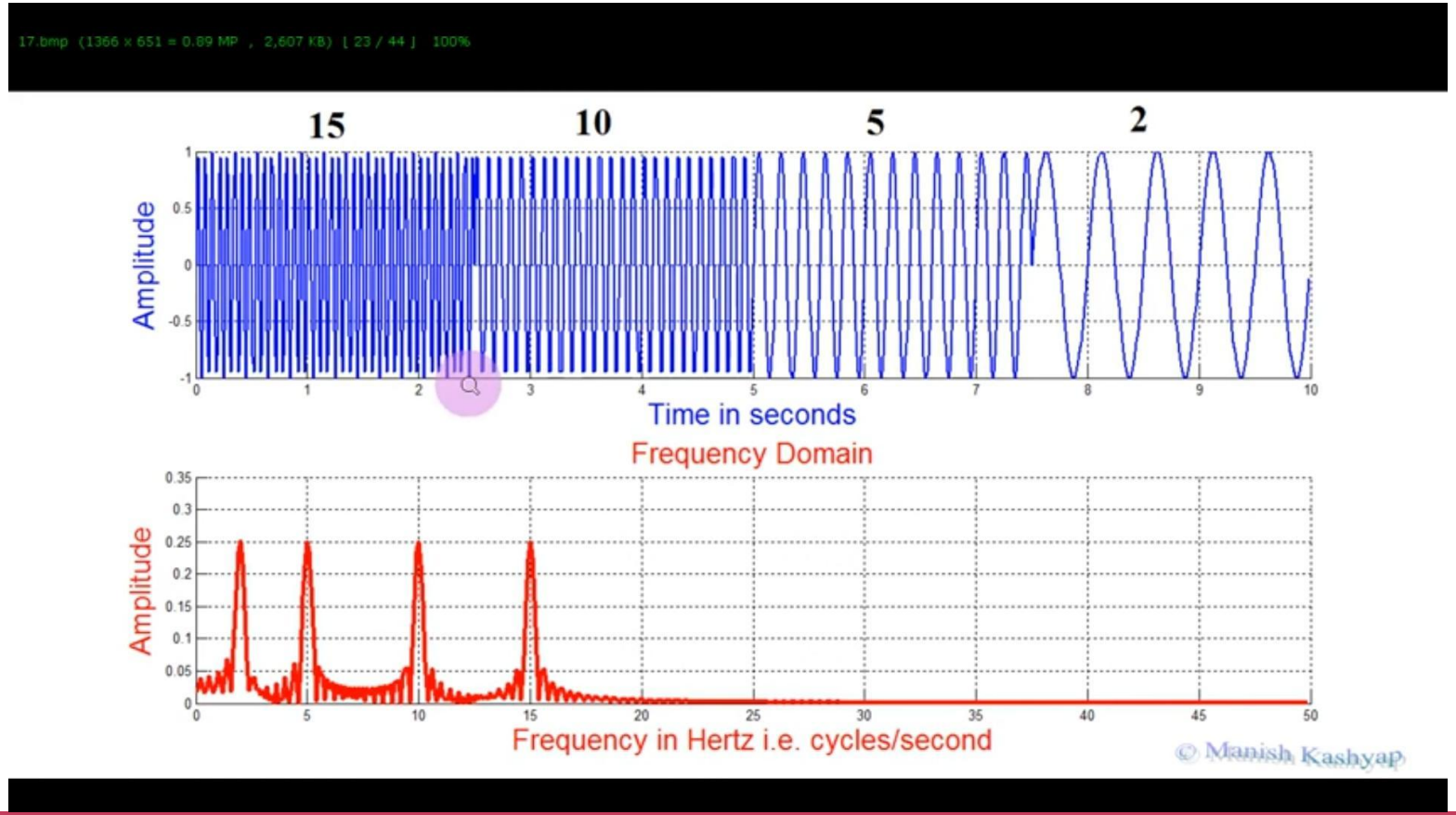
Diagram of the JPEG2000 encoder



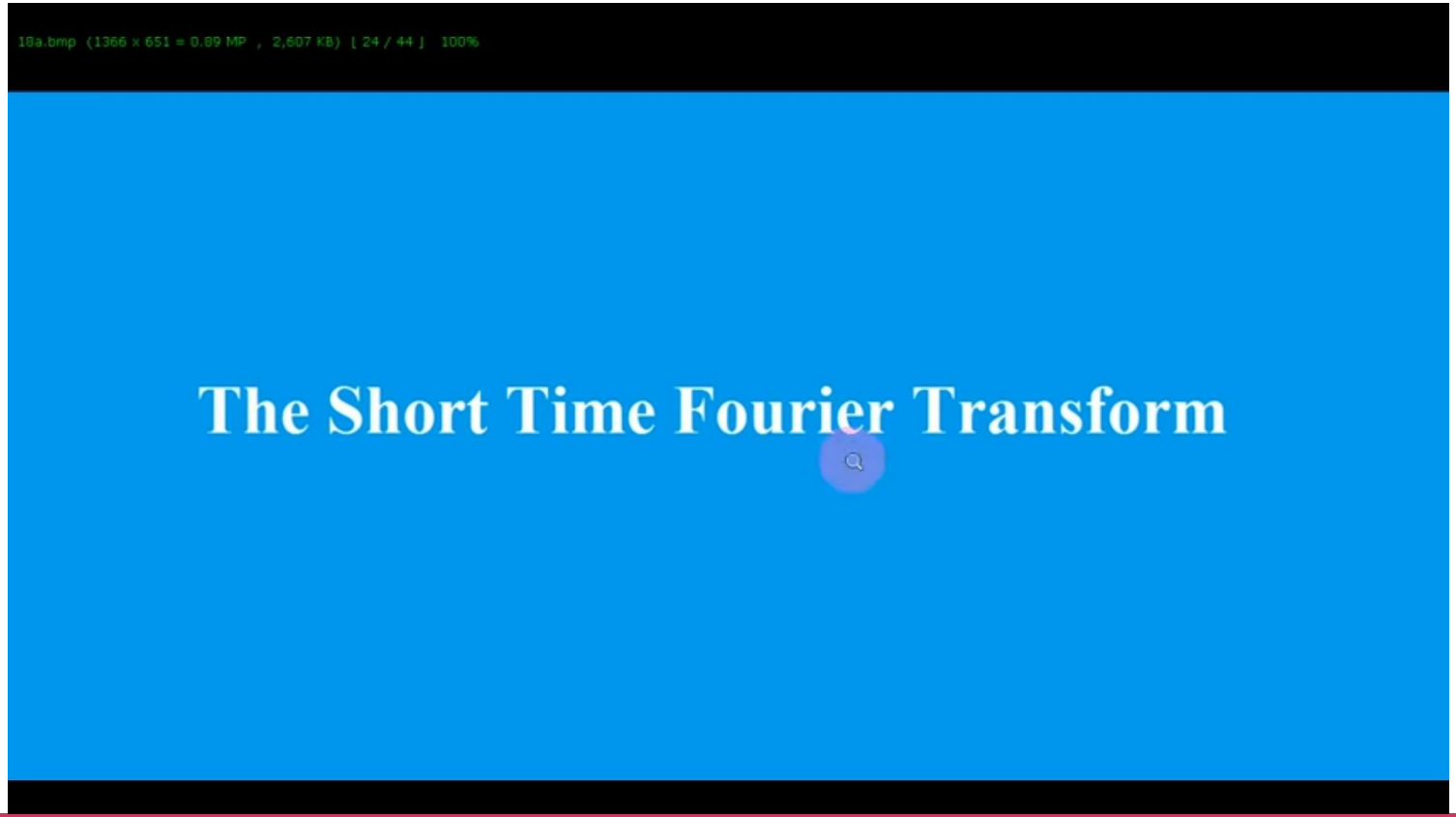
Let's recap. How to solve the quality loss problem? Where was the loss happening?


16b.bmp (1366 x 651 = 0.89 MP , 2,607 KB) [22 / 44] 100%
sin(2 × π × 2 × t) + sin (2 × π × 5 × t) + sin (2 × π × 10 × t) + sin (2 × π × 15 × t)
Amplitude
Time in seconds
Frequency Domain
Amplitude
Frequency in Hertz i.e. cycles/second
© Manish Kashyap


16a.bmp (1366 x 651 = 0.89 MP , 2,607 KB) [21 / 44] 100%
Loss of Location Information in Fourier Transform

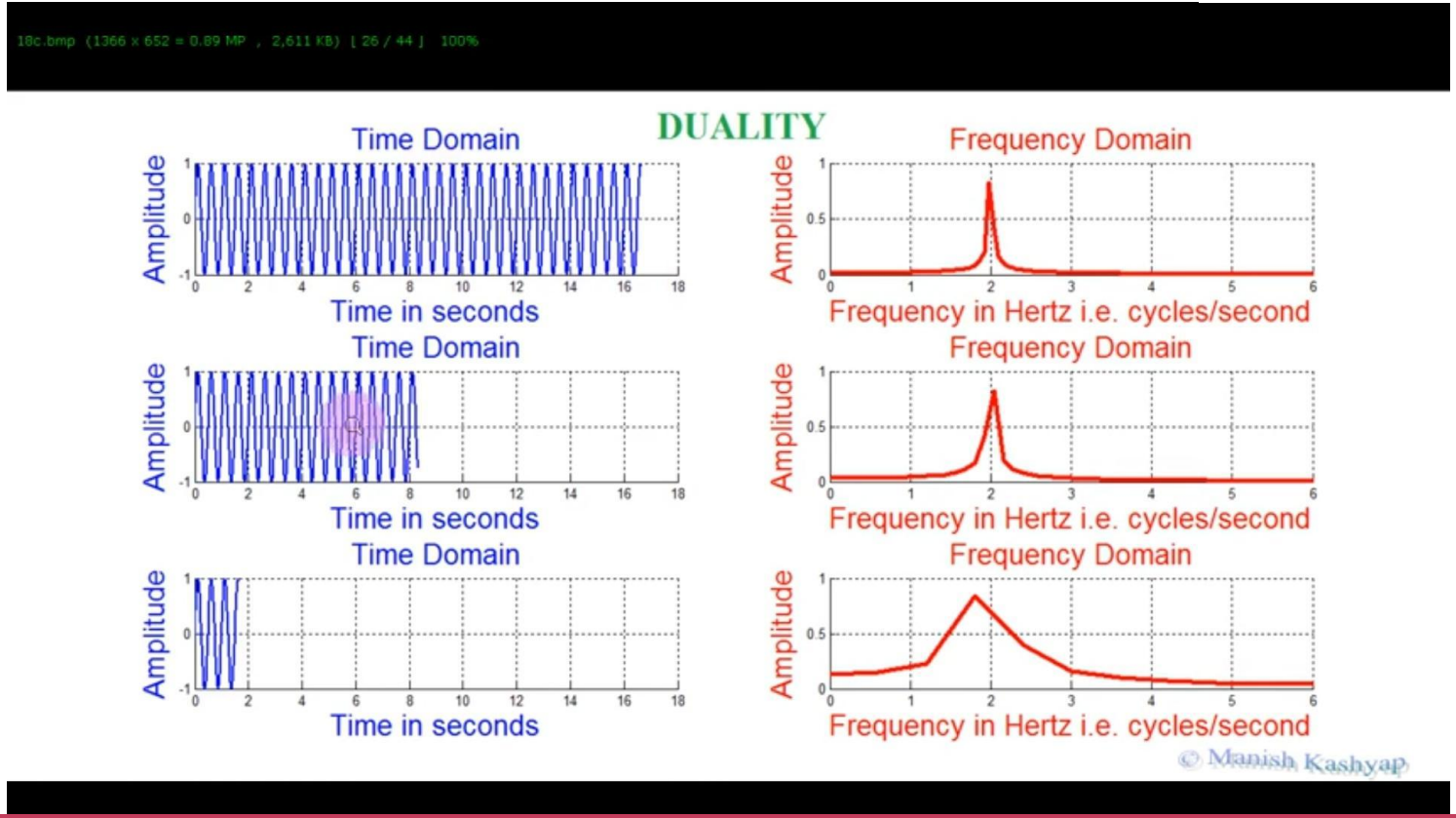

17.bmp (1366 x 651 = 0.89 MP , 2,607 KB) [23 / 44] 100%
15
10
5
2
Amplitude
Time in seconds
Frequency Domain
Amplitude
Frequency in Hertz i.e. cycles/second
© Manish Kashyap

upf. Universitat Pompeu Fabra Barcelona



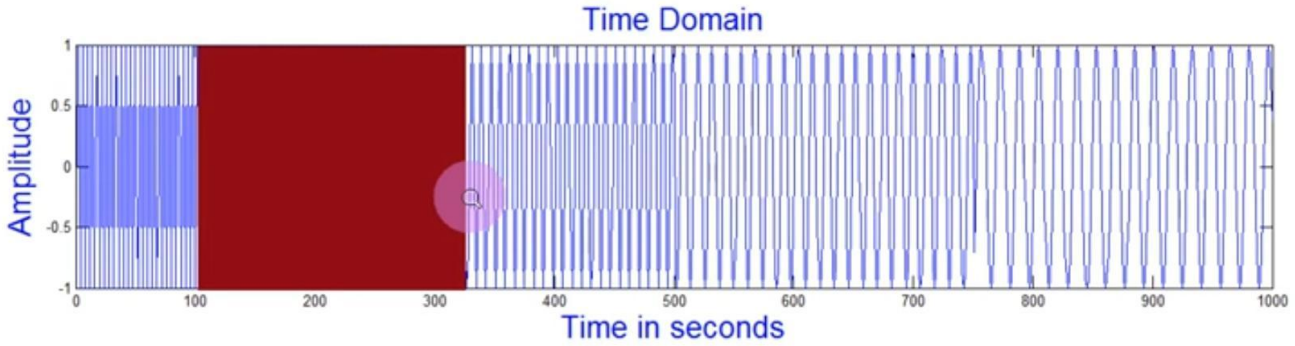
Good Time Resolution **DUALITY** **Bad Frequency Resolution**

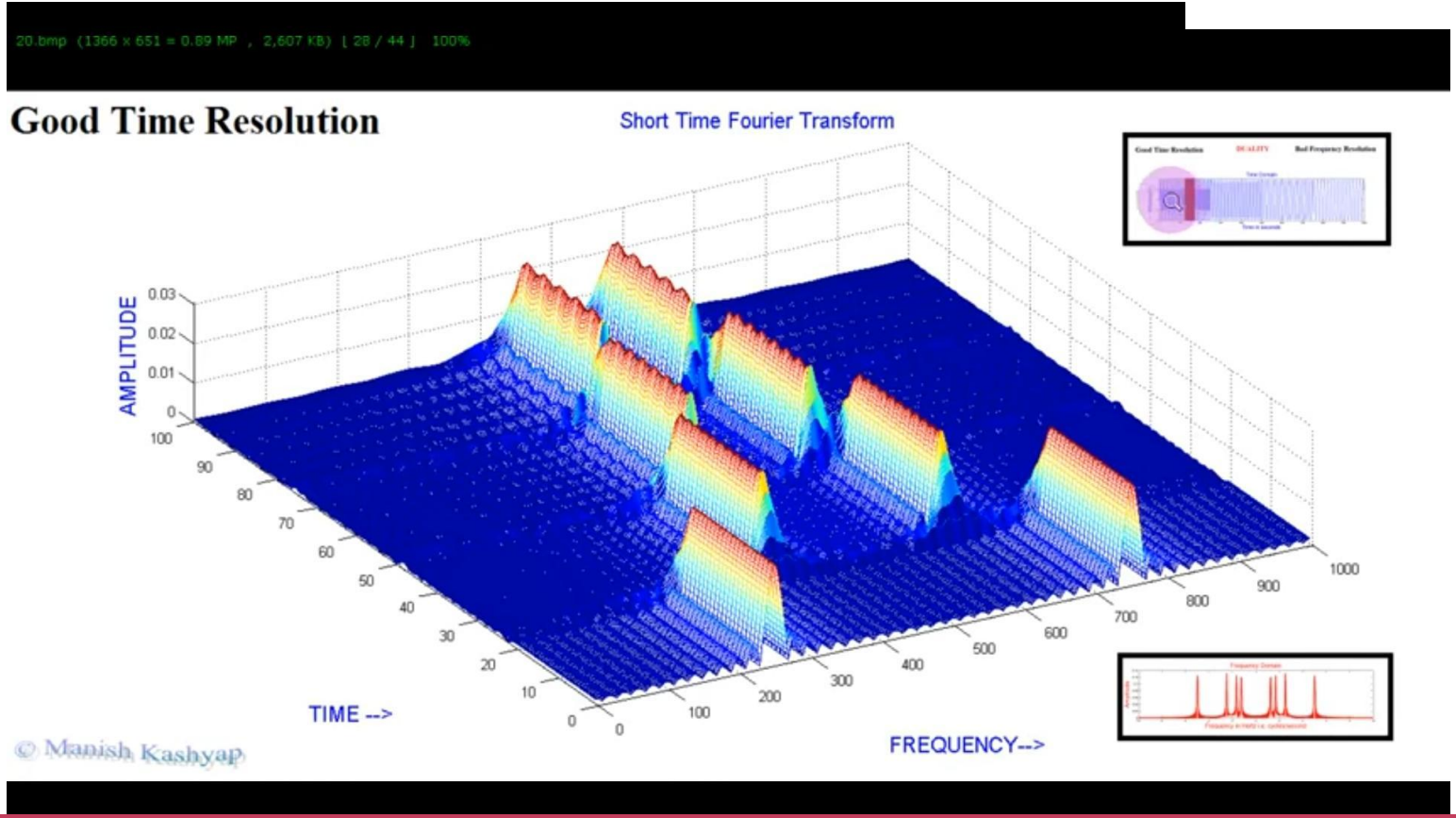
Start from the left, take Fourier Transform of the portion of signal overlaping with window and move the window forward to repeat till end of signal.

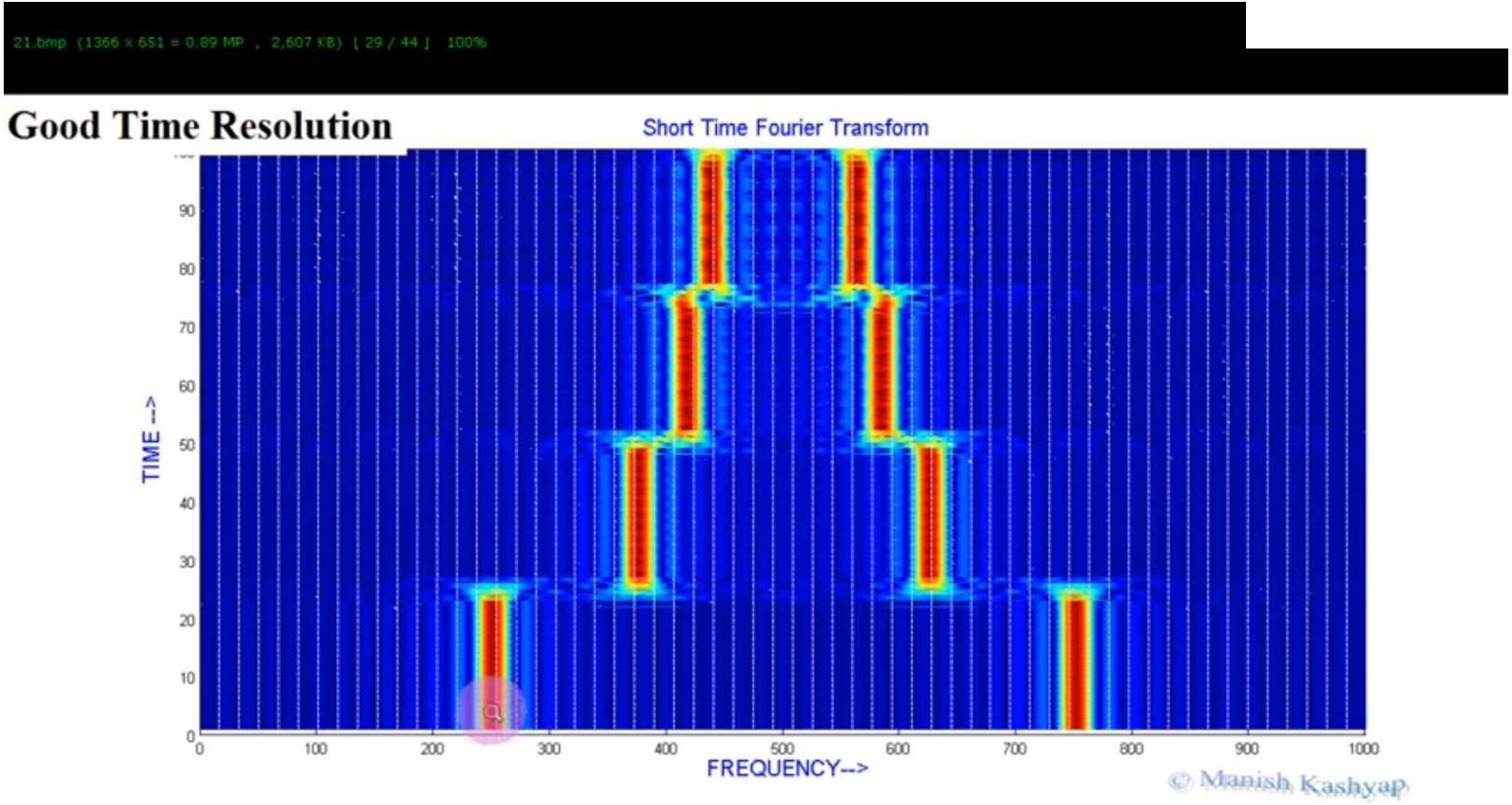

18c.bmp (1366 x 652 = 0.89 MP , 2,611 KB) [26 / 44] 100%
DUALITY
Time Domain
Amplitude
Time in seconds
Frequency Domain
Amplitude
Frequency in Hertz i.e. cycles/second
Time Domain
Amplitude
Time in seconds
Frequency Domain
Amplitude
Frequency in Hertz i.e. cycles/second
Time Domain
Amplitude
Time in seconds
Frequency Domain
Amplitude
Frequency in Hertz i.e. cycles/second
© Manish Kashyap

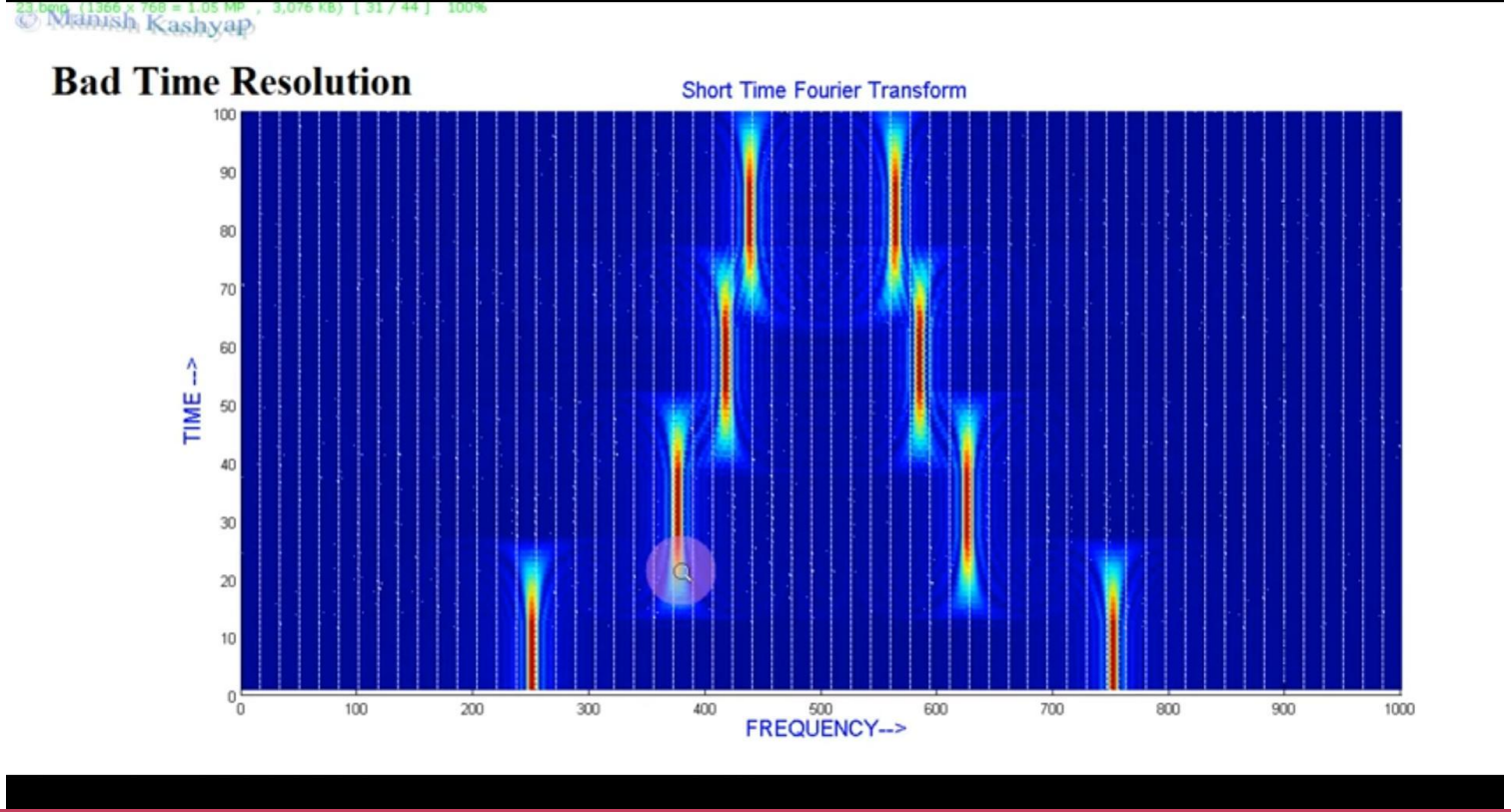
19.bmp (1366 x 651 = 0.89 MP , 2,607 KB) [27 / 44] 100%

Bad Time Resolution **DUALITY** **Good Frequency Resolution**

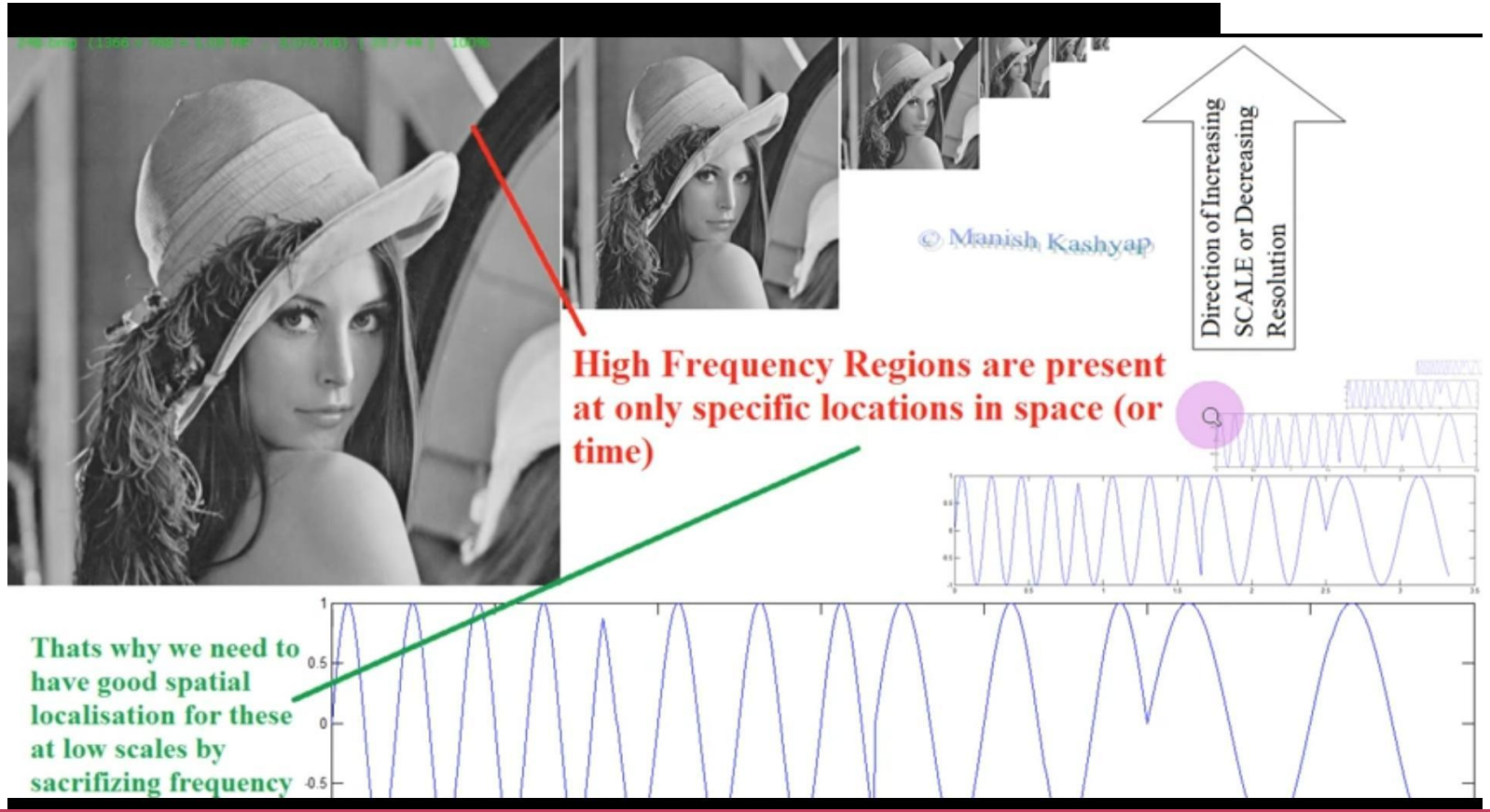



20.bmp (1366 x 651 = 0.89 MP , 2,607 KB) [28 / 44] 100%
Good Time Resolution
Short Time Fourier Transform
AMPLITUDE
0.03
0.02
0.01
0
100
90
80
70
60
50
40
30
20
10
0
TIME -->
0
100
200
300
400
500
600
700
800
900
1000
FREQUENCY-->
© Manish Kashyap


21.bmp (1366 x 651 = 0.89 MP , 2,607 KB) [29 / 44] 100%
Good Time Resolution
Short Time Fourier Transform
TIME -->
FREQUENCY-->
0
10
20
30
40
50
60
70
80
90
0
100
200
300
400
500
600
700
800
900
1000
© Manish Kashyap

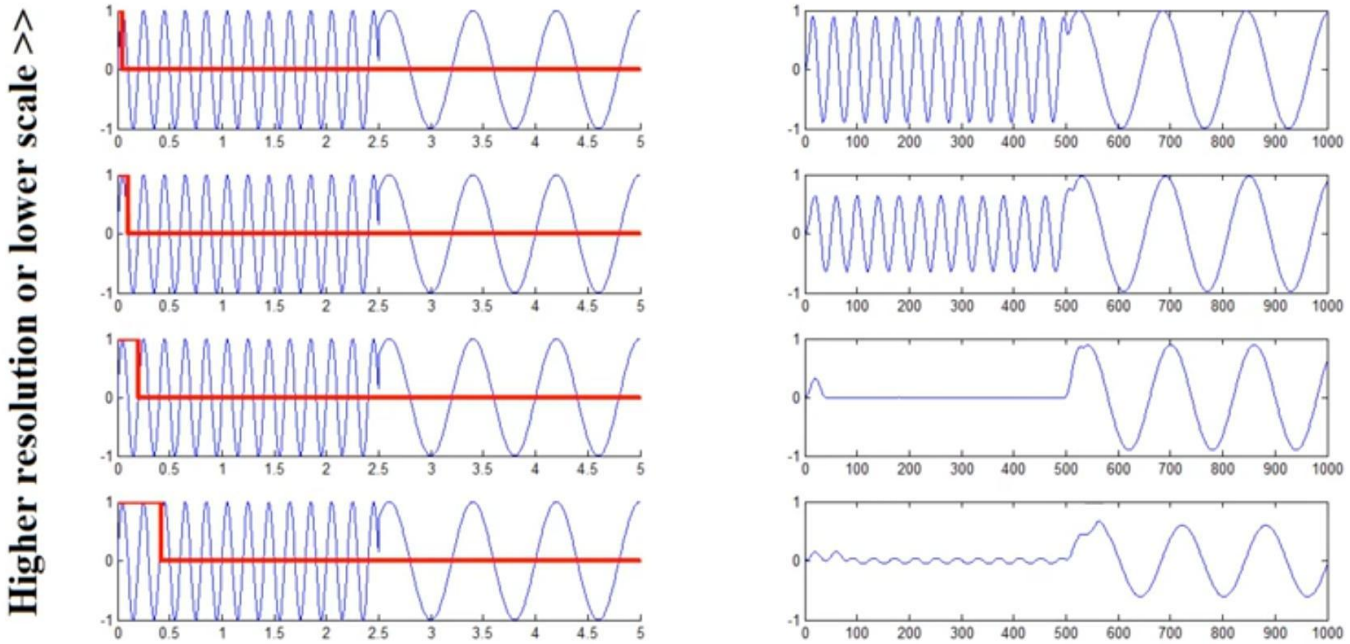


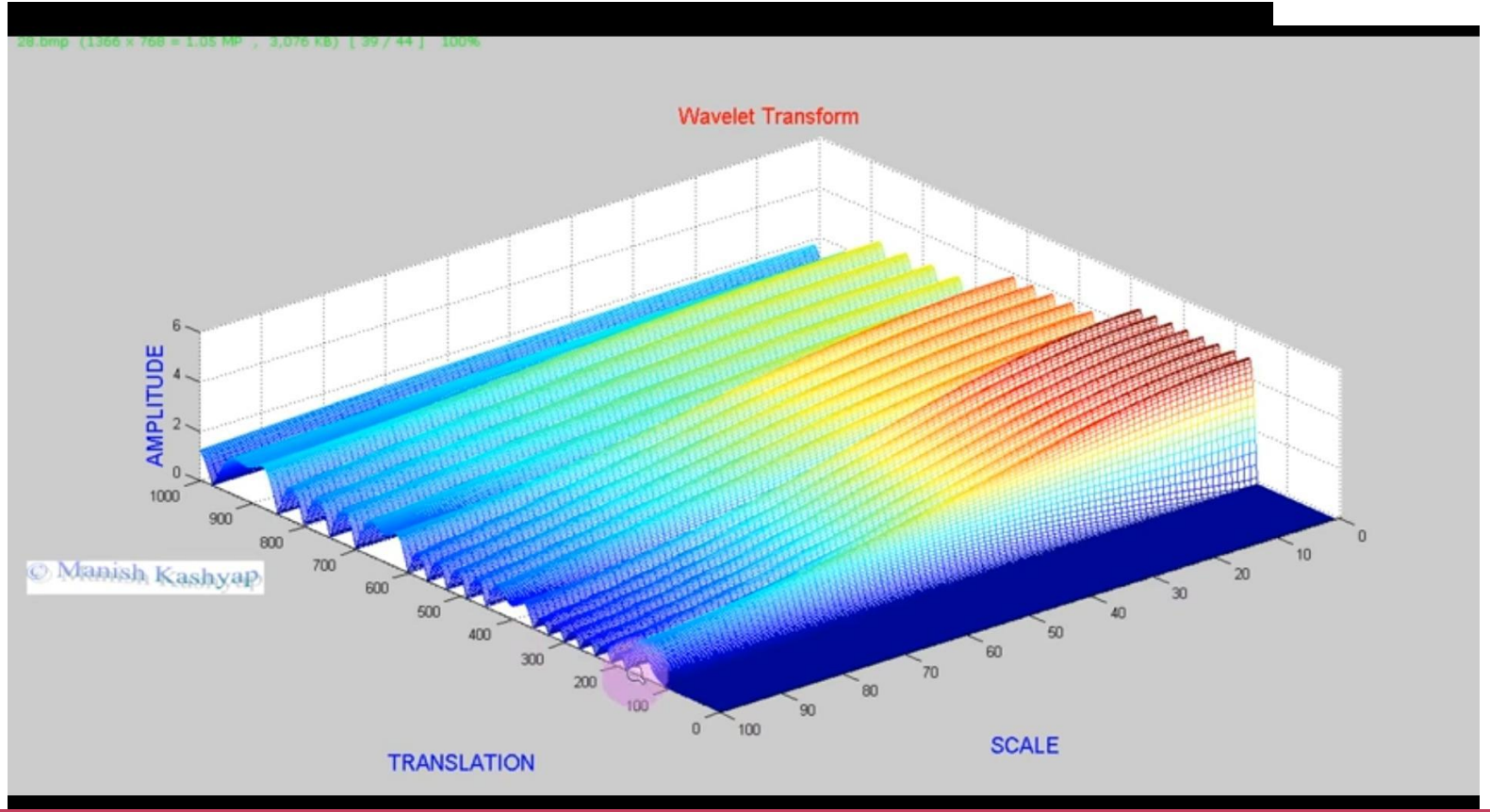
The problem in resolution of STFT gets solved with fixed window size

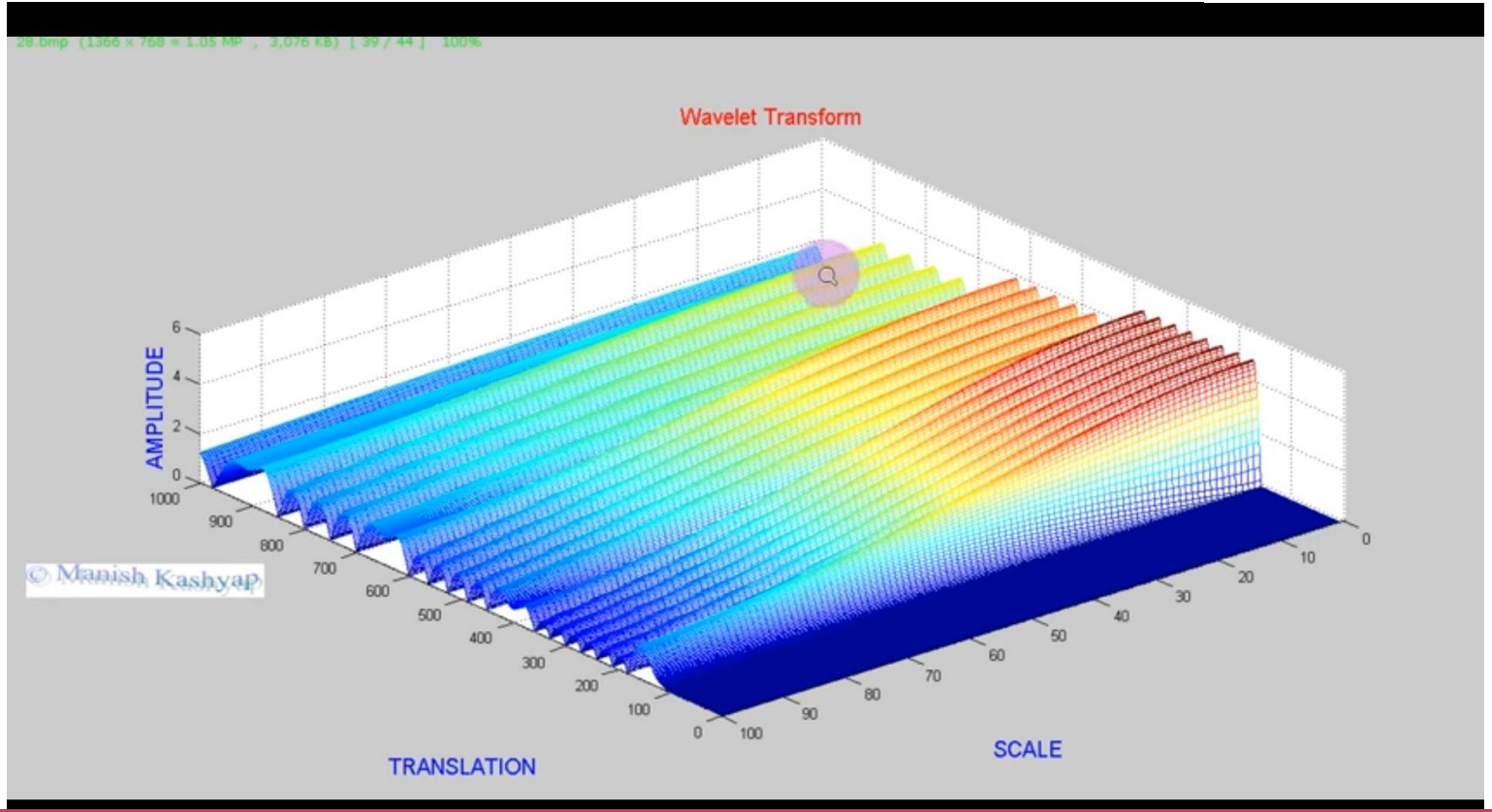

Direction of Increasing SCALE or Decreasing Resolution
© Manish Kashyap
High Frequency Regions are present at only specific locations in space (or time)
Thats why we need to have good spatial localisation for these at low scales by sacrifizing frequency
1
0.5
0
-0.5

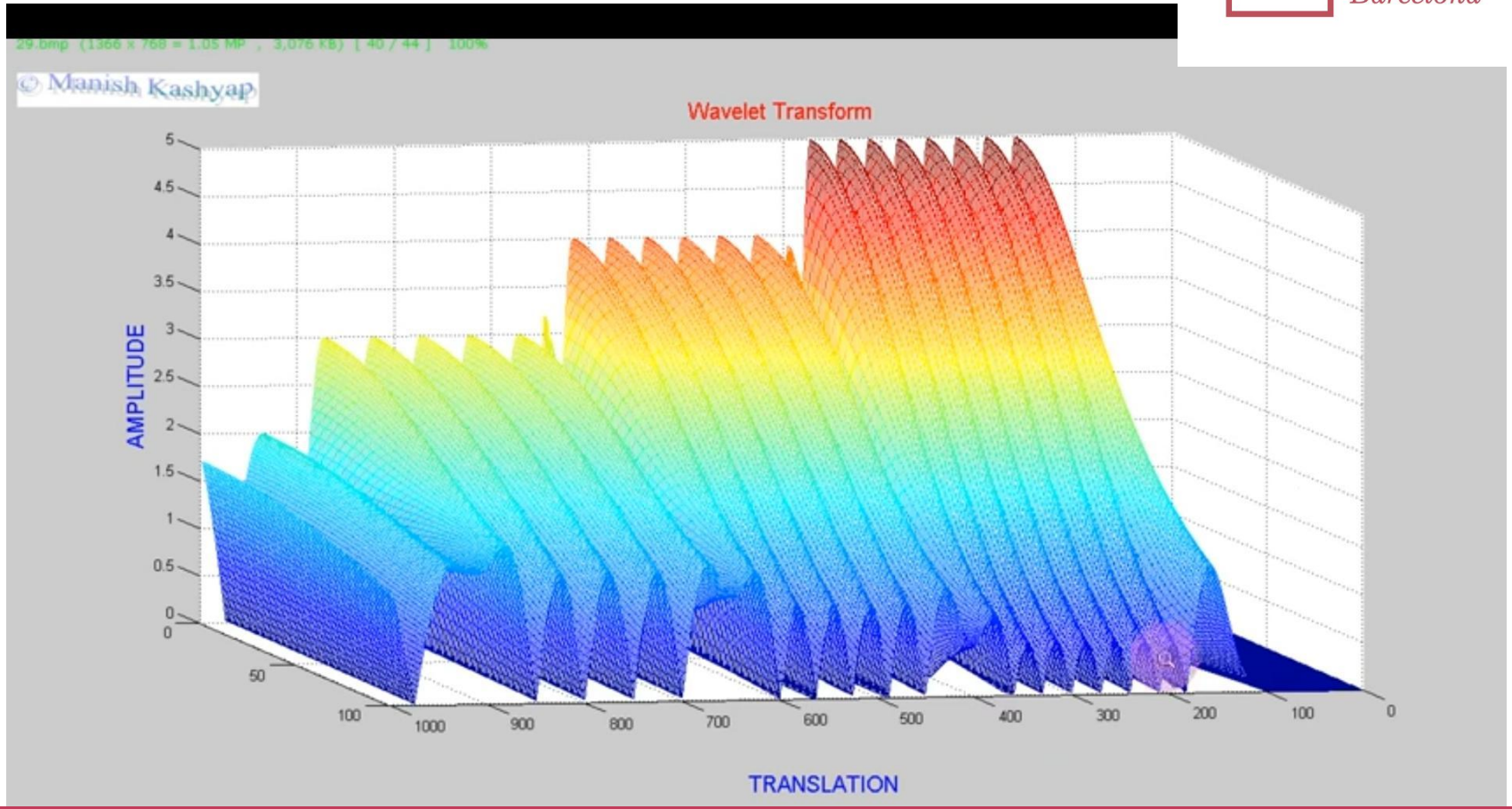
Wavelet transform

The Concept of SCALE




Wavelet Transform
AMPLITUDE
6
4
2
0
1000
900
800
700
600
500
400
300
200
100
0
TRANSLATION
0
10
20
30
40
50
60
70
80
90
100
SCALE
© Manish Kashyap

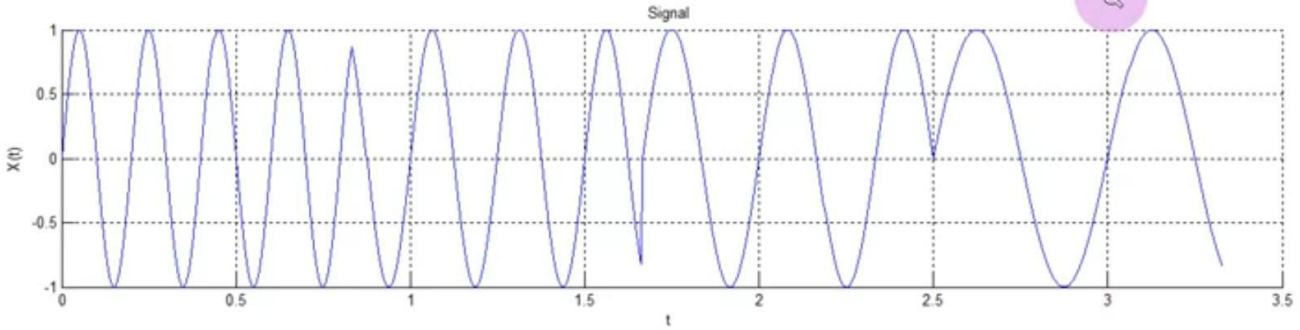

28.bmp (1366 x 768 = 1.05 MP , 3,076 KB) [39 / 44] 100%
Wavelet Transform
AMPLITUDE
6
4
2
0
1000
900
800
700
600
500
400
300
200
100
0
TRANSLATION
0
10
20
30
40
50
60
70
80
90
100
SCALE
© Manish Kashyap


© Manish Kashyap
Wavelet Transform
AMPLITUDE
TRANSLATION
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
50
100
1000
900
800
700
600
500
400
300
200
100
0

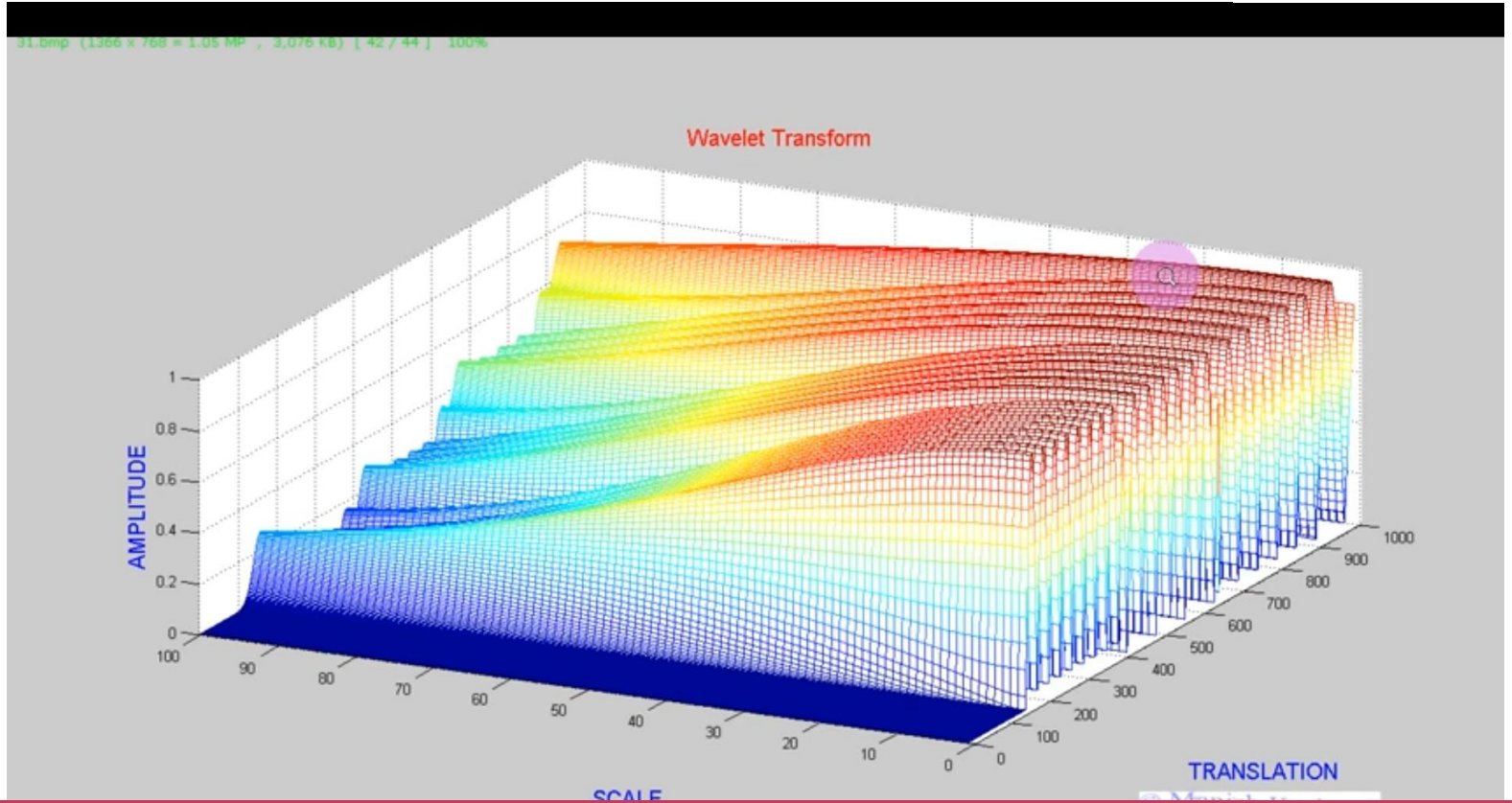
[redacted]

30.bmp (1366 x 768 = 1.05 MP , 3,076 KB) [41 / 44] 108%

5 4 3 2

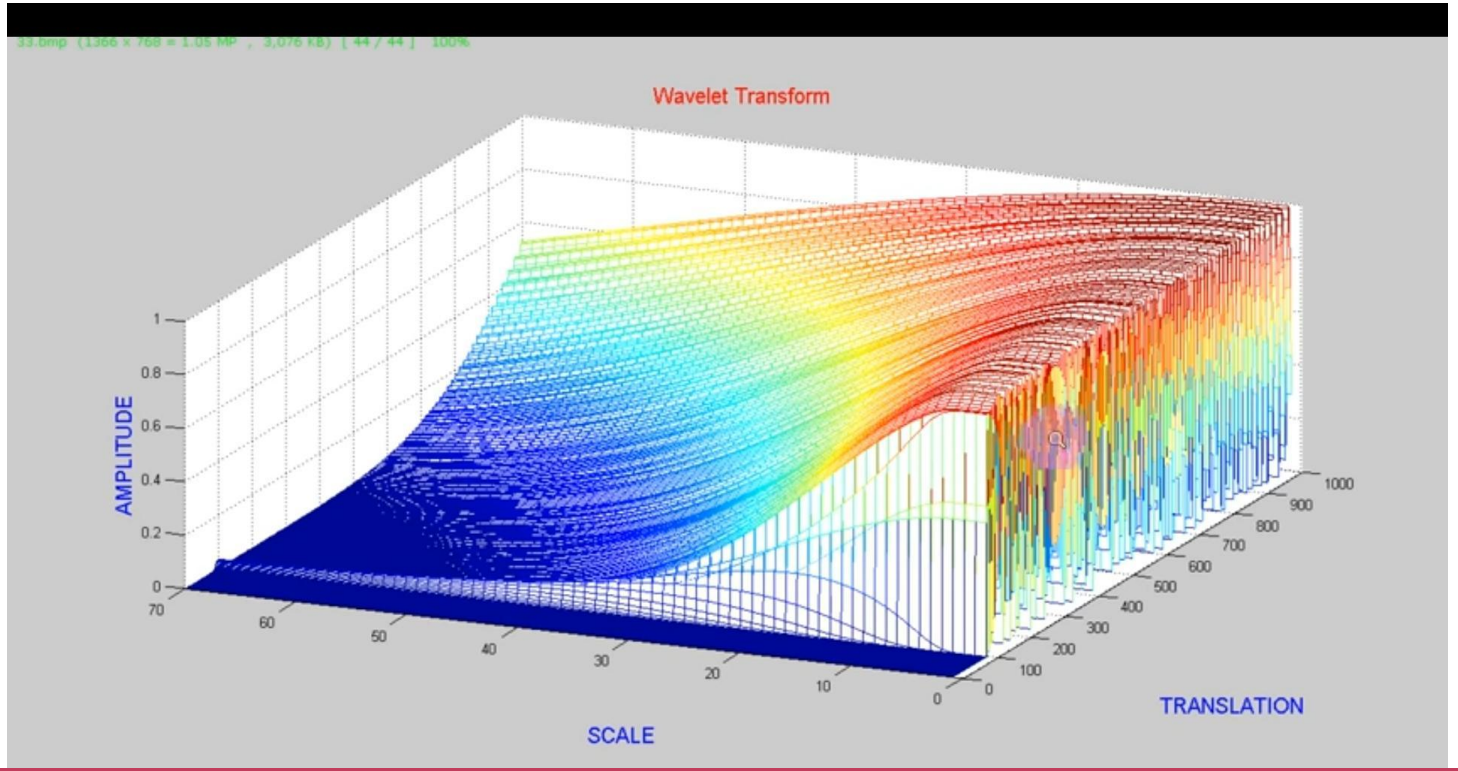


[redacted]


Wavelet Transform
AMPLITUDE
1
0.8
0.6
0.4
0.2
0
100
90
80
70
60
50
40
30
20
10
0
SCALE
0
100
200
300
400
500
600
700
800
900
1000
TRANSLATION

upf. Universitat Pompeu Fabra Barcelona

Example with chirp signal



DWT: Discrete Wavelet Transform

Discrete Wavelet Transform (DWT)

$$T_{m,n} = \int_{-\infty}^{\infty} x(t)\, \psi_{m,n}(t)\, dt$$

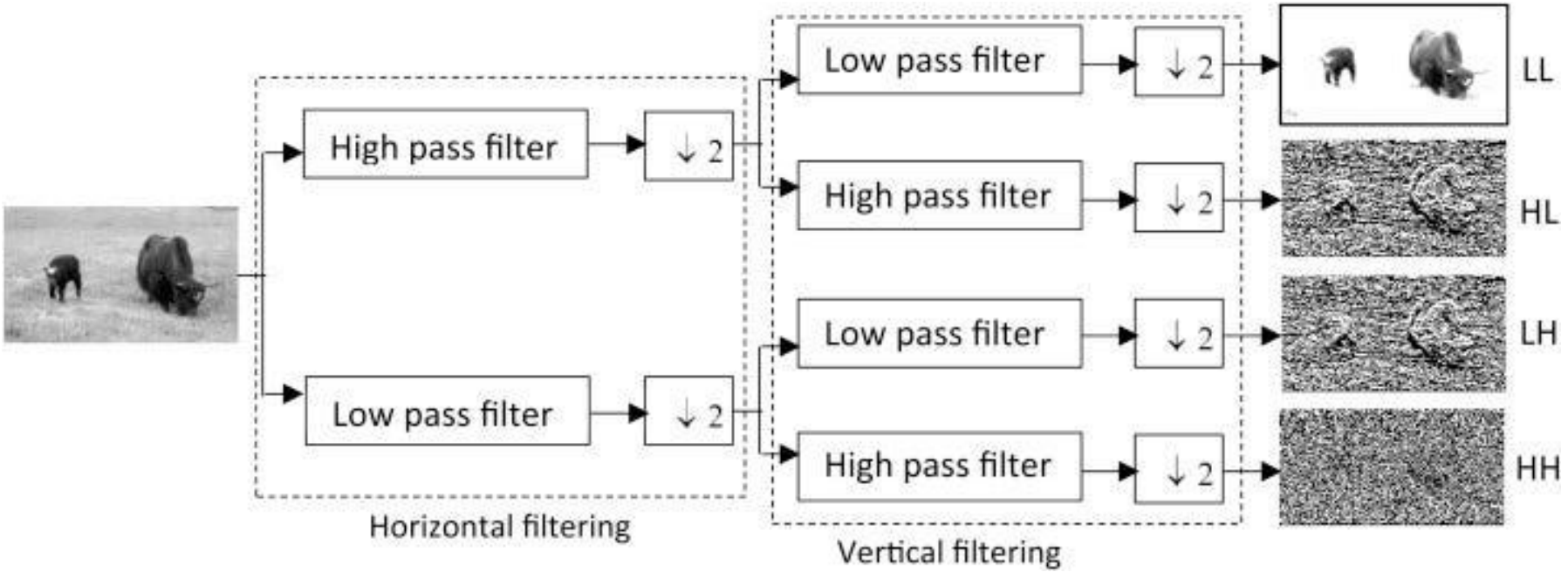
upf. Universitat Pompeu Fabra Barcelona

Imagine you have a picture, and you want to see both the overall scene and the small details separately.

To do this, you use two types of filters:

- Low-pass filter: This filter captures the big, smooth parts of the picture, like the sky or large areas of color.

- High-pass filter: This filter captures the fine details, like the edges of objects or small textures.


High pass filter
↓ 2
Low pass filter
↓ 2
Horizontal filtering
Low pass filter
↓ 2
High pass filter
↓ 2
Low pass filter
↓ 2
High pass filter
↓ 2
Vertical filtering
LL
HL
LH
HH

Downsampling

After using the filters, we have too much data to handle easily. So, we keep only every second sample (imagine throwing away half of the points), which is called downsampling. This makes the data smaller and easier to work with.

Iteration

We can repeat the filtering and downsampling on the big, smooth parts to get even more levels of detail. Each level gives us a different view of the picture, from very rough to very detailed.

Multi-Resolution Analysis

This is a fancy way of saying that DWT lets us see the picture at different levels of detail, like looking at it with different levels of zoom. This is super helpful for things like compressing images, where we want to keep the important parts but get rid of unnecessary details to save space.

Visualizing DWT in Images

When we use DWT on images, here's what happens:

2D Decomposition

We split the image into four parts (sub-bands):
LL (Low-Low): The big, smooth parts of the image.
LH (Low-High): The horizontal details.
HL (High-Low): The vertical details.
HH (High-High): The tiny, diagonal details.

We can keep splitting the LL part to get more levels of detail. It's like peeling an onion, layer by layer.

Comparision between 2 standards

Figure 3.12: Recovered images after JPEG compression with ratios $k = 1, 5, 10, 20$.

Comparision between 2 standards

Figure 5.2: Lenna's image compressed using JPEG 2000 (left) and JPEG (right) at rates 0.0625 (up), 0.125 (middle), and 0.25 (down).

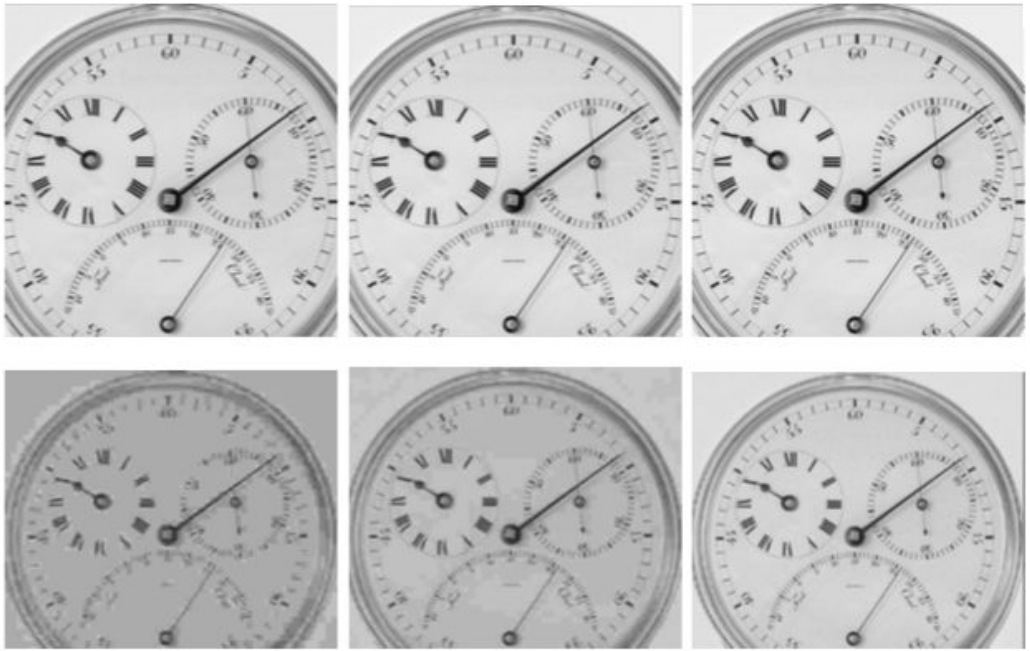


Figure 5.4: Chronometer image compressed using JPEG 2000 (up) and JPEG (down) at rates 0.0625 (left), 0.125 (middle), and 0.5 (right).

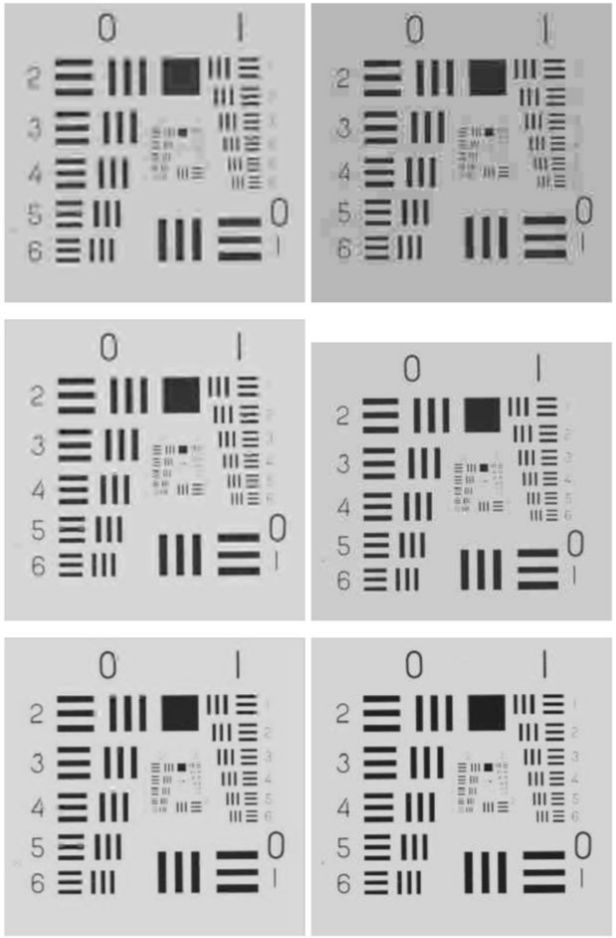


Figure 5.7: Pattern image compressed using JPEG 2000 (left) and JPEG (right) at rates 0.125 (up), 0.25 (middle), and 0.5 (down).

Why JPEG2000 failed?

- Patents
- Microsoft being Microsoft
- Browser support
- Hardware support
- Performance trade-offs (at its time)

Feature	JPEG	JPEG2000
Transformation	DCT	DWT
Artifacts	Blocks and edges	Smooth, no blocking artifacts
Lossless Compression	No	Yes
Scalability	No	Yes, multiple resolutions possible
High-Quality Performance	Moderate	Superior

upf. Universitat Pompeu Fabra Barcelona

‘ In our industry’s history, the best encoding solution is not always the standardized or most popular one.

In a capitalist, globalized society, the dominant codec often serves economic interests rather than being the most technically superior.’

Javi

Other standards which seemed to success further

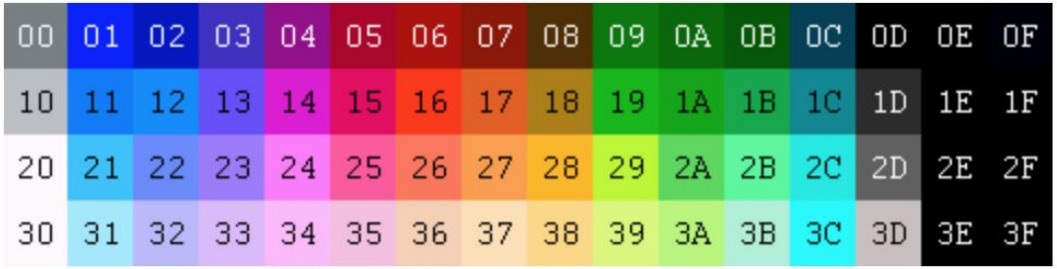
Graphical Image Interface

- 8 bits per pixel for each image, up to 256 different colors chosen from the 24-bit RGB color space

(Remember this slide?)

Other ways to encode a color image

Many other possible models may be used to represent the colors that make up an image. We could, for instance, use an indexed palette where we'd only need a single byte to represent each pixel instead of the 3 needed when using the RGB model. In such a model we could use a 2D matrix instead of a 3D matrix to represent our color, this would save on memory but yield fewer color options.



Graphical Image Interface

- Supports animations

- Supports separate palette of up to 256 colors for each frame.

Graphical Image Interface

- Less suitable for color photographs, enough for graphics or logos

- GIF images are compressed using the Lempel–Ziv–Welch (LZW) lossless data compression technique to reduce the file size without degrading the visual quality.

Graphical Image Interface

- Very common on the internet (specially 4chan and Forocoches) and nowadays in a lot of social media and messaging applications

Portable Network Graphics (.png)

- Non-patented, designed to substitute .gif
- Supports 24RGB or 32 bit RGBA
- Lossless codec, based on DEFLATE encoding

(DEFLATE encoding: a mix of LZSS and Huffman)

Universitat Pompeu Fabra Barcelona

/home/tilman/PNG-Gradient.png - Bless

File Edit View Search Tools Help

PNG-Gradient.png

```
00000000 89 50 4E 47 0D 0A 1A 0A 00 00 00 0D 49 48 44 52 00 .PNG........IHDR.
00000011 00 00 80 00 00 00 44 08 02 00 00 00 C6 25 AA 3E 00 ......D......%.>.
00000022 00 00 C2 49 44 41 54 78 5E ED D4 81 06 C3 30 14 40 ...IDATx^.....0.@
00000033 D1 B7 34 DD FF FF 6F B3 74 56 EA 89 12 6C 28 73 E2 ..4...o.tV...l(s.
00000044 AA 34 49 03 87 D6 FE D8 7B 89 BB 52 8D 3B 87 FE 01 .4I.....{..R.;...
00000055 00 80 00 00 10 00 00 02 00 40 00 00 08 00 00 01 00 .........@.......
00000066 20 00 00 04 00 80 00 00 10 00 00 02 00 40 00 00 08  ............@...
00000077 00 00 01 00 20 00 00 00 D4 5E 6A 64 4B 94 F5 98 7C .... ....^jdK...|
00000088 D1 F4 92 5C 5C 3E CF 9C 3F 73 71 58 5F AF 8B 79 5B ...\\>..?sqX_..y[
00000099 EE 96 B6 47 EB F1 EA D1 CE B6 E3 75 3B E6 B9 95 8D ...G.......u;....
000000aa C7 CE 03 39 C9 AF C6 33 93 7B 66 37 CF AB BF F9 C9 ...9...3.{f7.....
000000bb 2F 08 80 00 00 10 00 00 02 00 40 00 00 08 00 00 01 /.........@......
000000cc 00 20 00 00 04 00 80 00 00 10 00 00 02 00 40 00 00 . ............@..
000000dd 08 00 00 01 00 20 00 00 8C 37 DB 68 03 20 FB ED 96 ..... ...7.h. ...
000000ee 65 00 00 00 00 49 45 4E 44 AE 42 60 82           e....IEND.B`.
```

Signed 8 bit:	13	Signed 32 bit:	218765834	Hexadecimal:	0D 0A 1A 0A
Unsigned 8 bit:	13	Unsigned 32 bit:	218765834	Decimal:	013 010 026 010
Signed 16 bit:	3338	Float 32 bit:	4,255588E-31	Octal:	015 012 032 012
Unsigned 16 bit:	3338	Float 64 bit:	7,46625117388175E-246	Binary:	00001101 00001010 00

Show little endian decoding Show unsigned as hexadecimal ASCII Text:

Offset: 0x4 / 0xfa Selection: 0x1 to 0x3 (0x3 bytes) INS

Bonus track: if you want to know exactly how an image is get from the real world:

https://www.cambridgeincolour.com/tutorials/camera-sensors.htm

Thanks

franciscojavier.brines@upf.edu