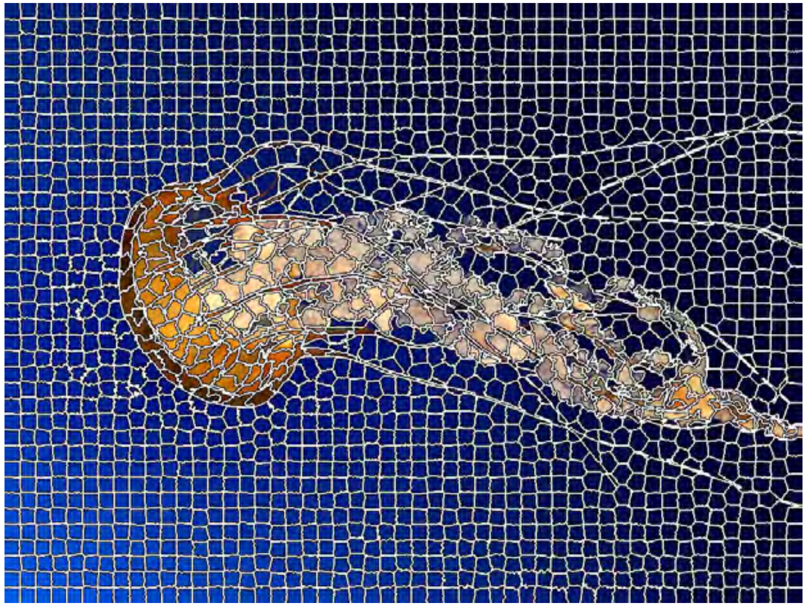
TRABALHO 1: IMAGEM

Implemente o algoritmo de superpixel SLICO descrito em:

https://infoscience.epfl.ch/record/177415/files/Superpixel_PAMI2011-2.pdf
https://infoscience.epfl.ch/record/177415

Seguindo as adaptações indicadas nos próximos slides.

Superpixels

O algoritmo SLIC (*Simple Linear Iterative Clustering*)

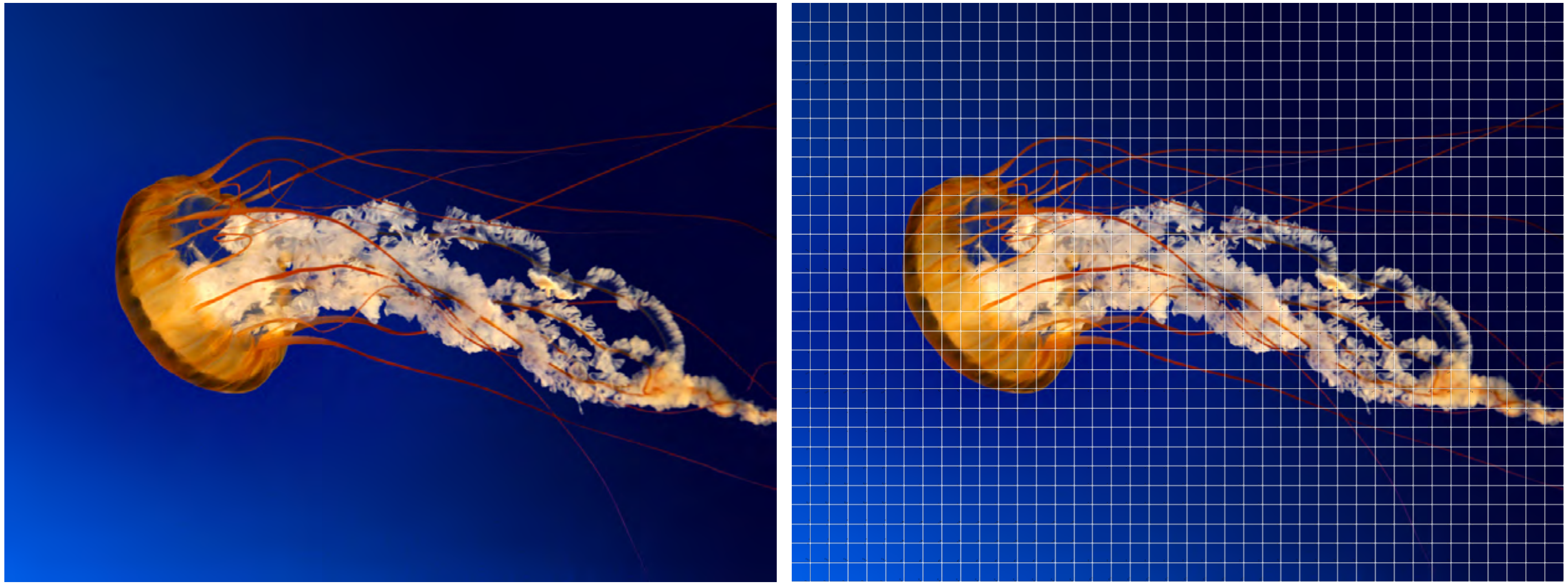
Entrada: n_s *(número aprox. de superpixels) e, opcionalmente,* m *(compacidade)*

1. Calcule as coordenadas CIE Lab de todos os pixels da imagem;

2. Calcule o tamanho, s, da cédula quadrada

$$s = \sqrt{\frac{w * h}{n_s}}$$

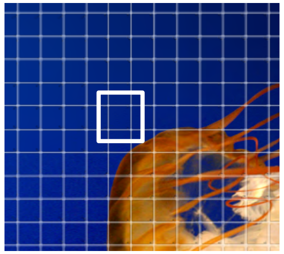
3. Initialize os representantes das cédulas $\mathbf{c}_i = [l_i; a_i; b_i; x_i; y_i]^T$ amostrando em uma grade de tamanho s, $i=0,\dots,(n_s-1)$



evite as bordas e pontos ruidosos: escolha na vizinhança 3x3 do centro da célula o pixel que tenha o menor gradiente para $\mathbf{c}_i$

O algoritmo SLIC (*cont.*)

4. Para cada superpixel $\boldsymbol{c}_k$ crie uma janela de tamanho $2s$ centrada em (x_k, y_k)



   a) *Para cada pixel neste janela que estiver atribuído a outro superpixel* $\boldsymbol{c}_j$, *verifique se a distância dele ao* $\boldsymbol{c}_k$ *é menor e, se for, atribua este pixel ao* $\boldsymbol{c}_k$ *.*

5. *Quando todos os superpixels tiverem sido visitados, recalcule o sua cor e centro através da média de seus pixels. Calcule também o deslocamento de seu centro e acumule numa medida de erro E.*

6. *Se o erro acumulado (de todos os superpixels) for pequeno ou se o número de iterações for excessivo, maior que* 10, *por exemplo, pare. Caso contrário volte para o passo* 4.

Cálculo de distância

Distância entre o pixel $\boldsymbol{p}_j = [l_j; a_j; b_j; x_j; y_j]^T$ e o superpixel $\boldsymbol{c}_i = [l_i; a_i; b_i; x_i; y_i]^T$

$$d_c = \sqrt{(l_i - l_j)^2 + (a_i - a_j)^2 + (b_i - b_j)^2}$$

$$d_s = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

$$d_t = \sqrt{\left(\frac{d_c}{m_c}\right)^2 + \left(\frac{d_s}{m_s}\right)^2}$$

onde m_c e m_s são as máximas distâncias esparadas no superpixel.
Opções:

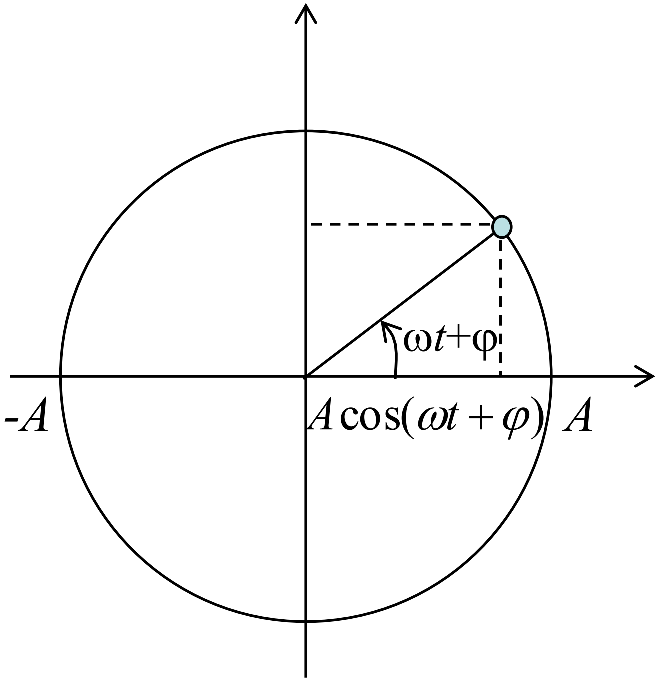
1. constantes: $m_s = s$ *e* m_c *é parâmetro de entrada.*
2. adaptativos: m_s *e* m_c *são calculados a partir da última iteração.*

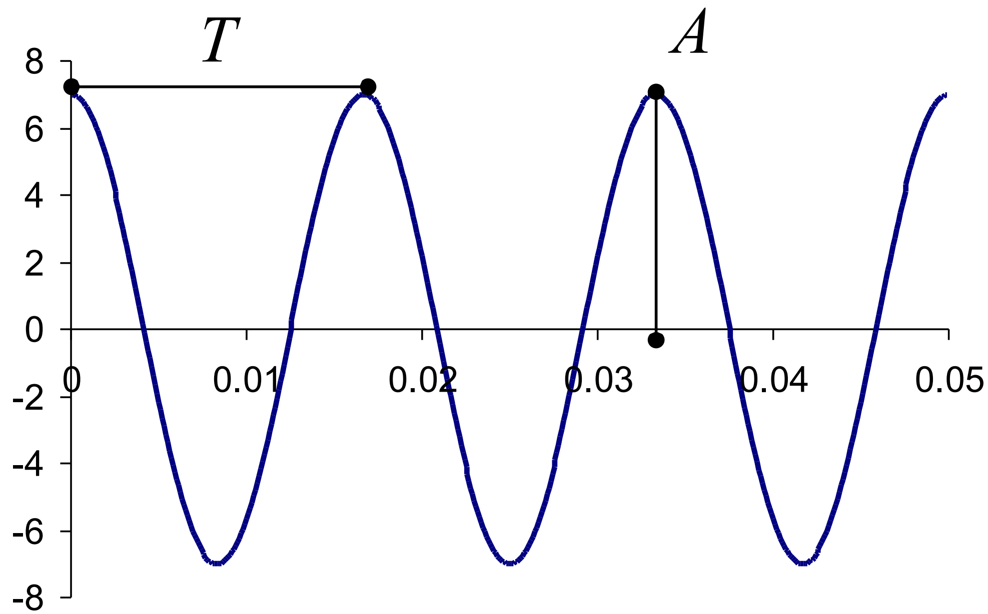
UM POUCO DE TEORIA DE SINAIS:
IMAGENS NO DOMÍNIO DA FREQUÊNCIA

- Transformada de Fourier
- Transformada de Haar

revisão

Harmônicos





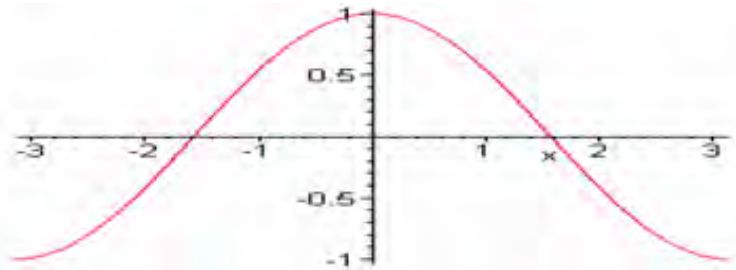
$$f = \frac{1}{T}\,(Hz)$$

$$\omega t = 2\pi f t = \frac{2\pi}{T}t \quad (rad)$$

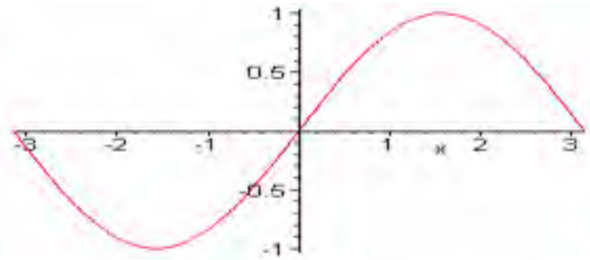
revisão

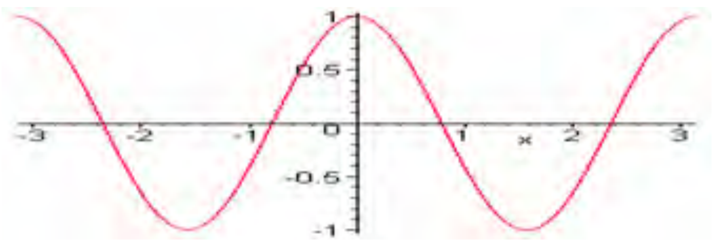
Integrais de senos e cosenos em [-π,π]

$\cos(nx)$ $\sin(nx)$

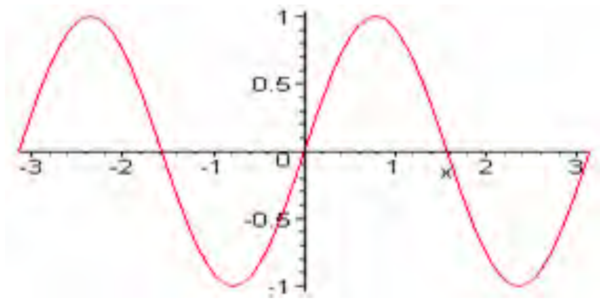


$n = 1$

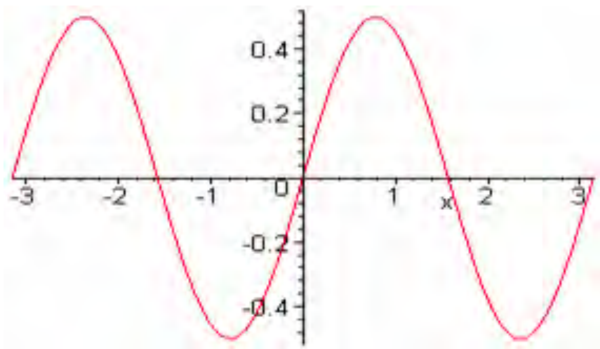




$n = 2$



$\sin(nx)\cos(nx)$



Áreas se compensam.
Integrais resultam em 0.

revisão

Integrais de senos e cosenos em [-π,π]

$$\int_{-\pi}^{\pi} \sin(mx)\sin(nx)\,dx = \pi\delta_{mn} \qquad \text{for } n, m \neq 0$$

$$\int_{-\pi}^{\pi} \cos(mx)\cos(nx)\,dx = \pi\delta_{mn} \qquad \text{for } n, m \neq 0$$

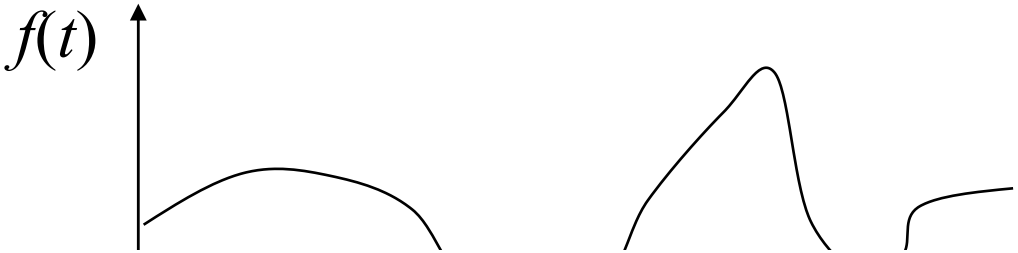
$$\int_{-\pi}^{\pi} \sin(mx)\cos(nx)\,dx = 0$$

$$\int_{-\pi}^{\pi} \sin(mx)\,dx = 0$$

$$\int_{-\pi}^{\pi} \cos(mx)\,dx = 0$$

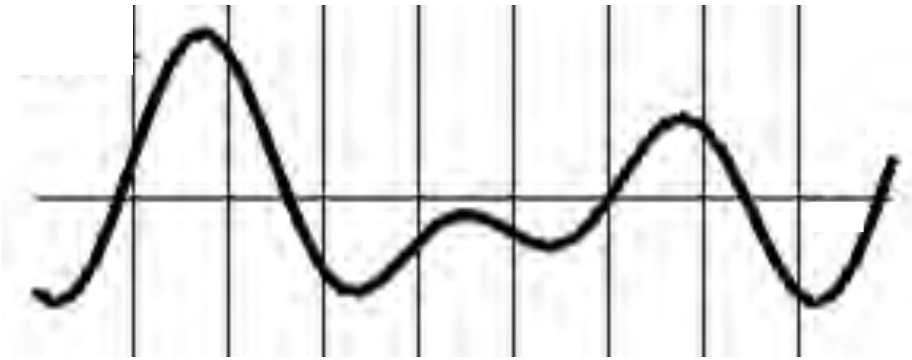
Funções ortogonais

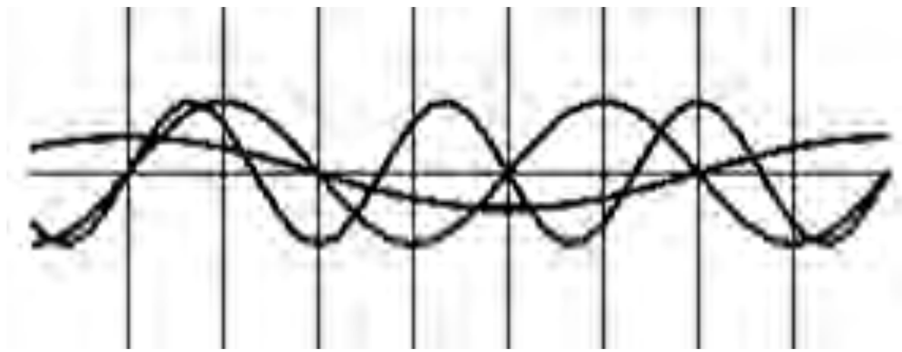
Série de Fourier



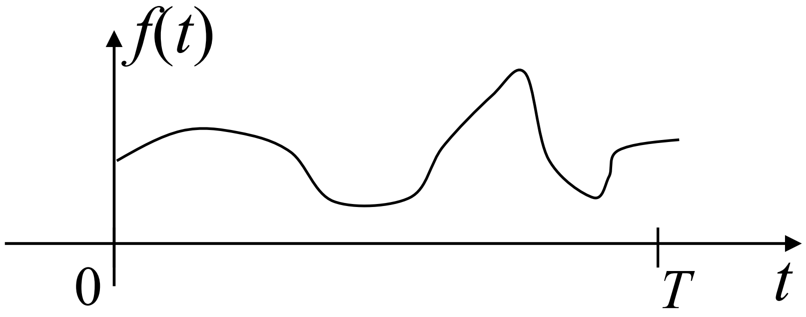
Jean Baptiste Joseph Fourier (1768-1830)
Paper de 1807 para o Institut de France:
Joseph Louis Lagrange (1736-1813), and Pierre Simon de Laplace (1749-1827).

$$f(t) = a_0 + 2\sum_{k=1}^{\infty}\left(a_k \cos\frac{2\pi kt}{T} + b_k \sin\frac{2\pi kt}{T}\right)$$





Série de Fourier: cálculo de a_0



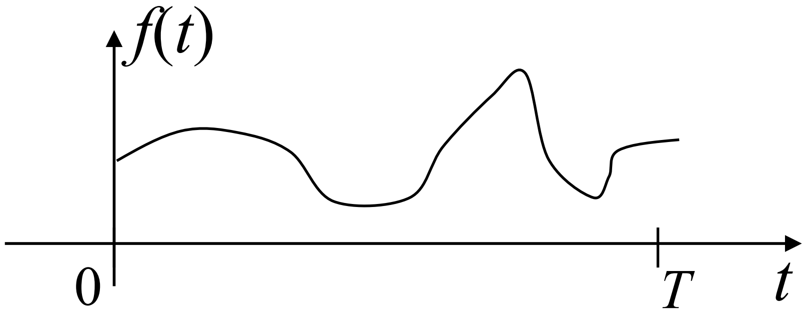
$$f(t) = a_0 + 2\sum_{k=1}^{\infty}(a_k \cos\frac{2\pi kt}{T} + b_k \sin\frac{2\pi kt}{T})$$

$$\int_0^T f(t)dt = \int_0^T a_o dt + \sum_{k=1}^{\infty}\left(a_k \int_0^T \cos(\frac{2\pi nkt}{T})dt + b_k \int_0^T \sin(\frac{2\pi kt}{T})dt \right)$$

$$\int_0^T f(t)dt = a_0 T + 0 + 0$$

$$a_0 = \frac{1}{T}\int_0^T f(t)dt$$

Série de Fourier: a_n e b_n



$$f(t) = a_0 + 2\sum_{k=1}^{\infty}\left(a_k \cos\frac{2\pi kt}{T} + b_k \sin\frac{2\pi kt}{T}\right)$$

$$\int_0^T \cos\left(\frac{2\pi nt}{T}\right) f(t)dt = 0 + 2\sum_{k=1}^{\infty} a_n \int_0^T \cos\left(\frac{2\pi n t}{T}\right)\cos\left(\frac{2\pi k t}{T}\right)dt + 0$$

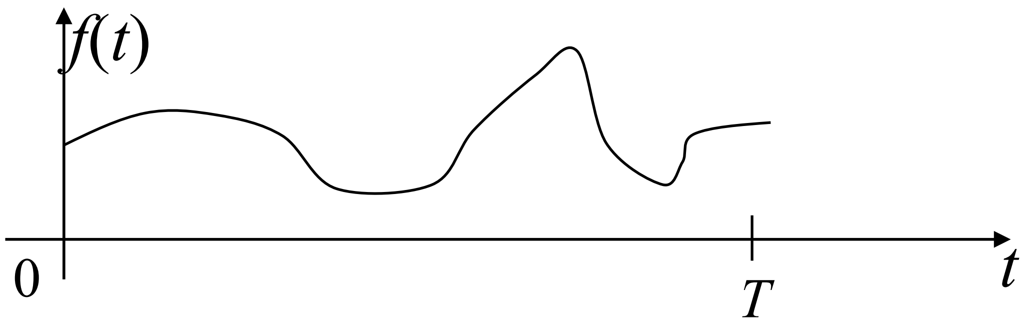
$$= T a_n$$

$$a_n = \frac{1}{T}\int_0^T f(t)\cos\left(\frac{2\pi n t}{T}\right)dt$$

...

$$b_n = \frac{1}{T}\int_0^T f(t)\sin\left(\frac{2\pi n t}{T}\right)dt$$

Resumindo



$$f(t) = a_0 + 2\sum_{k=1}^{\infty}\left(a_k \cos\frac{2\pi kt}{T} + b_k \sin\frac{2\pi kt}{T}\right)$$

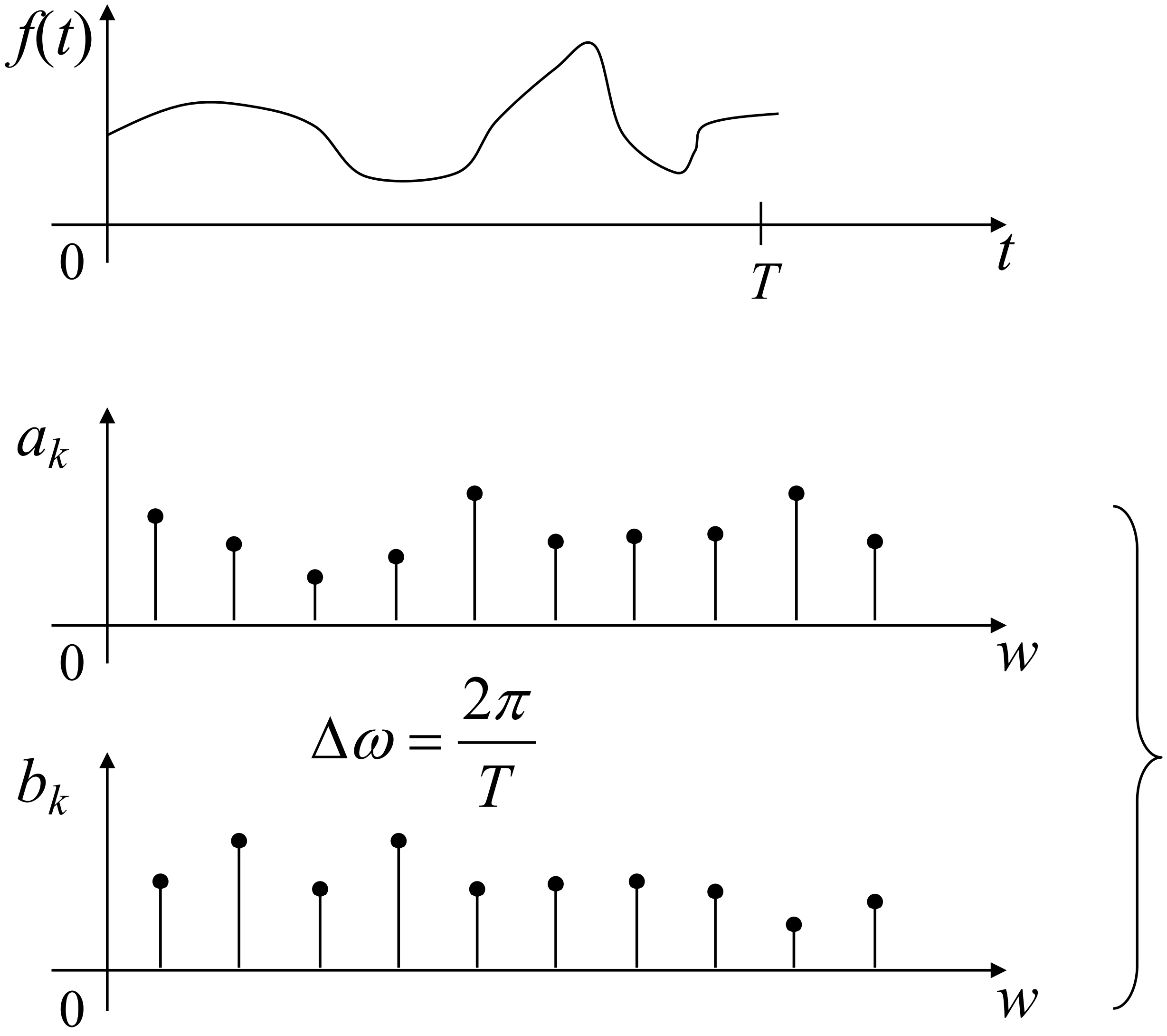
$$a_k = \frac{1}{T}\int_0^T f(t)\cos\left(\frac{2\pi kt}{T}\right)dt \quad k = 0,1,2,3,\ldots$$

$$b_k = \frac{1}{T}\int_0^T f(t)\sin\left(\frac{2\pi kt}{T}\right)dt \quad k = 1,2,3,\ldots$$

$$\omega_k = \frac{2\pi\, k}{T}$$

$$\Delta\omega = \frac{2\pi}{T}$$

Domínios

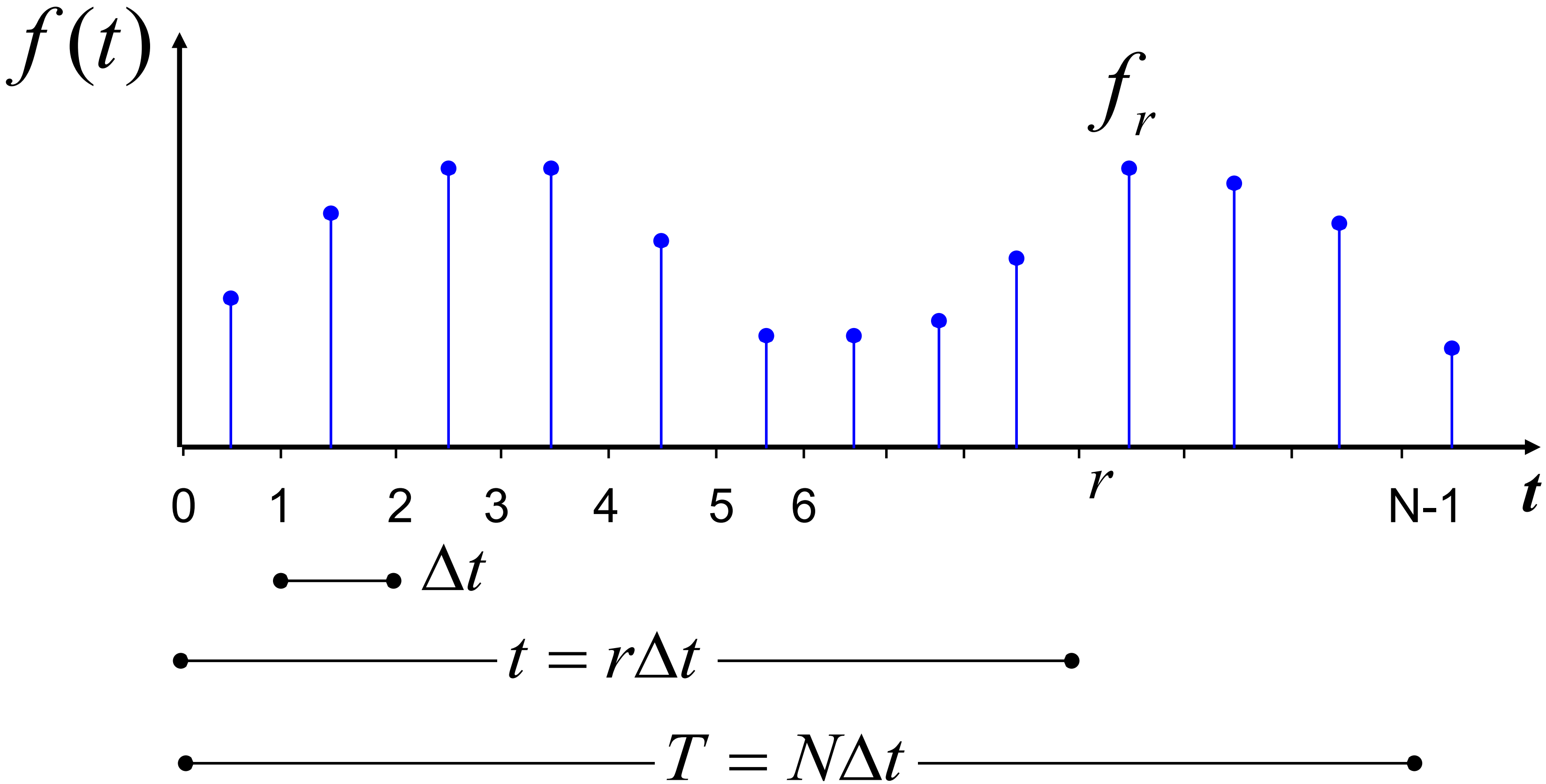


$$\Delta\omega = \frac{2\pi}{T}$$

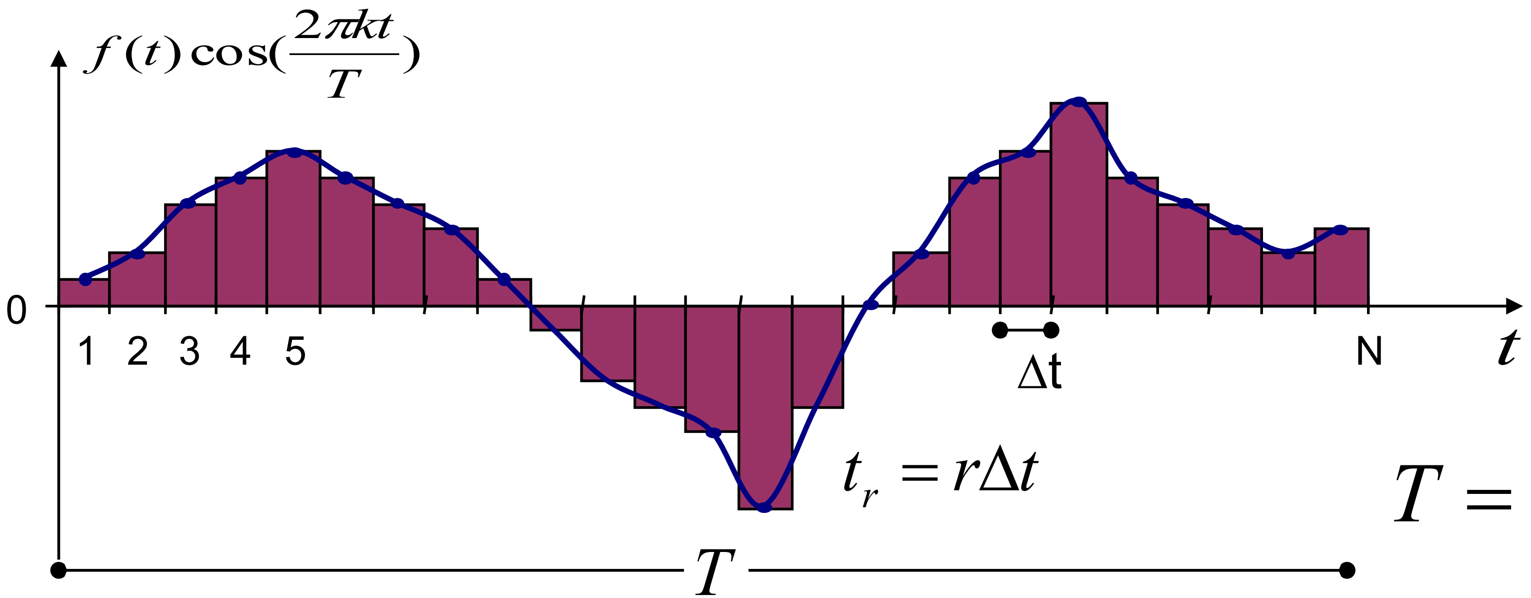
tempo ou
espaço

freqüencia

Sinal discreto



$$\left(f_o, f_1, f_2, \ldots, f_r, \ldots, f_{N-2}, f_{N-1},\right)$$



$$T = N\Delta t$$

$$a_k = \frac{1}{T}\int_0^T f(t)\cos(\frac{2\pi kt}{T})dt \qquad \cong \frac{1}{N\Delta t}\sum_{r=0}^{N-1} f_k \cos\left(\frac{2\pi kr\Delta t}{N\Delta t}\right)\Delta t$$

$$a_k \cong \frac{1}{N}\sum_{r=0}^{N-1} f_r \cos(\frac{2\pi kr}{N}) \quad \ldots \qquad b_k \cong \frac{1}{N}\sum_{r=0}^{N-1} f_k \sin\left(\frac{2\pi kr}{N}\right)$$

$$a_k \cong \frac{1}{N}\sum_{r=0}^{N-1} f_r \cos(\frac{2\pi kr}{N}) \qquad b_k \cong \frac{1}{N}\sum_{r=0}^{N-1} f_k \sin\left(\frac{2\pi kr}{N}\right)$$

$$\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{N-1} \end{bmatrix} = \frac{1}{N}\begin{bmatrix} c_{00} & c_{01} & \cdots & c_{0(N-1)} \\ c_{10} & c_{11} & \cdots & c_{1(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ c_{(N-1)0} & c_{(N-1)1} & \cdots & c_{(N-1)(N-1)} \end{bmatrix}\begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{bmatrix}$$

onde:

$$c_{kr} = \cos(\frac{2\pi kr}{N})$$

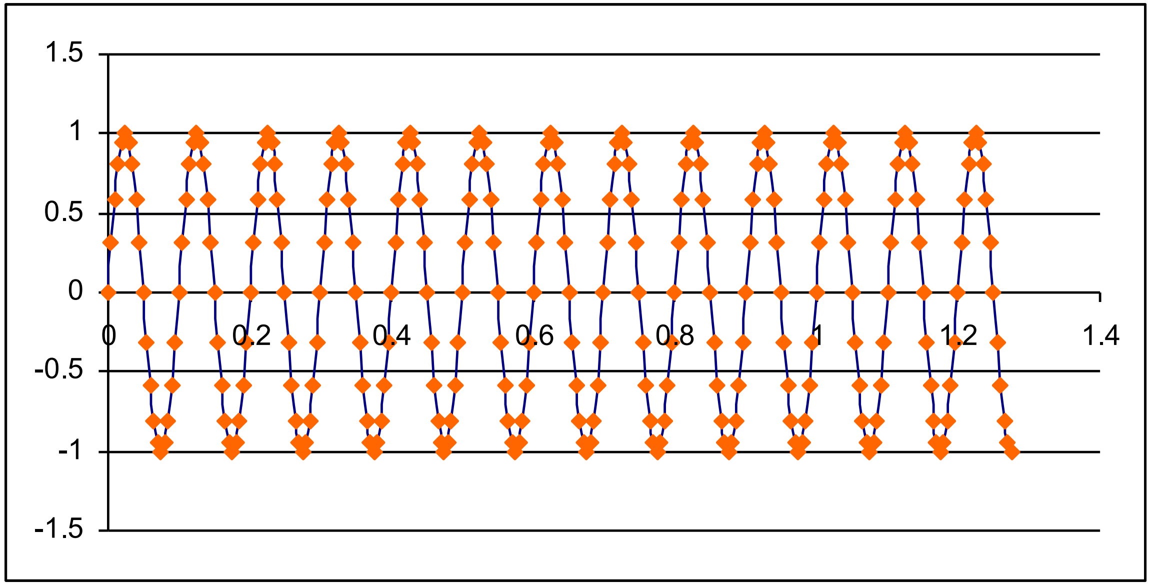
$$\begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{N-1} \end{bmatrix} = \frac{1}{N}\begin{bmatrix} s_{00} & s_{01} & \cdots & s_{0(N-1)} \\ s_{10} & s_{11} & \cdots & s_{1(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ s_{(N-1)0} & s_{(N-1)1} & \cdots & s_{(N-1)(N-1)} \end{bmatrix}\begin{bmatrix} f_0 \\ f_1 \\ \vdots \\ f_{N-1} \end{bmatrix}$$

onde:

$$s_{kr} = \sin(\frac{2\pi kr}{N})$$

Transformada Discreta

$$f(t) = \sin(2\pi 10 t)$$



$$f_a = 200Hz$$

$$N = 256$$

$$\Delta t = \frac{1}{f_a} = 0.005\sec$$

$$T = N \cdot \Delta t = \frac{N}{f_a}$$

$$T = 0.005 \times 256 = 1.28\sec$$

$$f_s = \sin(2\pi 10 \frac{sT}{N})$$

T - não é o período do sinal!

Transformada Discreta de Fourier

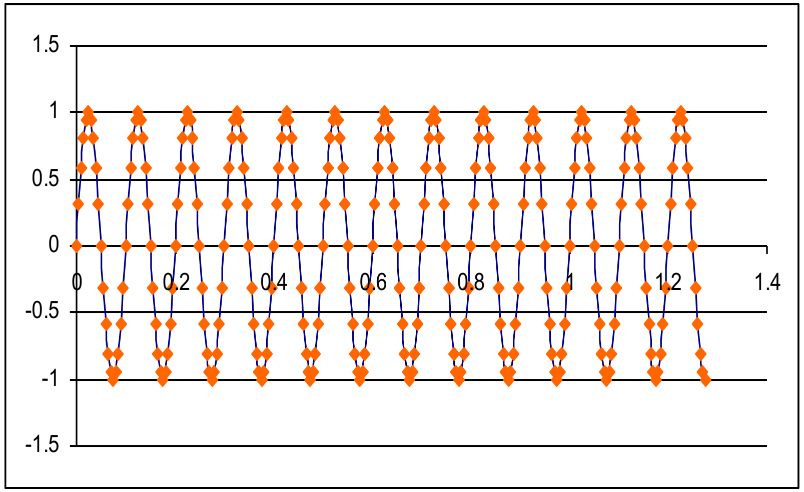
$$f_s = \sin(2\pi 10 \frac{sT}{N})$$

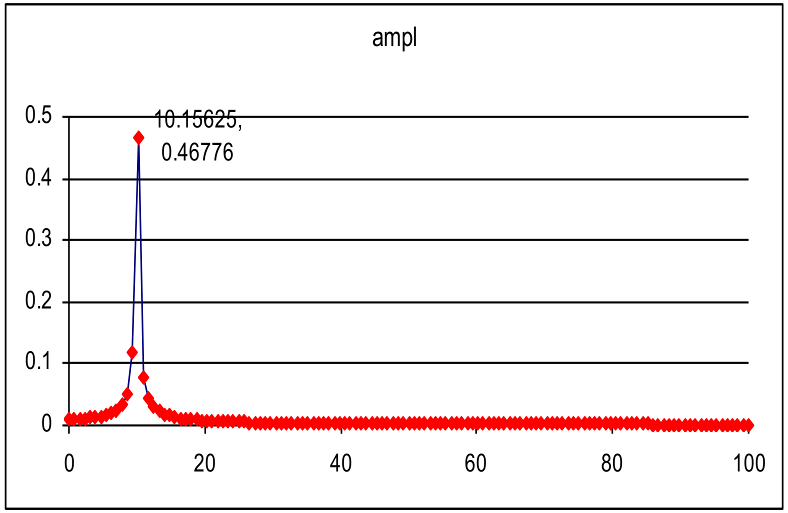
$$F_k \equiv \frac{1}{N}\sum_{s=0}^{N-1} f_s e^{-i(\frac{2\pi k s}{N})}$$

$$k = 1$$

$$\Delta f = \frac{1}{T} = 0.7813\ /\mathrm{sec}$$

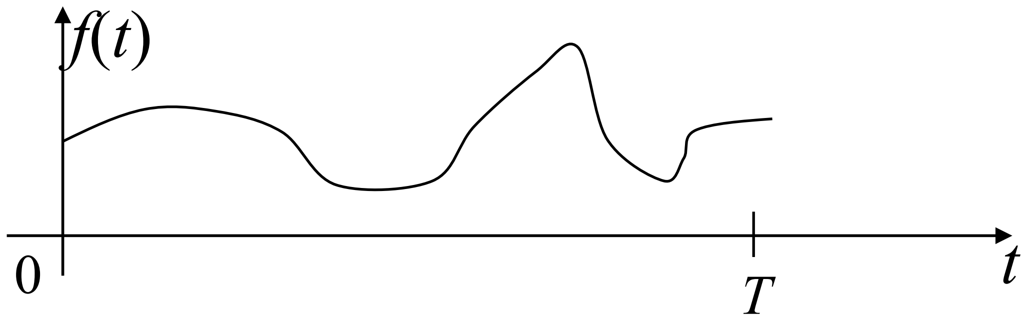
$$\Delta\omega = \frac{2\pi}{T} = 4.91\ rad\ /\mathrm{sec}$$



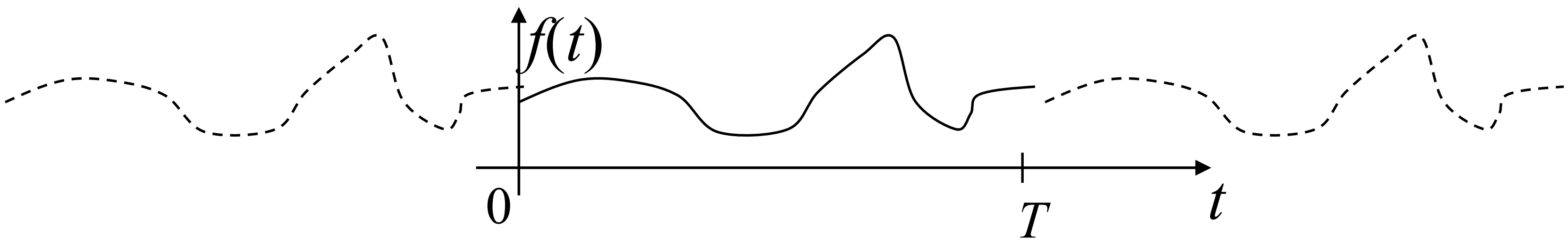


todas as feqüências computadas são multiplas destas

Periodicidade da Série de Fourier

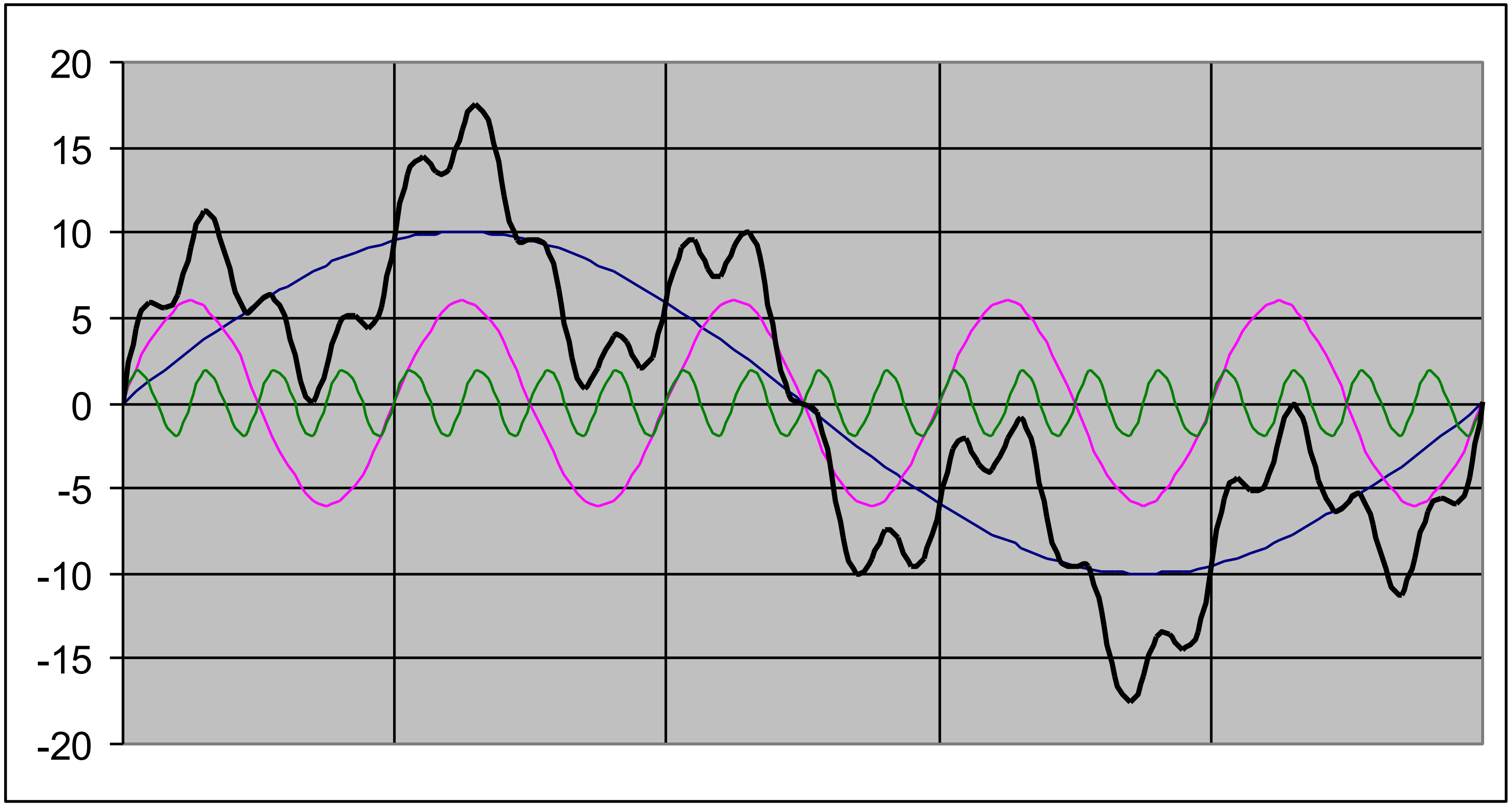


$$f(t+T) = a_0 + 2\sum_{k=1}^{\infty}\left(a_k \cos\left(\frac{2\pi k}{T}(t+T)\right) + b_k \sin\left(\frac{2\pi k}{T}(t+T)\right)\right) = f(t)$$

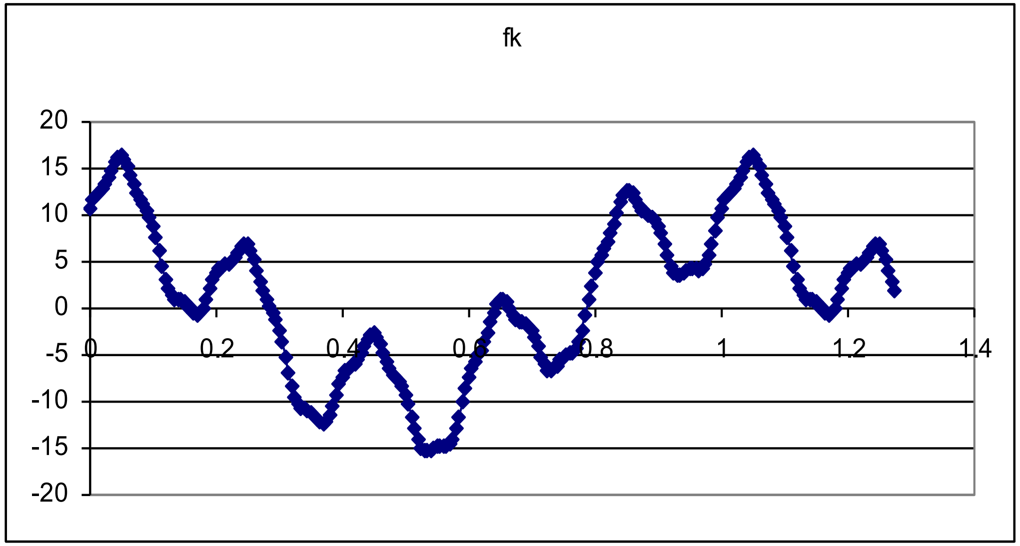


Outro exemplo

$$f3(t) := 10\cos(2\pi t) + 6\sin(10\pi t) + .8\cos(40\pi t)$$

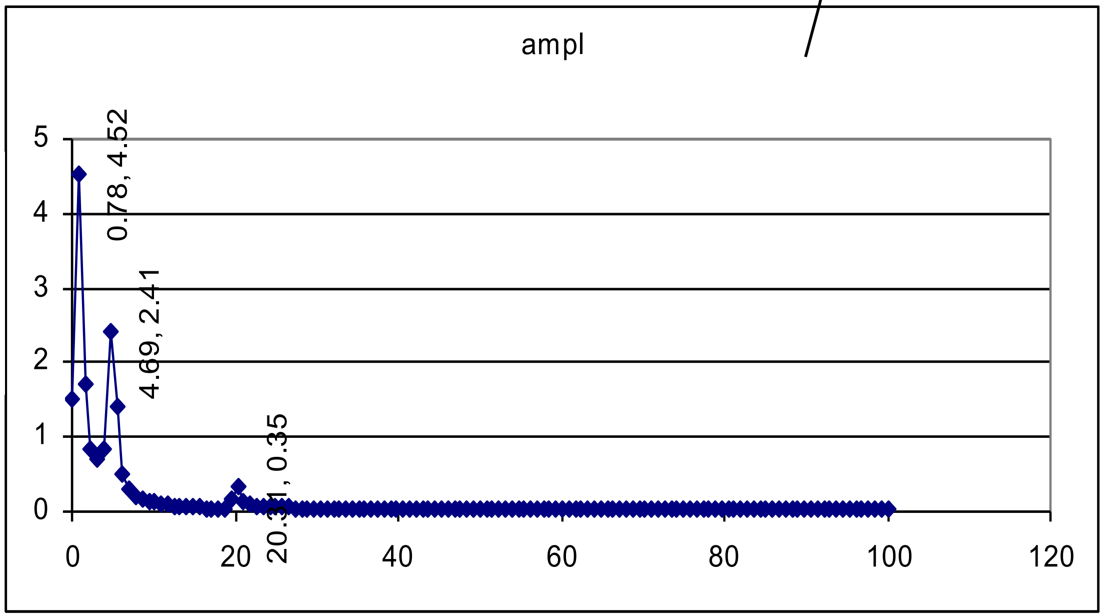


Transformada

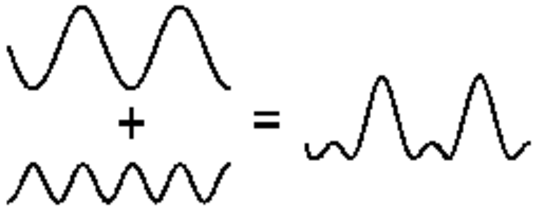


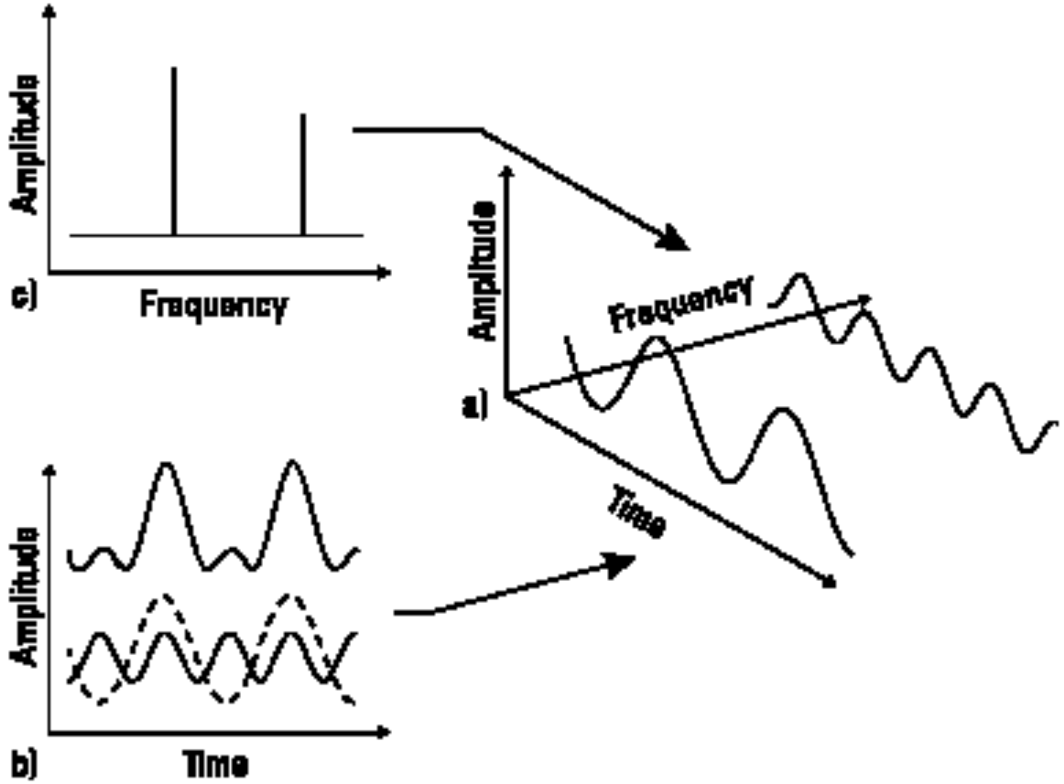
$$f_k \equiv \sum_{r=0}^{N-1} F_r e^{i\left(\frac{2\pi k r}{N}\right)}$$

$$F_k \equiv \frac{1}{N} \sum_{s=0}^{N-1} f_s e^{-i\left(\frac{2\pi k s}{N}\right)}$$

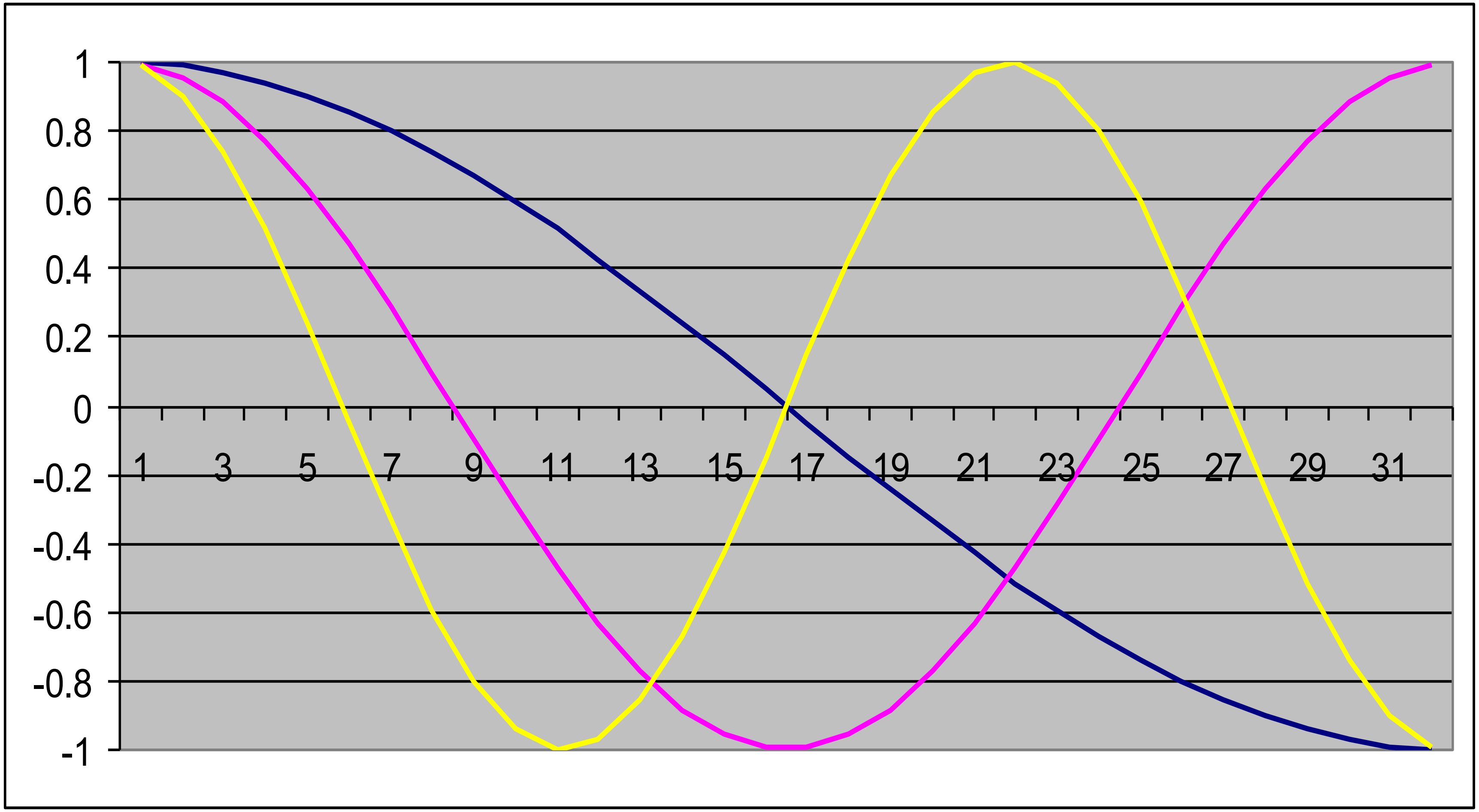


Eixo de freqüência





$$\cos(\alpha - \tfrac{\pi}{2}) = sen(\alpha)$$



o cosseno pode substituir o seno e ficarmos apenas com uma série de cossenos

Discrete Cosine Transformation (DCT)

$$C_k \equiv \frac{\Lambda(k)}{\sqrt{N}} \sum_{s=0}^{N-1} f_s \cos\left(\frac{(2s+1)k}{2N} \pi \right)$$

$$f_s \equiv \sum_{k=0}^{N-1} \frac{\Lambda(k)}{\sqrt{N}} C_r \cos\left(\frac{(2s+1)k}{2N} \pi \right)$$

$$\Lambda(k) = \begin{cases} 1 & k = 0 \\ \sqrt{2} & k \neq 0 \end{cases}$$

Filtro

- Um **filtro** é um operador que atenua ou realça uma determinada freqüência
- Fácil de visualizar no domínio da freqüência onde:

$$F(\omega) \leftarrow f(t)$$

$$H(\omega) = F(\omega)G(\omega)$$

$$h(t) \leftarrow H(\omega)$$

$h(t)$ é o $f(t)$ filtrado

Tipos de Filtros

$$H(\omega) = F(\omega)G(\omega)$$

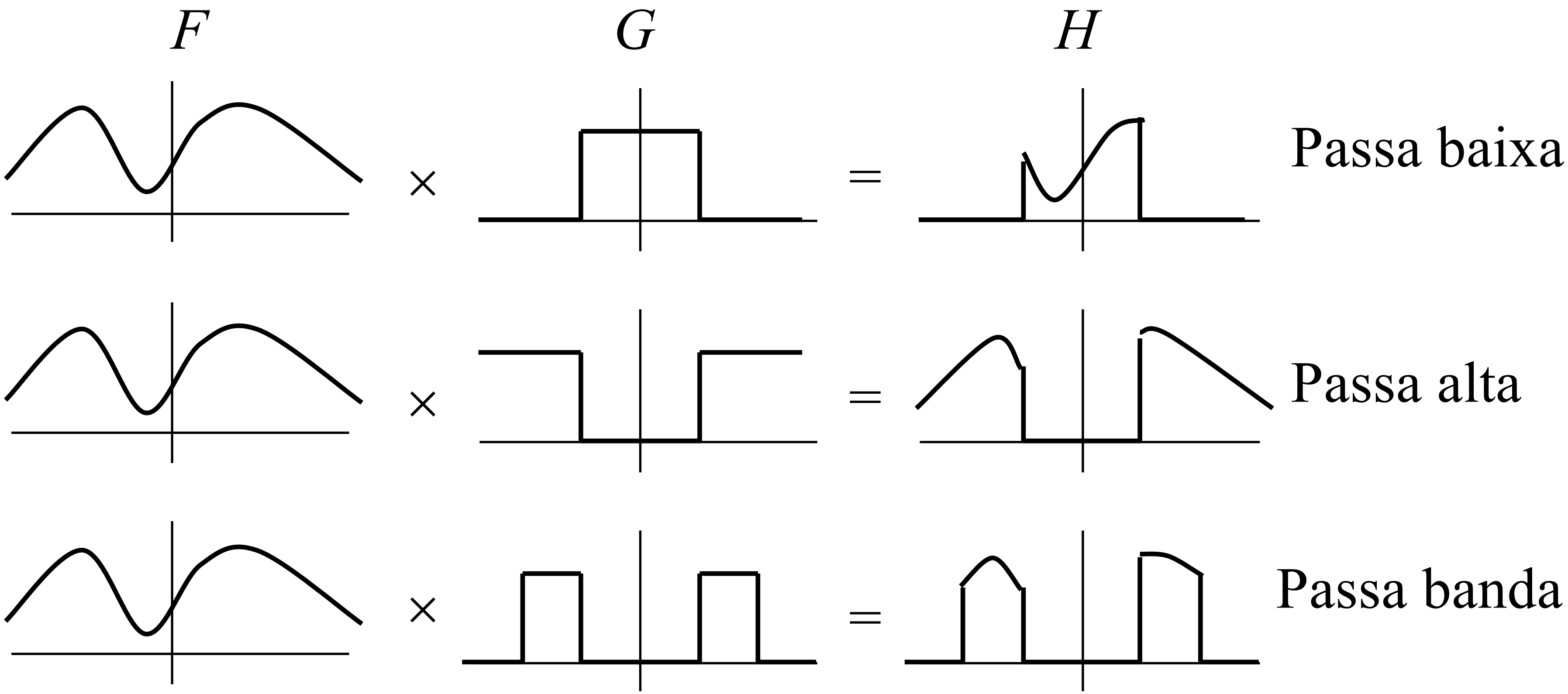


Imagem filtrada com um filtro passa baixa

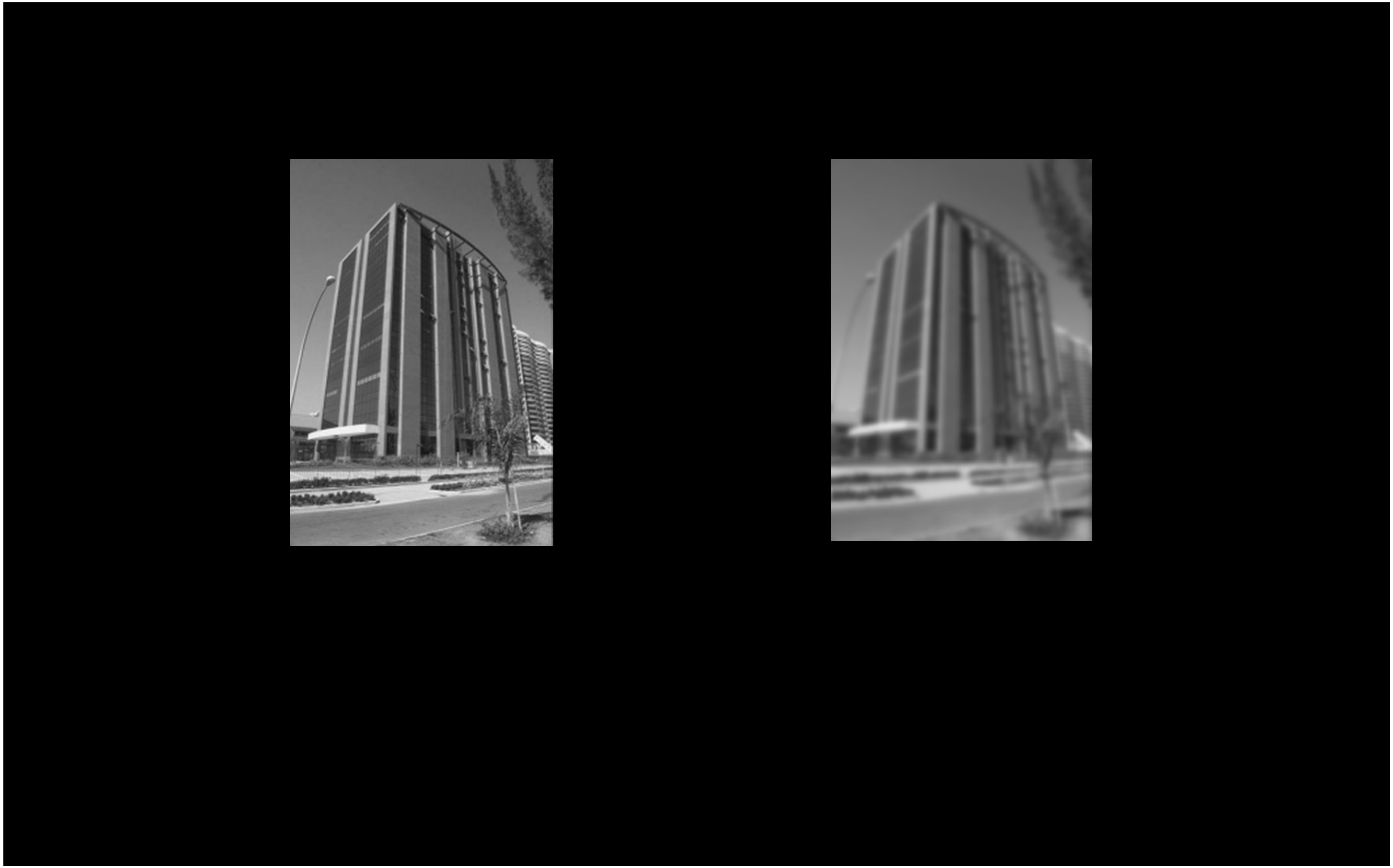


Imagem filtrada com um filtro passa alta

Filtragem no domínio espacial

$$F(\omega) \leftarrow f(x) \qquad G(\omega) \leftarrow g(x)$$

$$H(\omega) = F(\omega)G(\omega)$$

$$h(x) \leftarrow H(\omega)$$

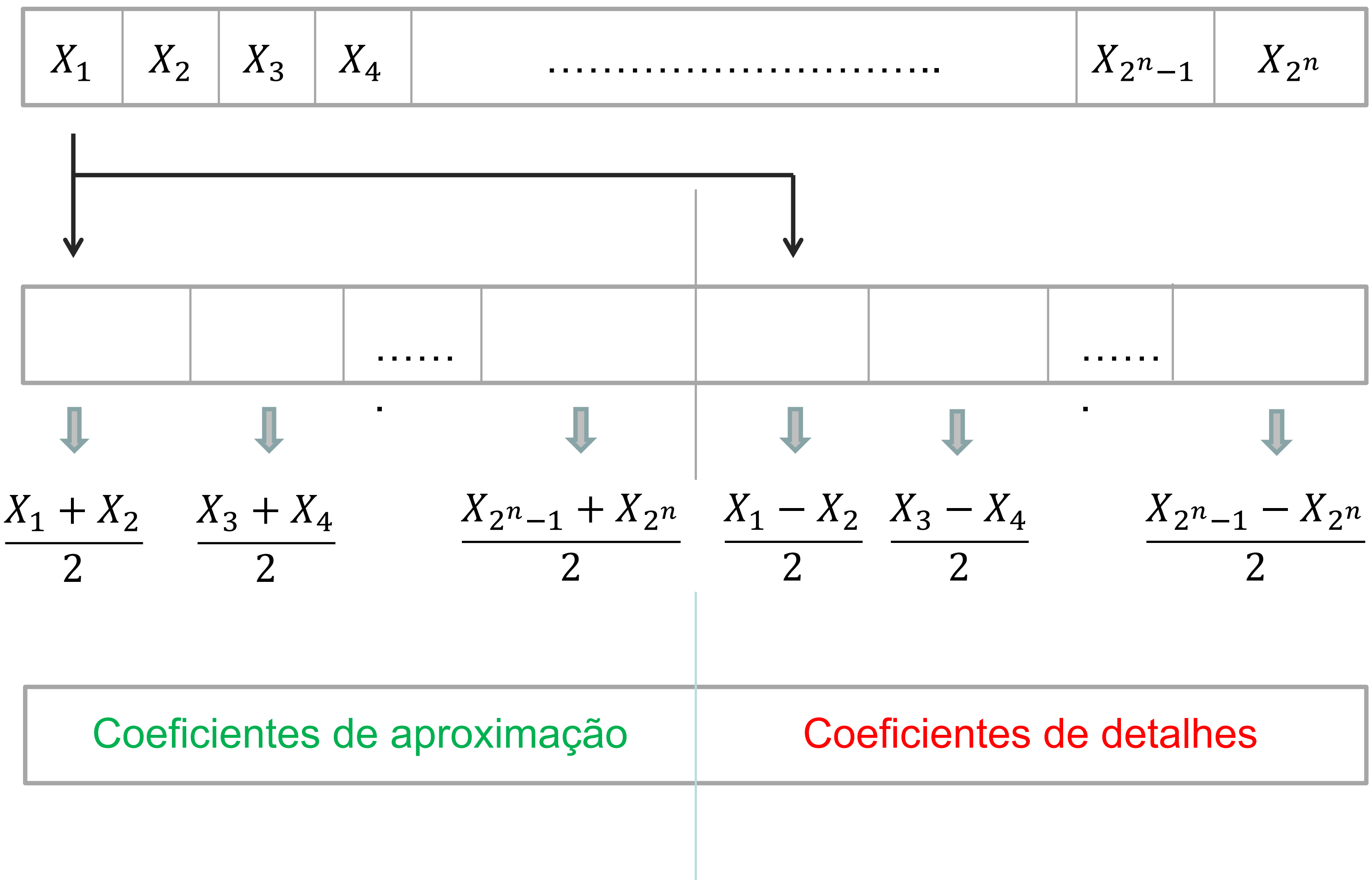
ou:

$$\boxed{h(x) = f \otimes g = \int_{-\infty}^{\infty} f(u)g(x-u)du}$$

- Filtragem no domínio espaci
pela *convolution* (e vi e-v

Na realidade é ao
contrário: a TF é
uma ferramenta
para filtragem.

Transformada Haar



Transformada Haar

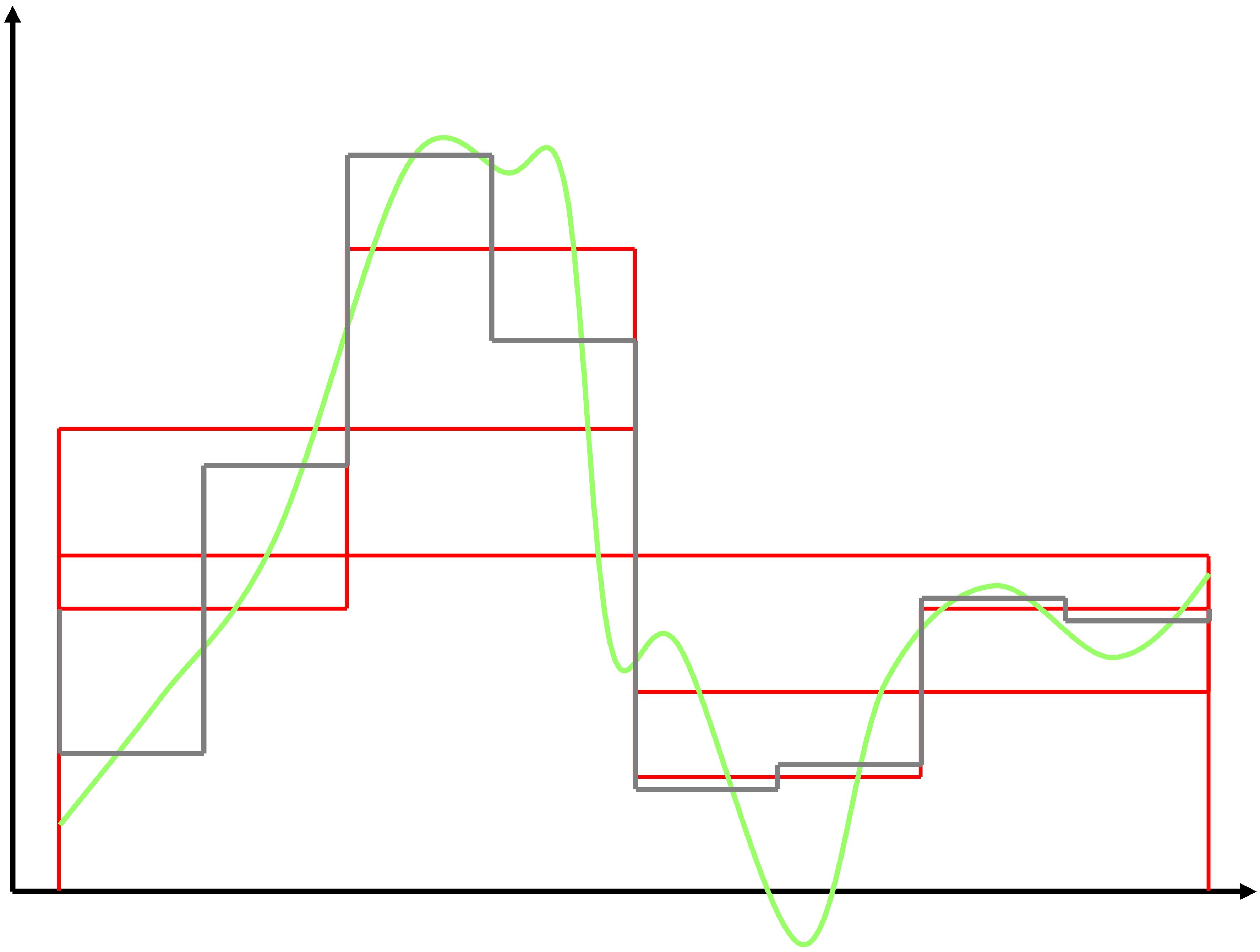
10	4	6	12	8	4	2	6

7	9	6	4	3	-3	2	-2

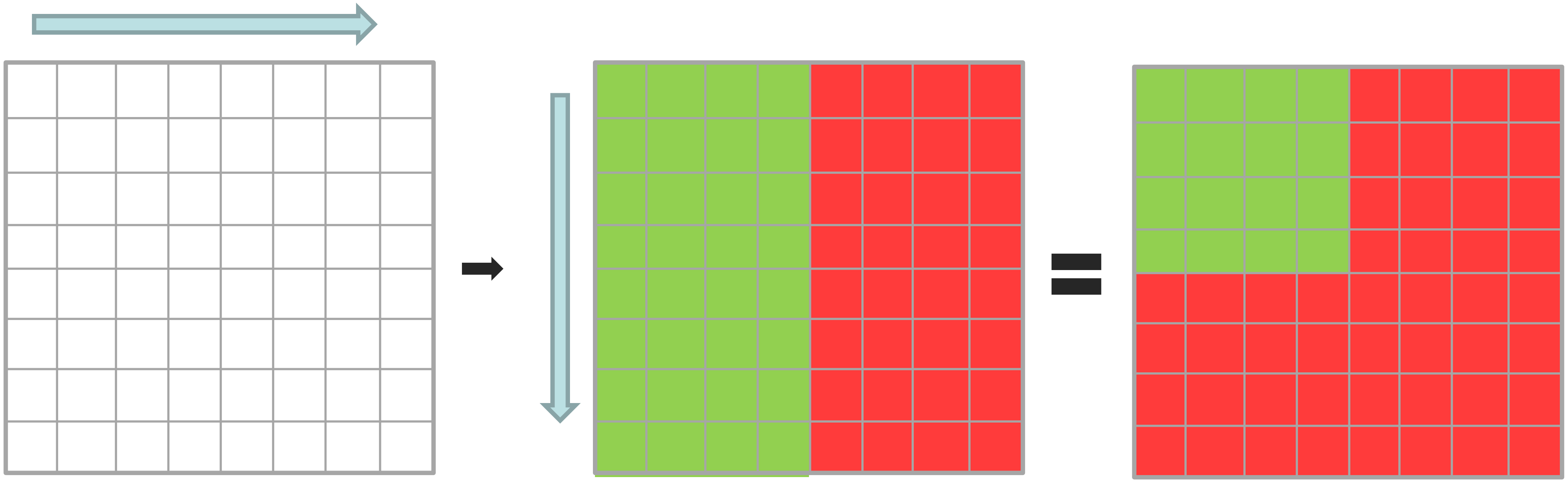
8	5	-1	1	3	-3	2	-2

6,5	1,5	-1	1	3	-3	2	-2

Três iterações de Haar sobre um vetor



Transformada Haar



Aplicando a transformada de Haar na horizontal

Aplicando a transformada de Haar na vertical

Matriz resultante

Uma iteração da transformada de Haar sobre uma matriz

Transformada de Haar

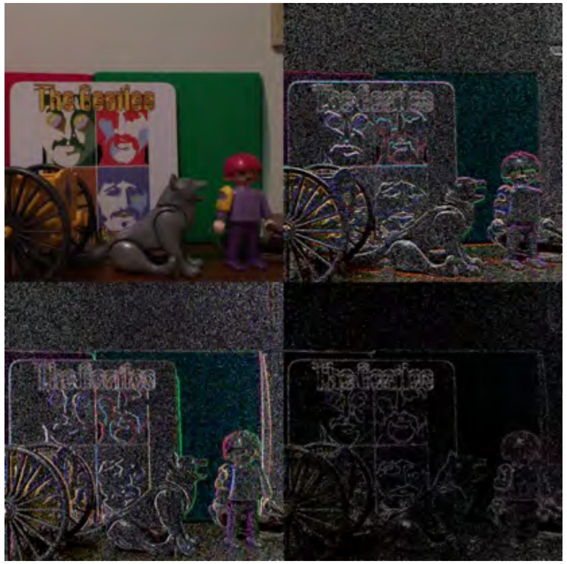


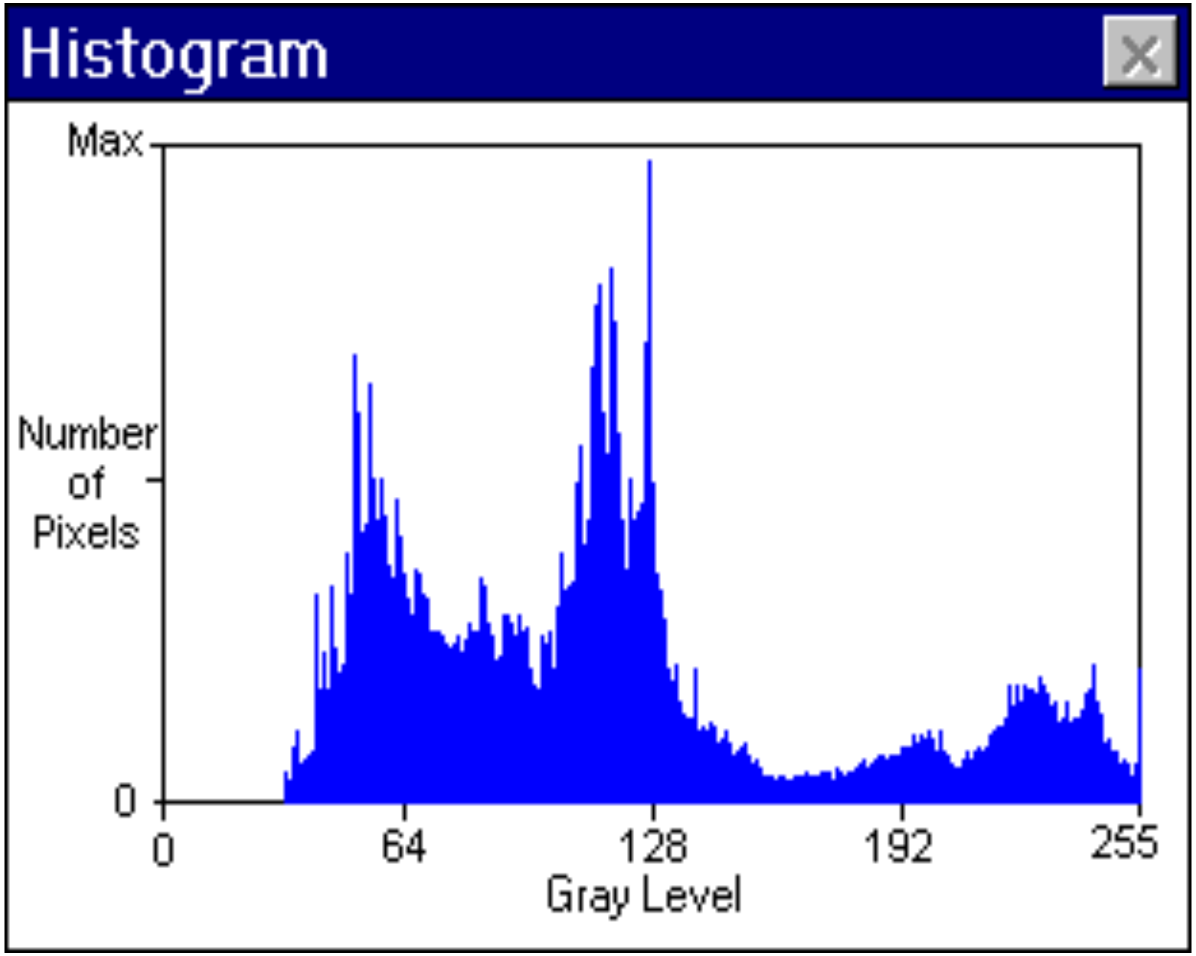
Imagem Digital
Conceitos, Processamento e Análise

Parte 3 - Processamentos apenas no espaço das cores

PROCESSAMENTO NO ESPAÇO DE COR

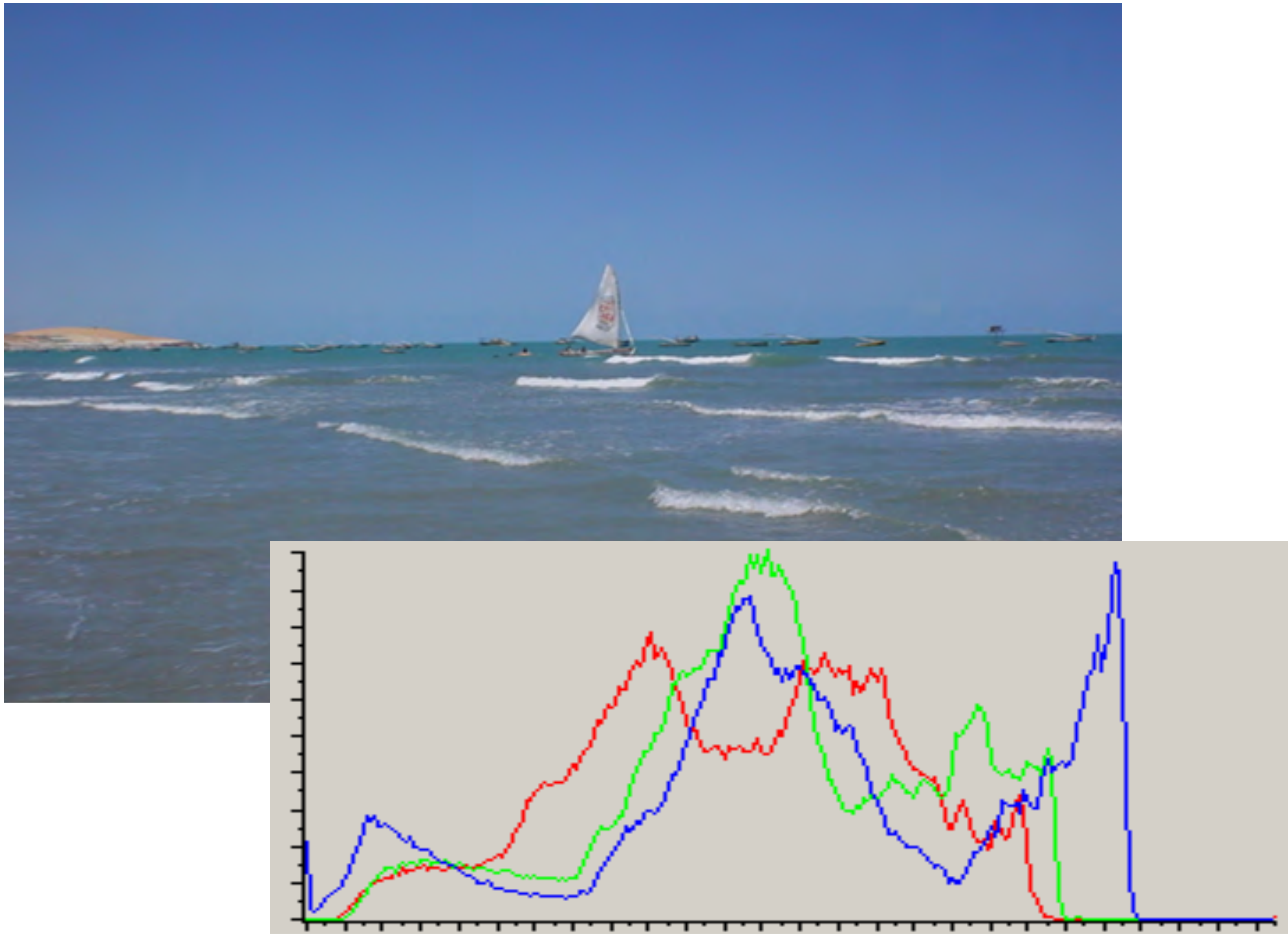
- Correção gama
- Equalização de histograma
- Quantização de cores
- Superpixels

Histogramas da Função



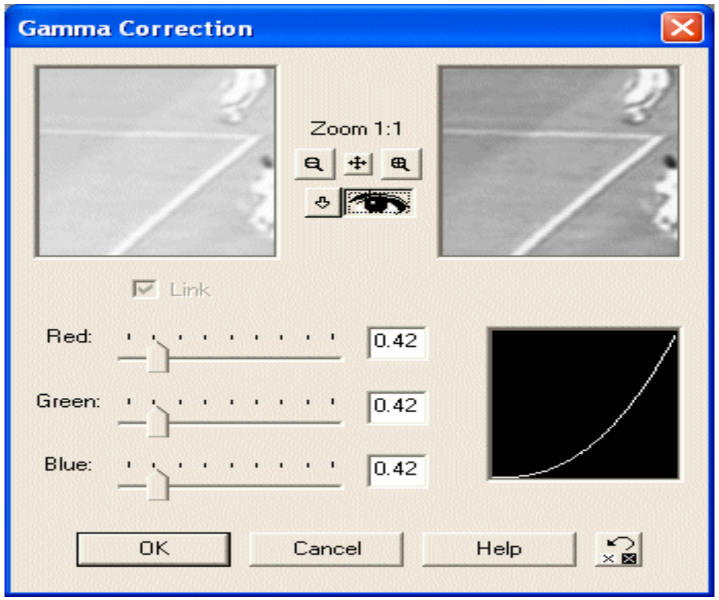
Uma outra maneira de ver a informação da imagem: probabilidade de ocorrência de um determinado valor, uso do intervalo [0,255], contraste,...

Histogramas de Imagem Colorida



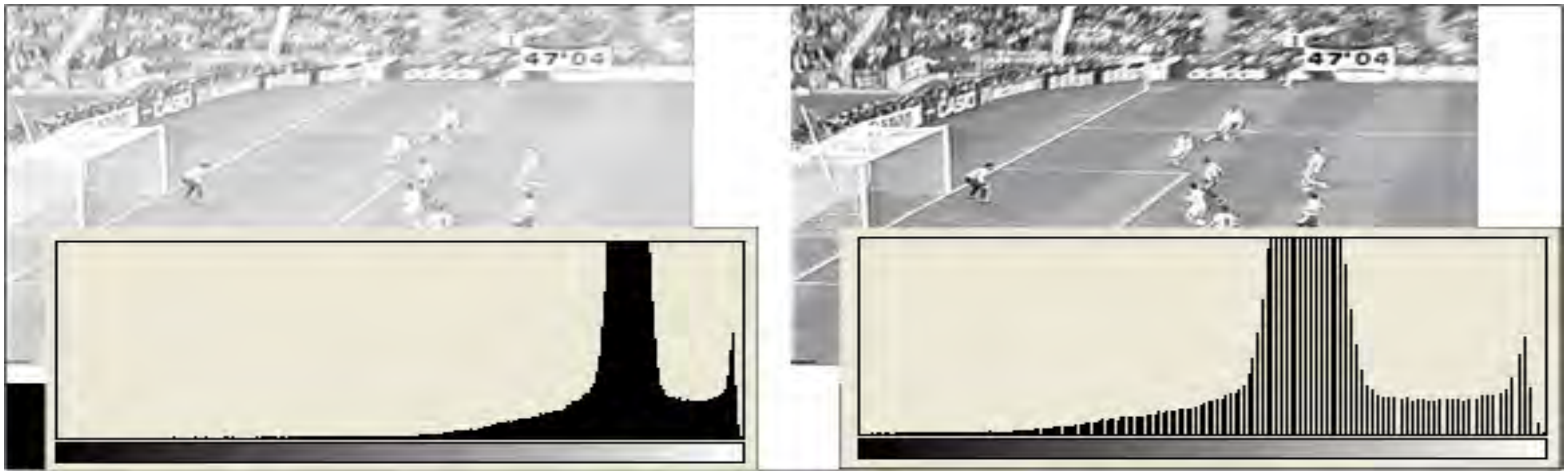
Correção gama

Ajustes de contraste e iluminação

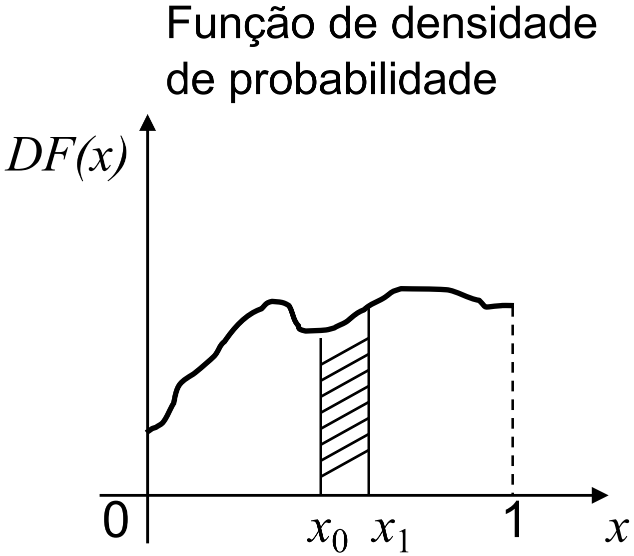


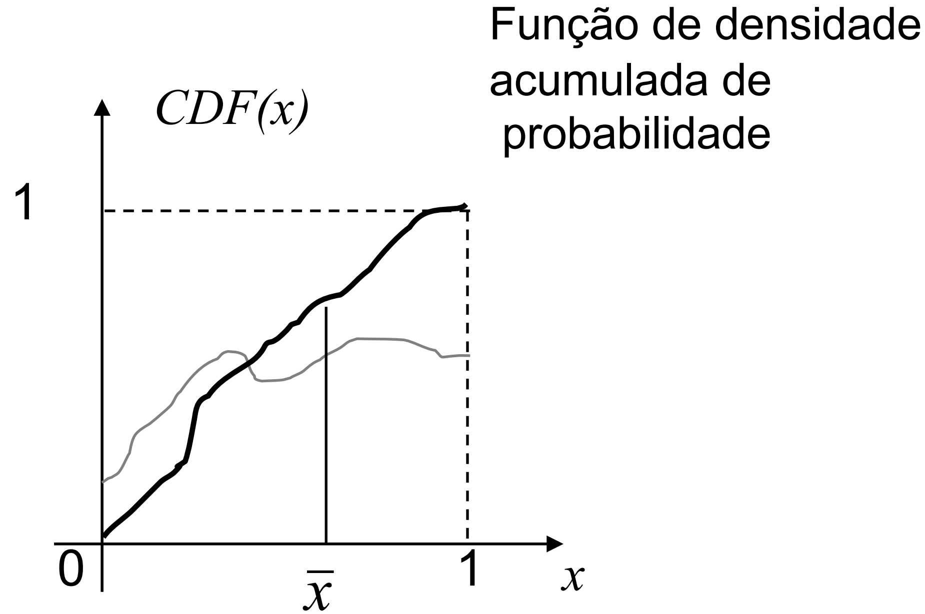
Correção gama

$$L \leftarrow \sqrt[\gamma]{L}$$



Probabilidade



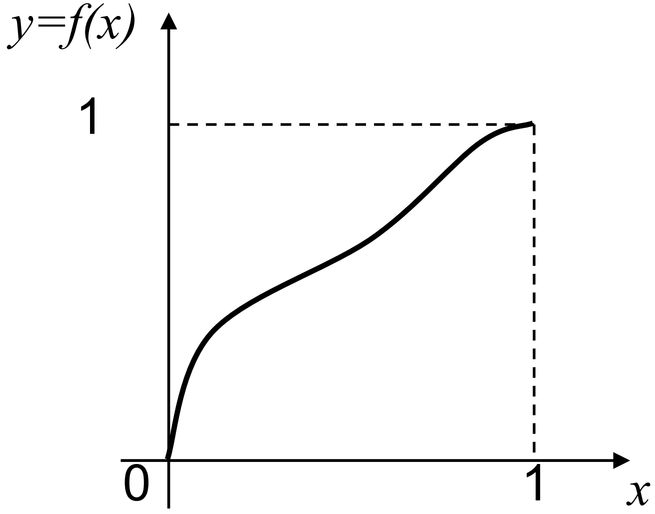


$$P\{x_0 \le x < x_1\} = \int_{x_0}^{x_1} DF(x)dx$$

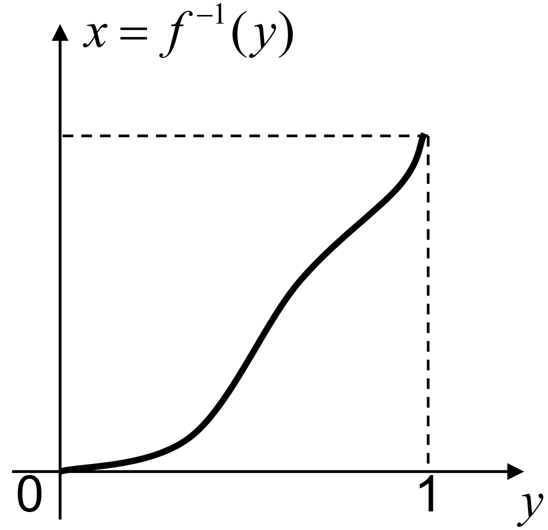
$$CDF(\bar{x}) = P\{0 \le x < \bar{x}\} = \int_0^{\bar{x}} DF(x)dx$$

$$DF(x) = \frac{d}{dx} CDF(\bar{x})$$

Mudança de variavel $y = f(x)$



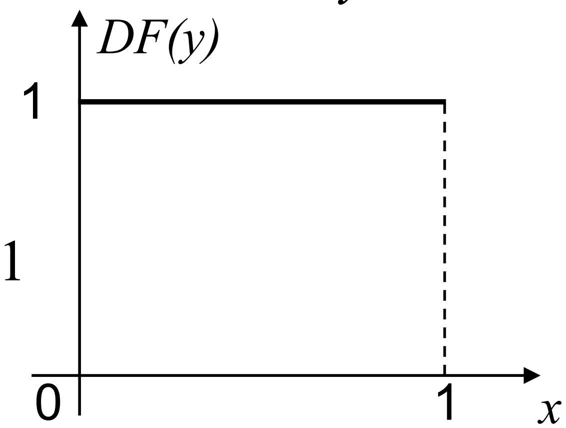
Transformação monotônica e limitada ao intervalo [0,1]



$$DF(y) = \frac{d}{dy} CDF(y) = \frac{d}{dx} CDF(x) \frac{dx}{dy} \qquad = DF(x) \frac{dx}{dy}$$

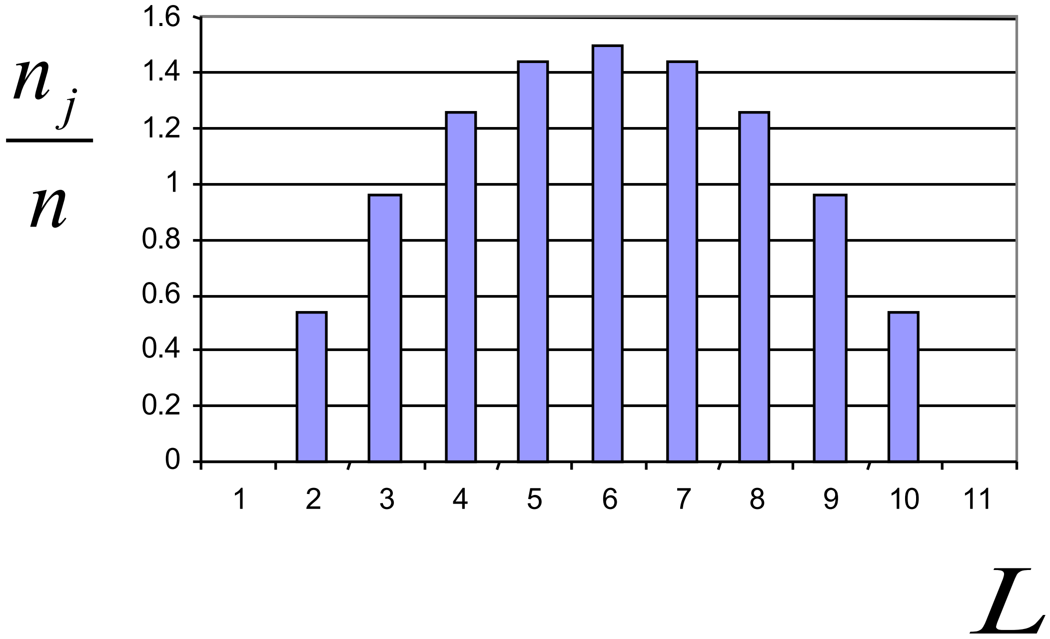
$$f(x) = CDF(x) \qquad \frac{dy}{dx} = DF(x)$$

$$DF(y) = \frac{DF(x)}{DF(x)} = 1$$

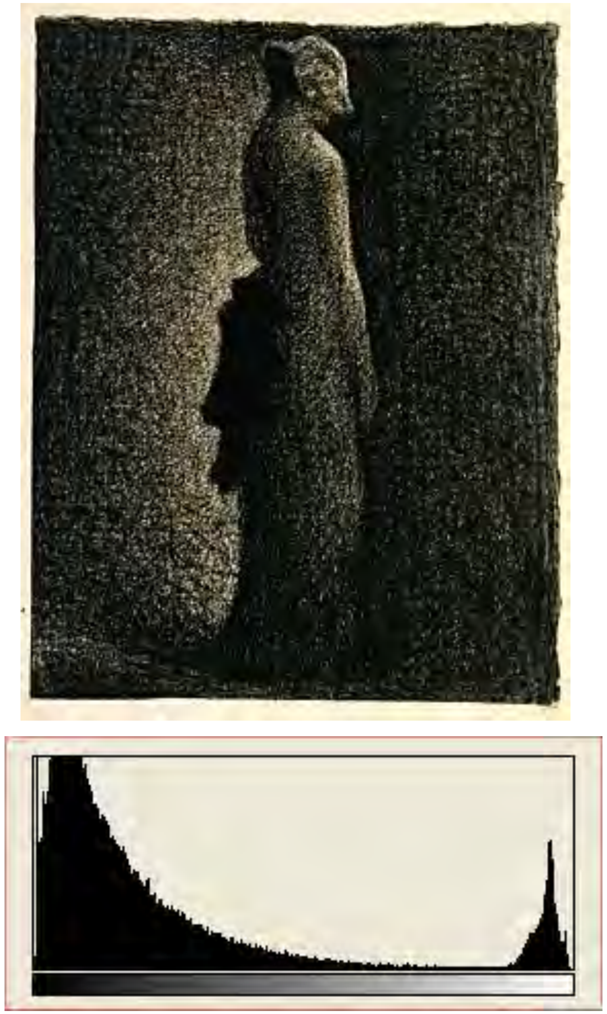


Equalização de Histograma

$$L' = f(L) = \sum_{j=0}^{L} \frac{n_j}{n}$$



Equalização do histograma



Tons de cinza e negativo

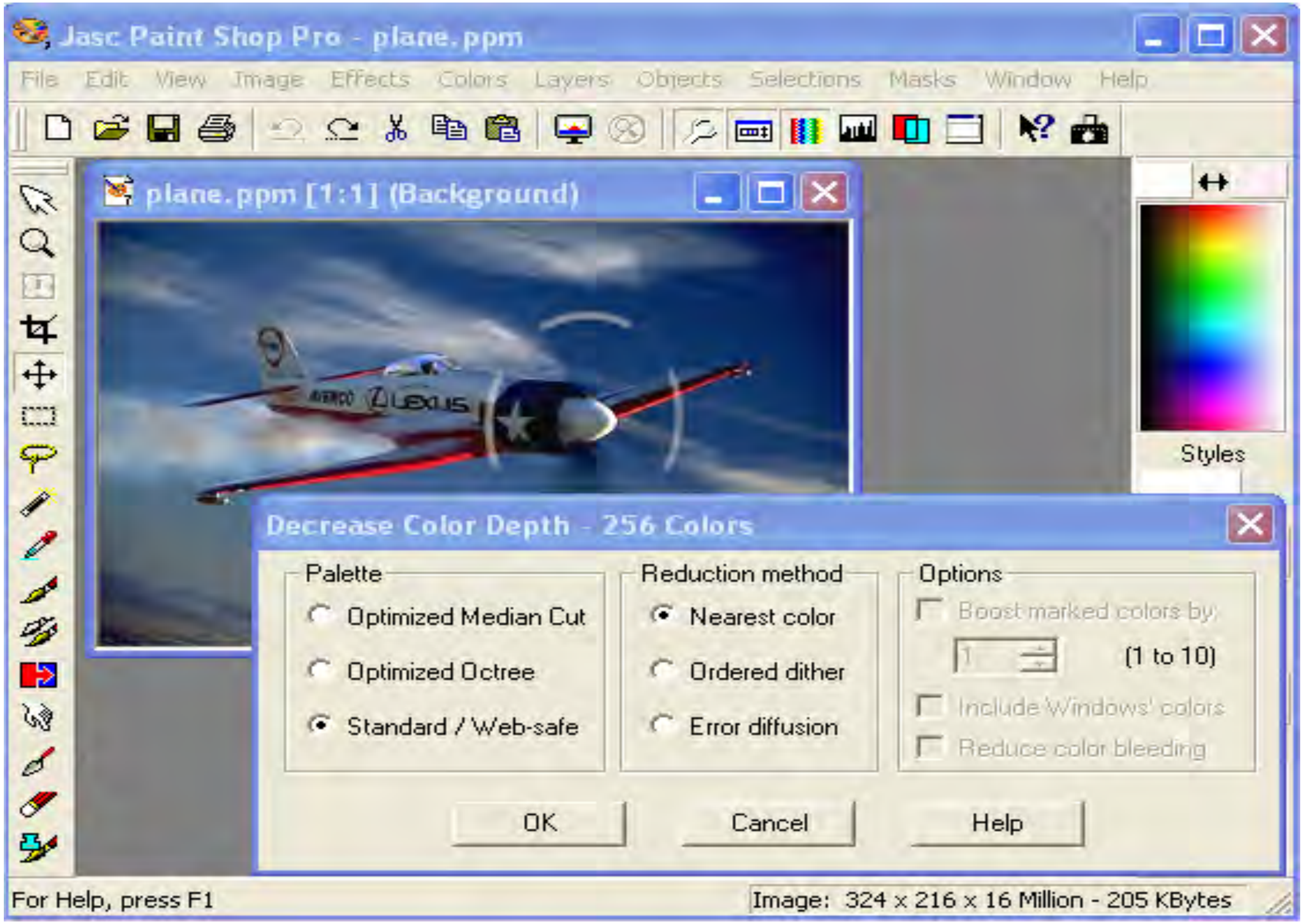
$$L_{x,y} = 0.299\boldsymbol{R}_{x,y} + 0.587\boldsymbol{G}_{x,y} + 0.114\boldsymbol{B}_{x,y}$$

tons de cinza

$$L_{x,y} = 255 - L_{x,y}$$

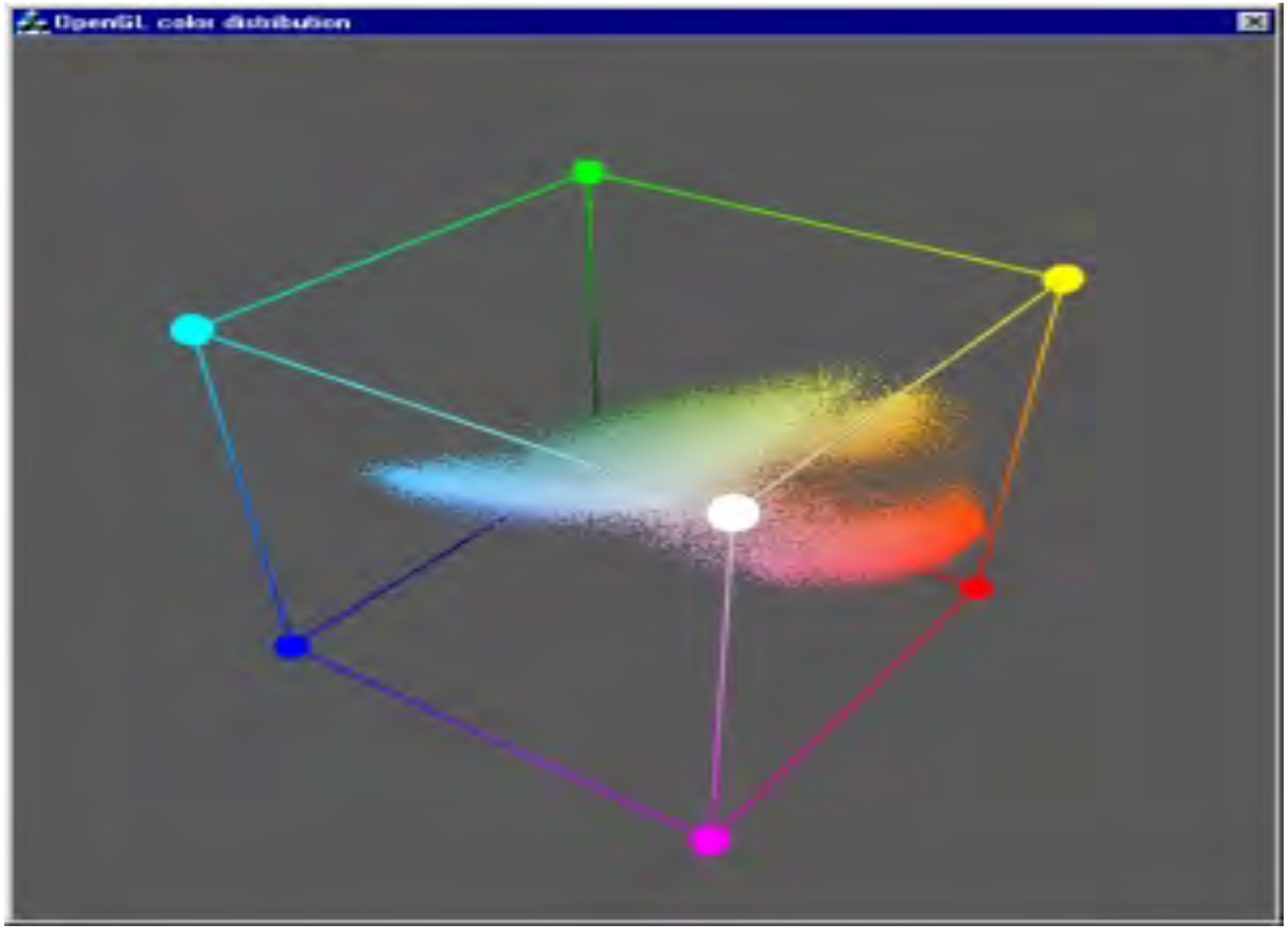
Quantização de cores

Quantização de 24 para 8 *bits*

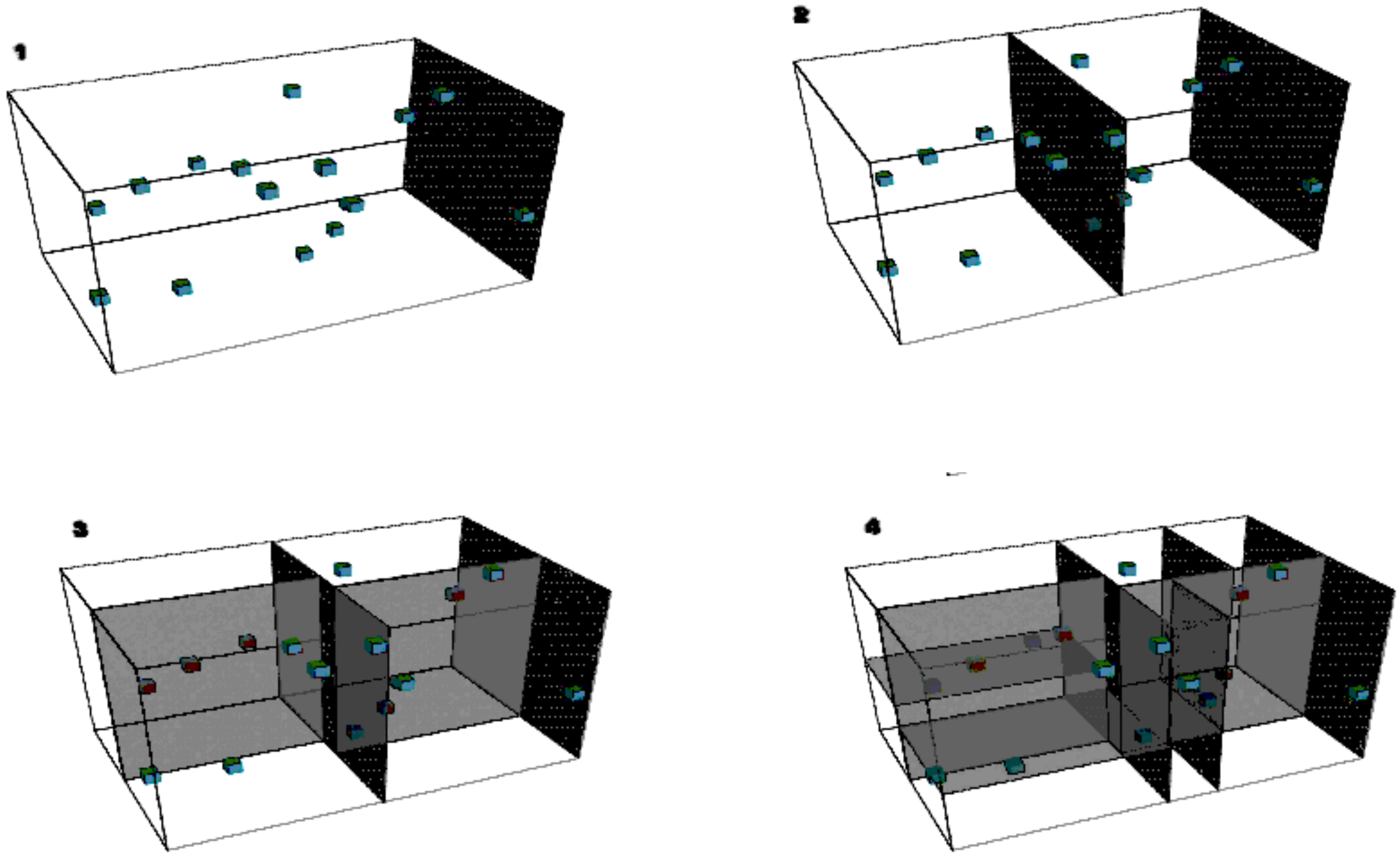


A qualidade depende da imagem

Corte mediano



Corte mediano



ARMAZENAMENTO DE IMGES

Compressão

Um pouco de teoria da informação
Codificação uniforme

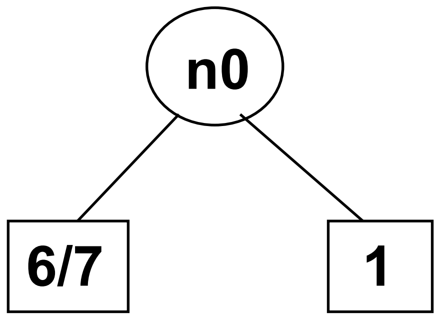
		Uniforme		
tons	# *pixels*	código	tam.	# *bits*
0	**1900**	**000**	**3**	**5700**
1/7	**2500**	**001**	**3**	**7500**
2/7	**2100**	**010**	**3**	**6300**
3/7	**1600**	**011**	**3**	**4800**
4/7	**800**	**100**	**3**	**2400**
5/7	**600**	**101**	**3**	**1800**
6/7	**300**	**110**	**3**	**900**
1	**200**	**111**	**3**	**600**
			TOTAL	**30000**

Podemos melhorar?

Construção da Árvore Huffman

1/7	2500
2/7	2100
0	1900
3/7	1600
4/7	800
5/7	600
6/7	300
1	200

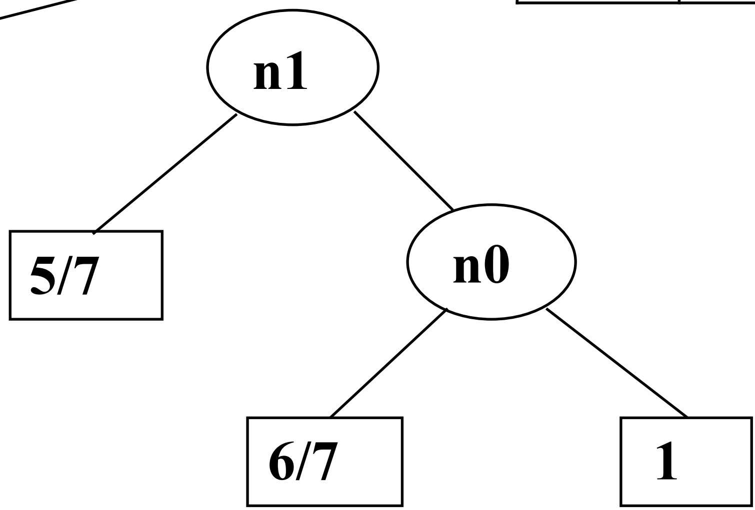
1/7	2500
2/7	2100
0	1900
3/7	1600
4/7	800
5/7	600
n0	**500**



Construção da Árvore Huffman

1/7	2500
2/7	2100
0	1900
3/7	1600
4/7	800
5/7	600
n0	500

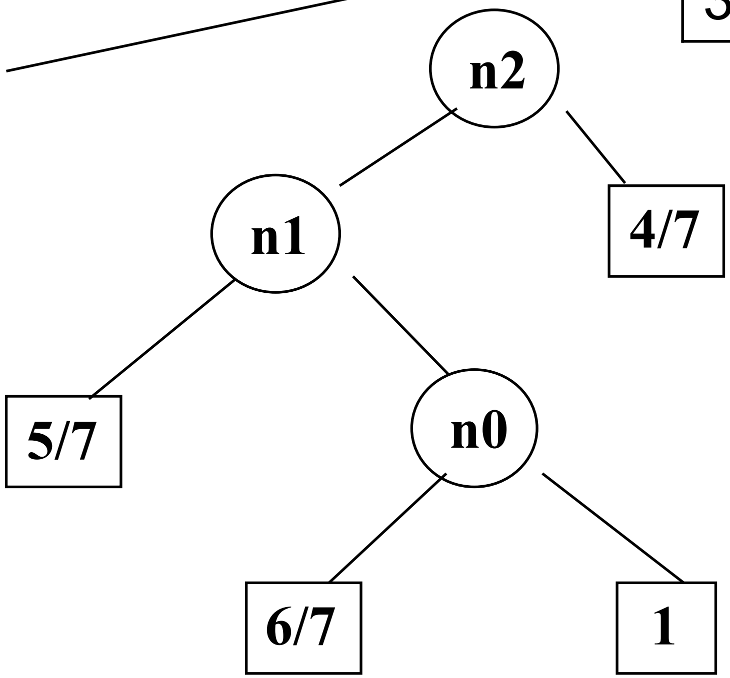
1/7	2500
2/7	2100
0	1900
3/7	1600
n1	**1100**
4/7	800



Construção da Árvore Huffman

1/7	2500
2/7	2100
0	1900
3/7	1600
n1	1100
4/7	800

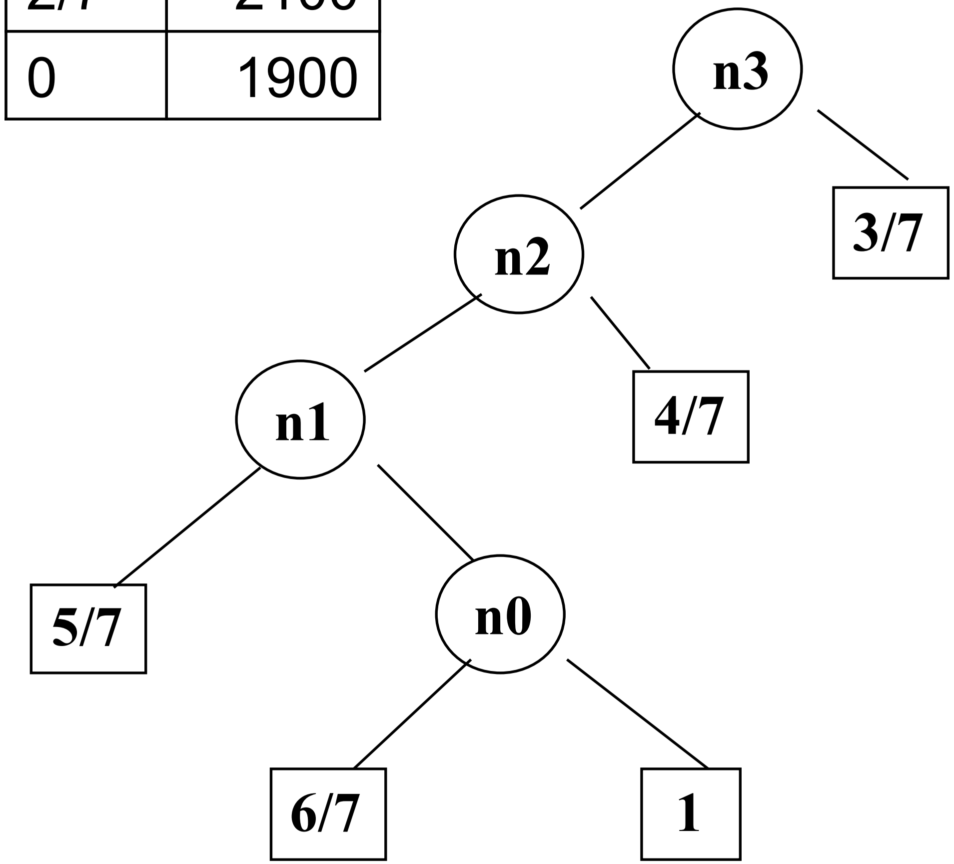
1/7	2500
2/7	2100
0	1900
n2	**1900**
3/7	1600



Construção da Árvore Huffman

1/7	2500
2/7	2100
0	1900
n2	1900
3/7	1600

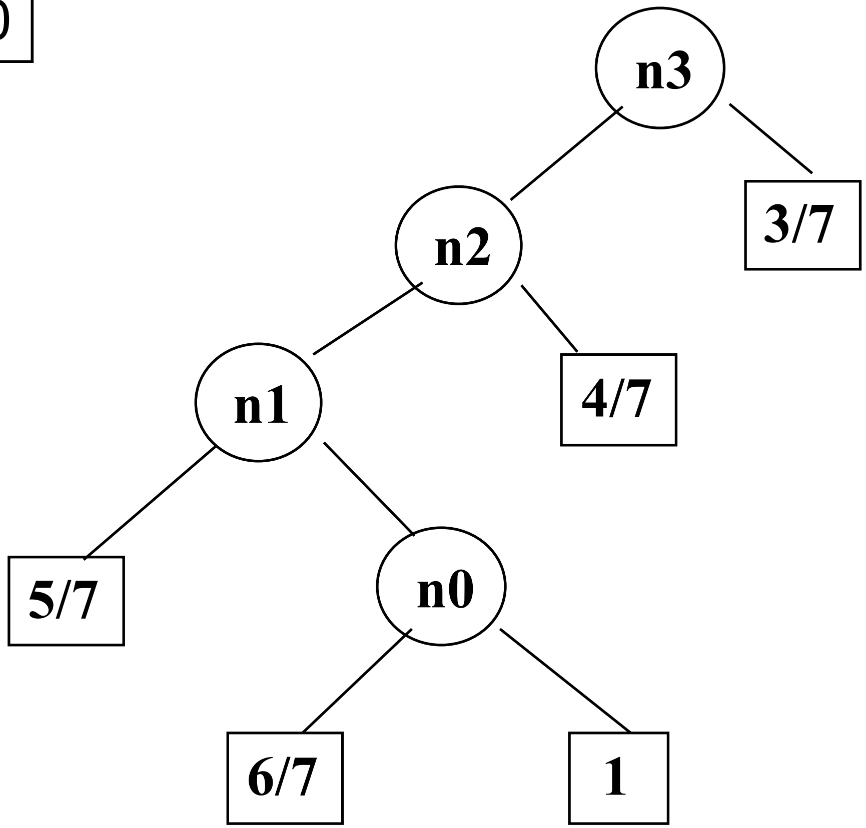
n3	**3500**
1/7	2500
2/7	2100
0	1900

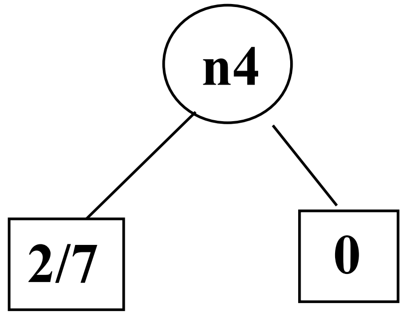


Construção da Árvore Huffman

n3	3500
1/7	2500
2/7	2100
0	1900

n4	4000
n3	3500
1/7	2500

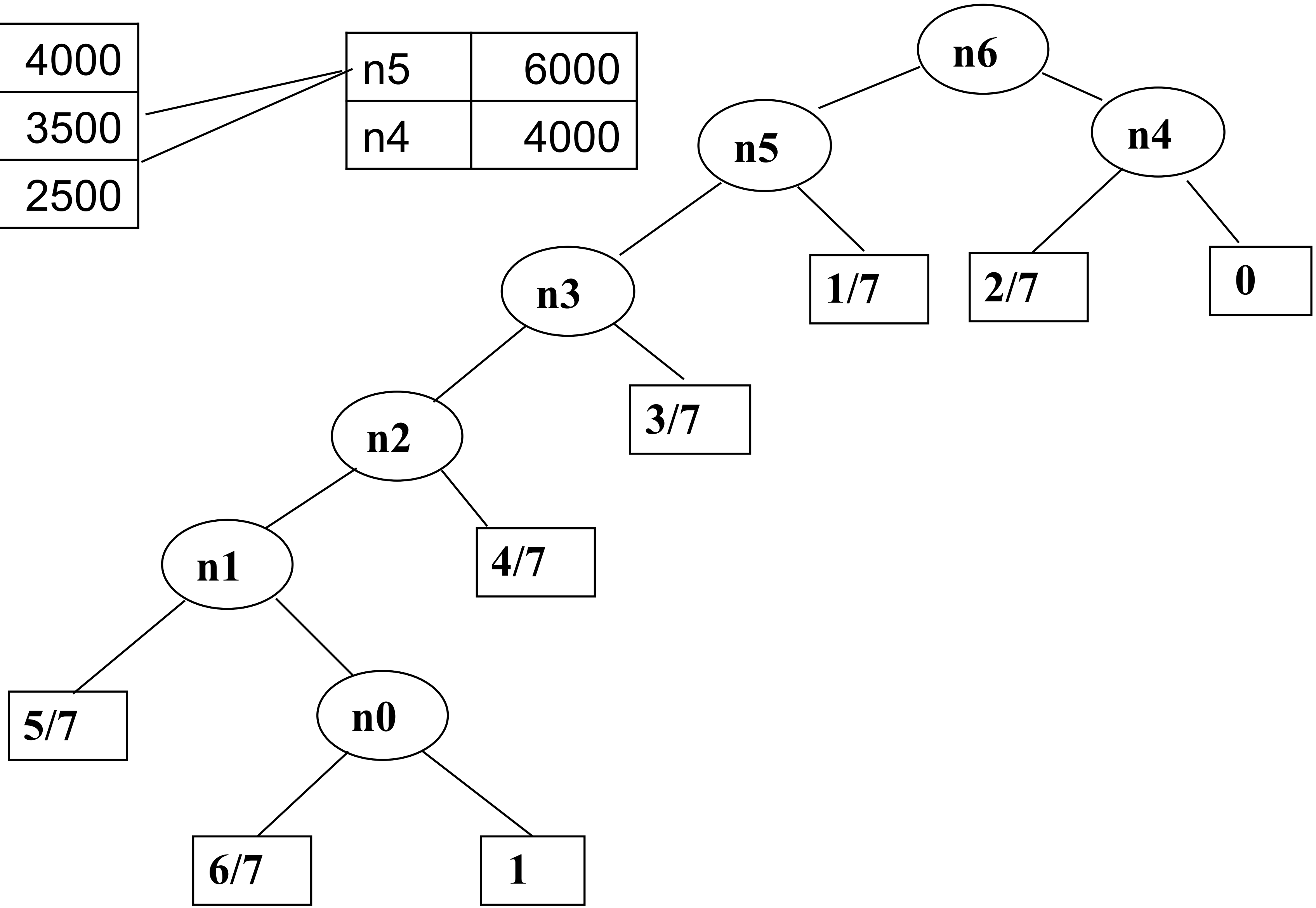




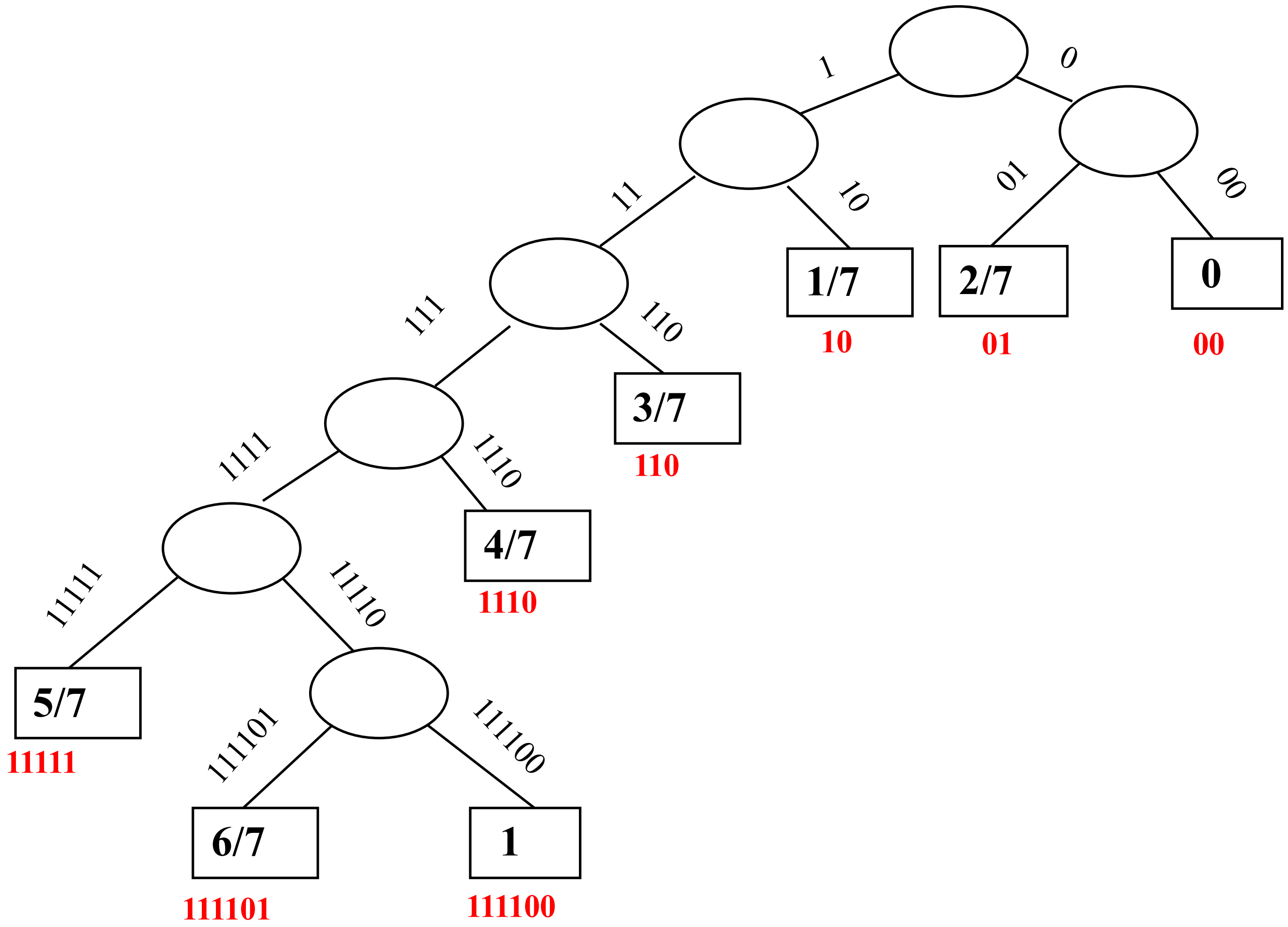
Construção da Árvore Huffman

n4	4000
n3	3500
1/7	2500

n5	6000
n4	4000



Construção da Árvore Huffman



Codificação de Huffman

		Uniforme			Huffman		
tons	# *pixels*	código	tam.	# *bits*	código	tam.	# *bits*
0	**1900**	**000**	**3**	**5700**	**00**	**2**	**3800**
1/7	**2500**	**001**	**3**	**7500**	**10**	**2**	**5000**
2/7	**2100**	**010**	**3**	**6300**	**01**	**2**	**4200**
3/7	**1600**	**011**	**3**	**4800**	**110**	**3**	**4800**
4/7	**800**	**100**	**3**	**2400**	**1110**	**4**	**3200**
5/7	**600**	**101**	**3**	**1800**	**11111**	**5**	**3000**
6/7	**300**	**110**	**3**	**900**	**111101**	**6**	**1800**
1	**200**	**111**	**3**	**600**	**111100**	**6**	**1200**
			TOTAL	**30000**		**TOTAL**	**27000**

Redundância de Codificação

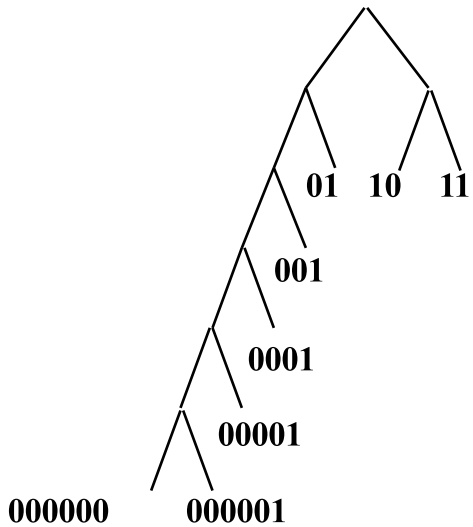
r	p(r)	Code 1	l(r)	l(r)p(r)	Code 2	l(r)	l(r)p(r)
0	0.19	000	3	0.57	11	2	0.38
1/7	0.25	001	3	0.75	01	2	0.50
2/7	0.21	010	3	0.63	10	2	0.42
3/7	0.16	011	3	0.48	001	3	0.48
4/7	0.08	100	3	0.24	0001	4	0.32
5/7	0.06	101	3	0.18	00001	5	0.30
6/7	0.03	110	3	0.09	000001	6	0.18
1	0.02	111	3	0.06	000000	6	0.12
	1.00		L_{avg}=	3.00		L_{avg}=	2.70

r_k = tons de cinza em uma imagem, k=0, 1, ..., $\tau-1$

$p(r_k) = n_k / n$

onde n_k = número de pixels com tom r_k
n = número de pixels da imagem

$$L_{avg} = \sum_{k=0}^{\tau-1} l(r_k)p(r_k)$$



Resultado da Teoria da Informação

$$l_{opt}(r_k) = \log_2\left(\frac{1}{p(r_k)}\right) \qquad \textit{número de bits}$$

r	p(r)	Code 1	l(r)	l(r)p(r)	Code 2	l(r)	l(r)p(r)	log(1/p)	log(1/p)*p
0	0.19	000	3	0.57	11	2	0.38	2.4	0.46
1/7	0.25	001	3	0.75	01	2	0.50	2.0	0.50
2/7	0.21	010	3	0.63	10	2	0.42	2.3	0.47
3/7	0.16	011	3	0.48	001	3	0.48	2.6	0.42
4/7	0.08	100	3	0.24	0001	4	0.32	3.6	0.29
5/7	0.06	101	3	0.18	00001	5	0.30	4.1	0.24
6/7	0.03	110	3	0.09	000001	6	0.18	5.1	0.15
1	0.02	111	3	0.06	000000	6	0.12	5.6	0.11
$\sum$=1.00			L_{avg} =	3.00		L_{avg} =	2.70	L_{opt} =	2.65

Compressão de imagens

Compressão de Imagens

Sem Perda

- Preserva exatamente o conteúdo da imagem
- Taxas de compressão 3 : 1

Com Perda

- Preserva de forma controlada o nível de qualidade da imagem
- Taxas de compressão que chegam a valores de mais de 100 : 1

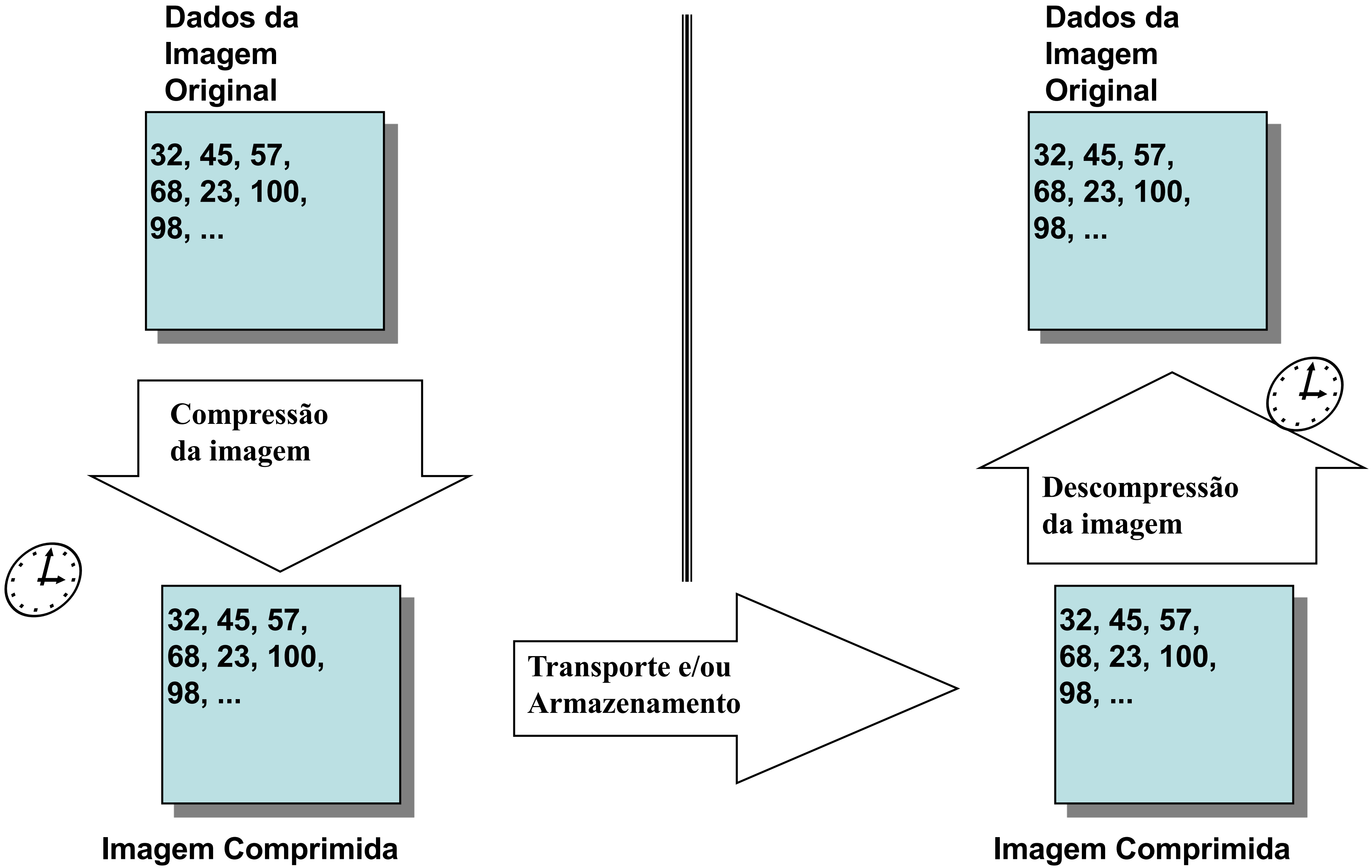
Métodos de compressão

- *Sem perdas*
  - *Run length encoding (RLE) - **repetição***
  - *Huffman coding - **histograma***
  - *Predictive coding **- diferenças***
  - *Block coding (LZW) **- dicionário***

- *Com perdas*
  - *Truncation coding - **reduz a representação***
  - *Predictive coding - **descarta diferenças altas***
  - *Block coding - **dicionário aproximado***
  - *Transform coding - **descarta frequencias altas***

Métodos compostos: JPEG, MPEG

Processo de compressão e descompressão

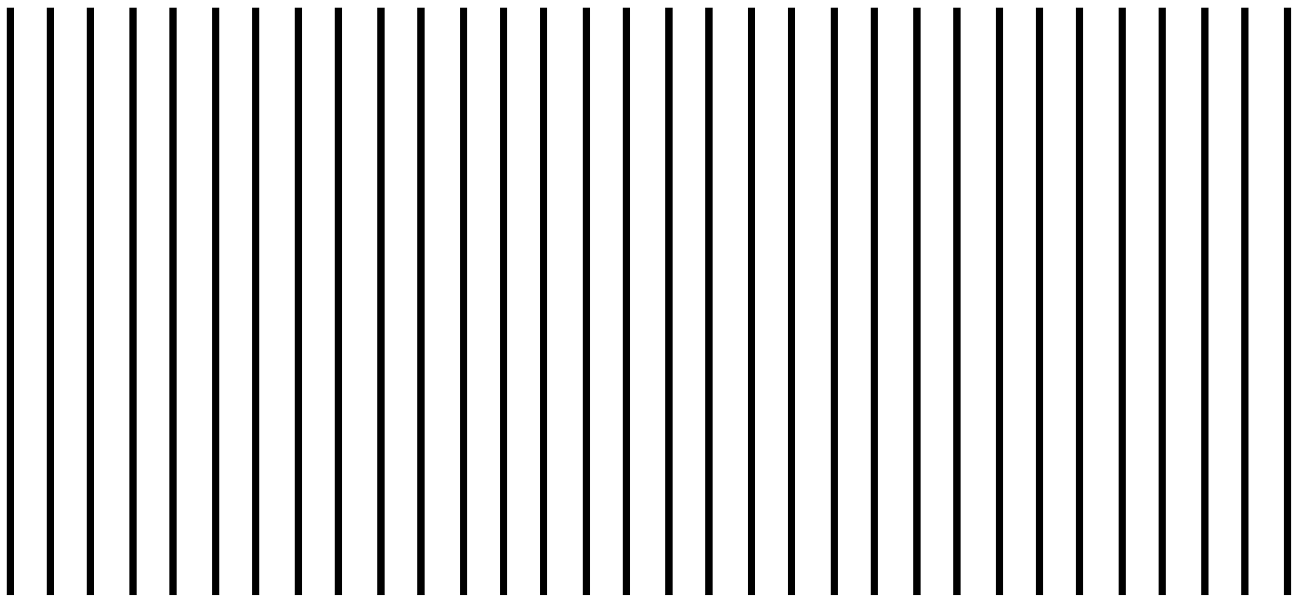


Fundamentos da Compressão de Imagens

A compressão de uma imagem é obtida quando se elimina a redundância de:

- codificação
- entre *pixels*
- psico-visual

Redundância entre *pixels*



640 colunas x 480 linhas x 1 byte/pixel = 300 KBytes

480*(1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0) = ~18 Kbytes

onde 1 = 32 bytes de preto e 0 = 32 bytes de branco

Compressão - RLE

Objetivo

Reduzir a quantidade de dados redundantes.

Exemplo

AAAAAAxxx → **6A3x**

Caracterísiticas

Simples e rápido, porém a eficiência depende da imagem a ser comprimida.

Run-Length Encoding

76	**76**	**76**	**76**	**76**	**78**	**79**	**79**	**79**	**79**	**80**	**80**

76 | 5 **78 | 1** **79 | 4** **80 | 2**

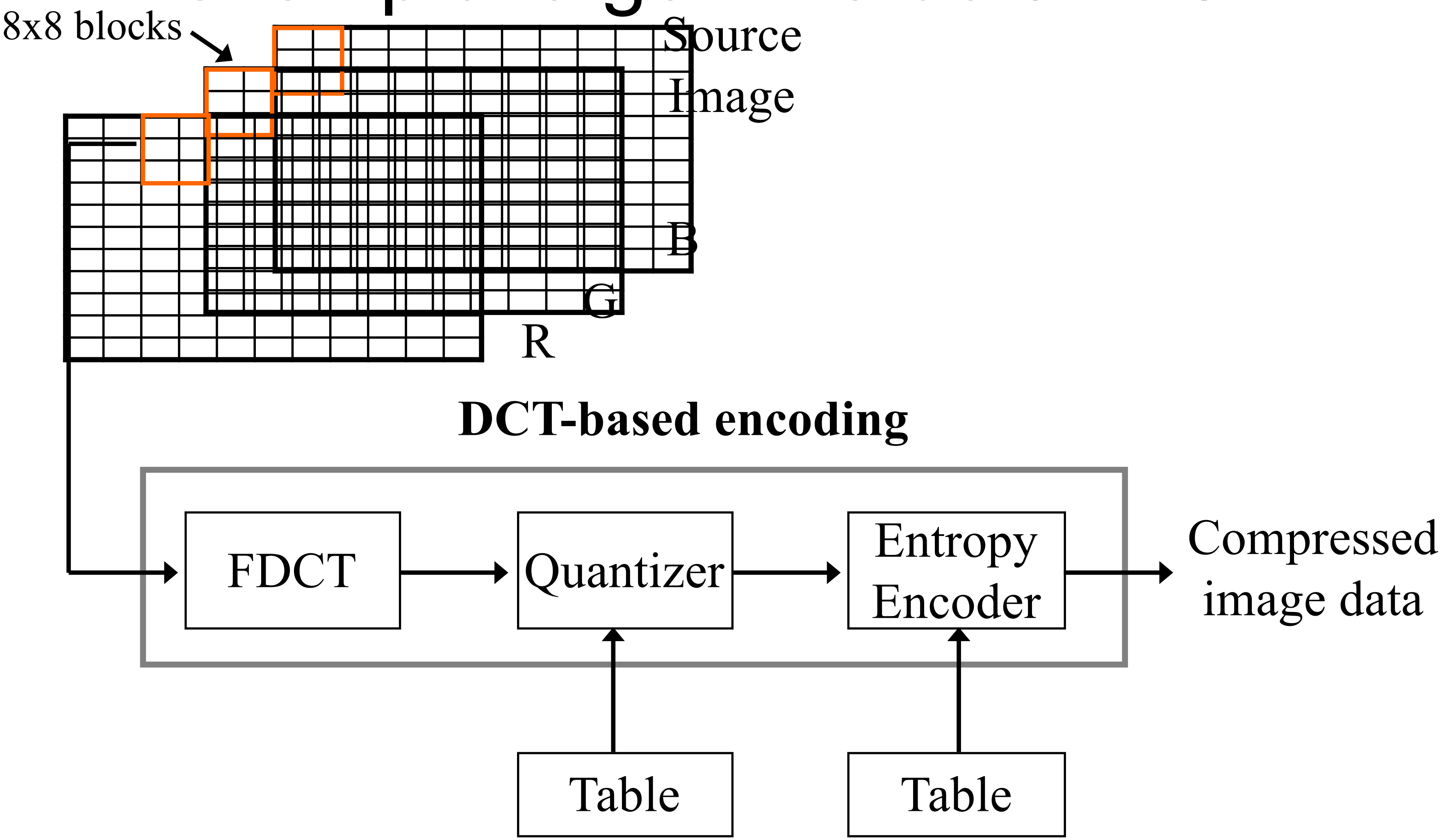
imagem binária

0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 . . .

7 4 5

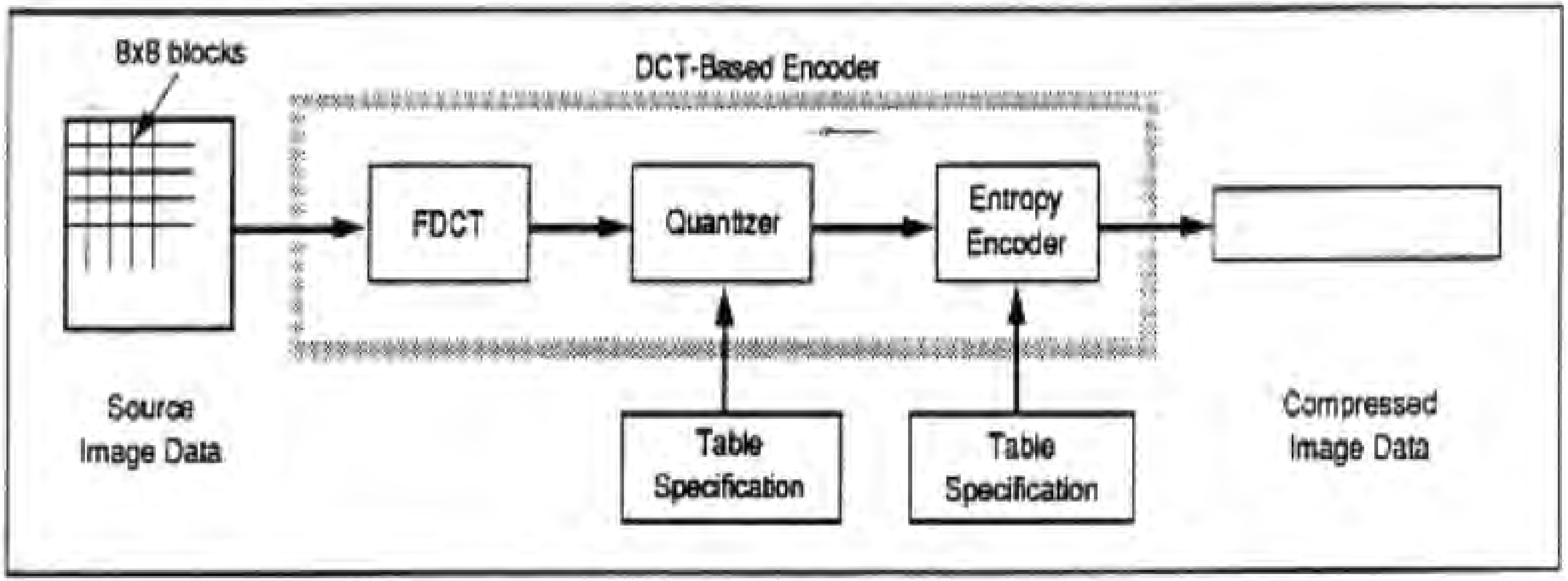
Compressão do jpeg

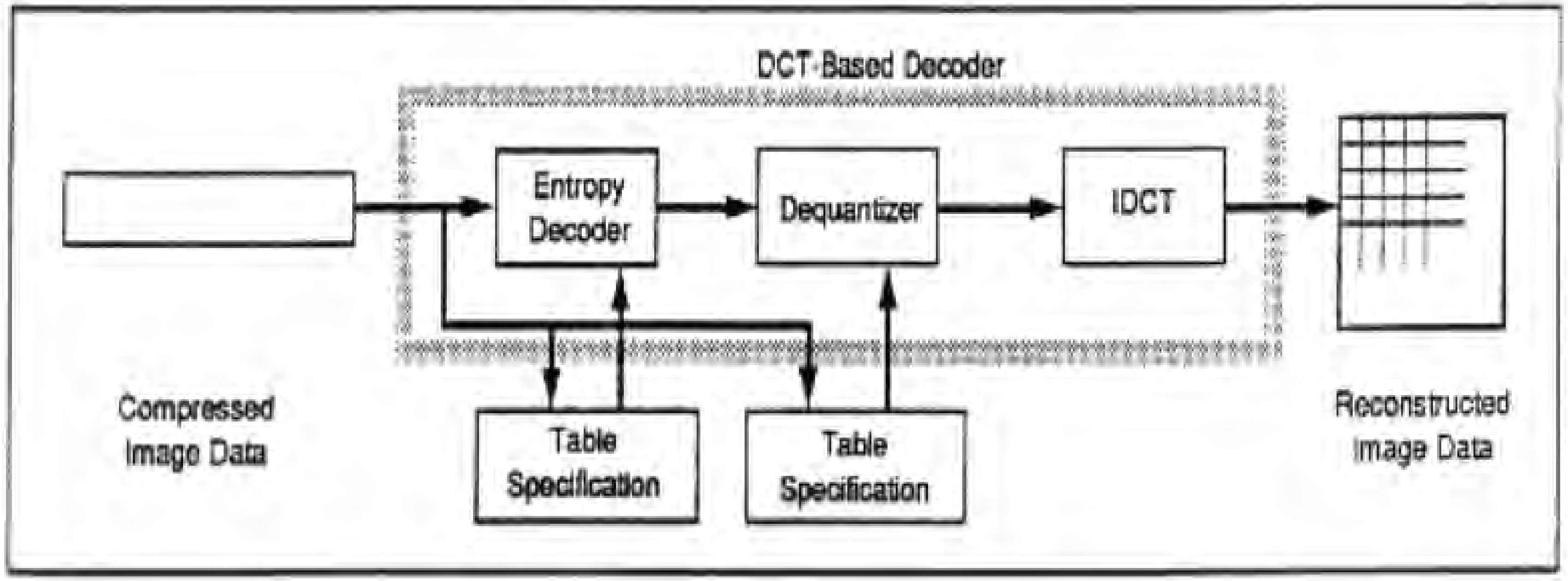
Aplicações são tecnologicamente complexas: exemplo: algoritmo do JPEG



University of Southern Denmark

The Maersk Mc-Kinney Moller Institute for Production Technology





Jpeg image compression

Run   Load image   Prefactor = 1.0

```
Size of original bitmap, 1 byte per pixel (kbyte): 94.720
Quantization prefactor: 1.00
Compressed size of image (kbyte): 6.887
Kompression ratio:  13.8
The compressed image is stored on the disk as D:\COURSES\AM37-2~1\DEMOS\JPEG\GEORGE.###
```

Jpeg image compression

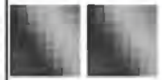
Run Load image Prefactor = 1.0

Graytones of original image

71	111	155	180	185	174	142	113
54	85	134	169	181	184	165	139
58	71	107	152	176	183	182	158
60	56	74	118	158	179	183	172
64	58	62	92	134	169	179	180
79	71	63	73	105	150	172	177
69	69	63	55	76	118	158	173
80	80	75	63	60	90	138	165

DCT without quantization

961	-290	-30	21	-13	8	-1	0
135	20	-171	8	1	2	0	0
-14	87	-10	-44	-13	0	1	0
4	11	3	11	-8	-8	3	-2
-6	14	0	-1	6	-2	0	0
-3	0	2	3	0	0	-1	2
9	14	0	0	1	1	-2	0
-1	-1	-1	-1	0	0	-1	0



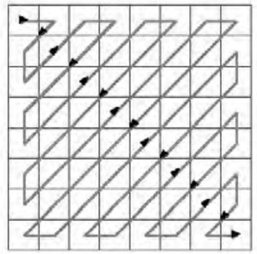
Original and reconstructed block(enlarged)

Quantization tabel

8	12	14	14	18	24	49	72
11	12	13	17	22	35	64	92
10	14	16	22	37	55	78	95
16	19	24	29	56	64	87	98
24	26	40	51	68	81	103	112
40	58	57	87	109	104	121	100
51	60	69	80	103	113	120	103
61	55	56	62	77	92	101	99

DCT after quantization

120	-24	-2	1	0	0	0	0
12	1	-13	0	0	0	0	0
-1	6	0	-2	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0



Reconstructed graytones

64	101	151	183	185	167	146	133
61	90	133	169	183	175	157	143
57	74	106	145	175	183	172	158
56	61	80	118	159	182	182	172
61	57	64	93	135	168	180	179
71	63	60	75	106	141	166	179
82	74	65	64	80	112	149	174
89	82	70	59	64	93	137	169

DCT after dequantization

960	-288	-28	14	0	0	0	0
132	12	-169	0	0	0	0	0
-10	84	0	-44	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Equations for JPEG DCT

- Forward DCT:

$$DCT(x,y)=\frac{1}{4}C_xC_y\sum_{x=0}^{7}\sum_{y=0}^{7}Spixel(i,j)\cdot\cos\frac{(2i+1)\cdot x\cdot\pi}{16}\cdot\cos\frac{(2j+1)\cdot y\cdot\pi}{16}$$

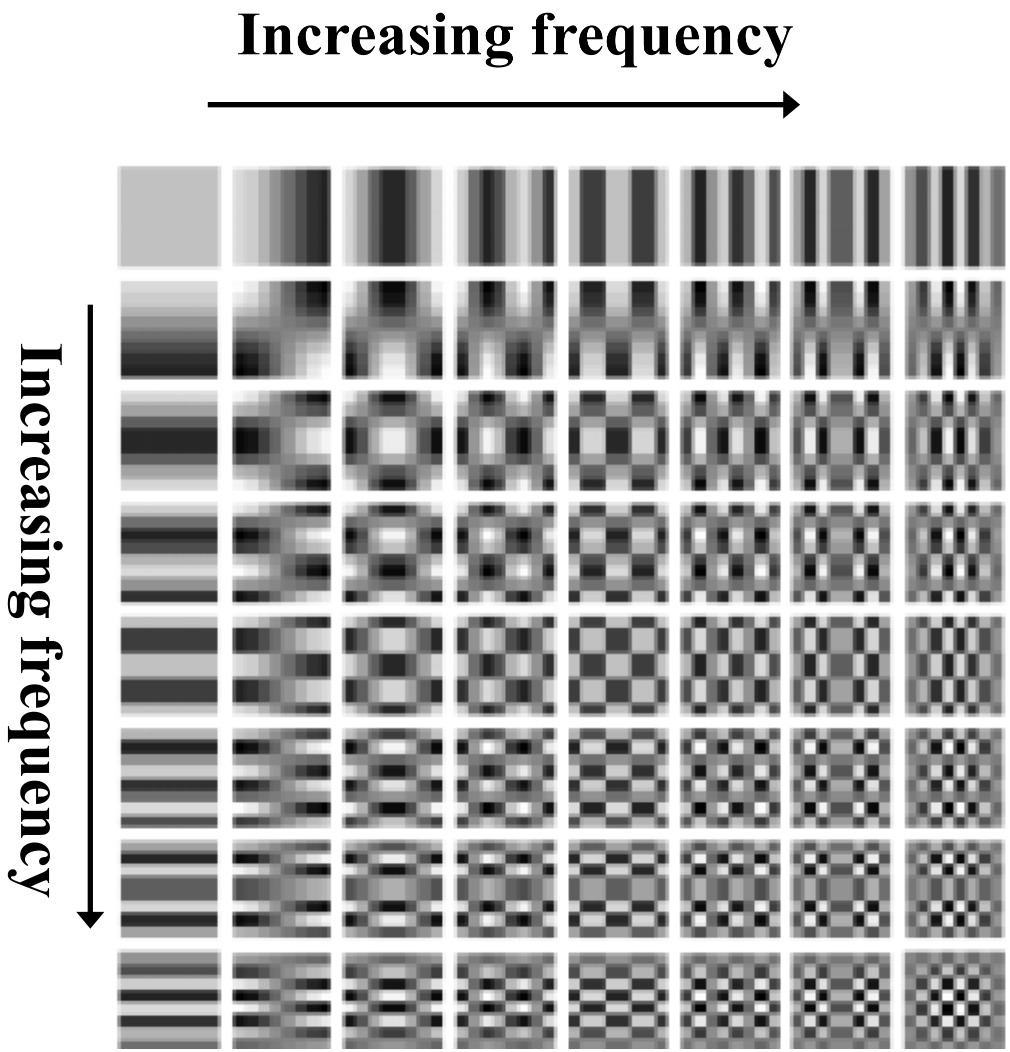
$$\text{where } C_xC_y=\frac{1}{\sqrt{2}} \text{ for x, y} = 0\text{; otherwise } C_x, C_y = 1.$$

- Inverse DCT:

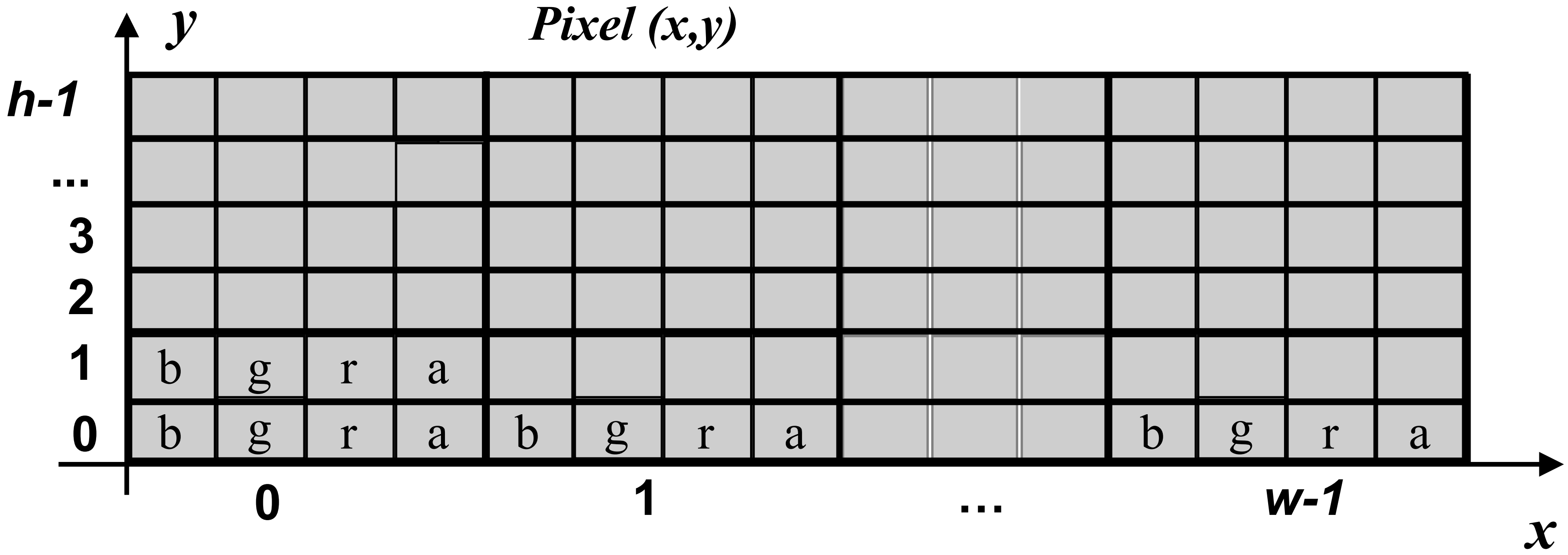
$$pixel(x,y)=\frac{1}{4}\sum_{x=0}^{7}\sum_{y=0}^{7}C_iC_jDCT(i,j)\cdot\cos\frac{(2x+1)j\cdot\pi}{16}\cdot\cos\frac{(2y+1)}{i\cdot\pi}$$

$$\text{where } C_i, C_j=\frac{1}{\sqrt{2}} \text{ for } i,j=0\text{; otherwise } C_i, C_j = 1.$$

Visualization of Basis Functions

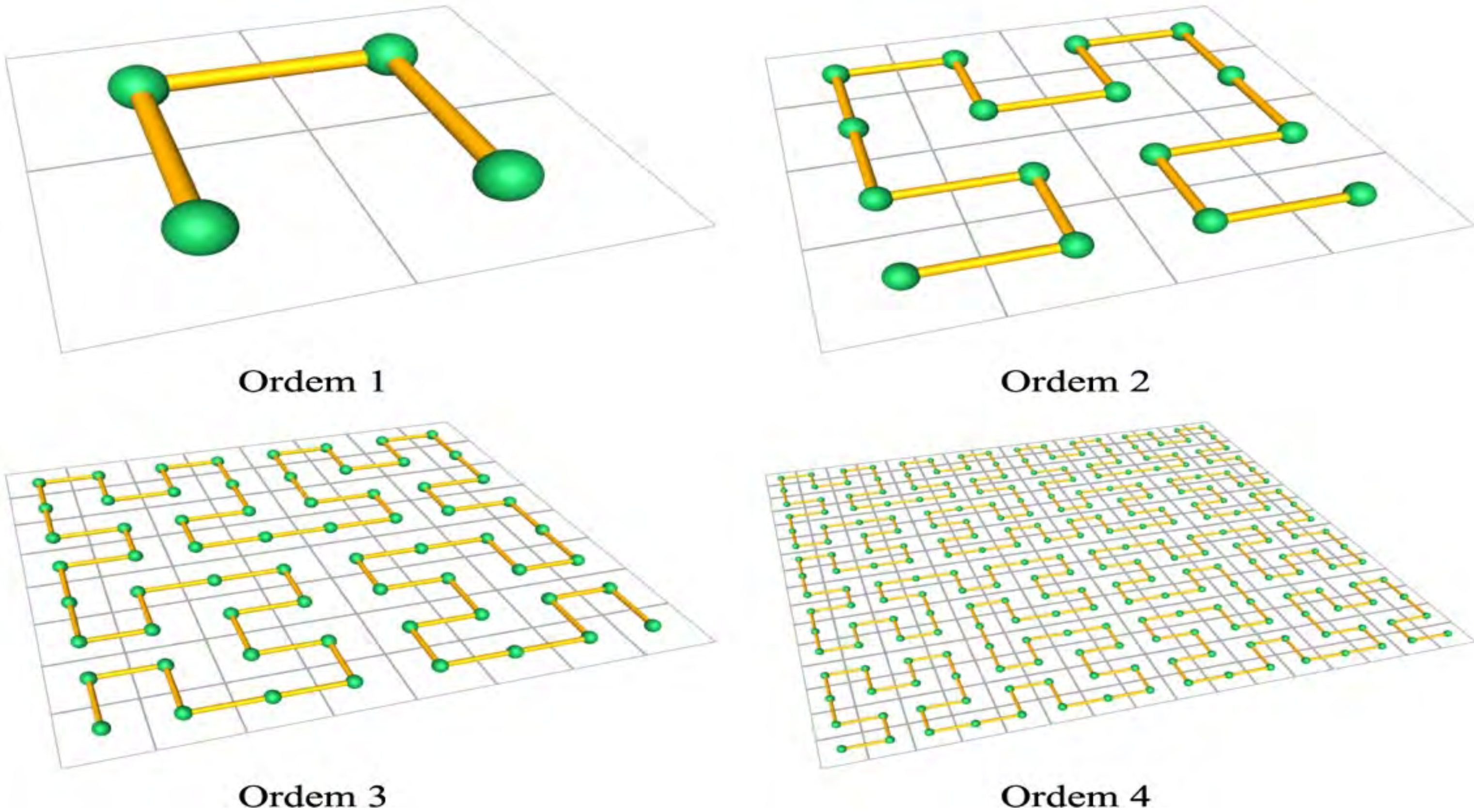


Organização de *pixels* num *array* no formato TGA (targa)




```
unsigned char *bgra_vector;
…
offset=4*(w*y+x);
blue  = bgra_vector[offset];
green = bgra_vector[offset+1];
red   = bgra_vector[offset+2];
alpha = bgra_vector[offset+3];
```

Outra ordem no plano



Tipo Abstrato Imagem

```
/*- implementação do tipo Imagem  */
struct image_imp {
  int width;    /* largura (width) em pixels   */
  int height;   /* altura (height) em pixels   */
  float  *buf;    /* buffer RGB               */
};
```

```
Image *imgCreate (int w, int h);
void   imgDestroy (Image *image);

int imgGetWidth(Image * image);
int imgGetHeight(Image * image);
float * imgGetRGBData(Image * image);

void imgSetPixel3fv(Image *image, int x, int y, float * color);
void imgSetPixel3ubv(Image *image, int x, int y, unsigned char *color);
void imgGetPixel3fv(Image *image, int x, int y, float *color);
void imgGetPixel3ubv(Image *image, int x, int y, unsigned char *color);

Image * imgReadBMP(char *filename);
int imgWriteBMP(char *filename, Image * image);

Image * imgCopy(Image * image);
Image * imgGrey(Image * image);
Image * imgResize(Image * img0, int w1, int h1);
```