Digital Signal Processing

Module 8: Image Processing

Module Overview:

- Module 8.1: Introduction to Images and Image Processing
- Module 8.2: Affine Transforms
- Module 8.3: 2D Fourier Analysis
- Module 8.4: Image Filters
- Module 8.5: Image Compression
- Module 8.6: The JPEG Compression Standard

Digital Signal Processing

Module 8.1: Image Processing

Overview:

- Images as multidimensional digital signals
- 2D signal representations
- Basic signals and operators

Overview:

- Images as multidimensional digital signals
- 2D signal representations
- Basic signals and operators

Overview:

- Images as multidimensional digital signals
- 2D signal representations
- Basic signals and operators

Please meet ...

Digital images

- two-dimensional signal $x[n_1, n_2]$, $n_1, n_2 \in \mathbb{Z}$
- indices locate a point on a grid $\rightarrow$ pixel
- grid is usually regularly spaced
- values $x[n_1, n_2]$ refer to the pixel's appearance

Digital images

- two-dimensional signal $x[n_1, n_2]$, $n_1, n_2 \in \mathbb{Z}$
- indices locate a point on a grid $\rightarrow$ pixel
- grid is usually regularly spaced
- values $x[n_1, n_2]$ refer to the pixel's appearance

Digital images

- two-dimensional signal $x[n_1, n_2]$, $n_1, n_2 \in \mathbb{Z}$
- indices locate a point on a grid $\rightarrow$ pixel
- grid is usually regularly spaced
- values $x[n_1, n_2]$ refer to the pixel's appearance

Digital images

- two-dimensional signal $x[n_1, n_2]$, $n_1, n_2 \in \mathbb{Z}$
- indices locate a point on a grid $\rightarrow$ pixel
- grid is usually regularly spaced
- values $x[n_1, n_2]$ refer to the pixel's appearance

Digital images: grayscale vs color

- grayscale images: scalar pixel values
- color images: multidimensional pixel values in a color space (RGB, HSV, YUV, etc)
- we can consider the single components separately:

Digital images: grayscale vs color

- grayscale images: scalar pixel values
- color images: multidimensional pixel values in a color space (RGB, HSV, YUV, etc)
- we can consider the single components separately:

Digital images: grayscale vs color

- grayscale images: scalar pixel values
- color images: multidimensional pixel values in a color space (RGB, HSV, YUV, etc)
- we can consider the single components separately:

 $=$  $+$  $+$ 

From one to two dimensions...

- something still works
- something breaks down
- something is new

Image processing

From one to two dimensions...

- something still works
- something breaks down
- something is new

From one to two dimensions...

- something still works
- something breaks down
- something is new

Image processing

What works:

- linearity, convolution
- Fourier transform
- interpolation, sampling

What works:

- linearity, convolution
- Fourier transform
- interpolation, sampling

What works:

- linearity, convolution
- Fourier transform
- interpolation, sampling

What breaks down:

- Fourier analysis less relevant
- filter design hard, IIRs rare
- linear operators only mildly useful

What breaks down:

- Fourier analysis less relevant
- filter design hard, IIRs rare
- linear operators only mildly useful

Image processing

What breaks down:

- Fourier analysis less relevant
- filter design hard, IIRs rare
- linear operators only mildly useful

What's new:

- new manipulations: affine transforms
- images are finite-support signals
- images are (most often) available in their entirety $\rightarrow$ causality loses meaning
- images are very specialized signals, designed for a very specific processing system, i.e. the human brain! Lots of semantics that is extremely hard to deal with

What's new:

- new manipulations: affine transforms
- images are finite-support signals
- images are (most often) available in their entirety $\rightarrow$ causality loses meaning
- images are very specialized signals, designed for a very specific processing system, i.e. the human brain! Lots of semantics that is extremely hard to deal with

Image processing

What's new:

- new manipulations: affine transforms
- images are finite-support signals
- images are (most often) available in their entirety $\rightarrow$ causality loses meaning
- images are very specialized signals, designed for a very specific processing system, i.e. the human brain! Lots of semantics that is extremely hard to deal with

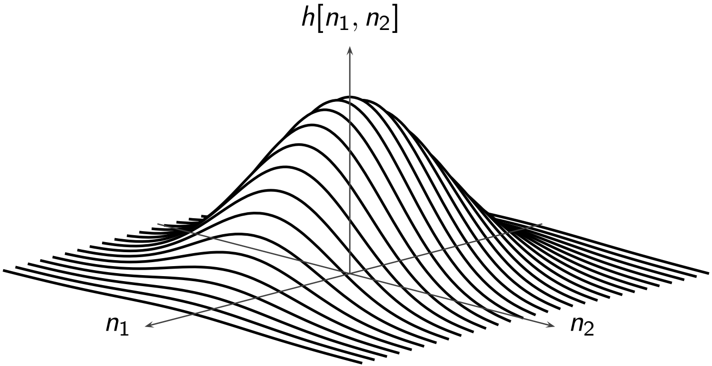
What's new:

- new manipulations: affine transforms
- images are finite-support signals
- images are (most often) available in their entirety $\rightarrow$ causality loses meaning
- images are very specialized signals, designed for a very specific processing system, i.e. the human brain! Lots of semantics that is extremely hard to deal with

A two-dimensional discrete-space signal:

$$x[n_1, n_2], \qquad n_1, n_2 \in \mathbb{Z}$$

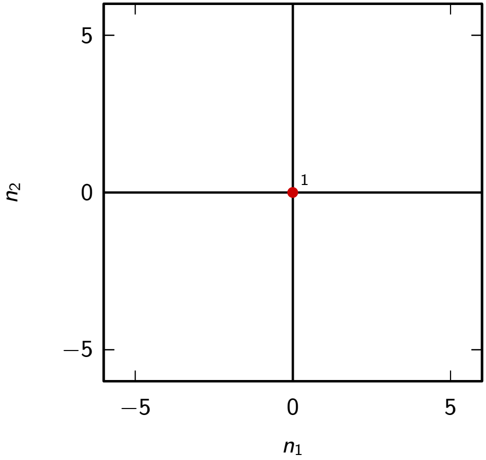
2D signals: Cartesian representation



2D signals: support representation

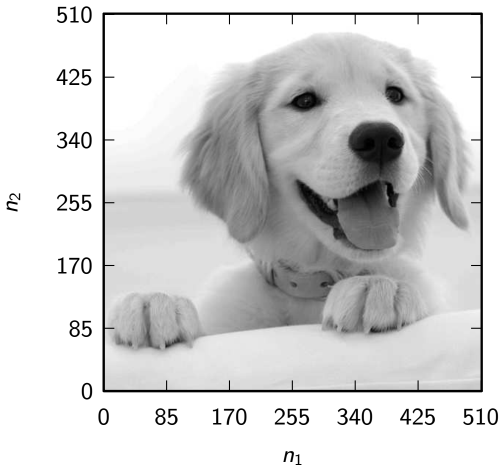
- just show coordinates of nonzero samples
- amplitude may be written along explicitly
- example:

$$\delta[n_1, n_2] = \begin{cases} 1 & \text{if } n_1 = n_2 = 0 \\ 0 & \text{otherwise.} \end{cases}$$



2D signals: image representation

- medium has a certain dynamic range (paper, screen)
- image values are quantized (usually to 8 bits, or 256 levels)
- the eye does the interpolation in space provided the pixel density is high enough



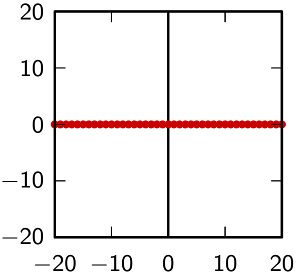
Why 2D?

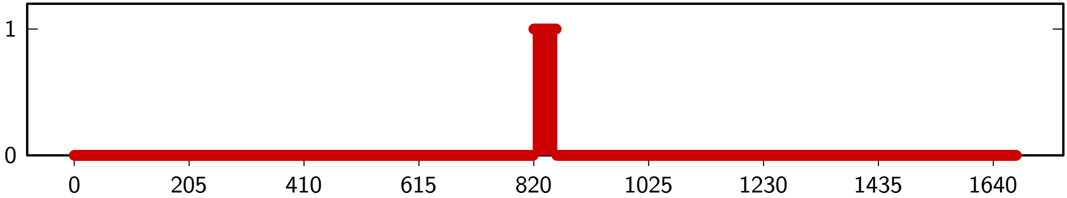
- images could be unrolled (printers, fax)
- but what about spatial correlation?

Why 2D?

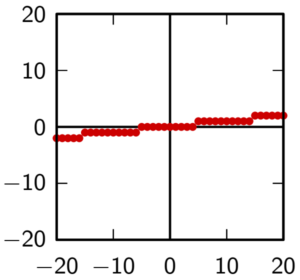
- images could be unrolled (printers, fax)
- but what about spatial correlation?

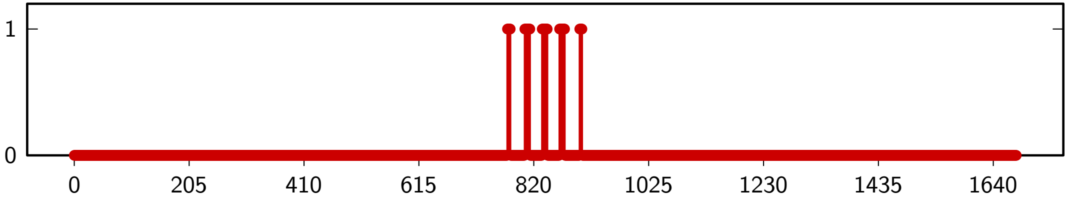
2D vs raster scan



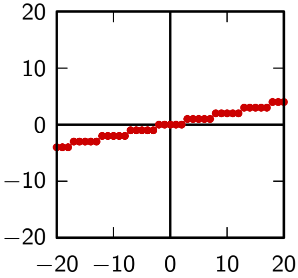


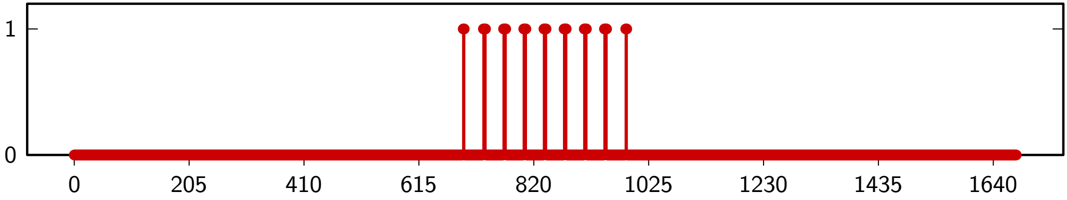
2D vs raster scan



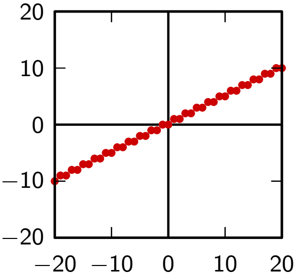


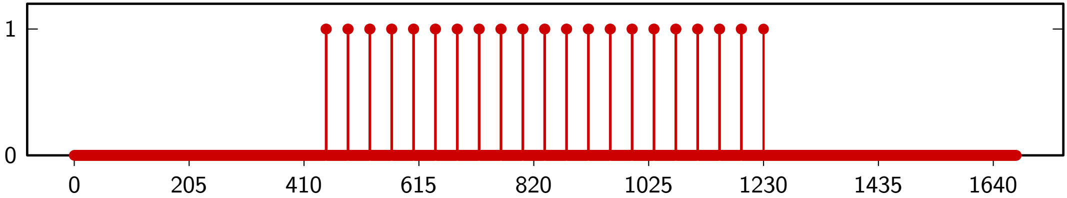
2D vs raster scan



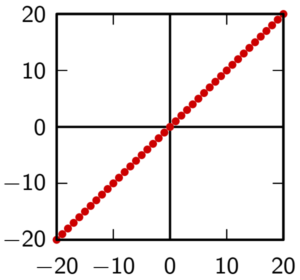


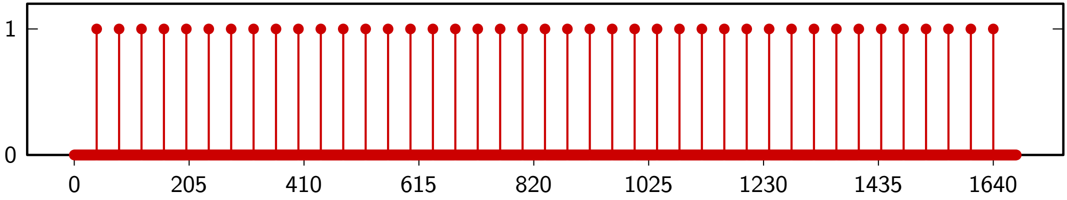
2D vs raster scan



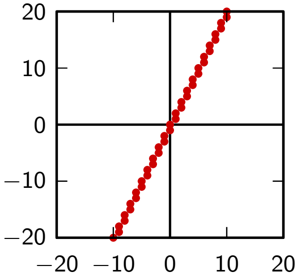


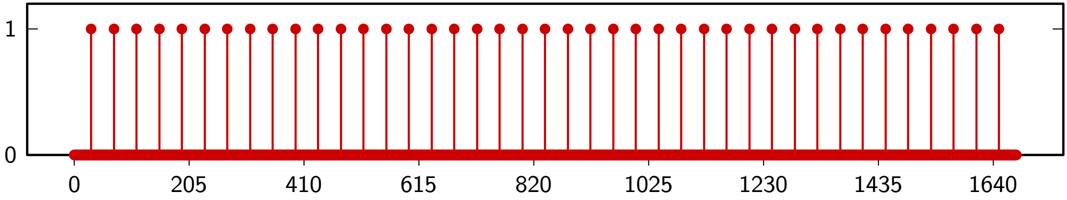
2D vs raster scan



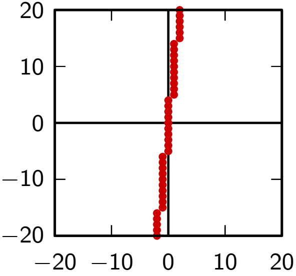


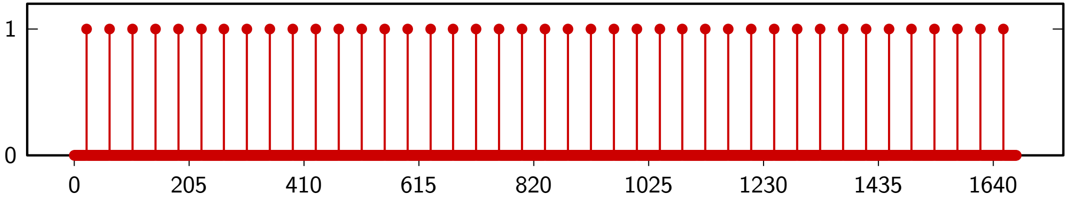
2D vs raster scan



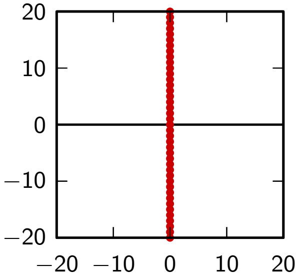


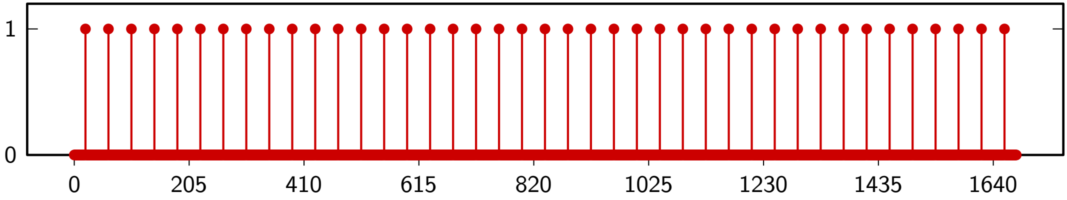
2D vs raster scan





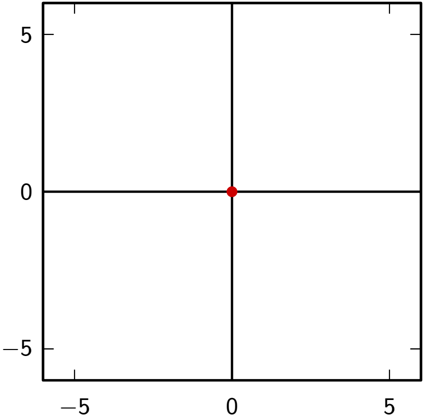
2D vs raster scan





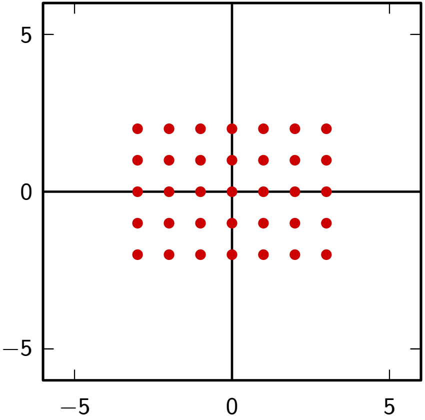
Basic 2D signals: the impulse

$$\delta[n_1, n_2] = \begin{cases} 1 & \text{if } n_1 = n_2 = 0 \\ 0 & \text{otherwise.} \end{cases}$$



Basic 2D signals: the rect

$$\text{rect}\left(\frac{n_1}{2N_1}, \frac{n_2}{2N_2}\right) = \begin{cases} 1 & \text{if } |n_1| < N_1 \\ & \text{and } |n_2| < N_2 \\ \\ 0 & \text{otherwise;} \end{cases}$$



Separability

$$x[n_1, n_2] = x_1[n_1]x_2[n_2]$$

Separable signals

$$\delta[n_1, n_2] = \delta[n_1]\delta[n_2]$$

$$\operatorname{rect}\left(\frac{n_1}{2N_1}, \frac{n_2}{2N_2}\right) = \operatorname{rect}\left(\frac{n_1}{2N_1}\right)\operatorname{rect}\left(\frac{n_2}{2N_2}\right).$$

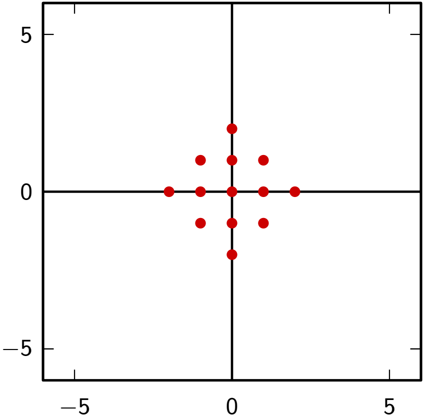
Separable signals

$$\delta[n_1, n_2] = \delta[n_1]\delta[n_2]$$

$$\mathrm{rect}\left(\frac{n_1}{2N_1}, \frac{n_2}{2N_2}\right) = \mathrm{rect}\left(\frac{n_1}{2N_1}\right)\mathrm{rect}\left(\frac{n_2}{2N_2}\right).$$

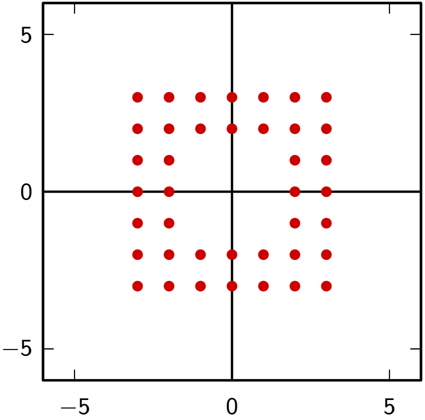
Nonseparable signal

$$x[n_1, n_2] = \begin{cases} 1 & \text{if } |n_1| + |n_2| < N \\ 0 & \text{otherwise} \end{cases}$$



Nonseparable signal

$$x[n_1, n_2] = \mathrm{rect}\left(\frac{n_1}{2N_1}, \frac{n_2}{2N_2}\right) - \mathrm{rect}\left(\frac{n_1}{2M_1}, \frac{n_2}{2M_2}\right)$$



2D convolution

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

2D convolution for separable signals

If $h[n_1, n_2] = h_1[n_1]h_2[n_2]$:

$$x[n_1, n_2] * h[n_1, n_2] = \sum_{k_1=-\infty}^{\infty} h_1[n_1 - k_1] \sum_{k_2=-\infty}^{\infty} x[k_1, k_2]h_2[n_2 - k_2]$$

$$= h_1[n_1] * (h_2[n_2] * x[n_1, n_2]).$$

2D convolution for separable signals

If $h[n_1, n_2]$ is an $M_1 \times M_2$ finite-support signal:

- non-separable convolution: $M_1 M_2$ operations per output sample
- separable convolution: $M_1 + M_2$ operations per output sample!

END OF MODULE 8.1

Digital Signal Processing

Module 8.2: Image Manipulations

Overview:

- Affine transforms
- Bilinear interpolation

Overview:

- Affine transforms
- Bilinear interpolation

Affine transforms

mapping $\mathbb{R}^2 \to \mathbb{R}^2$ that reshapes the coordinate system:

$$\begin{bmatrix} t_1' \\ t_2' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} - \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

$$\begin{bmatrix} t_1' \\ t_2' \end{bmatrix} = \mathbf{A} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} - \mathbf{d}$$

Affine transforms

mapping $\mathbb{R}^2 \rightarrow \mathbb{R}^2$ that reshapes the coordinate system:

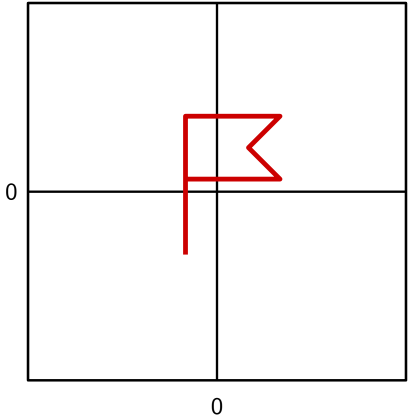
$$\begin{bmatrix} t_1' \\ t_2' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} - \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$

$$\begin{bmatrix} t_1' \\ t_2' \end{bmatrix} = \mathbf{A} \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} - \mathbf{d}$$

Translation

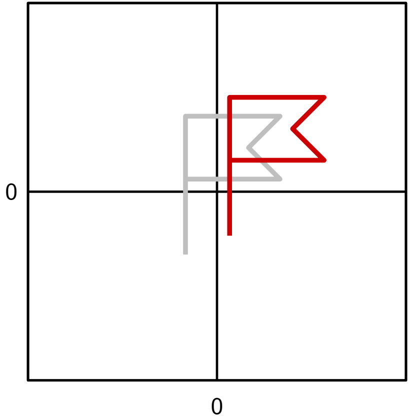
$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}$$

$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$



$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{I}$$

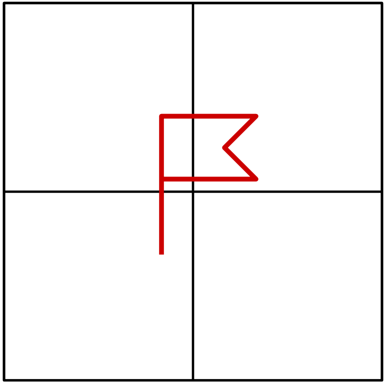
$$\mathbf{d} = \begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$$



$$\mathbf{A} = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \end{bmatrix}$$

$$\mathbf{d} = 0$$

if $a_1 = a_2$ the *aspect ratio* is preserved

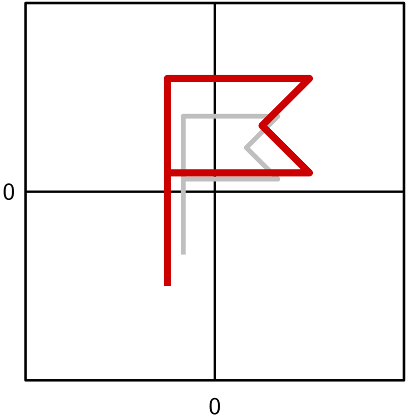


Scaling

$$\mathbf{A} = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \end{bmatrix}$$

$$\mathbf{d} = 0$$

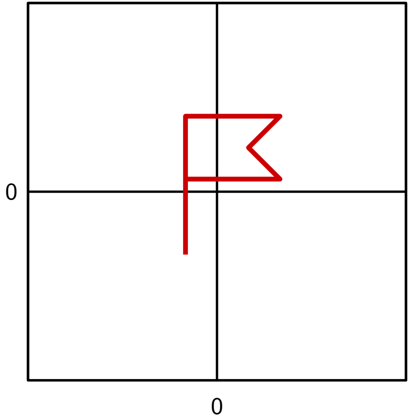
if $a_1 = a_2$ the *aspect ratio* is preserved



Rotation

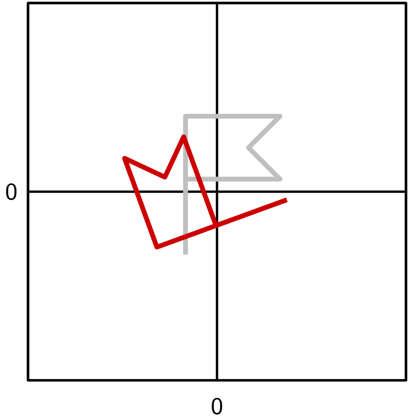
$$\mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{d} = 0$$



$$\mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

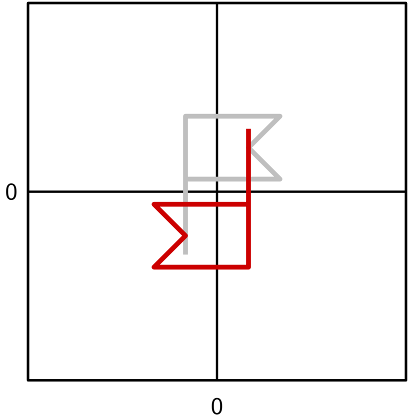
$$\mathbf{d} = 0$$



Rotation

$$\mathbf{A} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

$$\mathbf{d} = 0$$



Flips

Horizontal:

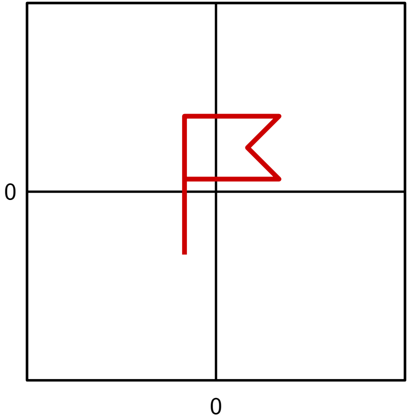
$$\mathbf{A} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{d} = 0$$

Vertical:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\mathbf{d} = 0$$



Horizontal:

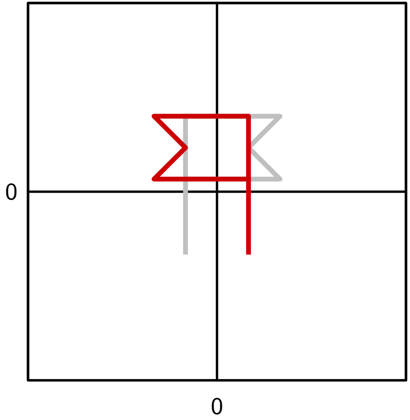
$$\mathbf{A} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{d} = 0$$

Vertical:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

$$\mathbf{d} = 0$$



Shear

Horizontal:

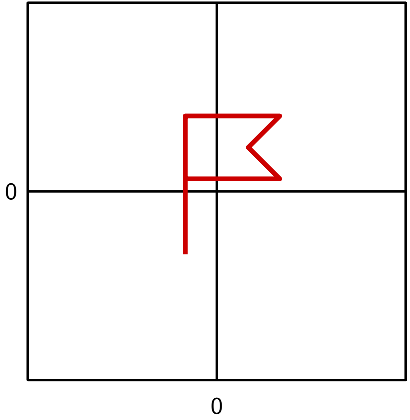
$$\mathbf{A} = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{d} = 0$$

Vertical:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}$$

$$\mathbf{d} = 0$$



Shear

Horizontal:

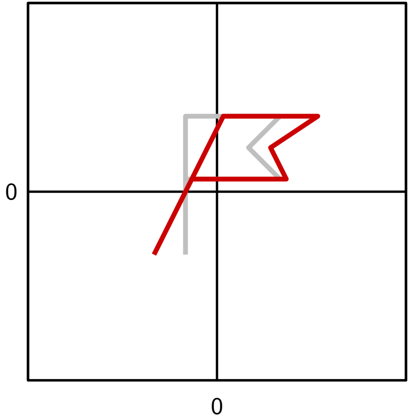
$$\mathbf{A} = \begin{bmatrix} 1 & s \\ 0 & 1 \end{bmatrix}$$

$$\mathbf{d} = 0$$

Vertical:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ s & 1 \end{bmatrix}$$

$$\mathbf{d} = 0$$



Affine transforms in discrete-space

$$\begin{bmatrix} t_1' \\ t_2' \end{bmatrix} = \mathbf{A} \begin{bmatrix} n_1 \\ n_2 \end{bmatrix} - \mathbf{d} \quad \in \mathbb{R}^2 \neq \mathbb{Z}^2$$

- apply the *inverse* transform:

$$\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \mathbf{A}^{-1} \begin{bmatrix} m_1 + d_1 \\ m_2 + d_2 \end{bmatrix};$$

- interpolate from the original grid point to the "mid-point"

$$(t_1, t_2) = (\eta_1 + \tau_1, \eta_2 + \tau_2), \qquad \eta_{1,2} \in \mathbb{Z}, \quad 0 \leq \tau_{1,2} < 1$$

Solution for images

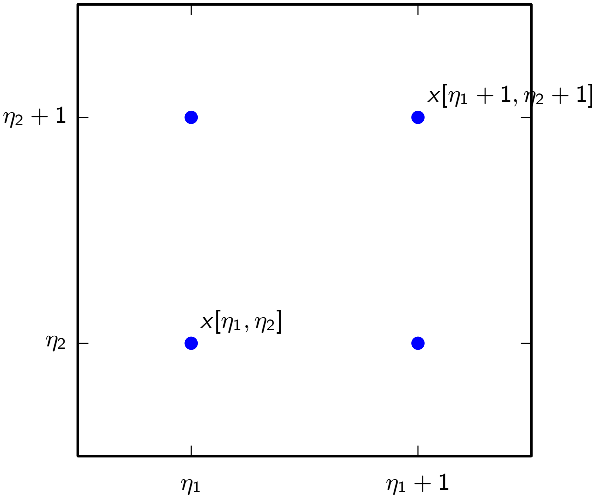
- apply the *inverse* transform:

$$\begin{bmatrix} t_1 \\ t_2 \end{bmatrix} = \mathbf{A}^{-1} \begin{bmatrix} m_1 + d_1 \\ m_2 + d_2 \end{bmatrix};$$

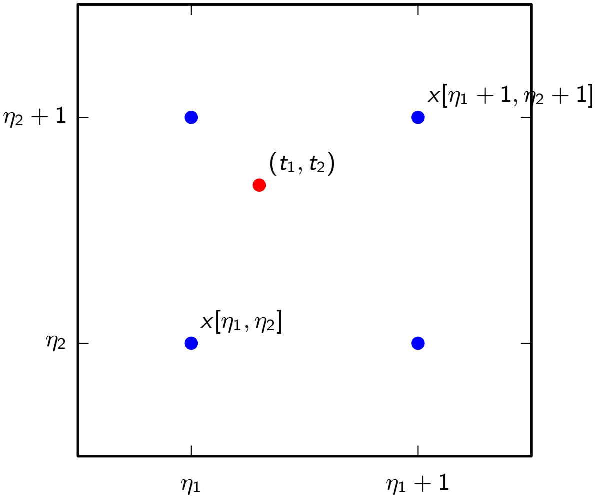
- interpolate from the original grid point to the "mid-point"

$$(t_1, t_2) = (\eta_1 + \tau_1, \eta_2 + \tau_2), \qquad \eta_{1,2} \in \mathbb{Z}, \quad 0 \leq \tau_{1,2} < 1$$

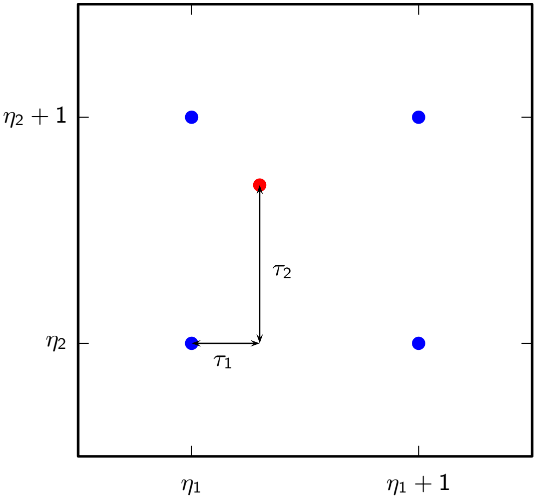
Bilinear Interpolation



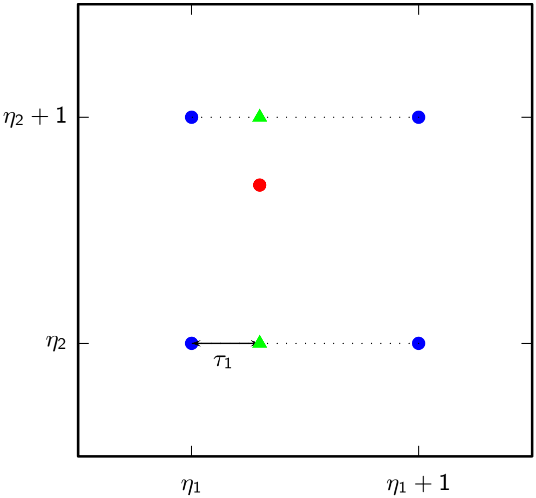
Bilinear Interpolation



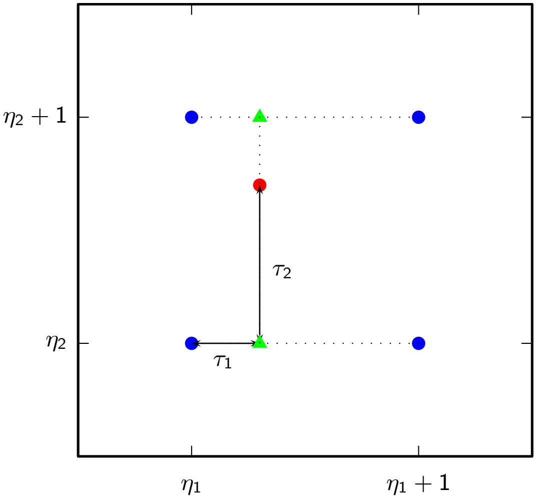
Bilinear Interpolation



Bilinear Interpolation



Bilinear Interpolation



Bilinear Interpolation

If we use a first-order interpolator:

$$\begin{aligned} y[m_1, m_2] = (1-\tau_1)(1-\tau_2)x[\eta_1, \eta_2] + \tau_1(1-\tau_2)x[\eta_1+1, \eta_2] \\ + (1-\tau_1)\tau_2 x[\eta_1, \eta_2+1] + \tau_1\tau_2 x[\eta_1+1, \eta_2+1] \end{aligned}$$

Shearing

END OF MODULE 8.2

Digital Signal Processing

Module 8.3: Frequency Analysis

Overview:

- DFT
- Magnitude and phase

Overview:

- DFT
- Magnitude and phase

2D DFT

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_1}n_1 k_1} e^{-j\frac{2\pi}{N_2}n_2 k_2}$$

$$x[n_1, n_2] = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X[k_1, k_2] e^{j\frac{2\pi}{N_1}n_1 k_1} e^{j\frac{2\pi}{N_2}n_2 k_2}$$

2D DFT

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_1} n_1 k_1} e^{-j\frac{2\pi}{N_2} n_2 k_2}$$

$$x[n_1, n_2] = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X[k_1, k_2] e^{j\frac{2\pi}{N_1} n_1 k_1} e^{j\frac{2\pi}{N_2} n_2 k_2}$$

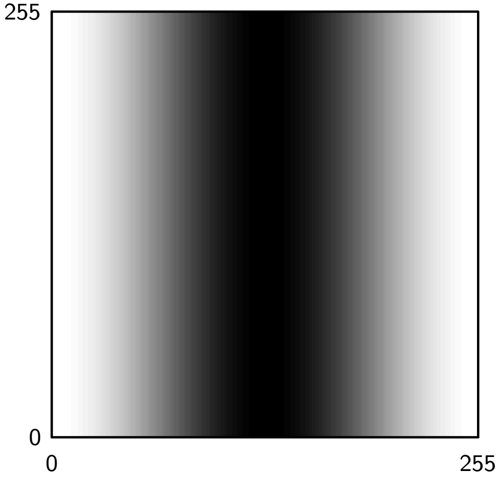
2D-DFT Basis Vectors

There are $N_1 N_2$ orthogonal basis vectors for an $N_1 \times N_2$ image:

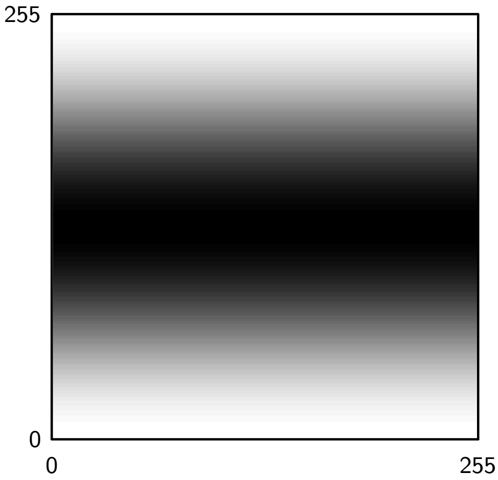
$$w_{k_1,k_2}[n_1, n_2] = e^{j\frac{2\pi}{N_1}n_1 k_1} e^{j\frac{2\pi}{N_2}n_2 k_2}$$

for $n_1, k_1 = 0, 1, \ldots, N_1 - 1$ and $n_2, k_2 = 0, 1, \ldots, N_2 - 1$

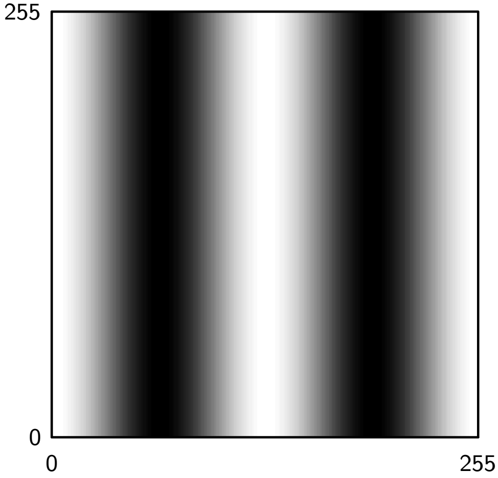
2D-DFT basis vectors for $k_1 = 1, k_2 = 0$ (real part)



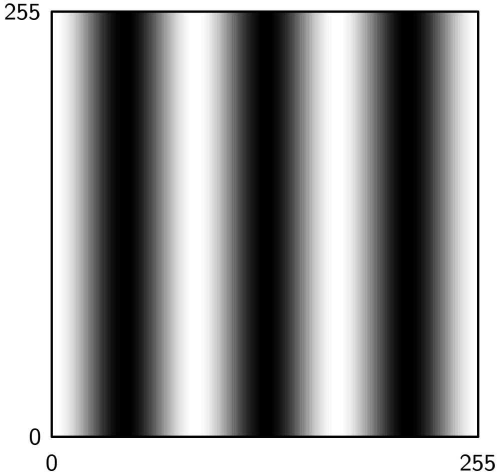
2D-DFT basis vectors for $k_1 = 0, k_2 = 1$ (real part)



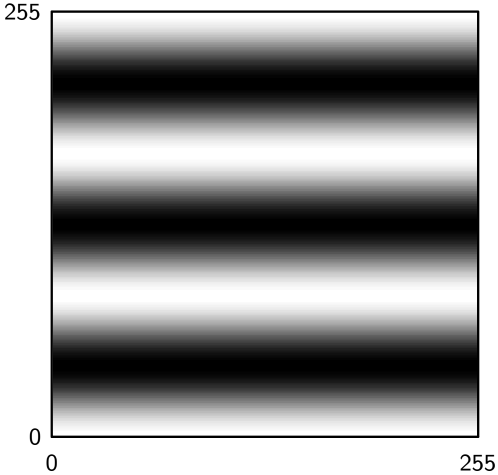
2D-DFT basis vectors for $k_1 = 2, k_2 = 0$ (real part)



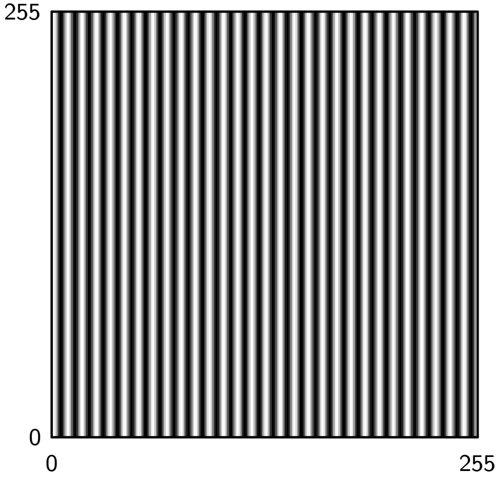
2D-DFT basis vectors for $k_1 = 3, k_2 = 0$ (real part)



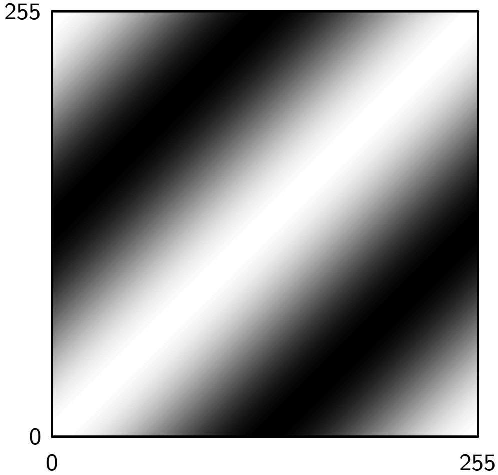
2D-DFT basis vectors for $k_1 = 0, k_2 = 3$ (real part)



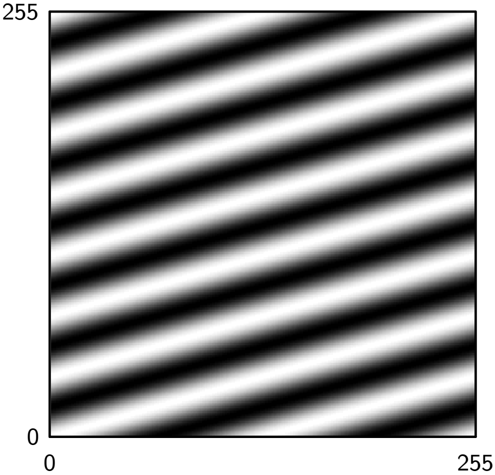
2D-DFT basis vectors for $k_1 = 30, k_2 = 0$ (real part)



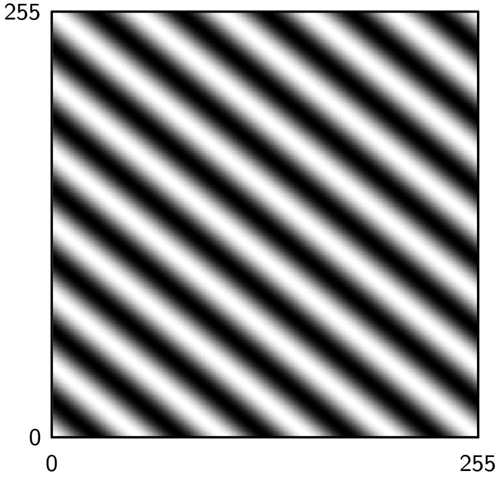
2D-DFT basis vectors for $k_1 = 1, k_2 = 1$ (real part)



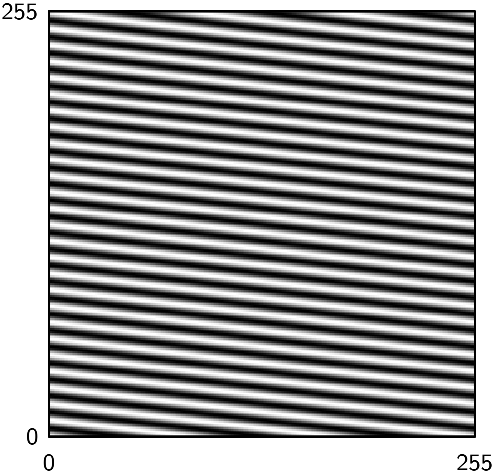
2D-DFT basis vectors for $k_1 = 2, k_2 = 7$ (real part)



2D-DFT basis vectors for $k_1 = 5, k_2 = 250$ (real part)



2D-DFT basis vectors for $k_1 = 3, k_2 = 230$ (real part)



2D DFT

2D-DFT basis functions are separable, and so is the 2D-DFT:

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_1}n_1k_1} e^{-j\frac{2\pi}{N_2}n_2k_2}$$

- 1D-DFT along n_2 (the columns)
- 1D-DFT along n_1 (the rows)

2D-DFT basis functions are separable, and so is the 2D-DFT:

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_2}n_2 k_2} \right] e^{-j\frac{2\pi}{N_1}n_1 k_1}$$

- 1D-DFT along n_2 (the columns)
- 1D-DFT along n_1 (the rows)

2D-DFT basis functions are separable, and so is the 2D-DFT:

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \left[\boxed{\sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_2} n_2 k_2}} \right] e^{-j\frac{2\pi}{N_1} n_1 k_1}$$

- 1D-DFT along n_2 (the columns)
- 1D-DFT along n_1 (the rows)

2D-DFT basis functions are separable, and so is the 2D-DFT:

$$X[k_1, k_2] = \boxed{\sum_{n_1=0}^{N_1-1} \left[\boxed{\sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_2} n_2 k_2}} \right] e^{-j\frac{2\pi}{N_1} n_1 k_1}}$$

- 1D-DFT along n_2 (the columns)
- 1D-DFT along n_1 (the rows)

2D DFT in matrix form

- finite-support 2D signal can be written as a matrix $\mathbf{x}$
- $N_1 \times N_2$ image is an $N_2 \times N_1$ matrix (n_1 spans the columns, n_2 spans the rows)
- recall also the $N \times N$ DFT matrix (Module 4.2):

$$\mathbf{W}_N = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W_N^1 & W_N^2 & W^3 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W^4 & W_N^6 & \dots & W_N^{2(N-1)} \\ & & & \dots & & \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & W_N^{3(N-1)} & \dots & W_N^{(N-1)^2} \end{bmatrix}$$

2D DFT in matrix form

- finite-support 2D signal can be written as a matrix $\mathbf{x}$
- $N_1 \times N_2$ image is an $N_2 \times N_1$ matrix (n_1 spans the columns, n_2 spans the rows)
- recall also the $N \times N$ DFT matrix (Module 4.2):

$$\mathbf{W}_N = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W_N^1 & W_N^2 & W^3 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W^4 & W_N^6 & \dots & W_N^{2(N-1)} \\ & & & \dots & & \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & W_N^{3(N-1)} & \dots & W_N^{(N-1)^2} \end{bmatrix}$$

2D DFT in matrix form

- finite-support 2D signal can be written as a matrix $\mathbf{x}$
- $N_1 \times N_2$ image is an $N_2 \times N_1$ matrix (n_1 spans the columns, n_2 spans the rows)
- recall also the $N \times N$ DFT matrix (Module 4.2):

$$\mathbf{W}_N = \begin{bmatrix} 1 & 1 & 1 & 1 & \dots & 1 \\ 1 & W_N^1 & W_N^2 & W^3 & \dots & W_N^{N-1} \\ 1 & W_N^2 & W^4 & W_N^6 & \dots & W_N^{2(N-1)} \\ & & & \dots & & \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & W_N^{3(N-1)} & \dots & W_N^{(N-1)^2} \end{bmatrix}$$

2D DFT in matrix form

$$X[k_1, k_2] = \boxed{\sum_{n_1=0}^{N_1-1} \left[\boxed{\sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_2} n_2 k_2}} \right] e^{-j\frac{2\pi}{N_1} n_1 k_1}}$$

$$X[k_1, k_2] = \boxed{\sum_{n_1=0}^{N_1-1} \left[\boxed{\sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_2} n_2 k_2}} \right] e^{-j\frac{2\pi}{N_1} n_1 k_1}}$$

$$\mathbf{V} = \mathbf{W}_{N_2}\, \mathbf{x}$$

$$\mathbf{V} \in \mathbb{C}^{N_2 \times N_1}$$

$$X[k_1, k_2] = \sum_{n_1=0}^{N_1-1} \left[\sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_2} n_2 k_2} \right] e^{-j\frac{2\pi}{N_1} n_1 k_1}$$

$$\mathbf{V} = \mathbf{W}_{N_2}\, \mathbf{x}$$

$$\mathbf{V} \in \mathbb{C}^{N_2 \times N_1}$$

$$\mathbf{X} = \mathbf{V}\, \mathbf{W}_{N_1}$$

$$\mathbf{X} \in \mathbb{C}^{N_2 \times N_1}$$

2D DFT in matrix form

$$X[k_1, k_2] = \boxed{\sum_{n_1=0}^{N_1-1} \left[\boxed{\sum_{n_2=0}^{N_2-1} x[n_1, n_2] e^{-j\frac{2\pi}{N_2} n_2 k_2}} \right] e^{-j\frac{2\pi}{N_1} n_1 k_1}}$$

$$\mathbf{V} = \mathbf{W}_{N_2}\,\mathbf{x}$$

$$\mathbf{V} \in \mathbb{C}^{N_2 \times N_1}$$

$$\mathbf{X} = \mathbf{V}\,\mathbf{W}_{N_1}$$

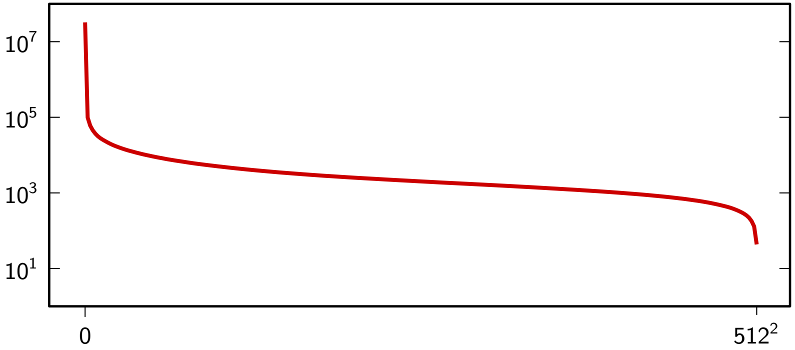
$$\mathbf{X} \in \mathbb{C}^{N_2 \times N_1}$$

$$\mathbf{X} = \mathbf{W}_{N_2}\,\mathbf{x}\,\mathbf{W}_{N_1}$$

How does a 2D-DFT look like?

- try to show the magnitude as an image
- problem: the range is too big for the grayscale range of paper or screen
- try to normalize: $|X'[n_1, n_2]| = |X[n_1, n_2]| / \max |X[n_1, n_2]|$
- but it doesn't work...

DFT coefficients sorted by magnitude



Dealing with HDR images

if the image is high dynamic range we need to compress the levels

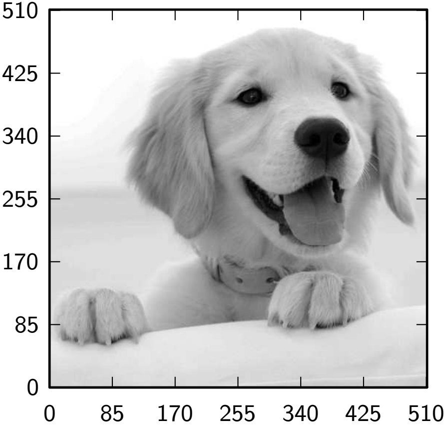
- remove flagrant outliers (e.g. $X[0,0] = \sum\sum x[n_1, n_2]$)
- use a nonlinear mapping: e.g. $y = x^{1/3}$ after normalization $(x \leq 1)$

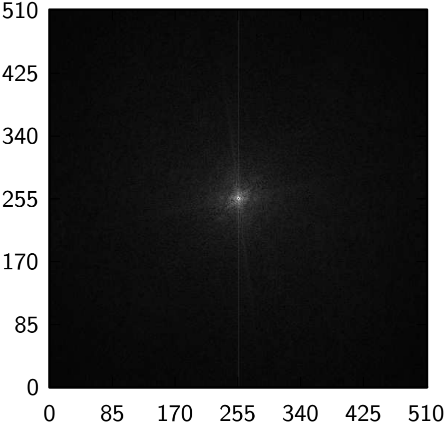
Dealing with HDR images

if the image is high dynamic range we need to compress the levels

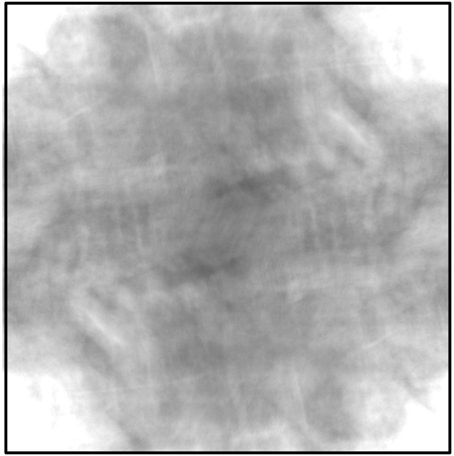
- remove flagrant outliers (e.g. $X[0,0] = \sum\sum x[n_1, n_2]$)
- use a nonlinear mapping: e.g. $y = x^{1/3}$ after normalization ($x \leq 1$)

How does a 2D-DFT look like?





DFT magnitude doesn't carry much information



DFT phase, on the other hand...

Image frequency analysis

- most of the information is contained in image's *edges*
- edges are points of abrupt change in signal's values
- edges are a "space-domain" feature $\rightarrow$ not captured by DFT's magnitude
- phase alignment is important for reproducing edges

END OF MODULE 8.3

Digital Signal Processing

Module 8.4: Filtering

Overview:

- Filters for image processing
- Classification
- Examples

Overview:

- Filters for image processing
- Classification
- Examples

Overview:

- Filters for image processing
- Classification
- Examples

Analogies with 1D filters

- linearity
- *space* invariance
- impulse response
- frequency response
- stability
- 2D CCDE

The problem with LSI operators

- interesting images contain lots of *semantics*: different information in different areas
- space-invariant filters process everything in the same way
- but we should process things differently
  - edges
  - gradients
  - textures
  - ...

The problem with LSI operators

- interesting images contain lots of *semantics*: different information in different areas
- space-invariant filters process everything in the same way
- but we should process things differently
  - edges
  - gradients
  - textures
  - ...

The problem with LSI operators

- interesting images contain lots of *semantics*: different information in different areas
- space-invariant filters process everything in the same way
- but we should process things differently
  - edges
  - gradients
  - textures
  - ...

The problem with LSI operators

- interesting images contain lots of *semantics*: different information in different areas
- space-invariant filters process everything in the same way
- but we should process things differently
  - edges
  - gradients
  - textures
  - ...

The problem with LSI operators

- interesting images contain lots of *semantics*: different information in different areas
- space-invariant filters process everything in the same way
- but we should process things differently
  - edges
  - gradients
  - textures
  - ...

The problem with LSI operators

- interesting images contain lots of *semantics*: different information in different areas
- space-invariant filters process everything in the same way
- but we should process things differently
  - edges
  - gradients
  - textures
  - ...

The problem with LSI operators

- interesting images contain lots of *semantics*: different information in different areas
- space-invariant filters process everything in the same way
- but we should process things differently
  - edges
  - gradients
  - textures
  - ...

Filter types

- IIR, FIR
- causal or noncausal
- highpass, lowpass, ...
  - lowpass → image smoothing
  - highpass → enhancement, edge detection

Filter types

- IIR, FIR
- causal or noncausal
- highpass, lowpass, ...
  - lowpass $\rightarrow$ image smoothing
  - highpass $\rightarrow$ enhancement, edge detection

Filter types

- IIR, FIR
- causal or noncausal
- highpass, lowpass, ...
  - lowpass $\longrightarrow$ image smoothing
  - highpass $\longrightarrow$ enhancement, edge detection

Filter types

- IIR, FIR
- causal or noncausal
- highpass, lowpass, ...
  - lowpass $\rightarrow$ image smoothing
  - highpass $\rightarrow$ enhancement, edge detection

Filter types

- IIR, FIR
- causal or noncausal
- highpass, lowpass, ...
  - lowpass $\rightarrow$ image smoothing
  - highpass $\rightarrow$ enhancement, edge detection

The problems with 2D IIRs

- nonlinear phase (edges!)
- border effects
- stability: the fundamental theorem of algebra doesn't hold in multiple dimensions!
- computability

The problems with 2D IIRs

- nonlinear phase (edges!)
- border effects
- stability: the fundamental theorem of algebra doesn't hold in multiple dimensions!
- computability

The problems with 2D IIRs

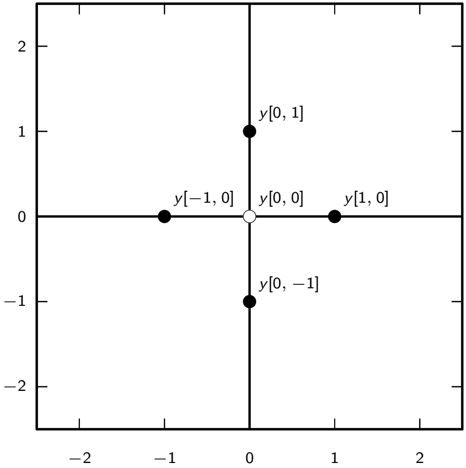
- nonlinear phase (edges!)
- border effects
- stability: the fundamental theorem of algebra doesn't hold in multiple dimensions!
- computability

The problems with 2D IIRs

- nonlinear phase (edges!)
- border effects
- stability: the fundamental theorem of algebra doesn't hold in multiple dimensions!
- computability

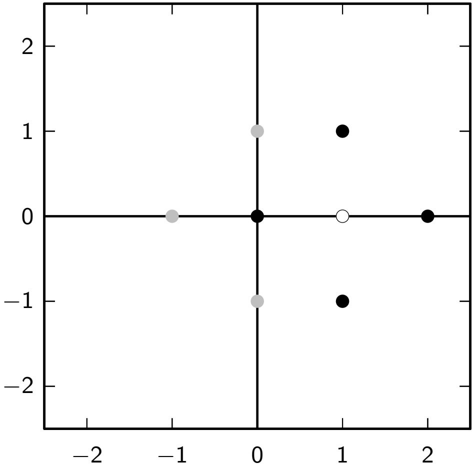
A noncomputable CCDE

$$y[n_1, n_2] = a_0 y[n_1 + 1, n_2] + a_1 y[n_1, n_2 - 1] + a_2 y[n_1 - 1, n_2] + a_3 y[n_1, n_2 + 1] + x[n_1, n_2];$$



A noncomputable CCDE

$$y[n_1, n_2] = a_0 y[n_1 + 1, n_2] + a_1 y[n_1, n_2 - 1] + a_2 y[n_1 - 1, n_2] + a_3 y[n_1, n_2 + 1] + x[n_1, n_2];$$



Practical FIR filters

- generally zero centered (causality not an issue) $\Rightarrow$ odd number of taps in both directions
- per-sample complexity:
  - $M_1 M_2$ for nonseparable impulse responses
  - $M_1 + M_2$ for separable impulse responses
- obviously always stable

Practical FIR filters

- generally zero centered (causality not an issue) $\Rightarrow$ odd number of taps in both directions
- per-sample complexity:
  - $M_1 M_2$ for nonseparable impulse responses
  - $M_1 + M_2$ for separable impulse responses
- obviously always stable

Practical FIR filters

- generally zero centered (causality not an issue) $\Rightarrow$ odd number of taps in both directions
- per-sample complexity:
  - $M_1 M_2$ for nonseparable impulse responses
  - $M_1 + M_2$ for separable impulse responses
- obviously always stable

Practical FIR filters

- generally zero centered (causality not an issue) $\Rightarrow$ odd number of taps in both directions
- per-sample complexity:
  - $M_1 M_2$ for nonseparable impulse responses
  - $M_1 + M_2$ for separable impulse responses
- obviously always stable

Practical FIR filters

- generally zero centered (causality not an issue) $\Rightarrow$ odd number of taps in both directions
- per-sample complexity:
  - $M_1 M_2$ for nonseparable impulse responses
  - $M_1 + M_2$ for separable impulse responses
- obviously always stable

Moving Average

$$y[n_1, n_2] = \frac{1}{(2N+1)^2} \sum_{k_1=-N}^{N} \sum_{k_2=-N}^{N} x[n_1 - k_1, n_2 - k_2]$$

$$h[n_1, n_2] = \frac{1}{(2N+1)^2} \mathrm{rect}\left(\frac{n_1}{2N}, \frac{n_2}{2N}\right)$$

$$y[n_1, n_2] = \frac{1}{(2N+1)^2} \sum_{k_1=-N}^{N} \sum_{k_2=-N}^{N} x[n_1 - k_1, n_2 - k_2]$$

$$h[n_1, n_2] = \frac{1}{(2N+1)^2} \mathrm{rect}\left(\frac{n_1}{2N}, \frac{n_2}{2N}\right)$$

Moving Average

$$h[n_1, n_2] = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Moving Average

11×11 MA

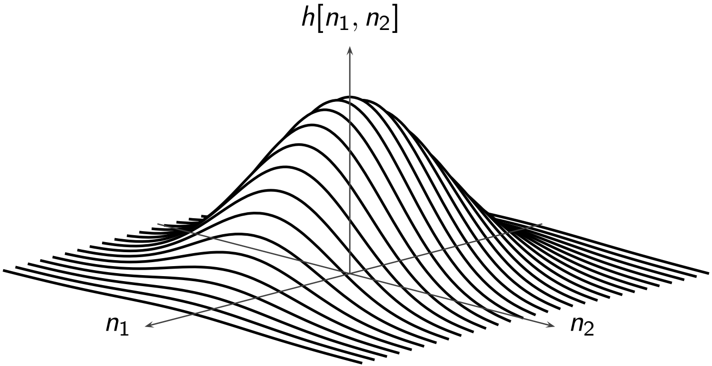
51×51 MA

Gaussian Blur

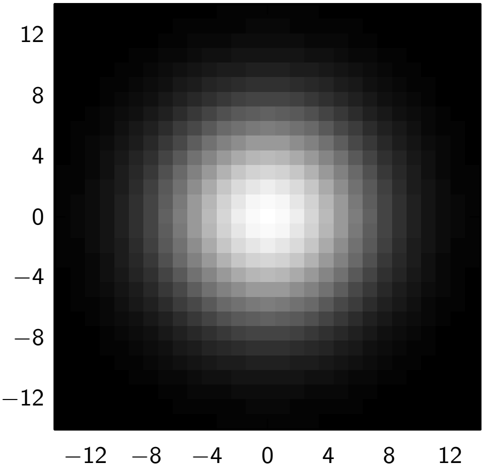
$$h[n_1, n_2] = \frac{1}{2\pi\sigma^2} e^{-\frac{n_1^2+n_2^2}{2\sigma^2}}, \qquad |n_1, n_2| < N$$

with $N \approx 3\sigma$

Gaussian Blur



Gaussian Blur



$\sigma = 5, N = 14$

$\sigma = 1.8, 11 \times 11$ blur

$\sigma = 8.7, 51 \times 51$ blur

Sobel filter

approximation of the first derivative in the horizontal direction:

$$s_o[n_1, n_2] = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

separability and structure:

$$s_o[n_1, n_2] = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Sobel filter

approximation of the first derivative in the horizontal direction:

$$s_o[n_1, n_2] = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

separability and structure:

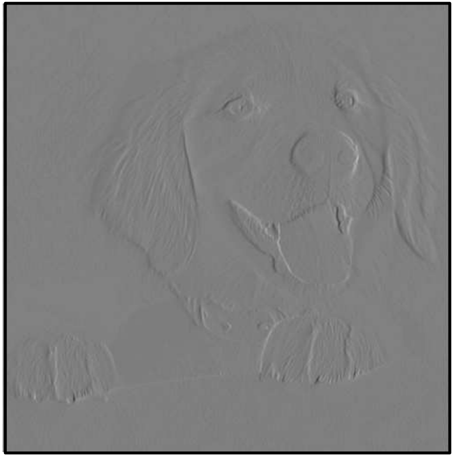
$$s_o[n_1, n_2] = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Sobel filter

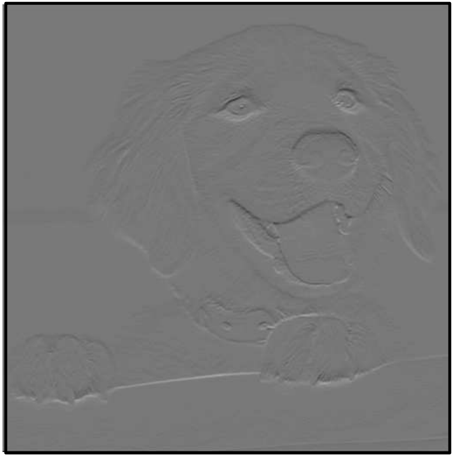
approximation of the first derivative in the vertical direction:

$$s_v[n_1, n_2] = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Sobel filter



horizontal Sobel filter



vertical Sobel filter

Sobel operator

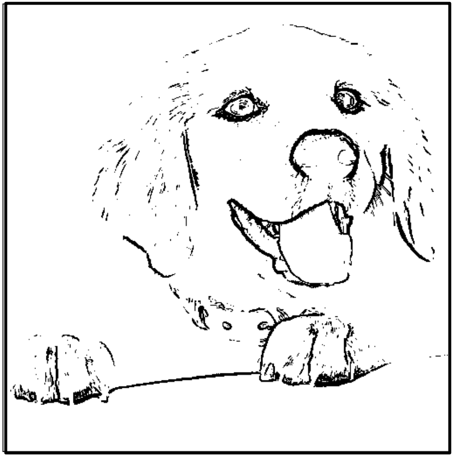
approximation for the square magnitude of the gradient:

$$|\nabla x[n_1, n_2]|^2 = |s_o[n_1, n_2] * x[n_1, n_2]|^2 + |s_v[n_1, n_2] * x[n_1, n_2]|^2$$

("operator" because it's nonlinear)

Gradient approximation for edge detection


Sobel operator


thresholeded Sobel operator

Laplacian operator

Laplacian of a function in continuous-space:

$$\Delta f(t_1, t_2) = \frac{\partial^2 f}{\partial t_1^2} + \frac{\partial^2 f}{\partial t_2^2}$$

approximating the Laplacian; start with a Taylor expansion

$$f(t+\tau) = \sum_{n=0}^{\infty} \frac{f^{(n)}(t)}{n!}\tau^n$$

and compute the expansion in $(t+\tau)$ and $(t-\tau)$:

$$f(t+\tau) = f(t) + f'(t)\tau + \frac{1}{2}f''(t)\tau^2$$

$$f(t-\tau) = f(t) - f'(t)\tau + \frac{1}{2}f''(t)\tau^2$$

Laplacian operator

by rearranging terms:

$$f''(t) = \frac{1}{\tau^2}(f(t-\tau) - 2f(t) + f(t+\tau))$$

which, on the discrete grid, is the FIR $h[n] = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}$

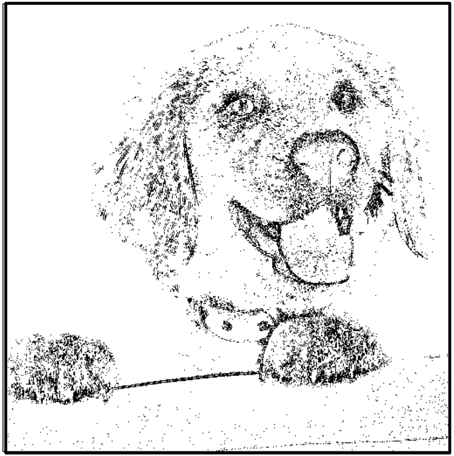
summing the horizontal and vertical components:

$$h[n_1, n_2] = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

If we use the diagonals too:

$$h[n_1, n_2] = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Laplacian operator



thresholeded Laplacian operator

END OF MODULE 8.4

Digital Signal Processing

Module 8.5: Image Compression

Overview:

- Redundancy in natural images
- Image coding ingredients

Overview:

- Redundancy in natural images
- Image coding ingredients

A thought experiment

- consider all possible 256×256, 8bpp images
- each image is 524,288 bits
- total number of possible images: $2^{524,288} \approx 10^{157,826}$
- number of atoms in the universe: 10^{82}

A thought experiment

- consider all possible 256×256, 8bpp images
- each image is 524,288 bits
- total number of possible images: $2^{524,288} \approx 10^{157,826}$
- number of atoms in the universe: 10^{82}

A thought experiment

- consider all possible 256×256, 8bpp images
- each image is 524,288 bits

- total number of possible images: $2^{524,288} \approx 10^{157,826}$
- number of atoms in the universe: 10^{82}

A thought experiment

- consider all possible 256×256, 8bpp images
- each image is 524,288 bits

- total number of possible images: $2^{524{,}288} \approx 10^{157{,}826}$
- number of atoms in the universe: 10^{82}

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number
- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number

- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number
- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number

- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an "encyclopedia of images"
- to indicate an image, simply give its number

- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

How many bits per image?

Another thought experiment

- take *all* images in the world and list them in an “encyclopedia of images”
- to indicate an image, simply give its number

- on the Internet: $M = 50$ billion
- raw encoding: 524,288 bits per image
- enumeration-based encoding: $\log_2 M \approx 36$ bits per image
- (of course, side information is HUGE)

Another approach:

- exploit "physical" redundancy
- allocate bits for things that matter (e.g. edges)
- use psychovisual experiments to find out what matters
- some information is discarded: *lossy* compression

Compression

Another approach:

- exploit "physical" redundancy
- allocate bits for things that matter (e.g. edges)
- use psychovisual experiments to find out what matters
- some information is discarded: *lossy* compression

Compression

Another approach:

- exploit "physical" redundancy
- allocate bits for things that matter (e.g. edges)
- use psychovisual experiments to find out what matters
- some information is discarded: *lossy* compression

Another approach:

- exploit "physical" redundancy
- allocate bits for things that matter (e.g. edges)
- use psychovisual experiments to find out what matters
- some information is discarded: *lossy* compression

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Compressing at pixel level

- reduce number bits per pixel
- equivalent to coarser quantization
- in the limit, 1bpp

Compressing at pixel level

- reduce number bits per pixel
- equivalent to coarser quantization
- in the limit, 1bpp

Compressing at pixel level

- reduce number bits per pixel
- equivalent to coarser quantization
- in the limit, 1bpp

Compressing at block level

- divide the image in blocks
- code the average value with 8 bits
- 3×3 blocks at 8 bits per block gives less than 1bpp

Compressing at block level

- divide the image in blocks
- code the average value with 8 bits
- 3×3 blocks at 8 bits per block gives less than 1bpp

Compressing at block level

- divide the image in blocks
- code the average value with 8 bits
- 3×3 blocks at 8 bits per block gives less than 1bpp

Compressing at block level

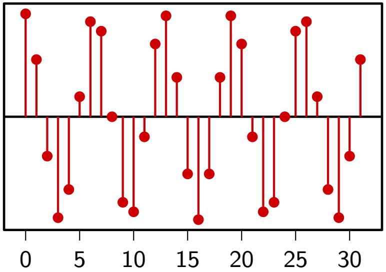
- exploit the local spatial correlation
- compress remote regions independently

Compressing at block level

- exploit the local spatial correlation
- compress remote regions independently

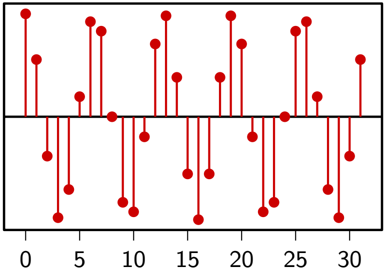
A simple example:

- take a DT signal, assume R bits per sample
- storing the signal requires NR bits
- now you take the DFT and it looks like this
- in theory, we can just code the two nonzero DFT coefficients!



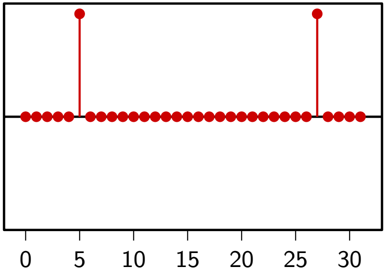
A simple example:

- take a DT signal, assume R bits per sample
- storing the signal requires NR bits
- now you take the DFT and it looks like this
- in theory, we can just code the two nonzero DFT coefficients!



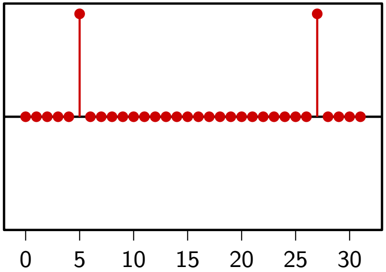
A simple example:

- take a DT signal, assume R bits per sample
- storing the signal requires NR bits
- now you take the DFT and it looks like this
- in theory, we can just code the two nonzero DFT coefficients!



A simple example:

- take a DT signal, assume R bits per sample
- storing the signal requires NR bits
- now you take the DFT and it looks like this
- in theory, we can just code the two nonzero DFT coefficients!



Transform coding

Ideally, we would like a transform that:

- captures the important features of an image block in a few coefficients
- is efficient to compute
- answer: the Discrete Cosine Transform

Ideally, we would like a transform that:

- captures the important features of an image block in a few coefficients
- is efficient to compute

- answer: the Discrete Cosine Transform

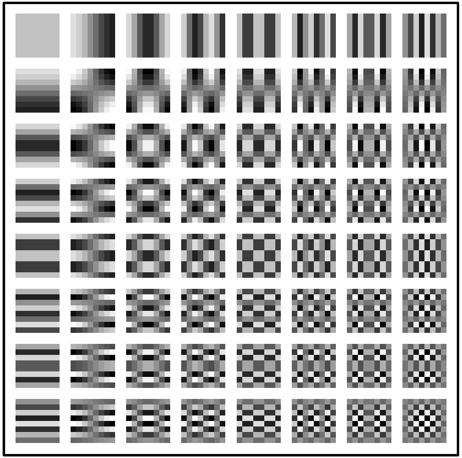
Transform coding

Ideally, we would like a transform that:

- captures the important features of an image block in a few coefficients
- is efficient to compute
- answer: the Discrete Cosine Transform

$$C[k_1, k_2] = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x[n_1, n_2] \cos\left[\frac{\pi}{N}\left(n_1 + \frac{1}{2}\right) k_1\right] \cos\left[\frac{\pi}{N}\left(n_2 + \frac{1}{2}\right) k_2\right]$$

DCT basis vectors for an 8×8 image



Smart quantization

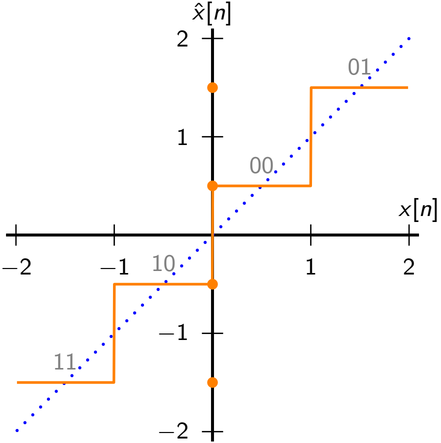
- deadzone
- variable step (fine to coarse)

Smart quantization

- deadzone
- variable step (fine to coarse)

Standard quantization:

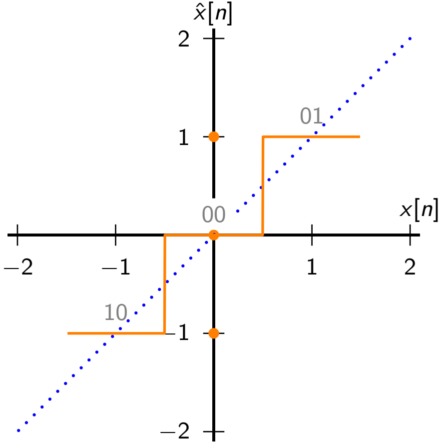
$$\hat{x} = \text{floor}(x) + 0.5$$



Quantization

Deadzone quantization:

$$\hat{x} = \text{round}(x)$$



Entropy coding

- minimize the effort to encode a certain amount of information
- associate short symbols to frequent values and vice-versa
- if it sounds familiar it's because it is...

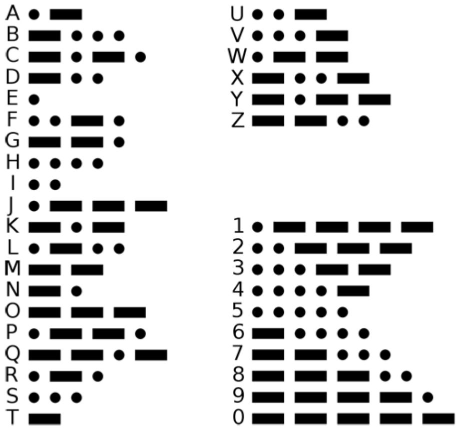
Entropy coding

- minimize the effort to encode a certain amount of information
- associate short symbols to frequent values and vice-versa
- if it sounds familiar it's because it is...

Entropy coding

- minimize the effort to encode a certain amount of information
- associate short symbols to frequent values and vice-versa

- if it sounds familiar it's because it is...


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T
U
V
W
X
Y
Z
1
2
3
4
5
6
7
8
9
0

END OF MODULE 8.5

Digital Signal Processing

Module 8.6: The JPEG Compression Algorithm

Key ingredients

- compressing at block level
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- split image into 8×8 non-overlapping blocks
- using a suitable transform (i.e., a change of basis)
- smart quantization
- entropy coding

Key ingredients

- split image into 8×8 non-overlapping blocks
- compute the DCT of each block
- smart quantization
- entropy coding

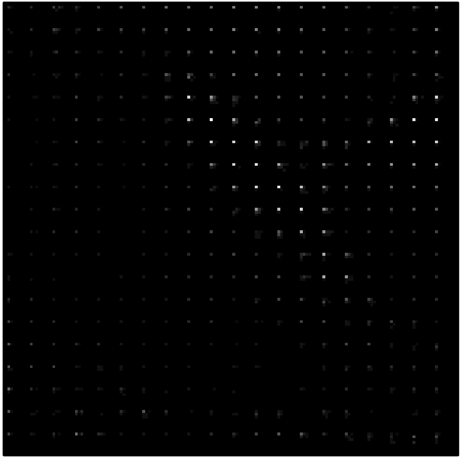
Key ingredients

- split image into 8×8 non-overlapping blocks
- compute the DCT of each block
- quantize DCT coefficients according to psycovisually-tuned tables
- entropy coding

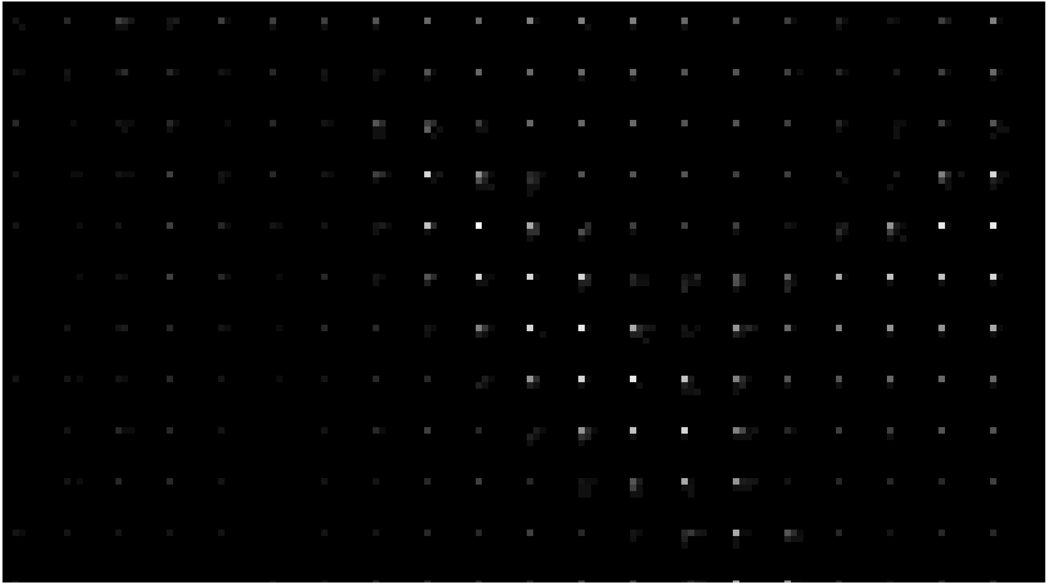
Key ingredients

- split image into 8×8 non-overlapping blocks
- compute the DCT of each block
- quantize DCT coefficients according to psycovisually-tuned tables
- run-length encoding and Huffman coding

DCT coefficients of image blocks (detail)



DCT coefficients of image blocks (detail)



Smart quantization

- most coefficients are negligible $\rightarrow$ captured by the deadzone
- some coefficients are more important than others
- find out the critical coefficients by experimentation
- allocate more bits (or, equivalently, finer quantization levels) to the most important coefficients

Smart quantization

- most coefficients are negligible $\rightarrow$ captured by the deadzone
- some coefficients are more important than others
- find out the critical coefficients by experimentation
- allocate more bits (or, equivalently, finer quantization levels) to the most important coefficients

Smart quantization

- most coefficients are negligible $\rightarrow$ captured by the deadzone
- some coefficients are more important than others
- find out the critical coefficients by experimentation
- allocate more bits (or, equivalently, finer quantization levels) to the most important coefficients

Smart quantization

- most coefficients are negligible $\rightarrow$ captured by the deadzone
- some coefficients are more important than others
- find out the critical coefficients by experimentation
- allocate more bits (or, equivalently, finer quantization levels) to the most important coefficients

Psychovisually-tuned quantization table

$$\hat{c}[k_1, k_2] = \mathrm{round}(c[k_1, k_2]/Q[k_1, k_2])$$

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

Advantages of nonuniform bit allocation

uniform

tuned

Advantages of nonuniform bit allocation (detail)

uniform

tuned

Efficient coding

- most coefficients are small, decreasing with index
- use zigzag scan to maximize ordering
- quantization will create long series of zeros

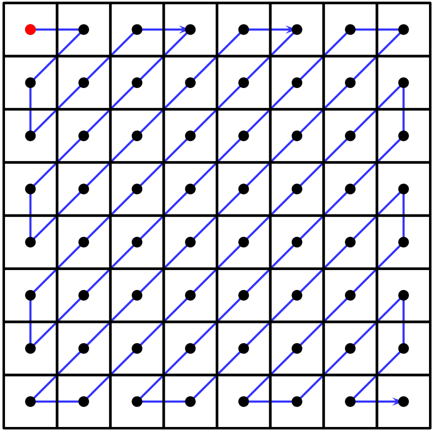
Efficient coding

- most coefficients are small, decreasing with index
- use zigzag scan to maximize ordering
- quantization will create long series of zeros

Efficient coding

- most coefficients are small, decreasing with index
- use zigzag scan to maximize ordering
- quantization will create long series of zeros

Zigzag scan



Example

$$
\begin{bmatrix}
100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
$$

100, -60, 0, 0, 0, 0, 6, 0, 0, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 13, 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

Example

$$\begin{bmatrix} 100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

100, -60, 0, 0, 0, 0, 6, 0, 0, 0, 13, 0, 0, 0, 0, 0, 0, 0, 0, 13, 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

Each nonzero value is encoded as the triple

$$[(r, s),\ c]$$

- r is the *runlength* i.e. the number of zeros before the current value
- s is the *size* i.e. the number of bits needed to encode the value
- c is the actual value
- $(0, 0)$ indicates that from now on it's only zeros (end of block)

Runlength encoding

Each nonzero value is encoded as the triple

$$[(r, s),\ c]$$

- r is the *runlength* i.e. the number of zeros before the current value
- s is the *size* i.e. the number of bits needed to encode the value
- c is the actual value
- $(0, 0)$ indicates that from now on it's only zeros (end of block)

Runlength encoding

Each nonzero value is encoded as the triple

$$[(r, s),\ c]$$

- r is the *runlength* i.e. the number of zeros before the current value
- s is the *size* i.e. the number of bits needed to encode the value
- c is the actual value
- $(0, 0)$ indicates that from now on it's only zeros (end of block)

Runlength encoding

Each nonzero value is encoded as the triple

$$[(r, s),\ c]$$

- r is the *runlength* i.e. the number of zeros before the current value
- s is the *size* i.e. the number of bits needed to encode the value
- c is the actual value
- $(0, 0)$ indicates that from now on it's only zeros (end of block)

Example

$$\begin{bmatrix} 100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[(0,7),\ 100],\ [(0,6),\ -60],\ [(4,3),\ 6],\ [(3,4),\ 13],\ [(8,1),\ -1],\ [(0,0)]$$

Example

$$\begin{bmatrix} 100 & -60 & 0 & 6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 13 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$[(0,7),\ 100],\ [(0,6),\ -60],\ [(4,3),\ 6],\ [(3,4),\ 13],\ [(8,1),\ -1], [(0,0)]$$

The runlength-size pairs

- by design, $(r, s) \in \mathcal{A}$ with $|\mathcal{A}| = 256$
- in theory, 8 bits per pair
- some pairs are much more common than others!
- a lot of space can be saved by being smart

The runlength-size pairs

- by design, $(r, s) \in \mathcal{A}$ with $|\mathcal{A}| = 256$
- in theory, 8 bits per pair
- some pairs are much more common than others!
- a lot of space can be saved by being smart

The runlength-size pairs

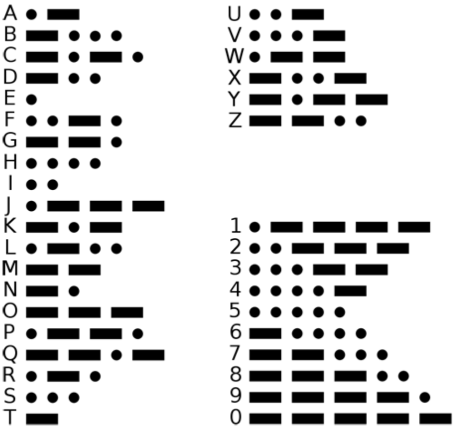
- by design, $(r, s) \in \mathcal{A}$ with $|\mathcal{A}| = 256$
- in theory, 8 bits per pair
- some pairs are much more common than others!
- a lot of space can be saved by being smart

The runlength-size pairs

- by design, $(r, s) \in \mathcal{A}$ with $|\mathcal{A}| = 256$
- in theory, 8 bits per pair
- some pairs are much more common than others!
- a lot of space can be saved by being smart

Variable-length encoding

great idea: shorter binary sequences for common symbols



Variable-length encoding

however: if symbols have different lengths, we must know how to parse them!

- in English, spaces separate words $\rightarrow$ extra symbol (wasteful)
- in Morse code, pauses separate letters and words (wasteful)
- can we do away with separators?

however: if symbols have different lengths, we must know how to parse them!

- in English, spaces separate words $\rightarrow$ extra symbol (wasteful)
- in Morse code, pauses separate letters and words (wasteful)
- can we do away with separators?

Variable-length encoding

however: if symbols have different lengths, we must know how to parse them!

- in English, spaces separate words $\rightarrow$ extra symbol (wasteful)
- in Morse code, pauses separate letters and words (wasteful)
- can we do away with separators?

Prefix-free codes

- no valid sequence can be the beginning of another valid sequence
- can parse a bitstream sequentially with no look-ahead
- extremely easy to understand graphically...

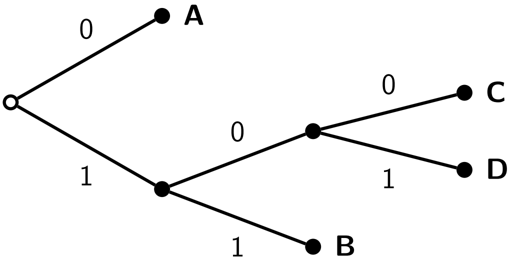
Prefix-free codes

- no valid sequence can be the beginning of another valid sequence
- can parse a bitstream sequentially with no look-ahead
- extremely easy to understand graphically...

Prefix-free codes

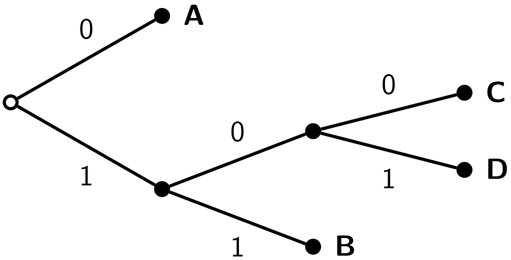
- no valid sequence can be the beginning of another valid sequence
- can parse a bitstream sequentially with no look-ahead
- extremely easy to understand graphically...

Prefix-free code



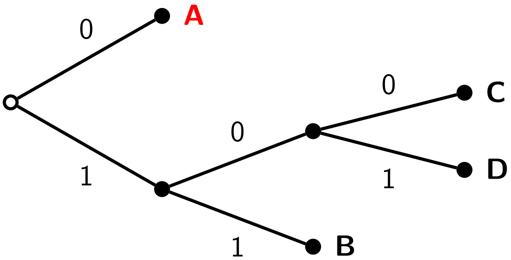
001100110101100

Prefix-free code



001100110101100

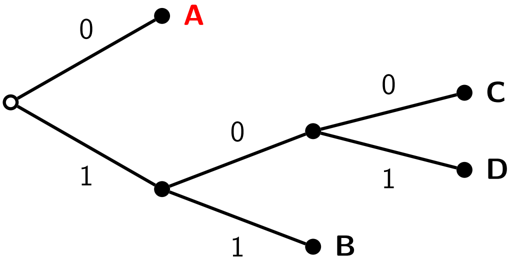
Prefix-free code



001100110101100

A

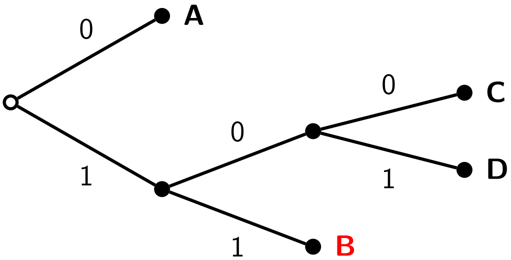
Prefix-free code



001100110101100

AA

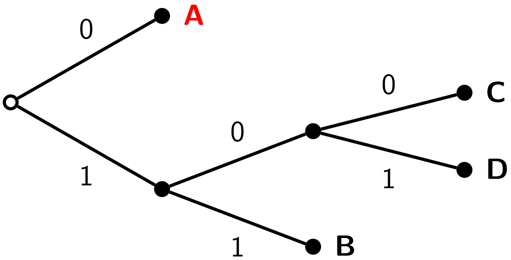
Prefix-free code



0011001101011100

AAB

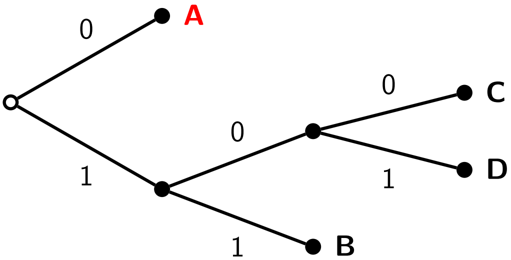
Prefix-free code



00110011010110 0

AABA

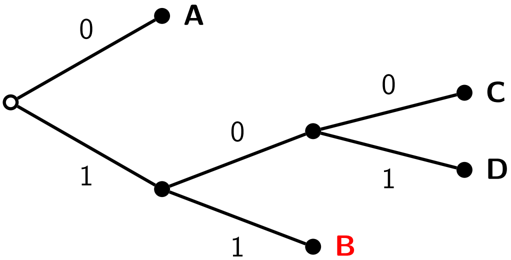
Prefix-free code



001100110101100

AABAA

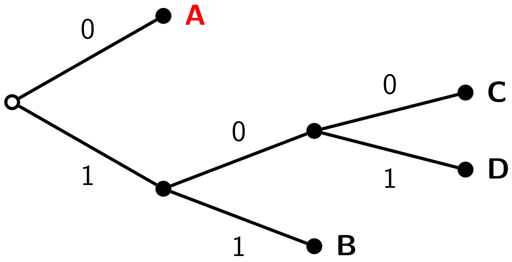
Prefix-free code



001100110101100

AABAAB

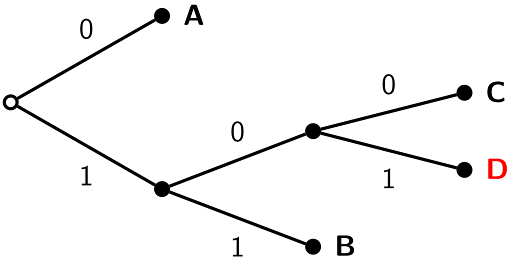
Prefix-free code



001100110101100

AABAABA

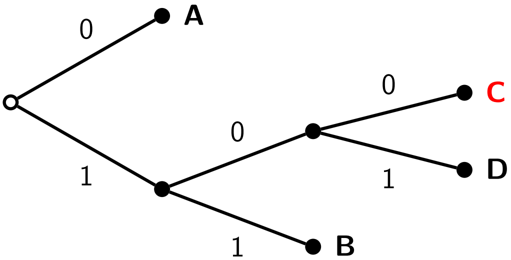
Prefix-free code



001100110101100

AABAABAD

Prefix-free code



0011001101011100

AABAABADC

Entropy coding

goal: minimize message length

- assign short sequences to more frequent symbols
- the Huffman algorithm builds the optimal code for a set of symbol probabilities
- in JPEG, you can use a "general-purpose" Huffman code or build your own (but then you pay a "side-information" price)

goal: minimize message length

- assign short sequences to more frequent symbols
- the Huffman algorithm builds the optimal code for a set of symbol probabilities
- in JPEG, you can use a "general-purpose" Huffman code or build your own
  (but then you pay a "side-information" price)

Entropy coding

goal: minimize message length

- assign short sequences to more frequent symbols
- the Huffman algorithm builds the optimal code for a set of symbol probabilities
- in JPEG, you can use a "general-purpose" Huffman code or build your own
  (but then you pay a "side-information" price)

Example

- four symbols: A, B, C, D
- probability table:

$$p(A) = 0.38 \qquad p(B) = 0.32$$

$$p(C) = 0.1 \qquad p(D) = 0.2$$

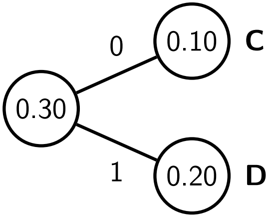
Example

- four symbols: A, B, C, D
- probability table:

$$p(A) = 0.38 \qquad p(B) = 0.32$$

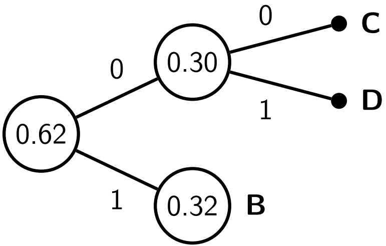
$$p(C) = 0.1 \qquad p(D) = 0.2$$

Building the Huffman code

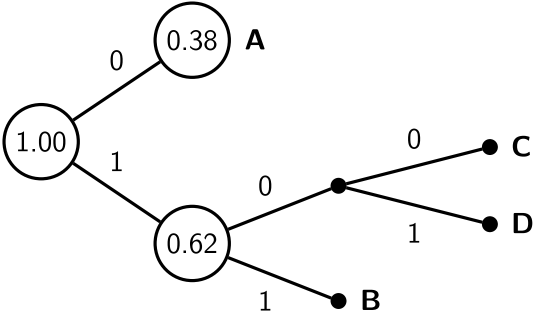


$p(A) = 0.38 \quad p(B) = 0.32 \quad p(C) = 0.1 \quad p(D) = 0.2$

Building the Huffman code



$p(A) = 0.38 \quad p(B) = 0.32 \quad p(C + D) = 0.3$



$p(A) = 0.38 \quad p(B + C + D) = 0.62$

END OF MODULE 8.6

END OF MODULE 8