

# Digital Image Processing



# Course organization

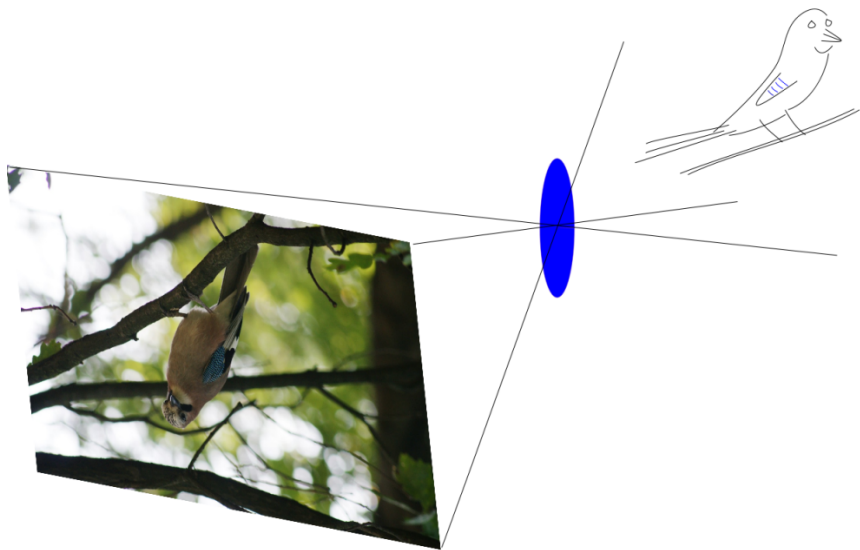
Teachers 2012:

- ▶ Lecturers: Václav Hlaváč, Ondřej Drbohlav
- ▶ Lab tutors: Jan Mačák, Tomáš Petříček

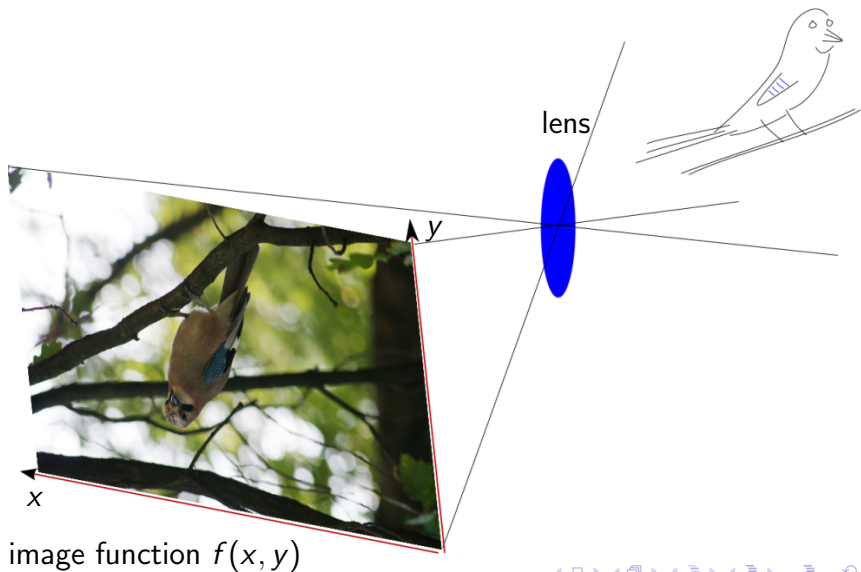
Courseware:

- ▶ <http://cw.felk.cvut.cz>
- ▶  $\Rightarrow$  online discussion of conditions and rules

# Digital image - Origin



# Digital image - Origin



# Image function $f(x, y)$

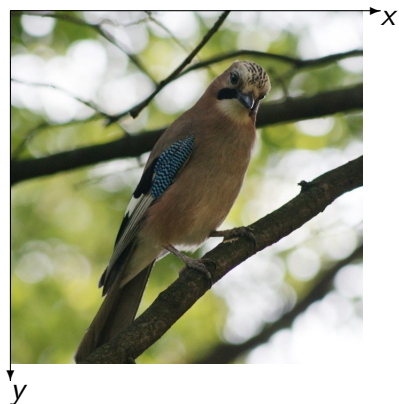


Image function is a mapping:

$$f : Q \mapsto R$$

	domain $Q$	range $R$
lives in	$Q \subset \mathbb{R}^2$	various: color $R \subset \mathbb{R}^3$
unit	$x, y$ each: [mm]	each channel [ $\text{Wm}^{-2}$ ]

# Image function $f(x, y)$



Image function is a mapping:

$$f : Q \mapsto R$$

	domain $Q$	range $R$
lives in	$Q \subset \mathbb{R}^2$	various: grayvalue $R \subset \mathbb{R}$
unit	$x, y$ each: [mm]	each channel [ $\text{Wm}^{-2}$ ]

## Image function $f(x, y)$ (2)



Image function is a mapping:

$$f : Q \mapsto R$$

This can be regarded as a **set** of ordered pairs ( $[x, y], value$ ).

Both  $Q$  and  $R$  are continuous!

The major part of this lecture will be concerned with how to **represent** the image function in a **digital** form.

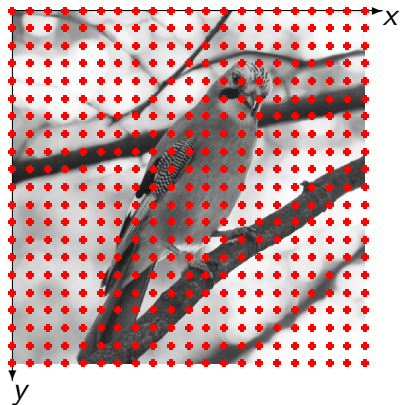
# Representing image function



This requires use of finite memory space.



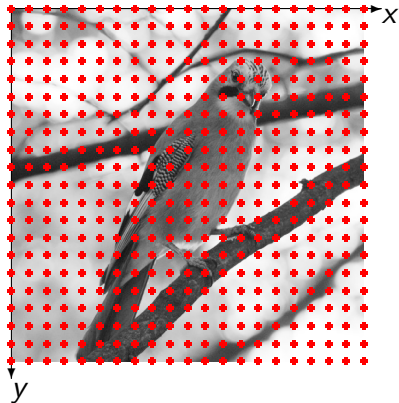
# Representing image function



This requires use of finite memory space.

- ▶ representing  $f$  by finite number of numbers  $\Rightarrow$  **sampling**

# Representing image function



This requires use of finite memory space.

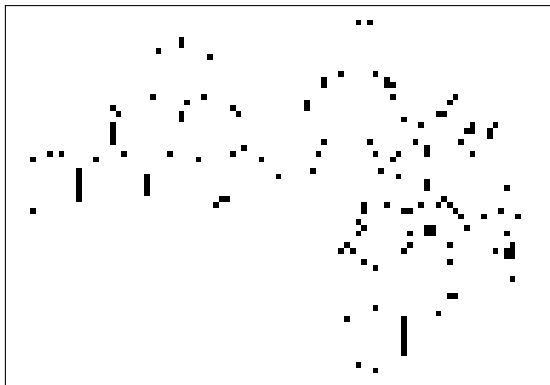
- ▶ representing  $f$  by finite number of numbers  $\Rightarrow$  **sampling**
- ▶ at each such point, store the *value* in finite precision  $\Rightarrow$  **quantization**.

# Sampling (1)

- ▶ Representing  $f$  using values sampled on a regular grid is by far the most common choice.
- ▶ There can be other representations (functional forms, etc.)
- ▶ There can be other sampling schemes (hexagonal, irregular, etc.)

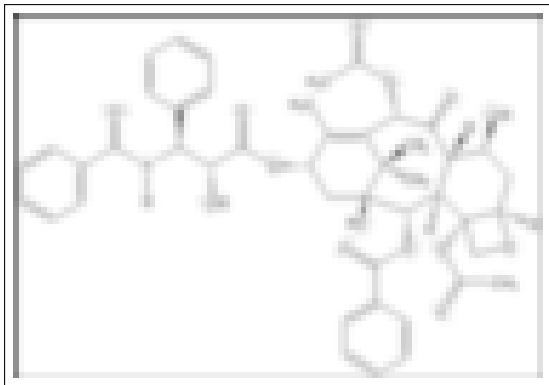
## Sampling (2)

How to sample properly? Intuitively, the function should not change much between two sampling points. Compare these 60x90 images ...



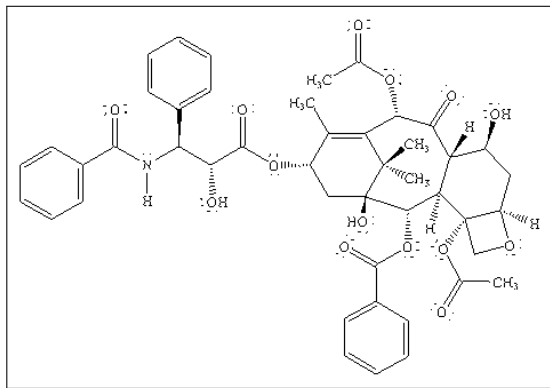
## Sampling (2)

How to sample properly? Intuitively, the function should not change much between two sampling points. Compare these 60x90 images ...



## Sampling (2)

How to sample properly? Intuitively, the function should not change much between two sampling points. Compare these 60x90 images ... and the source image function!



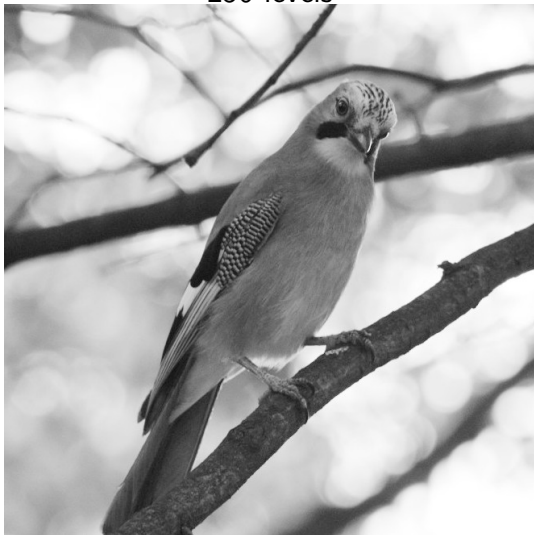
## Sampling (3)

- ▶ necessary to ensure that there are no high-frequency oscillations in the image function before sampling
- ▶ if necessary, filter the function before sampling
- ▶ this has relation to aliasing and Nyquist theory — we will be talking about it later.

link: [some blackboard scribble](#)

# Quantization

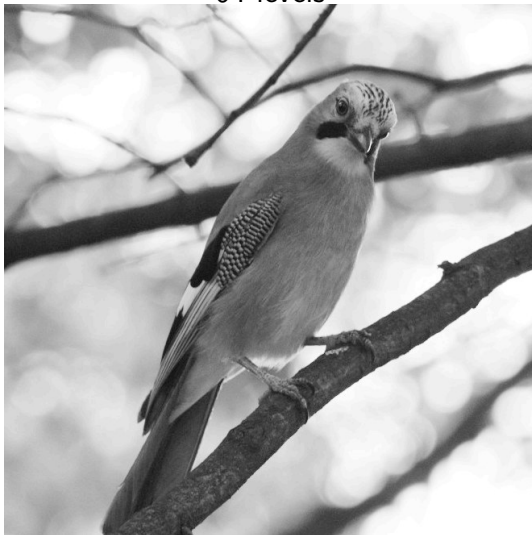
256 levels





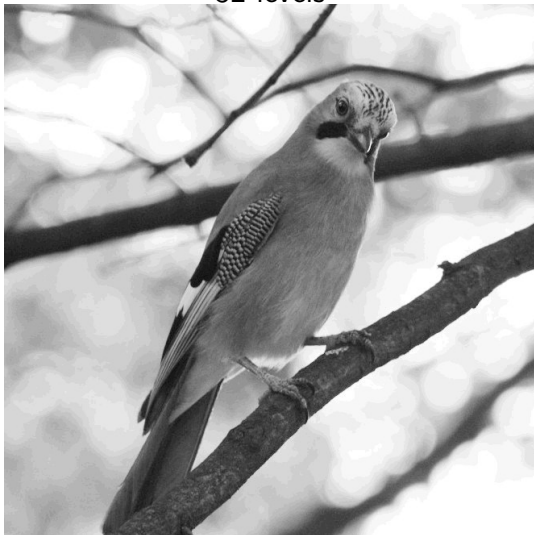
# Quantization

64 levels



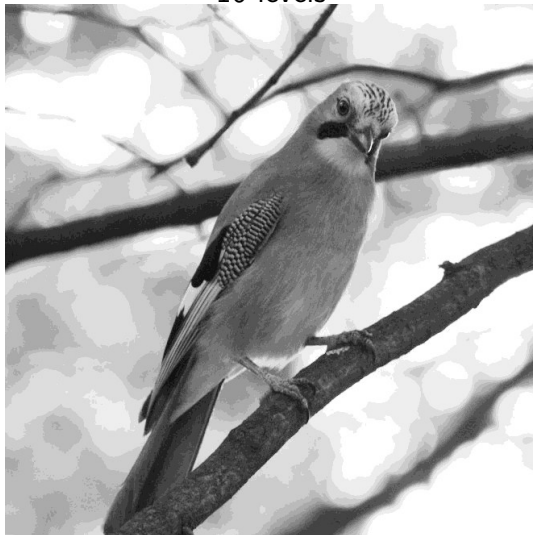
# Quantization

32 levels



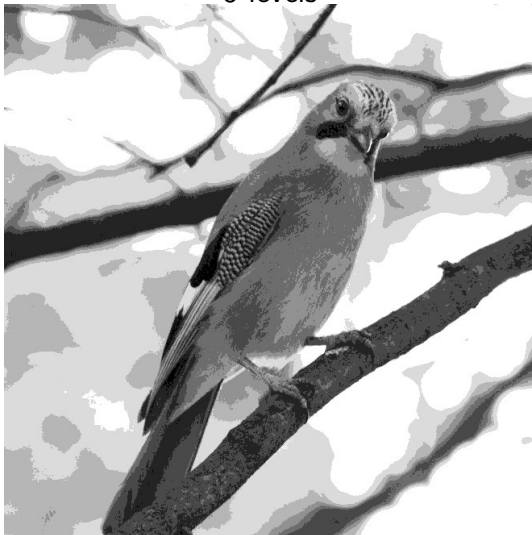
# Quantization

16 levels



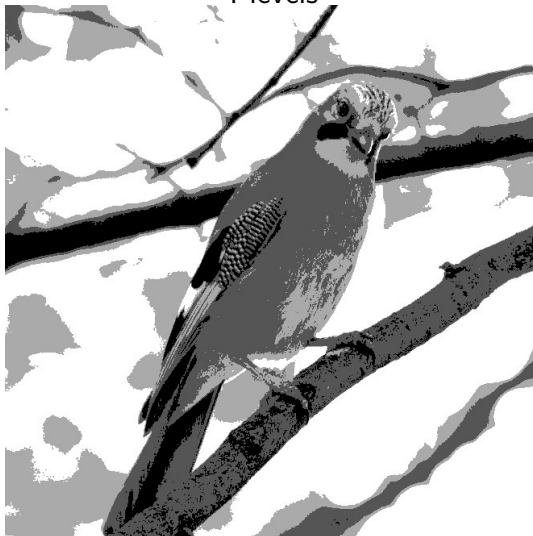
# Quantization

8 levels



# Quantization

4 levels



# Quantization

2 levels



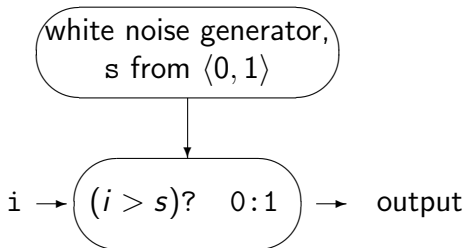
# Quantization & sampling — interplay

- ▶ Would it be possible to trade quantization for resolution?  
E.g. using only 2 levels but increasing sampling rate
- ▶ ... not attractive from coding/compression point of view
- ▶ ... but necessary for **creating** the image function at some output devices which use limited number of levels
- ▶ E.g. black & white printers
- ▶ Displays (Amazon Kindle)
- ▶ ⇒ **Dithering**

# Dithering (random) (1)

Simple but effective: random dithering

- ▶ Idea: represent a number  $i \in \langle 0, 1 \rangle$  by an ensemble of 0's and 1's such that their expected value is  $i$ .
- ▶ How:





# Dithering (random) (1)

Simple but effective: random dithering

- ▶ Idea: represent a number  $i \in \langle 0, 1 \rangle$  by an ensemble of 0's and 1's such that their expected value is  $i$ .

- ▶ How: (matlab code)

```
function o = dither_randomly(im);  
% function o = dither_randomly(im);  
% dithers a uint8 image using random  
% sampling.  
t = 255*rand(size(im));  
o = t < im;
```

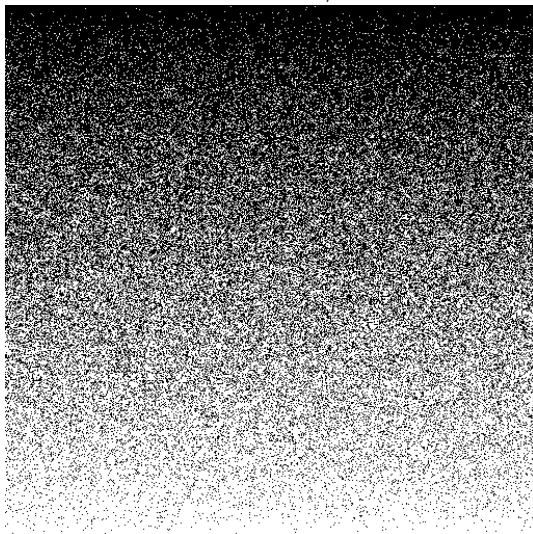
# Dithering (random) (2, Examples)

ramp, 0-255



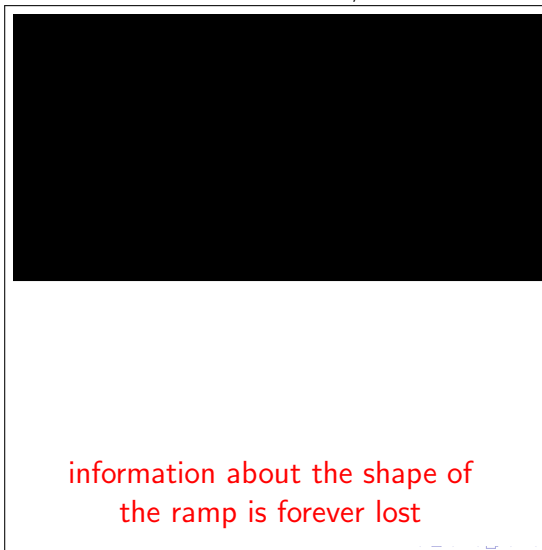
# Dithering (random) (2, Examples)

dithered, 0/1



## Dithering (random) (2, Examples)

thresholded, 0/1



## Dithering (random) (2, Examples)

original, 256 levels



# Dithering (random) (2, Examples)

dithered, 0/1



## Dithering (3)

Can we do better?

- ▶ with the previous approach, the advantage is simplicity
- ▶ ... but the problem is that the output image neighboring pixels are generated completely independently
- ▶ leading to sub-optimal result

## Dithering (3)

Can we do better?

- ▶ with the previous approach, the advantage is simplicity
- ▶ ... but the problem is that the output image neighboring pixels are generated completely independently
- ▶ leading to sub-optimal result
- ▶ another easy way: code and distribute the residuum to neighboring pixels
- ▶  $\Rightarrow$  Floyd-Steinberg dithering

link: [blackboard explanation](#)



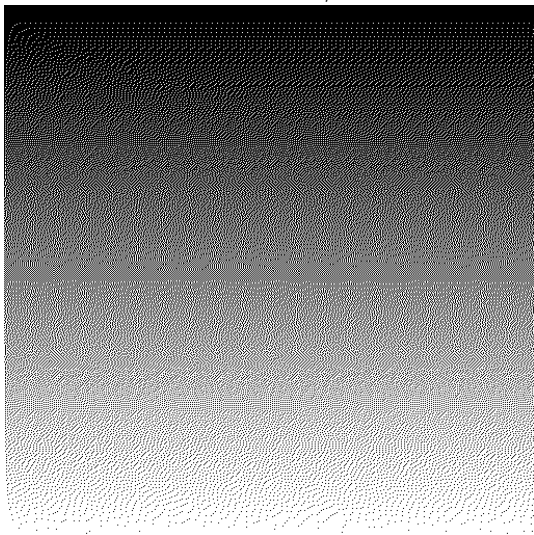
# Dithering (4, Floyd-Steinberg)

ramp, 0-255



# Dithering (4, Floyd-Steinberg)

dithered, 0/1



# Dithering (4, Floyd-Steinberg)

original, 256 levels



# Dithering (4, Floyd-Steinberg)

dithered, 0/1



# Dithering (5, Comparison)

random



Floyd-Steinberg



## Dithering (6, Comparison II)

filtered by a Gaussian,  $\sigma = 3$

random



Floyd-Steinberg



# Dithering (6, Comparison II)

original



## On Kindle: Ramp

Ramp image,  $800 \times 600$ , linear change from 0 to 255.  
No dithering.



E-ink Pearl,  
16 shades of gray



# On Kindle: Ramp

Ramp image,  $800 \times 600$ , linear change from 0 to 255.  
With dithering.



E-ink Pearl,  
16 shades of  
gray

# On Kindle: A photo

800 × 600 No dithering.



# On Kindle: A photo

800 × 600 With dithering.



# Information

- ▶ So far, we have seen that with different options of sampling/quantization, different amount of information is lost
- ▶ Connected to this is information-theoretic view of an image contents

# Histogram, entropy

- ▶ Histogram: stores frequencies  $q(i)$  for all values  $i$  in an image
- ▶ for a gray-scale, 8 bit image: 256 bins
- ▶ probability of a given intensity value is

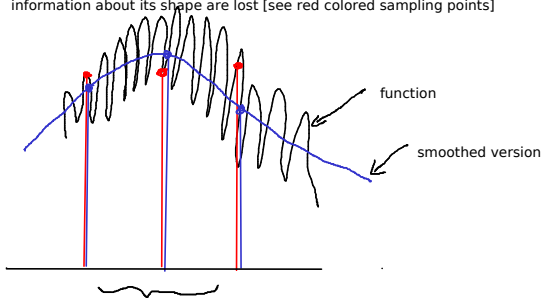
$$p(i) = q(i)/N,$$

$N$  is the number of pixels in an image

- ▶ entropy:

$$H = - \sum_{i=0}^{255} p(i) \log_2 p(i).$$

Problems with sampling an image function. When the function oscillates to a large extent in between the sampling locations, information about its shape are lost [see red colored sampling points]

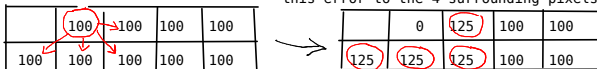


smoothing:  
average over an extended area

To retain at least some information about the shape of the function, the function has to be made smooth prior to sampling [see blue colored sampling points].

Floyd - Steinberg dithering algorithm - example. The goal is to represent an image by values which are either 0 or 255.

100 is closer to 0 than to 255. Replace this value by 0. Error is  $100-0=100$ . Distribute this error to the 4 surrounding pixels = increment them all by 25.



Then, continue with the next pixel in a left-to-right, top-to-bottom manner.

Note that this is the demonstration of the principle. In the actual Floyd-Steinberg algorithm, the error is not distributed to the four neighbors evenly, but by the following weights:

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