# TOČKOVNE HOMOGENE OPERACIJE

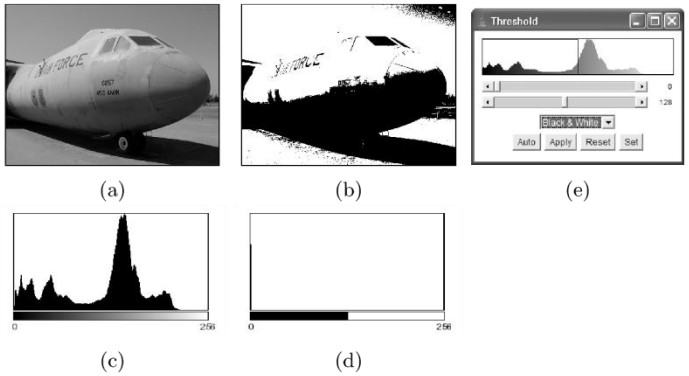
# Točkovne homogene operacije

- preslikave, odvisne samo od vrednosti slikovnega elementa
- neodvisne od slikovnih elementov v soseščini (filtri) in lokacije

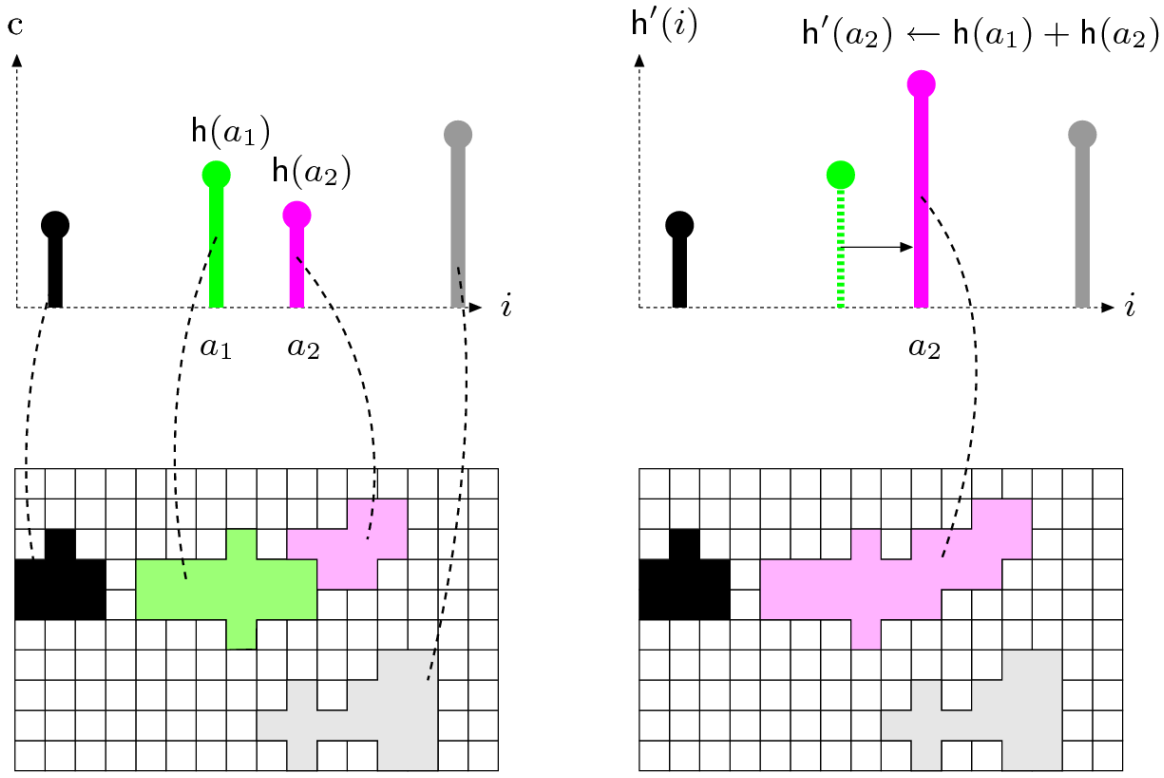
$$a' \leftarrow f(a)$$
$$I'(u,v) \leftarrow f\big(I(u,v)\big)$$

- **povečanje kontrasta** $f_{\text{contr}}(a) = a \cdot 1.5$ `ip.multiply(1.5)`
- **posvetlitev** $f_{\text{bright}}(a) = a + 10$ `ip.add(10)`
- **upragovljenje**

$$f_{\text{threshold}}(a) = \begin{cases} a_0 & \text{for } a < a_{\text{th}} \\ a_1 & \text{for } a \geq a_{\text{th}} \end{cases}$$



# Točkovne operacije in histogrami

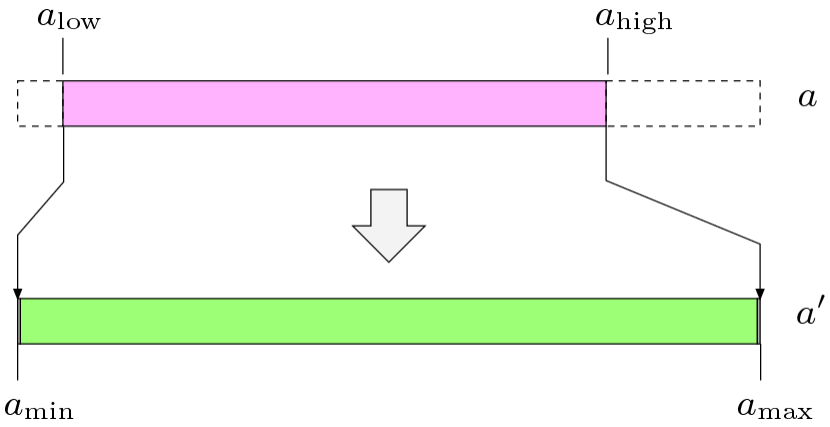



koši se lahko pomikajo (shift) ali pa združujejo (merge)

# Točkovne operacije: razteg histograma

- vrednosti intenzitet raztegnemo čez celoten interval
  - pomik najmanjše intenzitete $a_{low}$ v 0
  - skaliranje $(a_{max} - a_{min})/(a_{high} - a_{low})$
  - pomik v $a_{min}$

$$f_{ac}(a) = a_{min} + (a - a_{low}) \cdot \frac{a_{max} - a_{min}}{a_{high} - a_{low}}$$



# Točkovne operacije: histogram kot verjetnostna porazdelitev

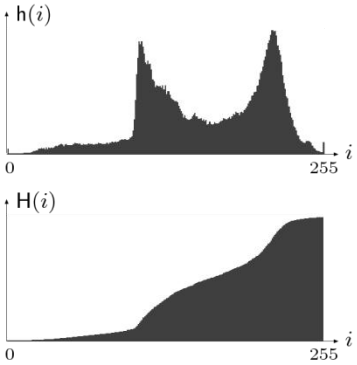
- predstavitev histograma, ki je neodvisna od velikosti slike
- histogram normaliziramo z $\sum_i \mathsf{h}(i) = M \cdot N$
- gostota verjetnosti diskretne spremenljivke

$$\mathsf{p}(i) = \frac{\mathsf{h}(i)}{MN}$$

- p(*i*) je verjetnost realizacije intenzitete *i*

# Točkovne operacije: kumulativna verjetnostna porazdelitev in histogram

- kumulativni histogram




$$\mathsf{H}(i) = \begin{cases} \mathsf{h}(0) & \text{for } \; i = 0 \\ \mathsf{H}(i-1) + \mathsf{h}(i) & \text{for } \; 0 < i < K \end{cases}$$

- kumulativna verjetnostna porazdelitev

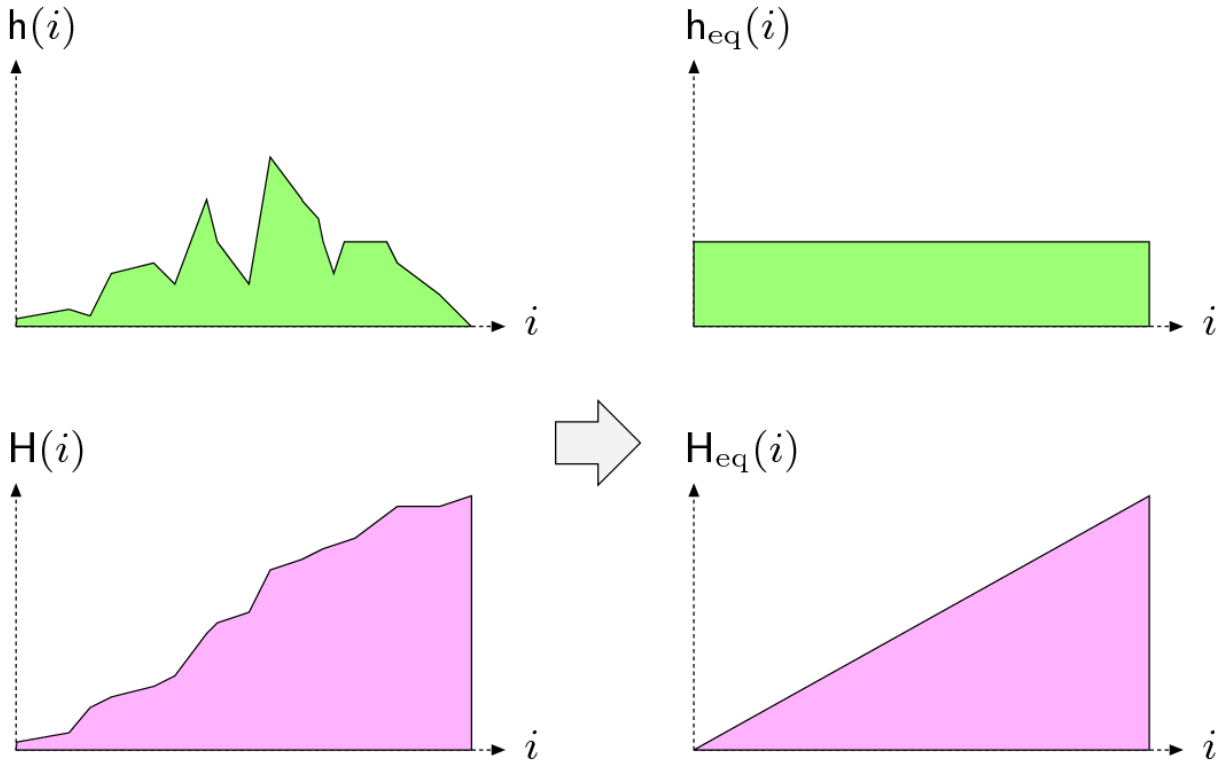
$$\mathsf{P}(i) = \frac{\mathsf{H}(i)}{\mathsf{H}(K-1)} = \frac{\mathsf{H}(i)}{MN} = \sum_{j=0}^{i} \frac{\mathsf{h}(j)}{MN}$$

$$= \sum_{j=0}^{i} \mathsf{p}(j) \qquad \text{for } \; 0 \leq i < K$$

- monotono naraščajoča funkcija

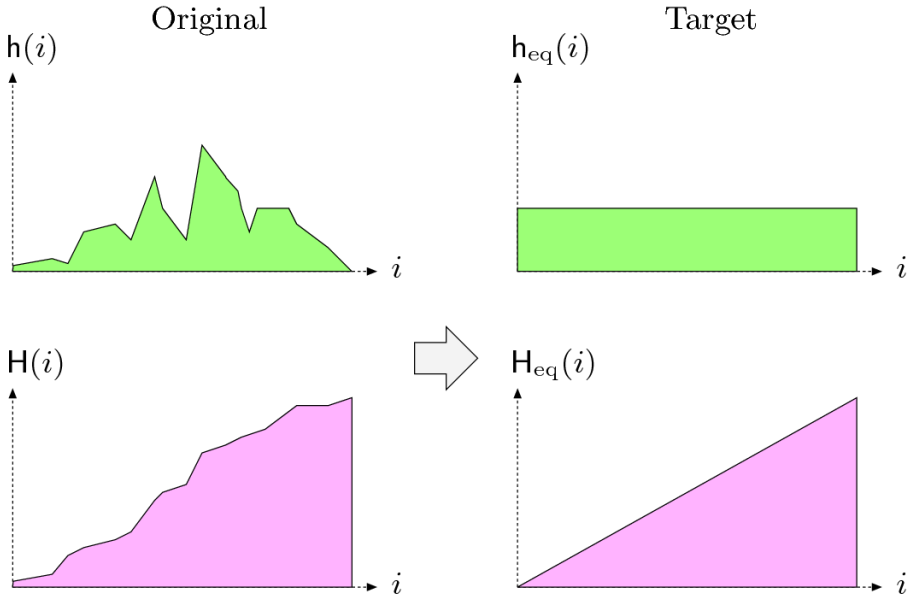
# Točkovne operacije: ekvalizacija histograma

- ekvalizacija: enakomerno porazdeljena verjetnost intenzitet



# Točkovne operacije: ekvalizacija histograma

- denimo da želimo sliko s histogramom $h(i)$, ki zavzema interval vrednosti $[i_{min}, i_{max}]$ pretvoriti v sliko z ekvaliziranim histogramom $h_{eq}(i')$
- ciljna slika naj zavzema vrednosti $[i'_{min}, i'_{max}]$



# Točkovne operacije: ekvalizacija histograma

pogoj $$\sum_{k=0}^{j} h_{eq}(i'_k) = \sum_{k=0}^{j} h(i_k)$$

zapišemo z vsotami verjetnostnih porazdelitev

$$p_{eq}(i') = \frac{h_{eq}(i')}{MN} = \frac{1}{i'_{\max} - i'_{\min}} \qquad p(i) = \frac{h(i)}{MN}$$

$$\sum_{k=0}^{j} \frac{1}{(i'_{\max} - i'_{\min})} = \sum_{k=0}^{j} \frac{h(i_k)}{MN}$$

in prevedemo v zvezen problem

$$\int_{i'_{\min}}^{i'} \frac{1}{(i'_{\max} - i'_{\min})} = \frac{(i' - i'_{\min})}{(i'_{\max} - i'_{\min})} = \frac{1}{MN} \int_{i_{\min}}^{i} h(s)ds$$

$$i' = i'_{\min} + \frac{(i'_{\max} - i'_{\min})}{MN} \int_{i_{\min}}^{i} h(s)ds$$

# Točkovne operacije: ekvalizacija histograma

$$i' = i'_{\min} + \frac{(i'_{\max} - i'_{\min})}{MN} \int_{i_{\min}}^{i} h(s)ds$$

nazaj v diskretno:

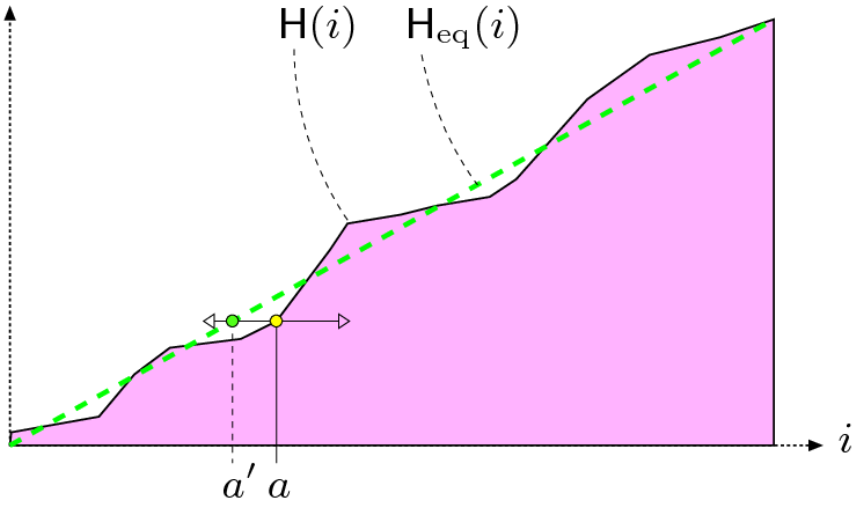
$$i'_j = f_{eq}(i_j) = i'_{\min} + \frac{(i'_{\max} - i'_{\min})}{MN} \sum_{k=i_{\min}}^{j} h(i_k)$$

Kumulativni histogram $H(i_j)$!

$$f_{eq}(i_j) = \frac{K}{MN} H(i_j); \quad [i'_{\min}, i'_{\max}] = [0, 255]$$

# Točkovne operacije: ekvalizacija histograma

$$a' = f_{eq}(a) = \frac{K}{MN} H(a)$$



## Histogram equalization

```
public void run(ImageProcessor ip) {
    int w = ip.getWidth();
    int h = ip.getHeight();
    int M = w * h;
    int K = 256;

    int[] H = ip.getHistogram();
    for (int j = 1; j < H.length; j++) {
       H[j] = H[j-1] + H[j];
    }

    for (int v = 0; v < h; v++) {
       for(int u = 0; u < w; u++) {
          int a = ip.get(u,v);
          int b = H[a] * (K-1) / M;
          ip.set(u, v, b);
       }
    }
}
```

izračun kumulativnega histograma

izračun preslikave

## Histogram equalization (lookup table)

```
public void run(ImageProcessor ip) {
    int w = ip.getWidth();
    int h = ip.getHeight();
    int M = w * h;
    int K = 256;

    int[] H = ip.getHistogram();
    for (int j = 1; j < H.length; j++) {
       H[j] = H[j-1] + H[j];
    }
    for (int j = 0; j < H.length; j++) {
       H[j] = H[j] * (K-1) / M;
    }

    ip.applyTable(H);
}
```

Za točkovne operacije lahko uporabimo predhodno izračunane lookup tabele

$$\mathbf{L} : [0, K-1] \xrightarrow{f} [0, K-1]$$

$$\mathbf{L}[a] \leftarrow f(a) \qquad \text{for } \; 0 \leq a < K$$

izračun kumulativnega histograma

izračun lookup tabele

uporabi tabelo na `ip`

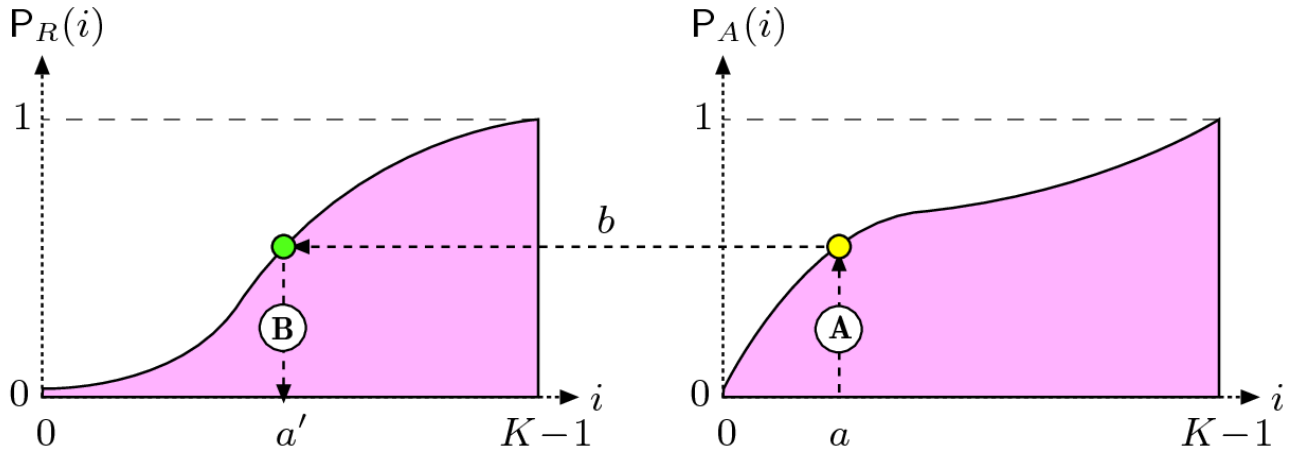
# Točkovne operacije: prilagajanje histograma

- Splošni princip prilagajanja histograma

$$a' = f_{\mathrm{hs}}(a)$$

- $\mathrm{P}_A$ slike $I_A$ naj ustreza referenčnem $\mathrm{P}_R$

$$\mathrm{P}_{A'}(i) \approx \mathrm{P}_R(i)$$

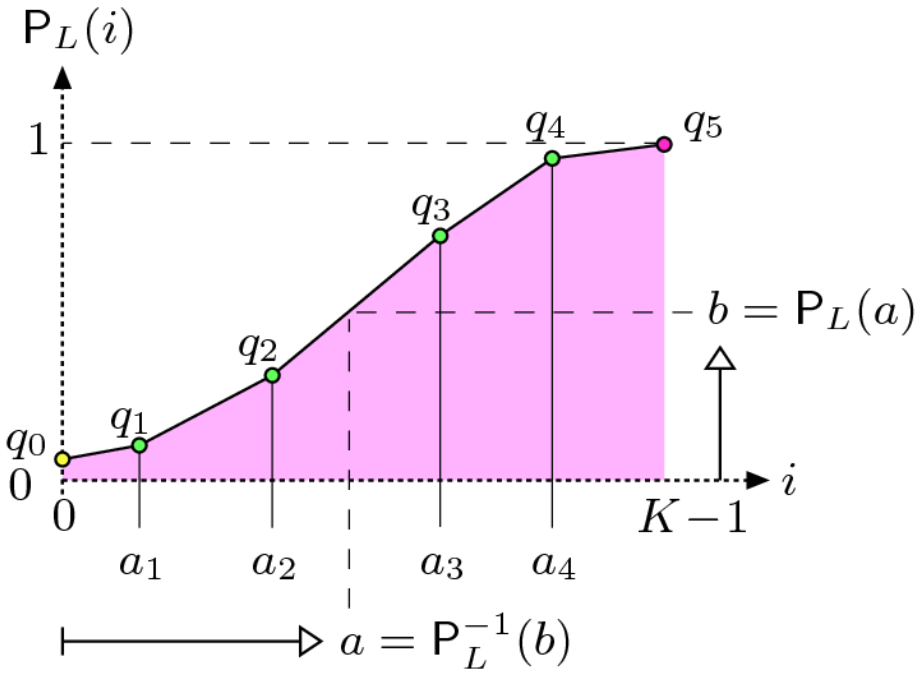



$$a' = \mathrm{P}_R^{-1}\big(\mathrm{P}_A(a)\big)$$

$$f_{\mathrm{hs}}(a) = a' = \mathrm{P}_R^{-1}\big(\mathrm{P}_A(a)\big)$$

# Točkovne operacije: prilagajanje histograma

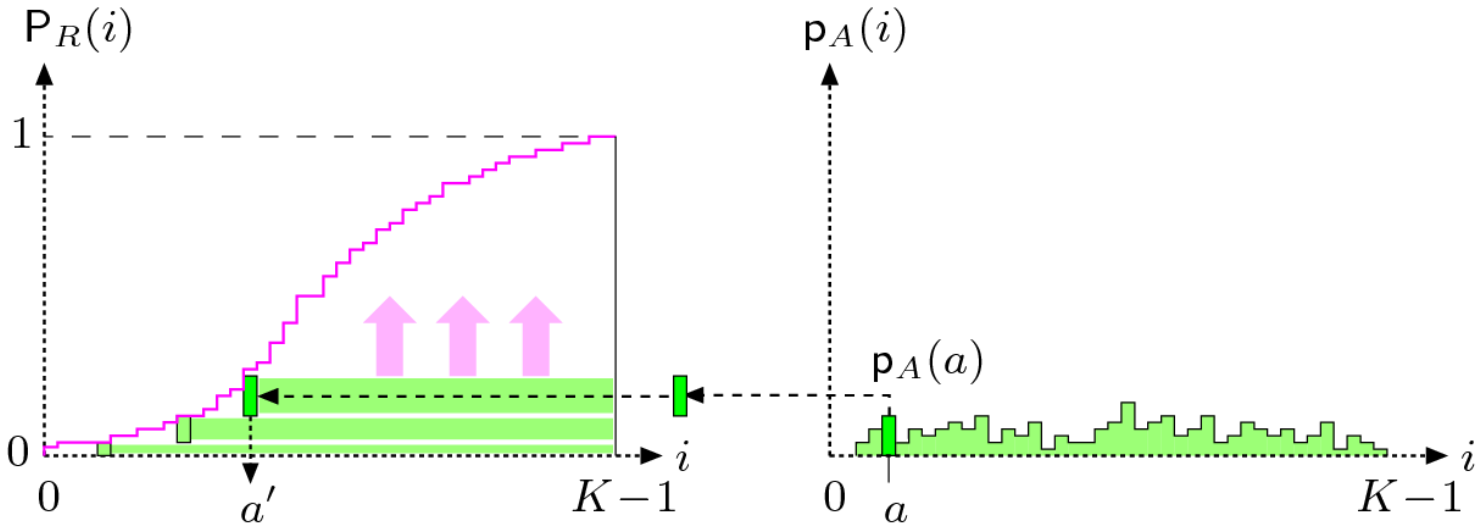
- Pogoj: funkcija $P_R(i)$ mora biti invertibilna
- referenčne histograme pogosto definiramo kot odsekoma linearno funkcijo



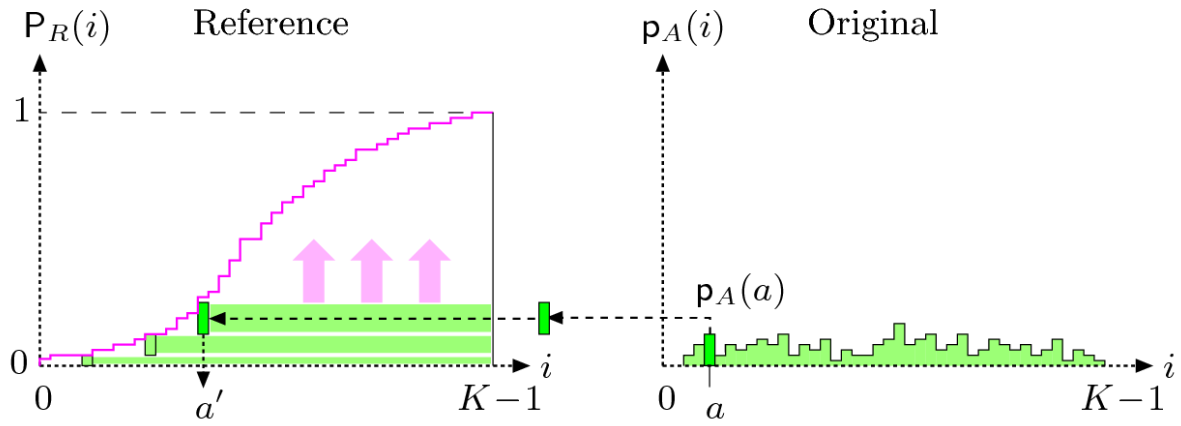
# Točkovne operacije: prilagajanje histograma

- Prilagajanje histograma histogramu referenčne slike:
  - če so nekateri koši v h(i) prazni, H(i) ni invertibilna

- "Tetris" prilagajanje



# Točkovne operacije: prilagajanje histograma




1: MatchHistograms($h_A$, $h_R$)

$h_A$: histogram of the target image
$h_R$: reference histogram (of same size as $h_A$)

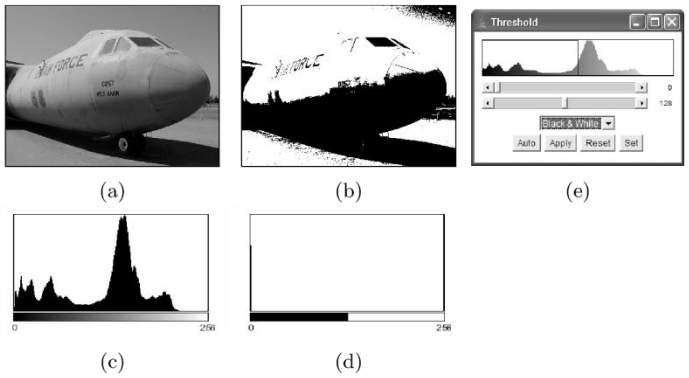
2: Let $K \leftarrow \text{Size}(h_A)$
3: Let $P_A \leftarrow \text{Cdf}(h_A)$ ▷ cdf for $h_A$ (Alg. 5.1)
4: Let $P_R \leftarrow \text{Cdf}(h_R)$ ▷ cdf for $h_R$ (Alg. 5.1)
5: Create a table $f_{\text{hs}}[\,]$ of size $K$ ▷ pixel mapping function $f_{\text{hs}}$
6: **for** $a \leftarrow 0 \ldots (K-1)$ **do**
7: $j \leftarrow K-1$
8: **repeat**
9: $f_{\text{hs}}[a] \leftarrow j$
10: $j \leftarrow j-1$
11: **while** $(j \geq 0) \wedge (P_A(a) \leq P_R(j))$
12: **return** $f_{\text{hs}}$.

# TOČKOVNA SEGMENTACIJA

# Točkovna segmentacija

Upragovljenje

$$f_{\text{threshold}}(a) = \begin{cases} a_0 & \text{for } a < a_{\text{th}} \\ a_1 & \text{for } a \geq a_{\text{th}} \end{cases}$$

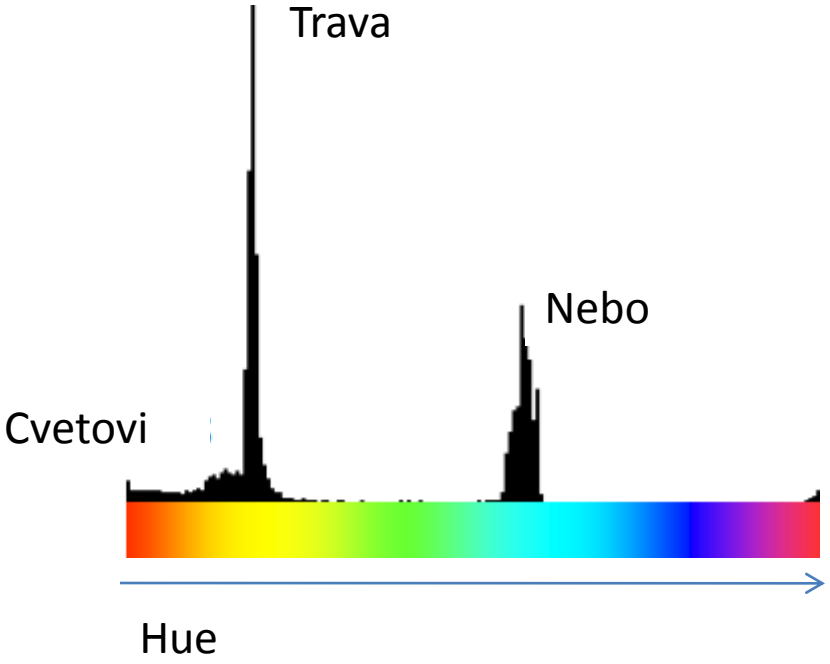



Točkovna segmentacija

- slikovne elemente razdelimo v n razredov glede na vrednost (sivinsko, RGB, Hue v HSV...)
- točkovna segmentacije ne upošteva lege in sosednosti
- regije niso zakjučene
- točkovna segmentacija kot gručenje

# Točkovna segmentacija

Primer



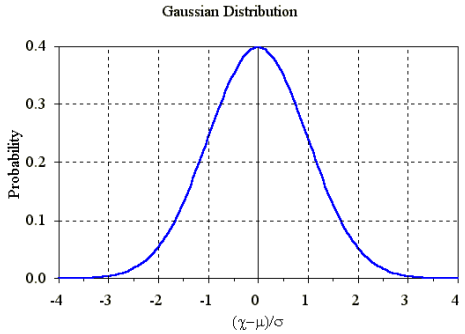
# Točkovna segmentacija

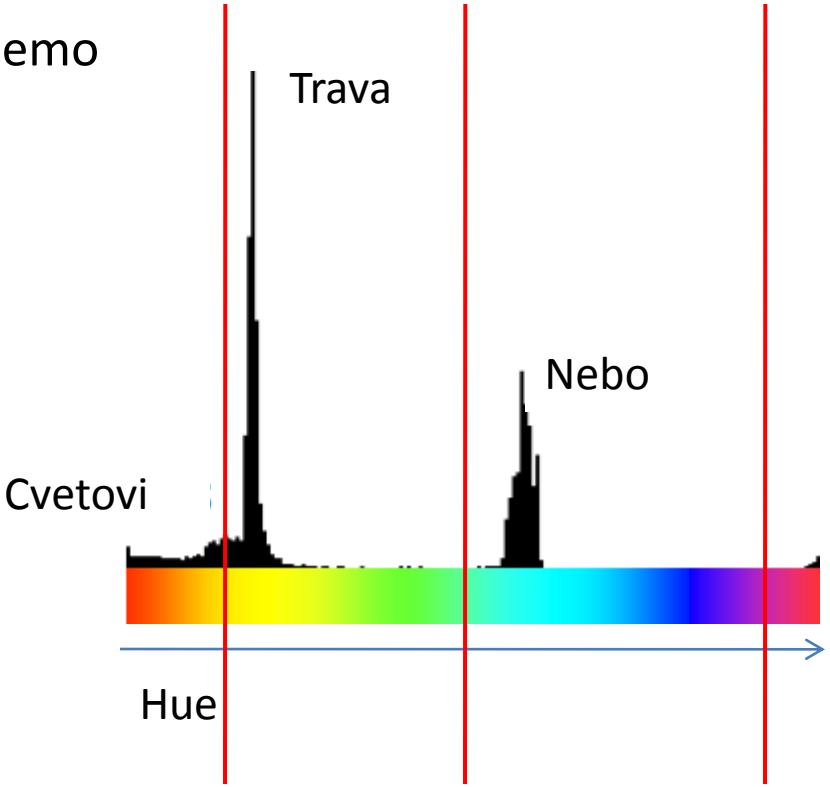
tri gruče slikovnih elementov

če poznamo število gruč, lahko poiščemo
ustrezne lokalne maksimume

potrebujemo še meje
med gručami

gruče lahko modeliramo z
Gaussovo porazdelitvijo



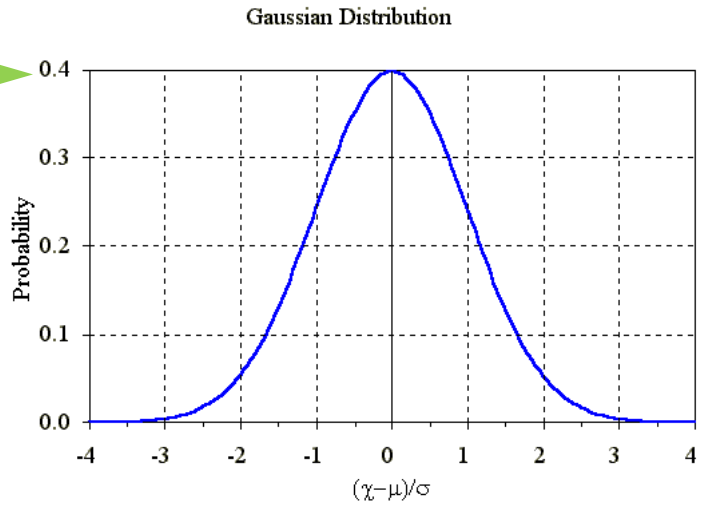



# Točkovna segmentacija

estimacija normalne porazdelitve

poznamo $\mu$, iščemo $\sigma$

$$G(x) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$




poiščemo lahko odmik kjer je $x=h/2$

$$G(x) = \frac{1}{2} G(\mu)$$

$$\frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(x-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(\mu-\mu)^2}{2\sigma^2}}$$

$$e^{\frac{-(x-\mu)^2}{2\sigma^2}} = \frac{1}{2} e^0$$

$$\frac{-(x-\mu)^2}{2\sigma^2} = \ln\left(\frac{1}{2}\right)$$

$$(x-\mu)^2 = -2\sigma^2 \ln\left(\frac{1}{2}\right)$$

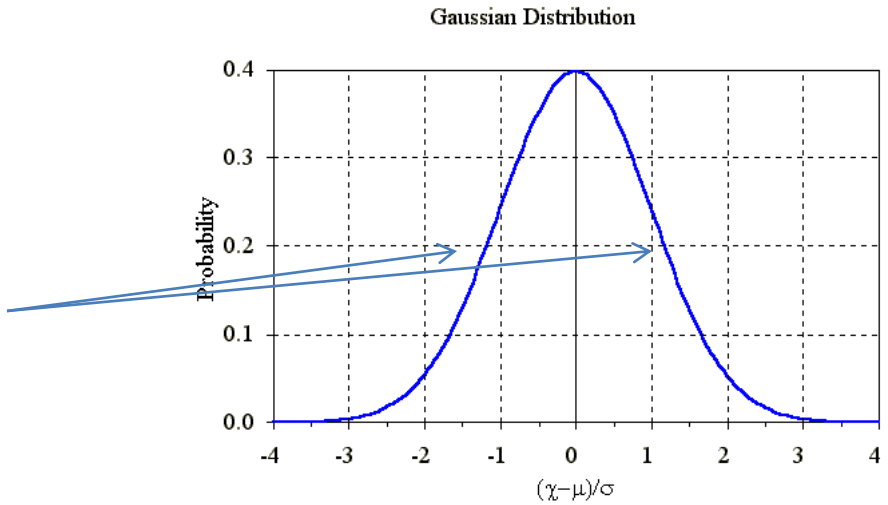
$$x = \mu \pm \sqrt{2\ln(2)}\sigma$$

$$x \approx \mu \pm 1.2\sigma$$

# Točkovna segmentacija

$$x \approx \mu \pm 1.2\sigma$$

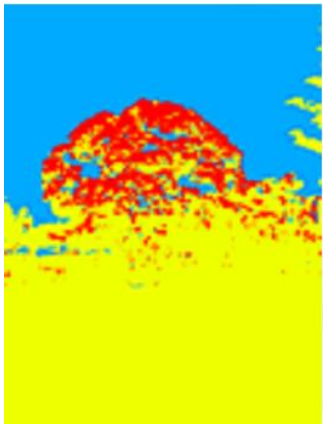
v histogramu poiščemo levi in desni $\mu$- $x_l$ in $\mu$+$x_d$ z vrednostmi $\approx$ h/2



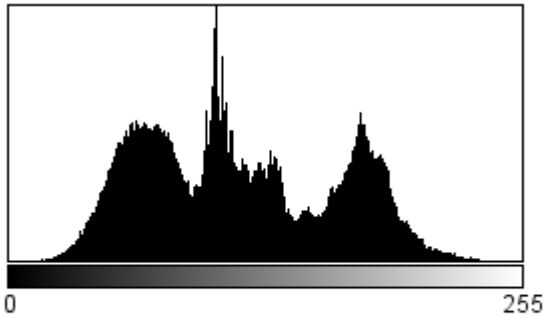

$\sigma = (x_l + x_d)/2.4$

porazdelitev N($\mu$, $\sigma$) lahko uporabimo za klasifikacijo slikovnih elementov

- npr. v razred uvrstimo slikovne elemente v intervalu $\mu$-$\epsilon$ $\sigma$, $\mu$+$\epsilon$ $\sigma$
- ali pa za slikovni element izračunamo verjetnost pripadnosti posameznemu razredu



# K-means segmentacija




Count: 166500
Mean: 114.469
StdDev: 47.756

Min: 6
Max: 250
Mode: 103 (2634)

Pogosto je težko določiti maksimume, predpodstavimo pa lahko število razredov
Uporabimo lahko algoritme samodejnega gručenja, npr. K-means

# K-means segmentacija

K-means gručenje hevristično minimizira varianco v gručah

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} (x_j - \mu_i)^2$$

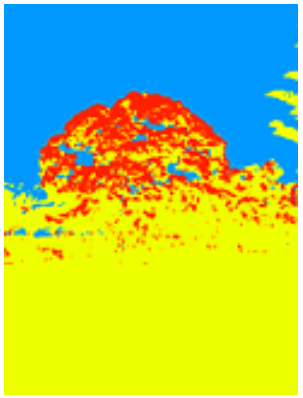
Izberi k začetnih središč gruč $\mu_i$, i=1 ⋯ k

Ponavljaj

vsako od vrednosti $x_i$ pridruži k najbližjemu središču

ko so vse vrednosti razdeljene, izračunaj nova središča gruč

do konvergence



# SEGMENTACIJA REGIJ

# Segmentacija regij

Slika kot množica regij

$R = \cup R_i$ , $R_i \cap R_j = 0$

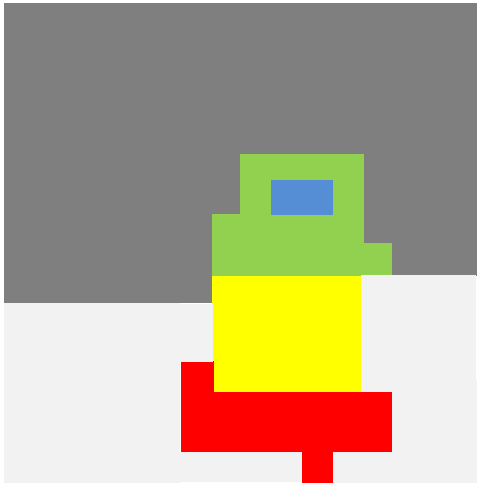
Kriterij za segmentacijo:

$H(R_i)$ homogenost posamezne regije

sosednje regije se razlikujejo

$H(R_i)$ = TRUE

$H(R_i \cap R_j)$ = FALSE

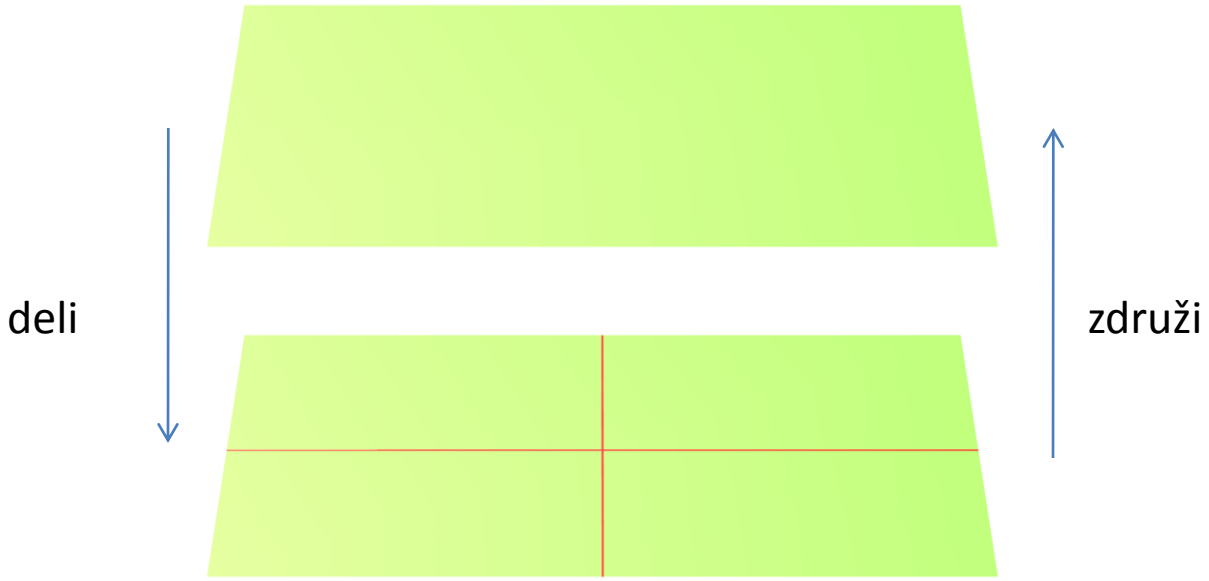


# Deli in združi

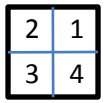
Želimo lokalno konsistentne regije

Princip **deli in združi** (split and merge)

Primer: *quadtree segmentacija*



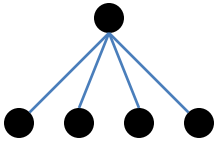
# Quadtree segmentacija

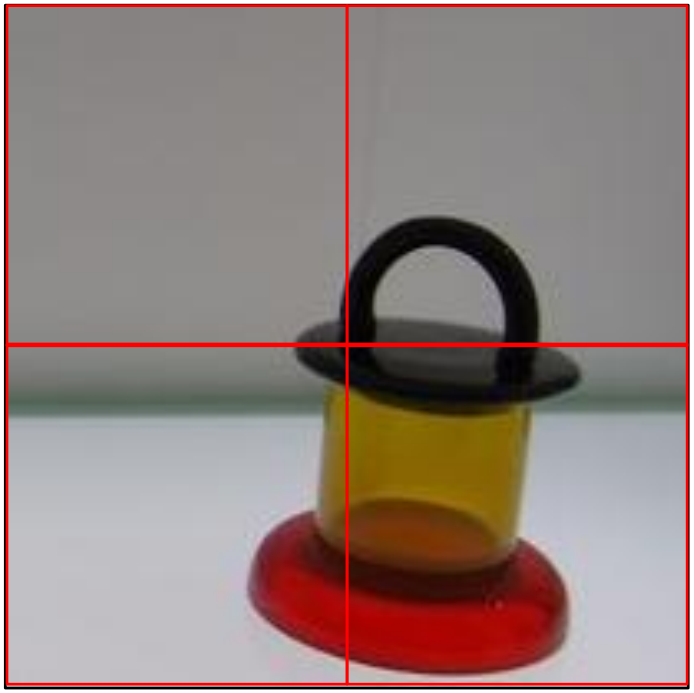



Želimo lokalno konsistentne regije

Princip **deli in združi** (split and merge)

Primer: *quadtree segmentacija*



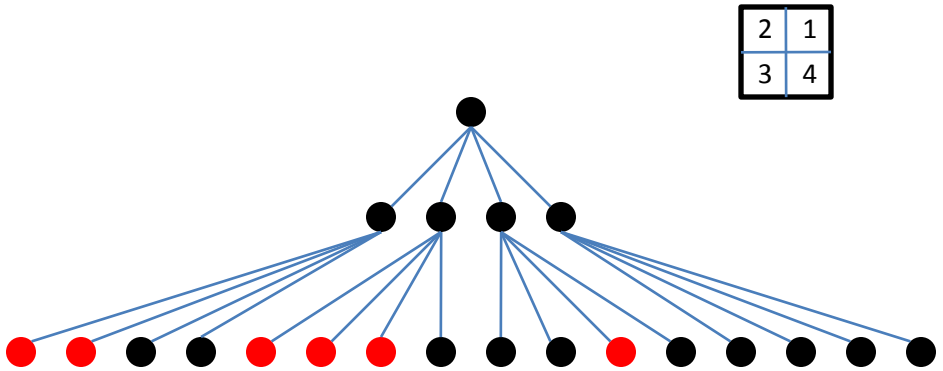


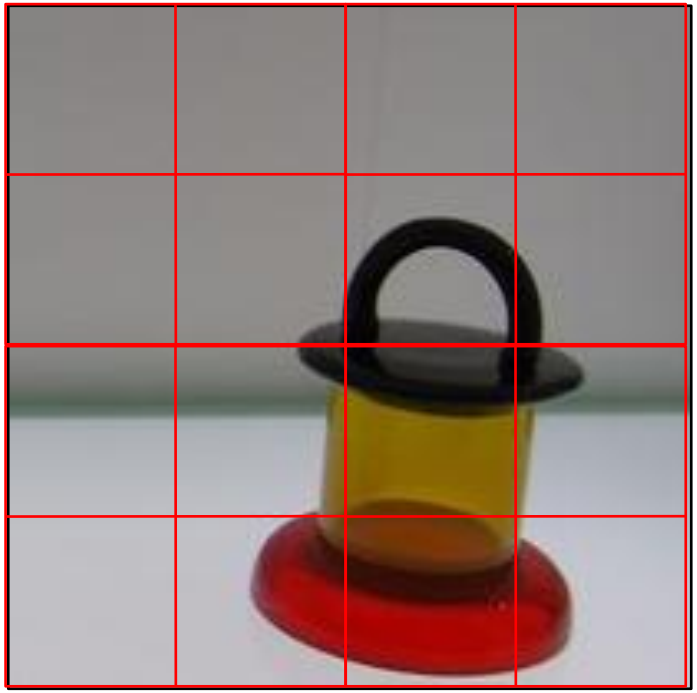
# Quadtree segmentacija

Želimo lokalno konsistentne regije

Princip **deli in združi** (split and merge)

Primer: *quadtree segmentacija*



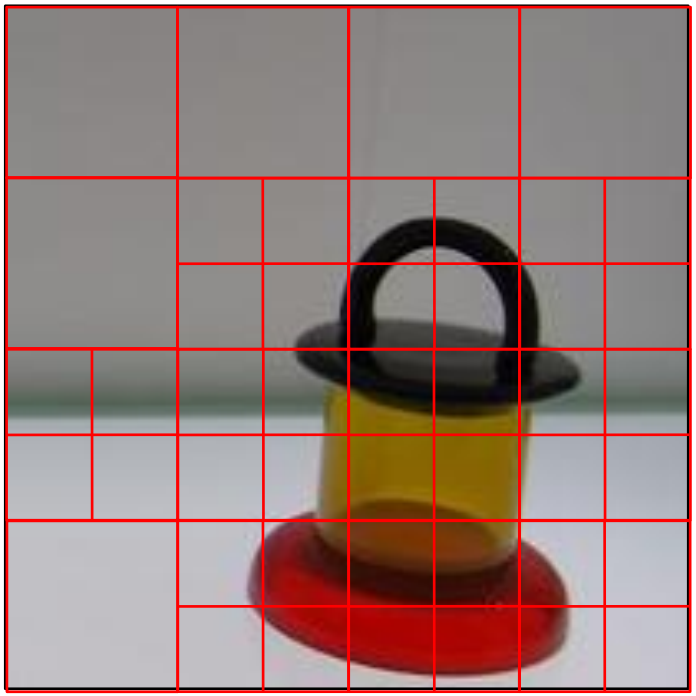



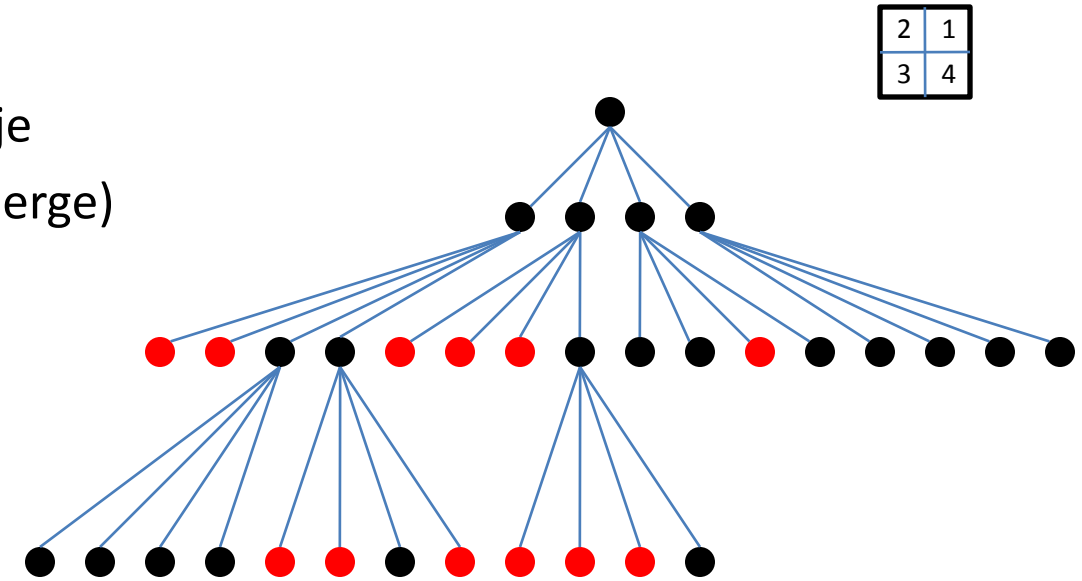
# Quadtree segmentacija

Želimo lokalno konsistentne regije

Princip **deli in združi** (split and merge)

Primer: *quadtree segmentacija*



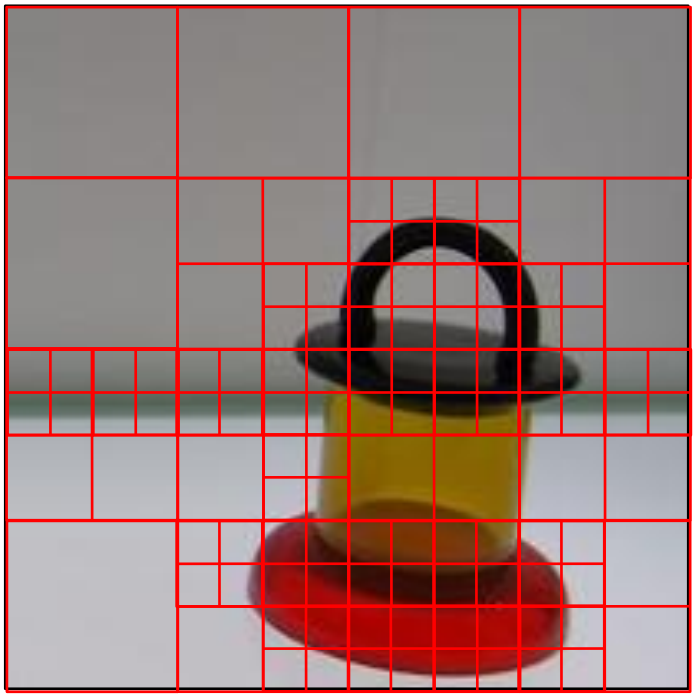


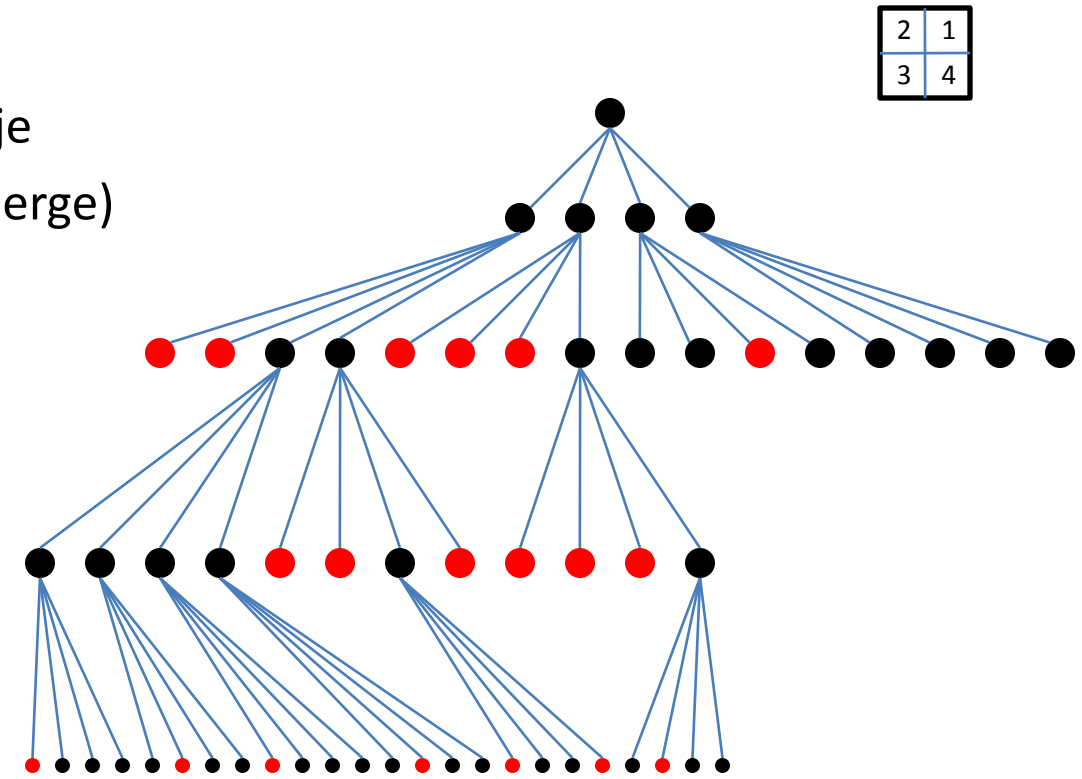
# Quadtree segmentacija

Želimo lokalno konsistentne regije

Princip **deli in združi** (split and merge)

Primer: *quadtree segmentacija*



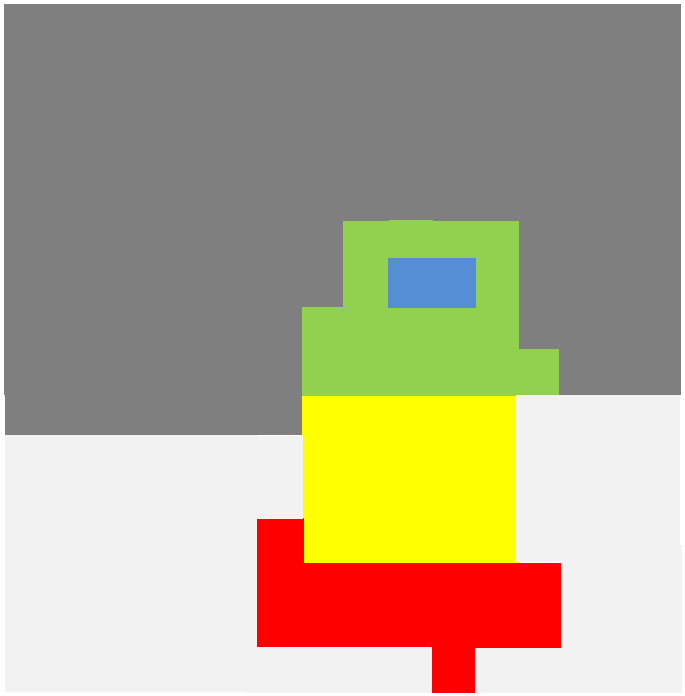


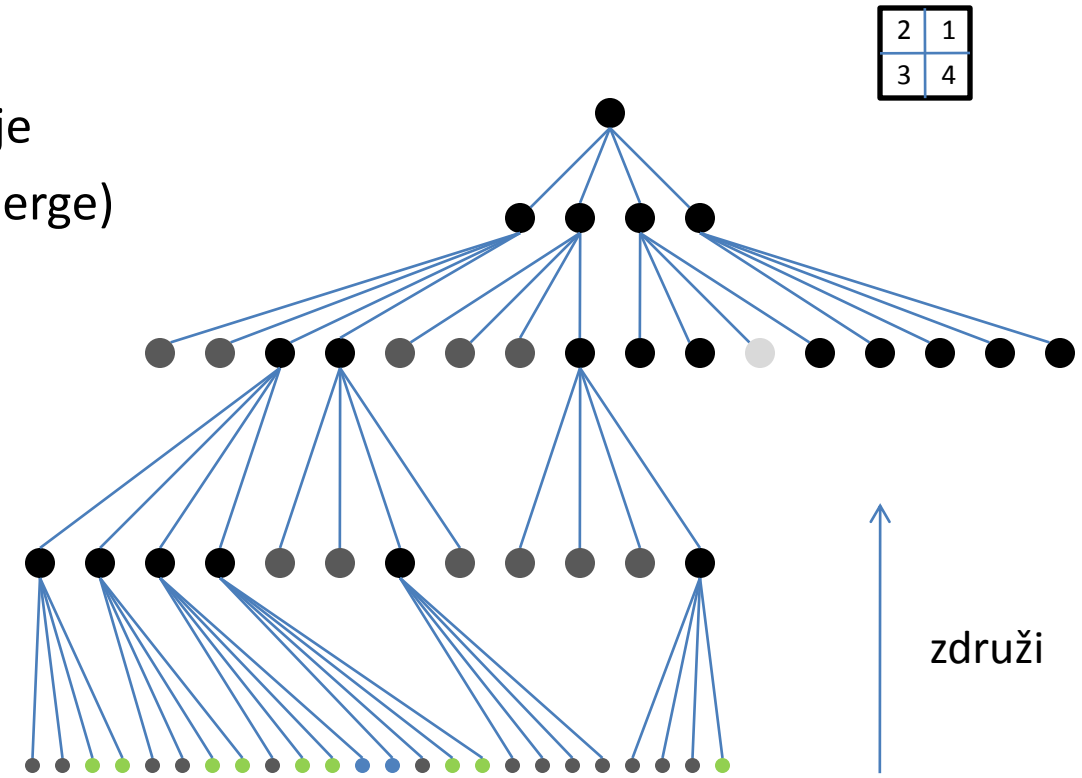
# Quadtree segmentacija

Želimo lokalno konsistentne regije

Princip **deli in združi** (split and merge)

Primer: *quadtree segmentacija*





## Segmentacija v regije

Želimo lokalno konsistentne regije

Princip **deli in združi** (split and merge)

Primer: *quadtree segmentacija*

```
ponavljaj
   razdeli nehomogena področja na 4 podpodročja
   združi podobna homogena podpodročja z istim staršem
dokler obstaja nehomogeno področje ali l<K
združi podobna homogena področja
```

# “Graph cuts” segmentacija

Quadtrees regije se delijo na 4 enake regije

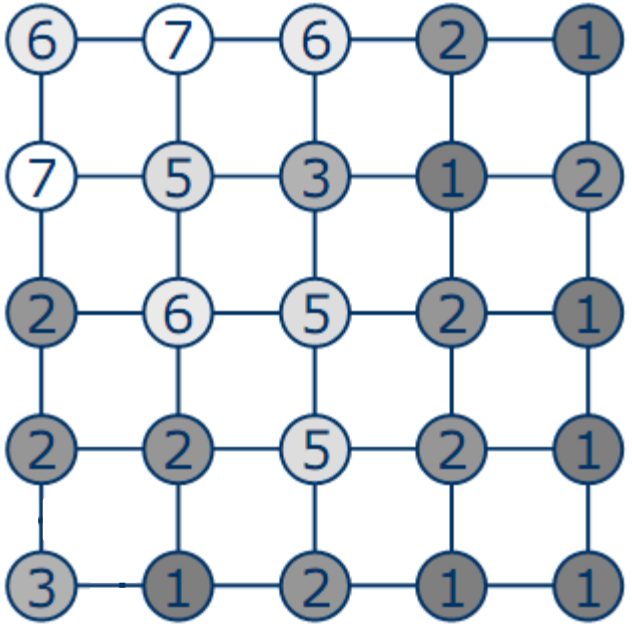
Iskanje sosednjih vozlišč zamudno (dodatna struktura – graf sosednosti)

Regije pa lahko ločujemo na nivoju slikovnega elementa

Primer: **Normalized graph cuts**

Slika kot graf G(V,E)

| | | | | |
|---|---|---|---|---|
| 6 | 7 | 6 | 2 | 1 |
| 7 | 5 | 3 | 1 | 2 |
| 2 | 6 | 5 | 2 | 1 |
| 2 | 2 | 5 | 2 | 1 |
| 3 | 1 | 2 | 1 | 1 |



# “Graph cuts” segmentacija

$c_{ij}$ – kapaciteta povezave

$c_{ij} \approx S(I(i), I(j))$

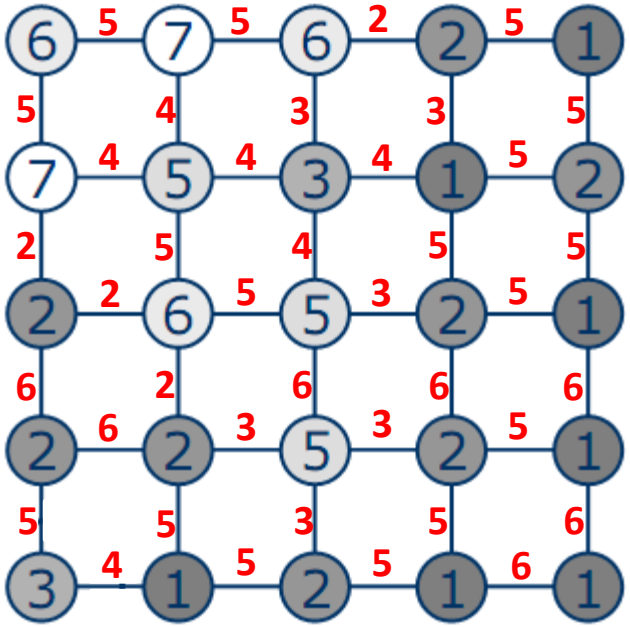
**Rez** *cut*(S, T) deli V na disjunkni množici S in T

kapaciteta reza

$$c(S,T) = \sum_{u \in S, v \in T | (u,v) \in E} c(u,v)$$

Idealna meja bo sliko delila na dve regiji z **minimalnim rezom**

c(S, T) -> min



# Normalized graph cuts

$c_{ij}$ – kapaciteta povezave

$c_{ij} \approx S(I(i), I(j))$ ; S je mera podobnosti

c(S, T) -> min

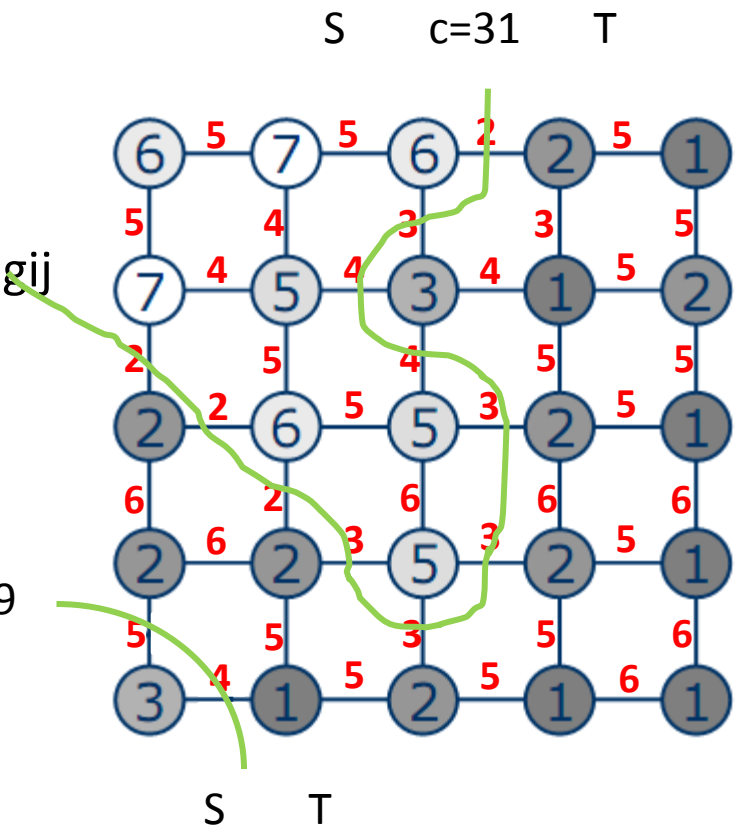
Vendar pogoj favorizira krajše reze

Normalizacija glede na kapacitete regij

$$a(S) = \sum_{u \in S, v \in V | (u,v) \in E} c(u, v)$$

**Normalized cut**

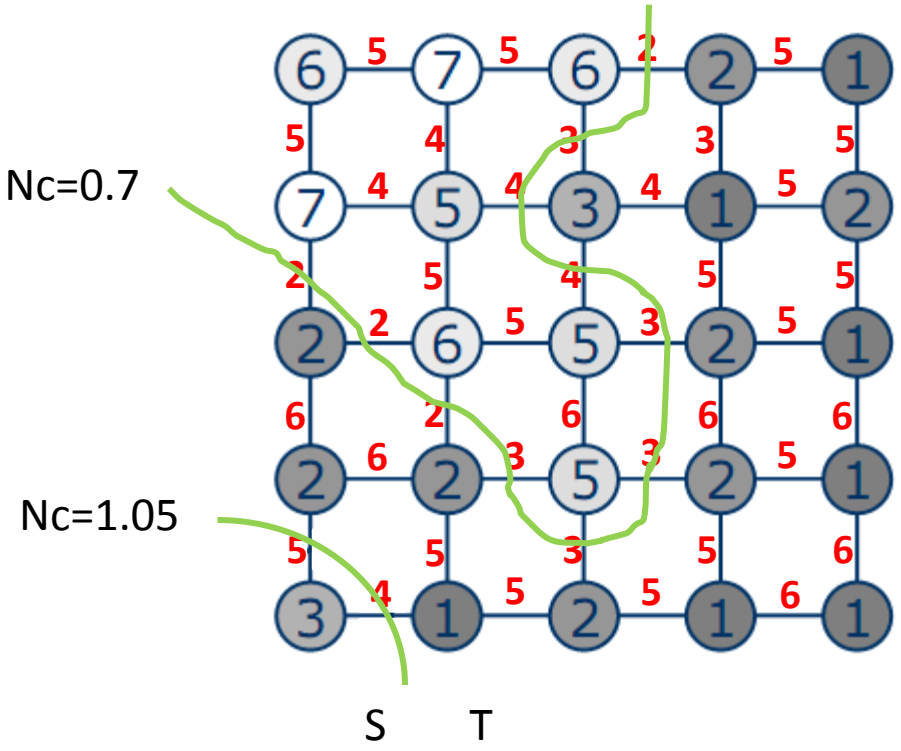
$$Nc(S,T) = c(S,T) \cdot \left( \frac{1}{a(S)} + \frac{1}{a(T)} \right)$$



# Normalized graph cuts

$$Nc(S, T) = c(S,T)(1/a(S) + 1/a(T))$$

**Normalized cut**



# Normalized graph cuts

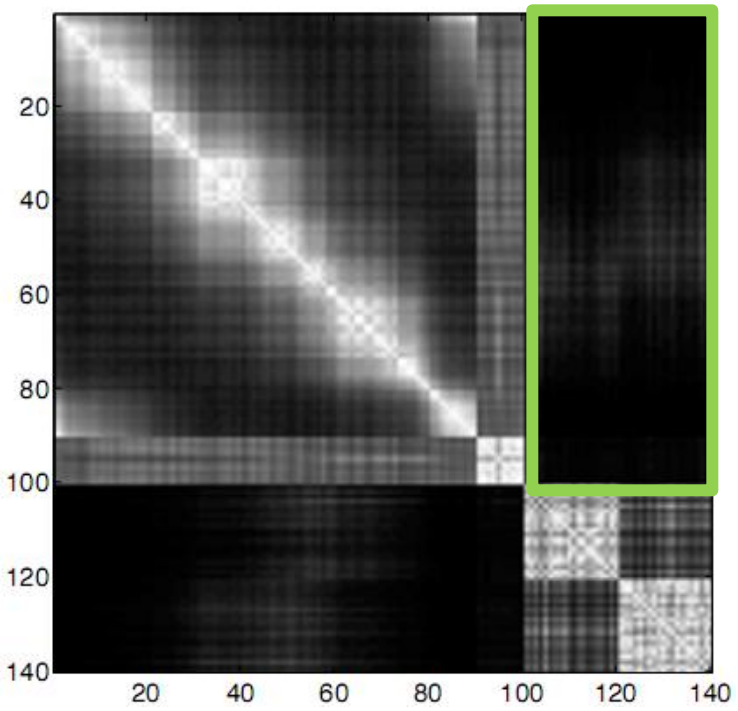
Število možnih rezov

$2^{|V|-2}$

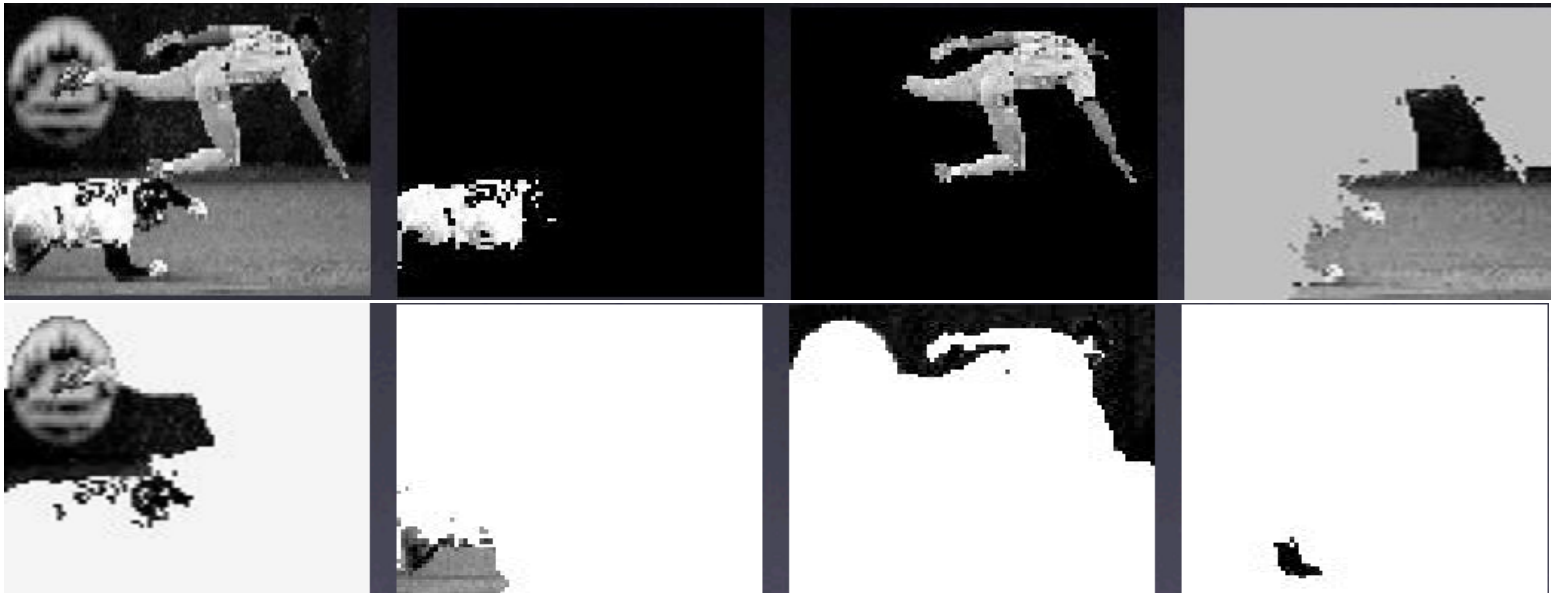
Kompleksnost

$O(2^{|V|})$

Rešitev lahko aproksimiramo z uporabo spektralne analize matrike podobnosti



# Normalized graph cuts



**Normalized Cuts and Image Segmentation**
Jianbo Shi and Jitendra Malik
*IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 888-905, August 2000.

# DISKRETNA FOURIEROVA TRANSFORMACIJA

# Diskretna fourierova transformacija

- **Slikovni vektorski prostor (Pixel space)**

$$\Re^{MN}$$

- Slika je linearna kombinacija baznih vektorjev (matrični vektorji)

$$I(x,y) = a_1 \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + a_2 \cdot \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + a_3 \cdot \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} + \ldots + a_n \cdot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Bazne funkcije razpenjajo prostor vseh slik.
  Prostor lahko transformiramo tako, da bodo podatki (slike) predstavljene v drugi bazi.

$$\vec{a} = UI(:)$$

$I(:)$ je vektorski zapis matrike

$U$ je matrični zapis nove baze

## Diskretna fourierova transformacija

- $U$ je ortonormirana vektorska baza
- Diskretni fourierov transform: U je baza harmoničnih komponent
- Enodimenzionalni transform:

$$I(x) \rightarrow F(u)$$

$$F(u) = \frac{1}{N} \sum_{x=0}^{N-1} I(x) \cdot e^{-j\frac{2\Pi}{n}ux}$$

$$I(x) = \frac{1}{N} \sum_{u=0}^{N-1} F(u) \cdot e^{-j\frac{2\Pi}{n}ux}$$

# Diskretna fourierova transformacija

- Implementacija

$$I(x) \rightarrow F(u)$$

$$F(u) = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} I(x) \cdot e^{-j\frac{2\Pi}{N}ux} =$$

$$= \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \underbrace{I_{\mathrm{Re}}(x) + i \cdot I_{\mathrm{Im}}(x)}_{I(x)} \cdot \left[ \underbrace{\cos \cdot \left( 2\pi \frac{ux}{N} \right)}_{C_u^N(x)} - i \cdot \underbrace{\sin \cdot \left( 2\pi \frac{ux}{N} \right)}_{S_u^N(x)} \right]$$

## Discrete Fourier Transform

```
Complex[] DFT(Complex[] img, boolean forward) {
   int N = img.length;
   double s = 1 / Math.sqrt(N);
   Complex[] F = new Complex[N];
   for (int u = 0; u < N; u++) {
       double sumRe = 0;
       double sumIm = 0;
       double phim = 2 * Math.PI * u / N;
       for (int x = 0; x < N; x++) {
            double imgRe = img[u].re;
            double imgIm = img[u].im;
            double cosw = Math.cos(phim * u);
            double sinw = Math.sin(phim * u);
            if (!forward)
                sinw = -sinw;
            sumRe += imgRe * cosw + imgIm * sinw;
            sumIm += imgIm * cosw – imgRe * sinw;
       }
       F[u] = new Complex(s * sumRe, s * sumIm);
   }
   return F;
}
```

## class Complex

```
class Complex {
   double re, im;

   Complex(double re, double im) {
       this.re = re;
       this.im = im;
   }
}
```

# Diskretna fourierova transformacija

- 2-D DFT je definiran kot

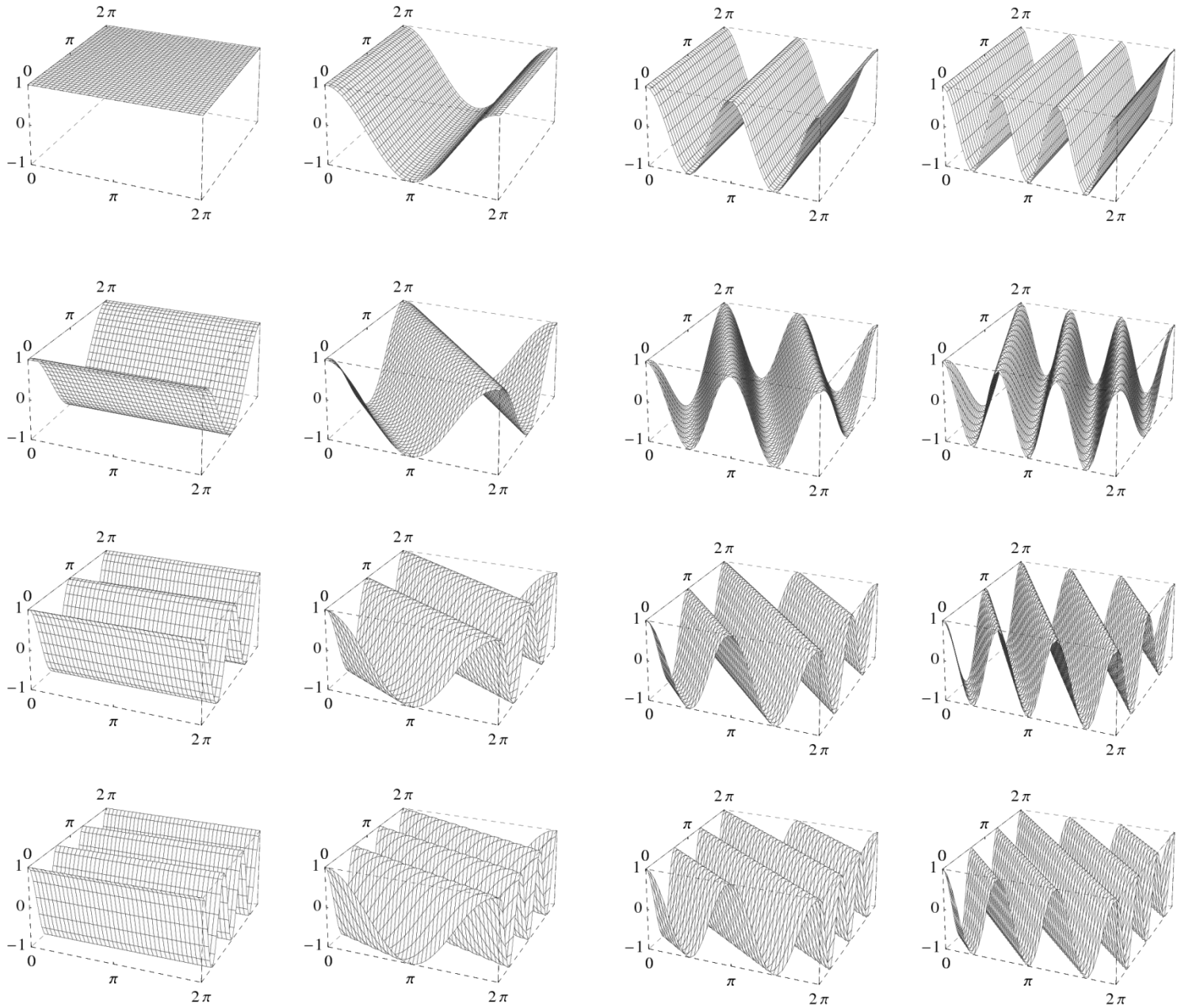
$$I(x,y) \rightarrow F(u,v)$$

$$F(u,v) = \frac{1}{MN} \sum_{y=0}^{M} \sum_{x=0}^{N} I(x,y) \cdot e^{-j2\Pi(\frac{ux}{N}+\frac{vy}{M})} =$$

$$= \frac{1}{MN} \sum_{y=0}^{M} \sum_{x=0}^{N} \underbrace{I_{\mathrm{Re}}(x,y) + i \cdot I_{im}(x,y)}_{I(x,y)} \cdot \underbrace{\left[ \cos\left[ 2\pi \cdot \left( \frac{ux}{M} + \frac{vy}{N} \right) \right]}_{C_{u,v}^{M,N}(x,y)} + i \cdot \underbrace{\sin\left[ 2\pi \left( \frac{ux}{M} + \frac{vy}{N} \right) \right]}_{S_{u,v}^{M,N}(x,y)} \right]$$

## 2-D harmonične komponente

$$C_{u,v}^{M,N}(x,y)$$



# Diskretna fourierova transformacija

2-D transform izračunamo **separabilno** iz 1-D transforma

$$I(x,y) \rightarrow F(u,v)$$

$$F(u,v) = \frac{1}{MN} \sum_{y=0}^{M} \sum_{x=0}^{N} I(x,y) \cdot e^{-j2\Pi(\frac{ux}{N}+\frac{vy}{M})} =$$

$$= \frac{1}{M} \sum_{y=0}^{M} \frac{1}{N} \sum_{x=0}^{N} I(x,y) \cdot e^{-j2\Pi\frac{ux}{N}} \cdot e^{-j2\Pi\frac{vy}{M}} =$$

$$= \frac{1}{M} \sum_{y=0}^{M} F(u,y) \cdot e^{-j2\Pi\frac{vy}{M}}$$

# Diskretna fourierova transformacija

2-D transform izračunamo **separabilno** iz 1-D transforma

```
for y = 0 ... N-1
        I(.,y) = DFT(I(.,y))
for x = 0 .. M-1
        I(x,.) = DFT(I(x,.))
```

## Lastnosti diskretnega 2-D Fourierovega transforma

- Periodična funkcija

$F(u,v) = F(u + k_1M, v + k_2N)$

- če je I(x,y) realna, je absolutna vrednost $|F(u,v)|$ sredinsko simetrična

$|F(u,v)| = |F(-u, -v)|$

- Realna in imaginarna komponenta $F_{Re}$ , $F_{Im}$

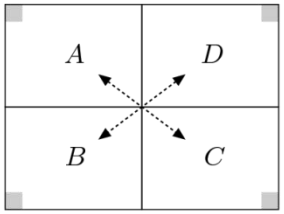
- Fazni in močnostni spekter

- Navadno prikazujemo $|F(u, v)|$ s premikom spektra v sredino slike

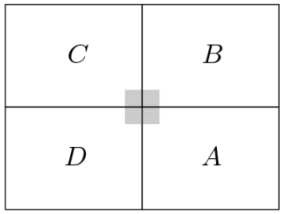
$-M / 2 <= u <= (M – 1) / 2$

$-N / 2 <= v <= (N - 1) / 2$

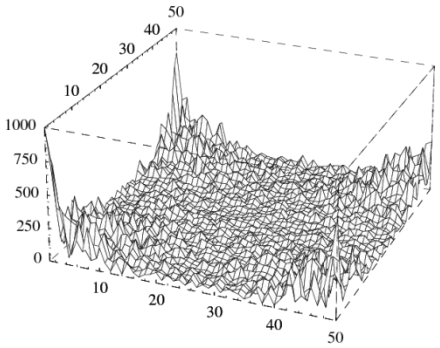
# Lastnosti diskretnega 2-D Fourierovega transforma



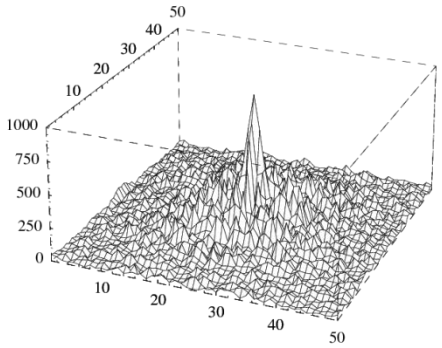

(a)



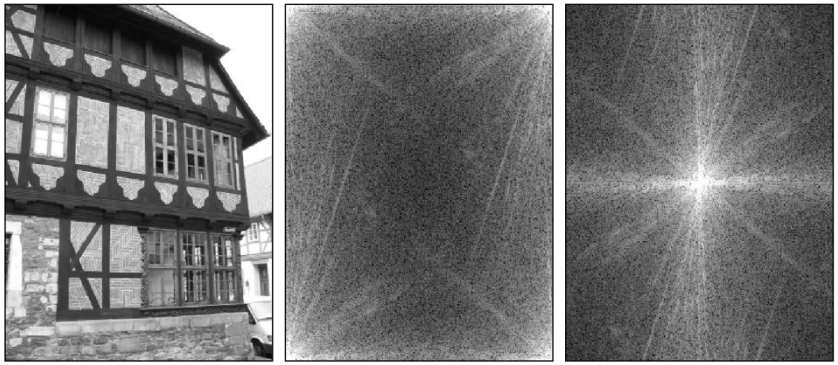

(b)




(c)




(d)

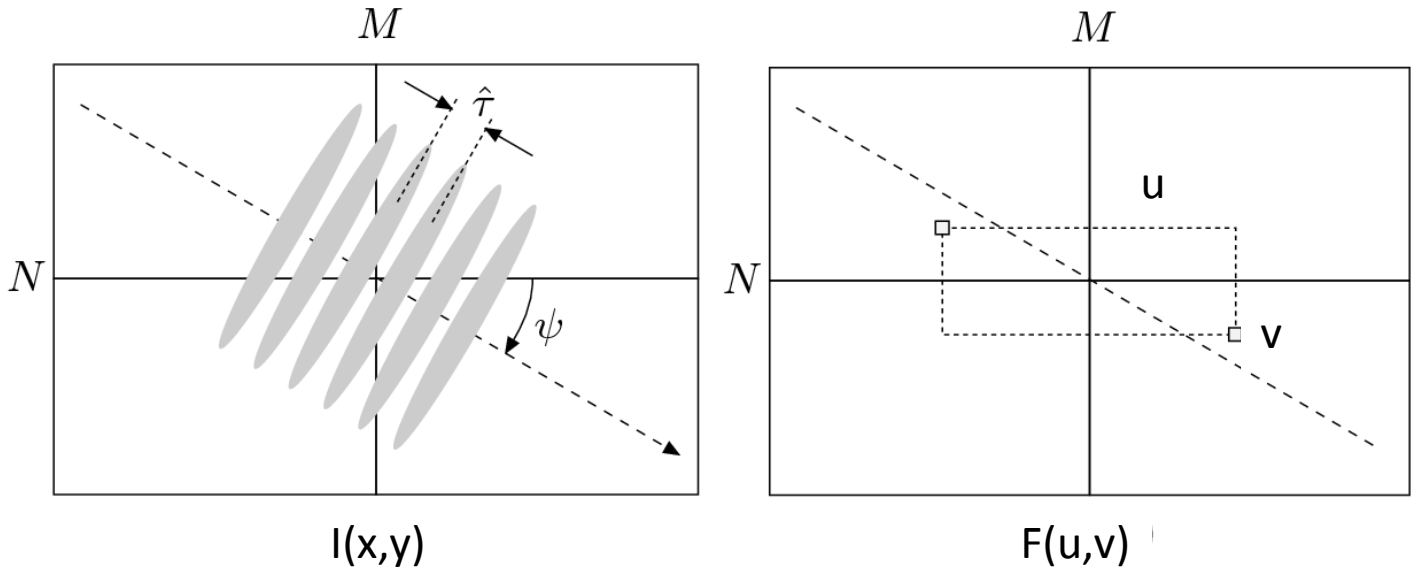


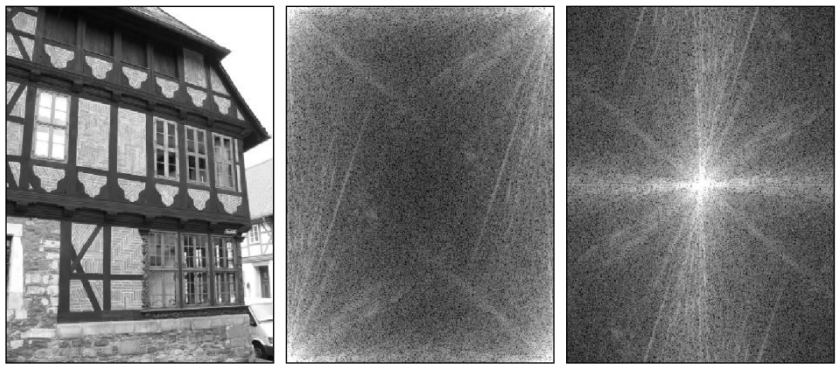
(a) (b) (c)

# Interpretacija 2-D Fourierovega transforma

$$\hat{f} = \frac{1}{\hat{\tau}}$$

$$(u,v) = \pm \hat{f} \cdot (M\cos\psi, N\sin\psi)$$

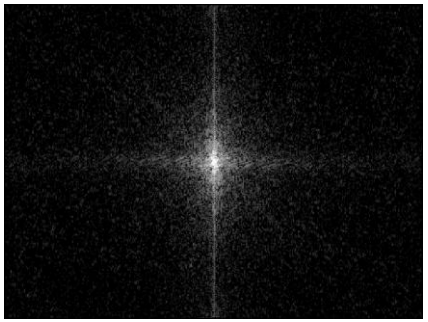


(a) (b) (c)

# Lastnosti diskretnega 2-D Fourierovega transforma

- Definicija transforma predvideva periodično I(x, y)



## Lastnosti 2D diskretnega Fourierovega transforma

- Rotacija

- Linearna kombinacija

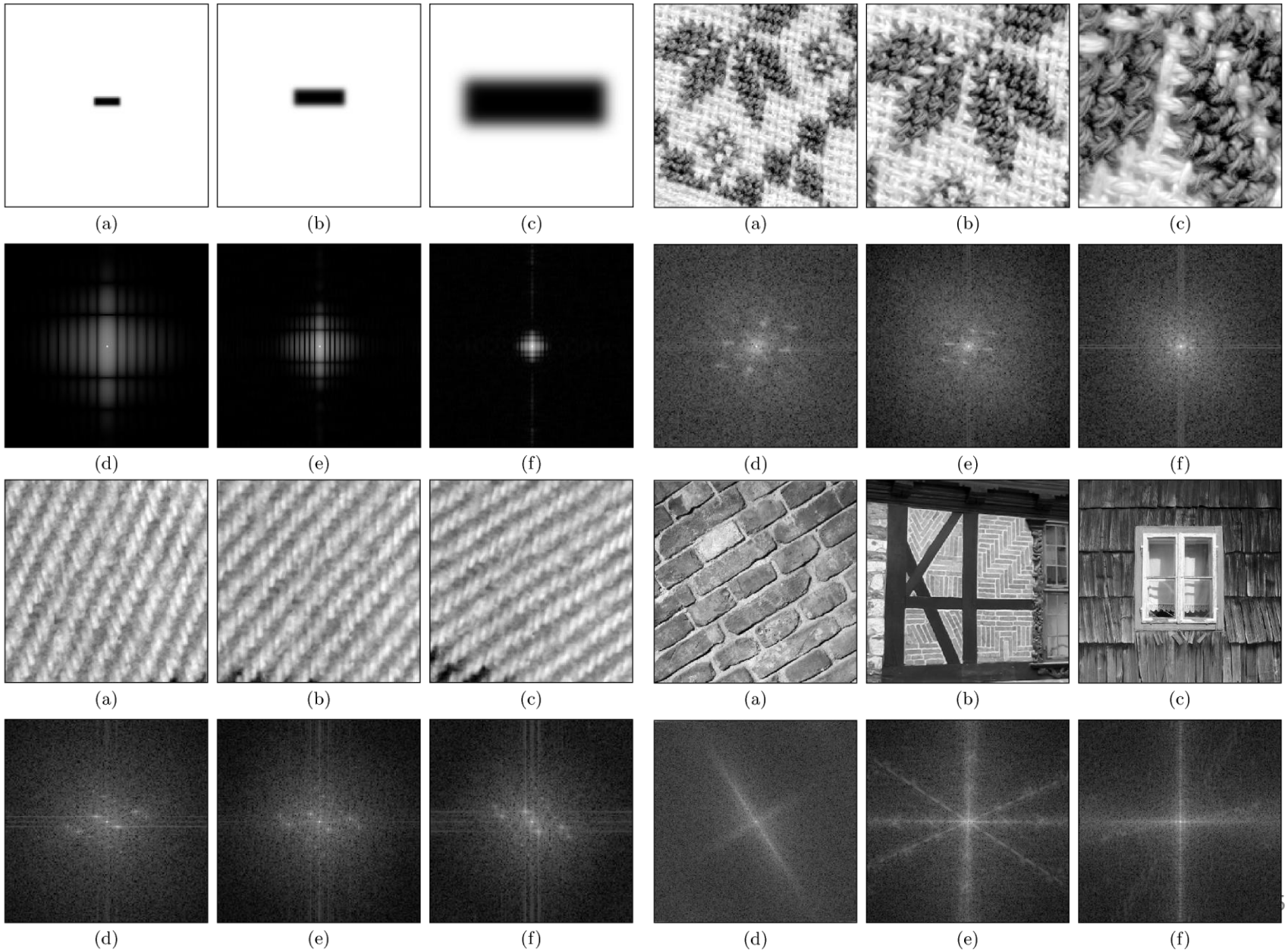
$$k_1 I_1(x,y) + k_2 I_2(x,y) \Leftrightarrow k_1 F_1(u,v) + k_2 F_2(u,v)$$

- Translacija

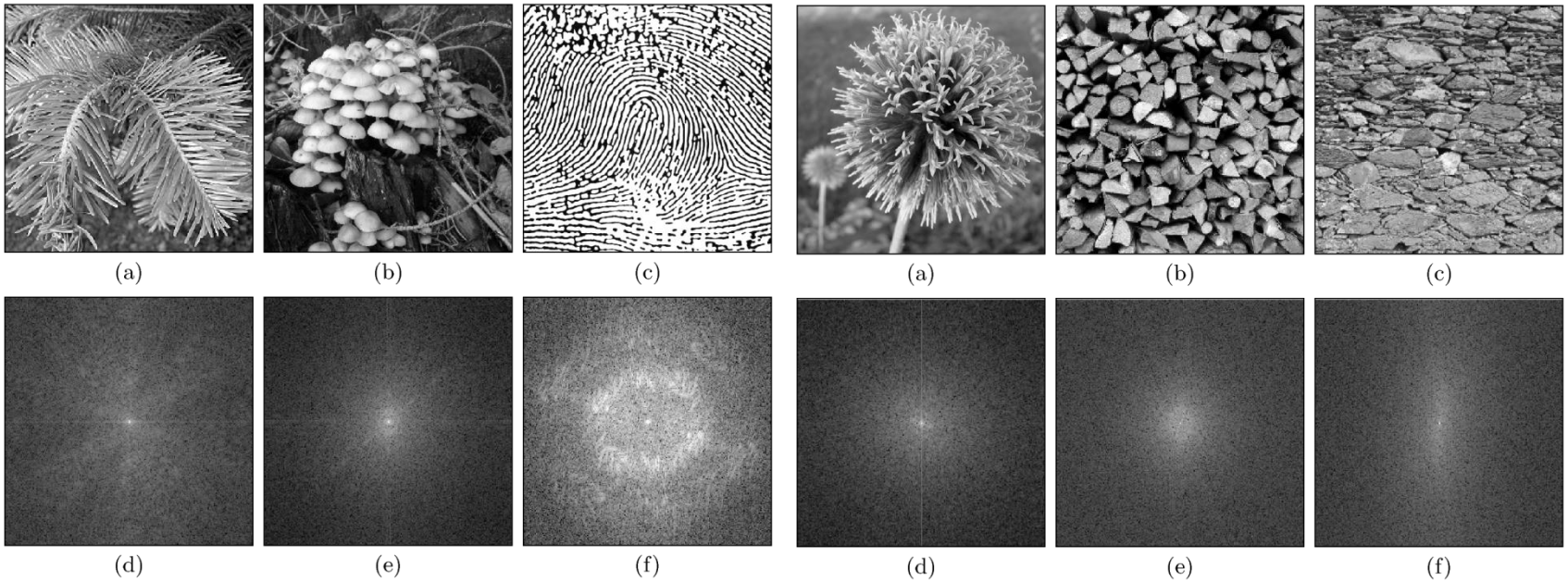
$$I(x - x_0, y - y_0) \Leftrightarrow F(u,v) e^{-j2\Pi(\frac{ux_0}{N} + \frac{vy_0}{M})}$$

- **Demonstriraj!**

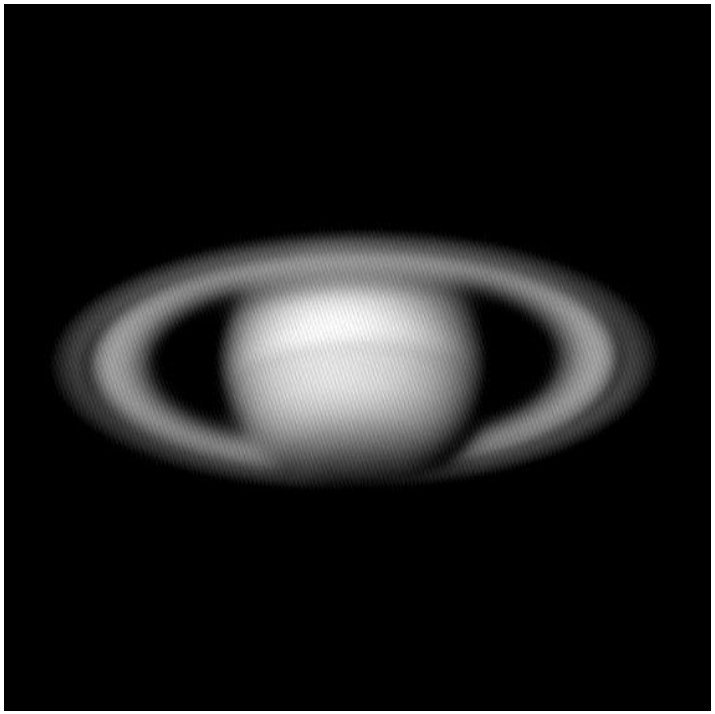
# Primeri 2-D DFT

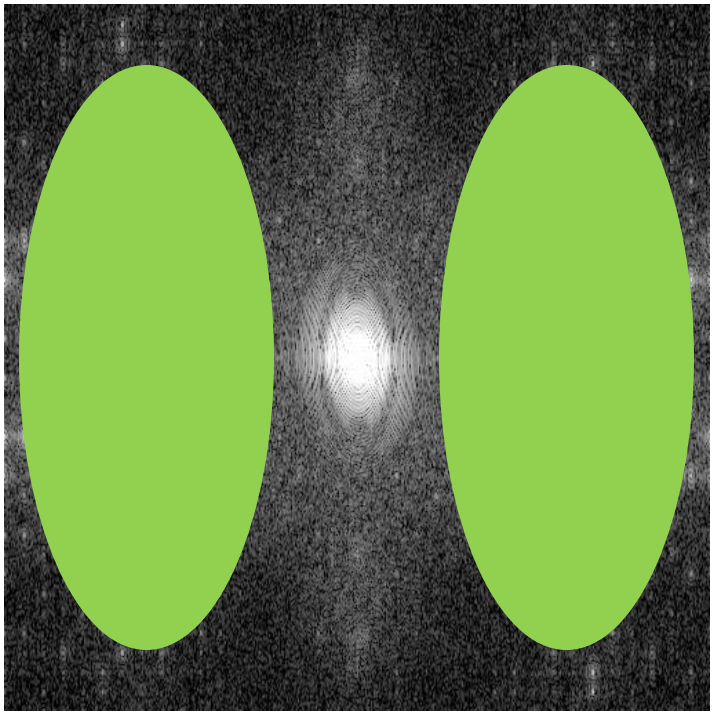


# Primeri 2-D DFT



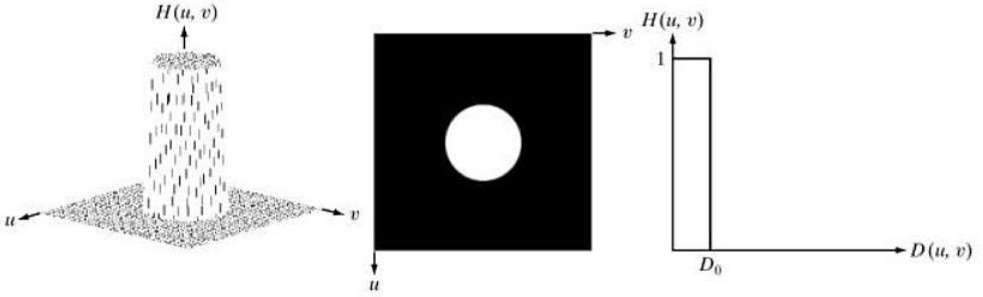
# Inverzni DFT



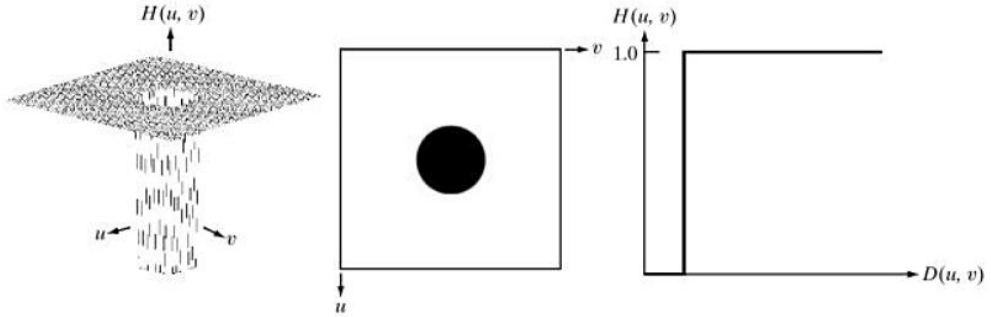


**Idealni filtri**

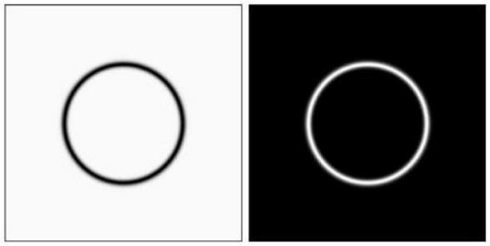
- Nizko-propustni



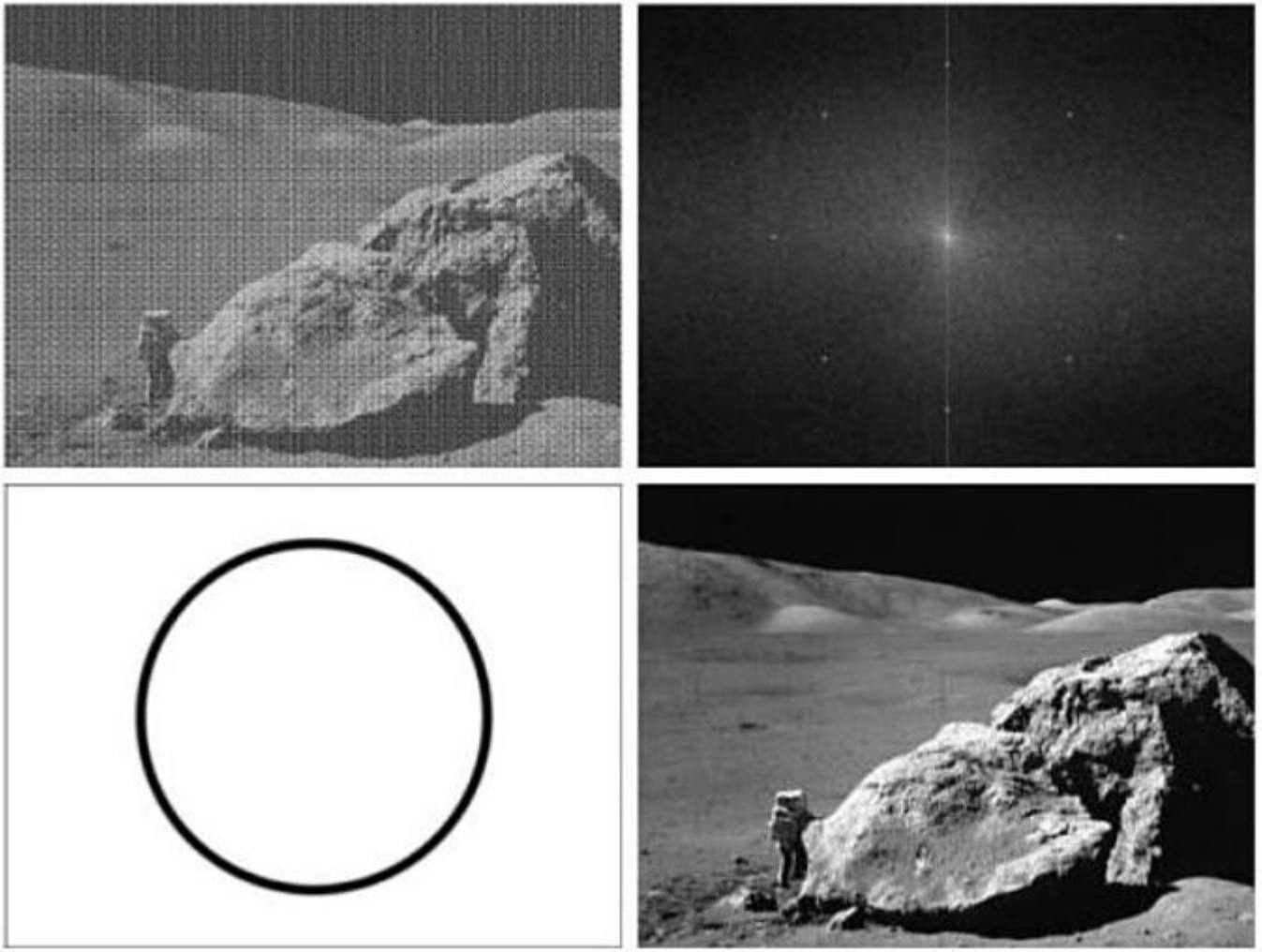

- Visoko-propustni




- Pasovno-propustni



# Idealni filtri



**Konvolucijski teorem**

- Množenje v frekvenčnem prostoru = konvolucija v slikovnem prostoru

$$I \otimes H = F^{-1}(F(I) \cdot F(H))$$

- Konvolucija

$$I_H(x,y) = I \otimes H = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} H(h,k) I(x-h, y-k)$$

## Filtriranje s konvolucijo

$$I_H(x,y) = I * H = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} H(h,k) I(x-h, y-k)$$

linearna konvolucija

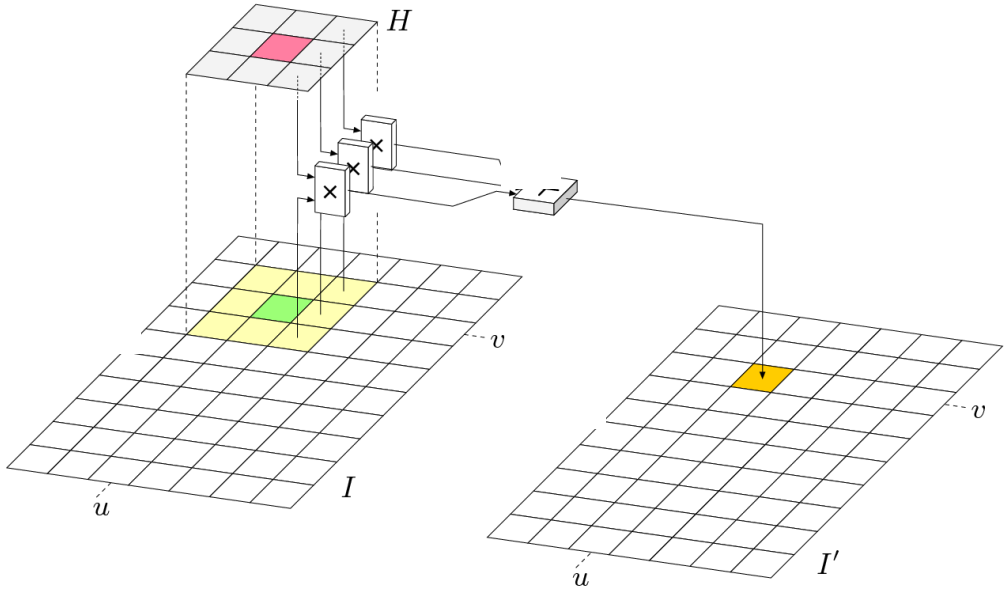
$$I_H(x,y) = I \otimes H = \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} H(h,k) I(x+h, y+k)$$

linearna korelacija

Jedro H dimenzije m x m

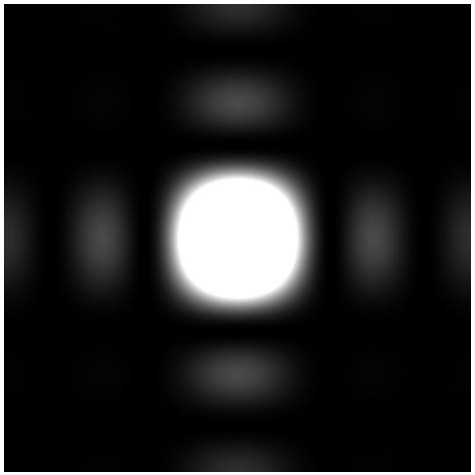
Primer: filter povprečenja

$$H_{avg} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$



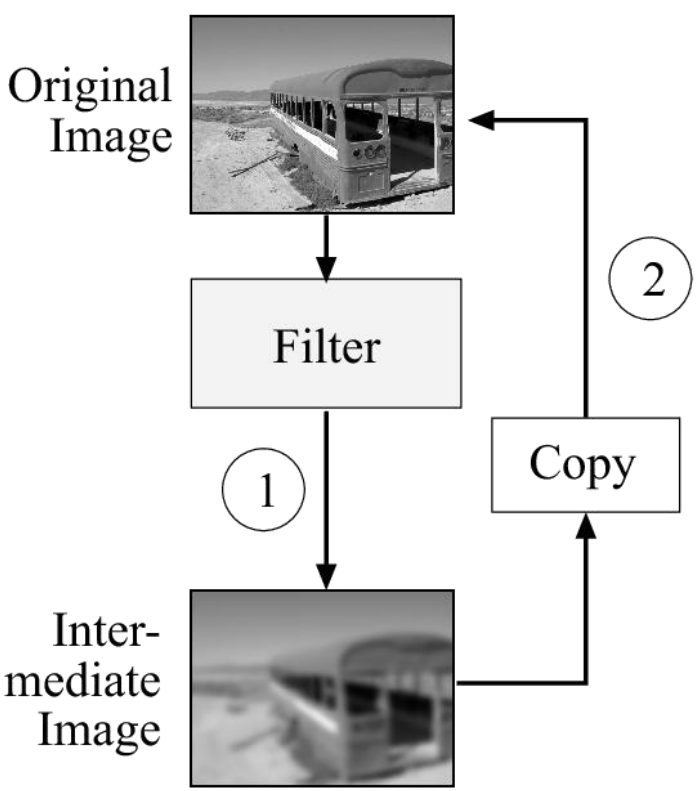
## Filtriranje s konvolucijo

$$H_{avg} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

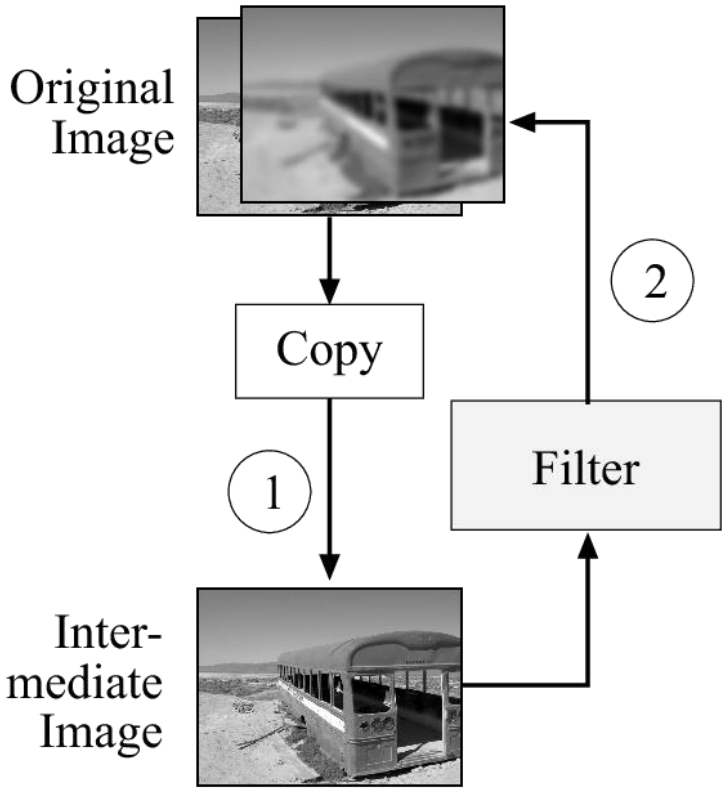


$F(H_{avg})$

**Realizacija linearnega filtriranja**




Version A




Version B

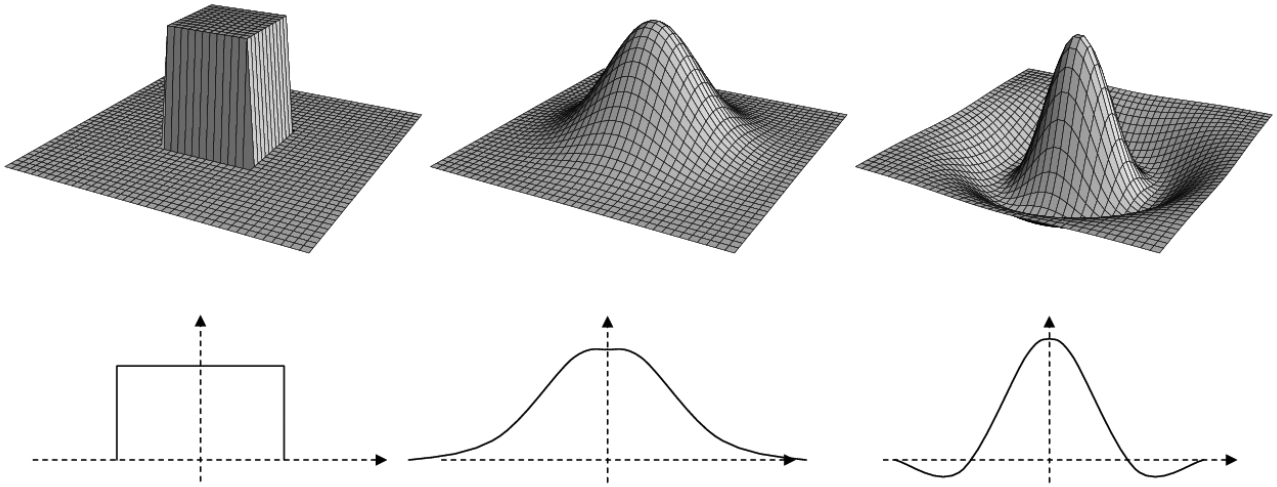
## 3x3 average filter

```
import ij.*;
import ij.pluginFilter.PluginFilter;
import ij.process.*;

public class Filter_Avg implements PlugInFilter {
    public void run(ImageProcessor img) {
        int w = img.getWidth();
        int h = img.getHeight();
        ImageProcessor copy = img.duplicate();

        for (int v = 1; v < h-2; v++) {
            for (int u = 1; u < w-2; u++) {
                int sum = 0;
                for (int j = -1; j <= 1; j++) {
                    for (int i = -1; i <= 1; i++) {
                        int p = copy.getPixel(u+i, v+j);
                        sum = sum + p;
                    }
                }
                int q = (int) Math.round(sum/9.0);
                img.putpixel(u, v, q);
            }
        }
    }
}
```

## Smoothing filter

```
public void run(ImageProcessor img) {
    int w = img.getWidth();
    int h = img.getHeight();
    double [][] filter = {
        {0.075, 0.125, 0.075}.
        {0.125, 0.200, 0.125},
        {0.075, 0.125, 0.075}
    };
    ImageProcessor copy = img.duplicate();

    for (int v = 1; v <= h-2; v++) {
        for (int u = 1; u <= w-2; u++) {
            double sum = 0;
            for (int j = -1; j <= 1; j++) {
                for (int i = -1; i <= 1; i++) {
                    int p = copy.getPixel(u+i, v+j);
                    double c = filter[j+1][i+1];
                    sum = sum + c * p;
                }
            }
            int q = (int) Math.round(sum);
            img.putpixel(u, v, q);
        }
    }
}
```

# Primeri linearnih filtrov



| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |

Povprečni filter

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 2 | 1 | 0 |
| 1 | 3 | 5 | 3 | 1 |
| 2 | 5 | 9 | 5 | 2 |
| 1 | 3 | 5 | 3 | 1 |
| 0 | 1 | 2 | 1 | 0 |

Gaussov filter

| | | | | |
|---|---|---|---|---|
| 0 | 0 | -1 | 0 | 0 |
| 0 | -1 | -2 | -1 | 0 |
| -1 | -2 | 16 | -2 | -1 |
| 0 | -1 | -2 | -1 | 0 |
| 0 | 0 | -1 | 0 | 0 |

Laplaceov filter

## Linearnost, asociativnost in separabilnost

$$(s \cdot I) * H \;=\; I * (s \cdot H) \;=\; s \cdot (I * H)$$

Linearnost

$$(I_1 + I_2) * H \;=\; (I_1 * H) + (I_2 * H)$$

$$A * (B * C) = (A * B) * C$$

Asociativnost

$$H = H_1 * H_2 * \ldots * H_n$$

$$\begin{aligned} I * H &= I * (H_1 * H_2 * \ldots * H_n) \\ &= \big(\ldots((I * H_1) * H_2) * \ldots * H_n\big) \end{aligned}$$

Separabilnost

**Linearnost, asociativnost in separabilnost in x/y separabilnost**

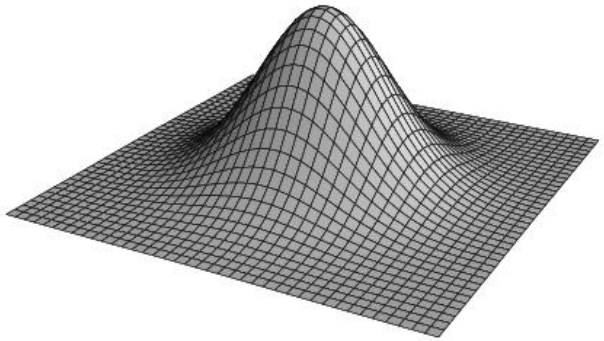
$$I' \leftarrow (I * H_x) * H_y = I * \underbrace{(H_x * H_y)}_{H_{xy}}$$ x/y separabilnost

$$H_x = \begin{bmatrix} 1 & 1 & \mathbf{1} & 1 & 1 \end{bmatrix} \quad \text{and} \quad H_y = \begin{bmatrix} 1 \\ \mathbf{1} \\ 1 \end{bmatrix}$$

$$H_{xy} = H_x * H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & \mathbf{1} & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

# Gaussov filter

$$G_\sigma(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$



x/y separabilnost

$$G_\sigma(x,y) \quad = \quad e^{-\frac{x^2+y^2}{2\sigma^2}} \quad = \quad e^{-\frac{x^2}{2\sigma^2}} \cdot e^{-\frac{y^2}{2\sigma^2}} \quad = \quad g_\sigma(x) \cdot g_\sigma(y)$$

Zaradi diskretnega vzorčenja mora biti okno dovolj veliko; tipično +/- 3$\sigma$

## 1D Gaussian Kernel

```
float[] makeGaussianKernel1D(double sigma) {

    int center = (int)(3.0*sigma);
    float[] kernel = new float[2*center+1];

    double sigma2 = sigma * sigma;
    for (int i = 0; i < kernel.lenght; i++) {
           double r = center – i;
           kernel[i] = (float) Math.exp(-0.5 * (r*r) / sigma2);
    }
    return kernel;
}
```