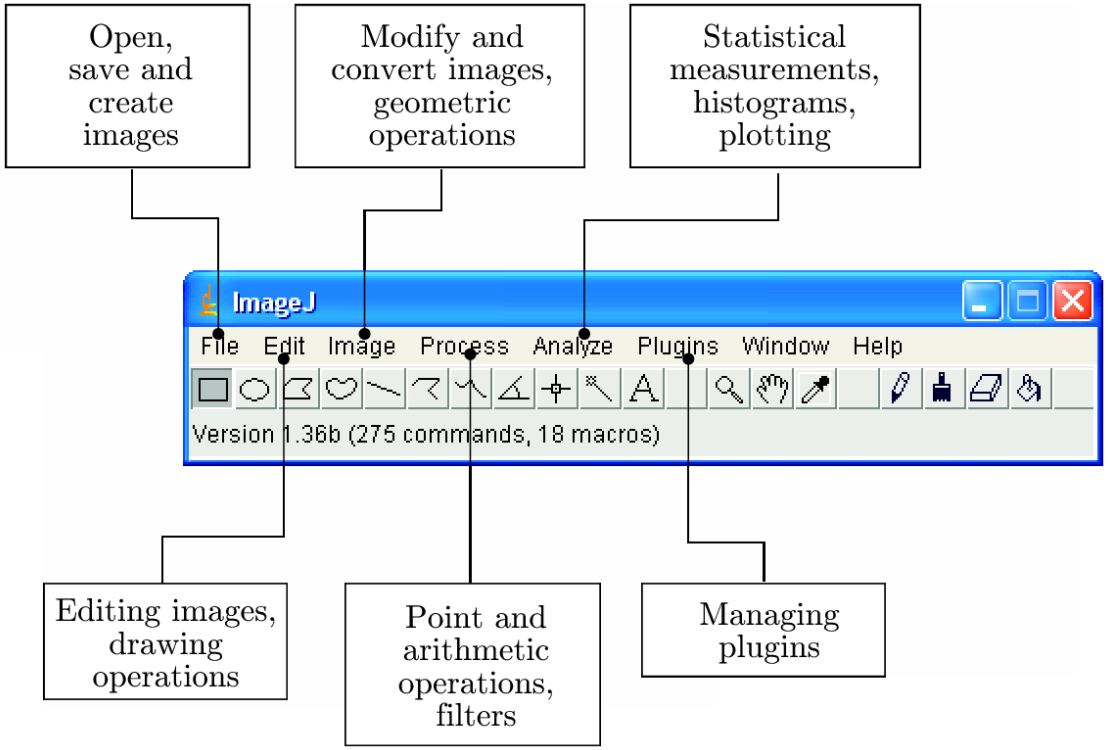
Vaje pri predmetu Računalniško zaznavanje 08/09

ImageJ

- http://rsb.info.nih.gov/ij/download.html
- Wayne Rasband (National Institute of Health)



ImageJ

- Pripravljena orodja za pregledovanje in interaktivno obdelavo
- Podpora specializiranim formatom (16b int, 32b float...)
- Podpora razvoju novih vtičnikov (**Plugins**)
- Zaradi dinamičnega izvajanja je primerna za prototipni razvoj
- Java Runtime Environment (**JRE**)
- V spletnem okolju se lahko izvaja kot applet ali strežniška komponenta
- Makro skriptni jezik

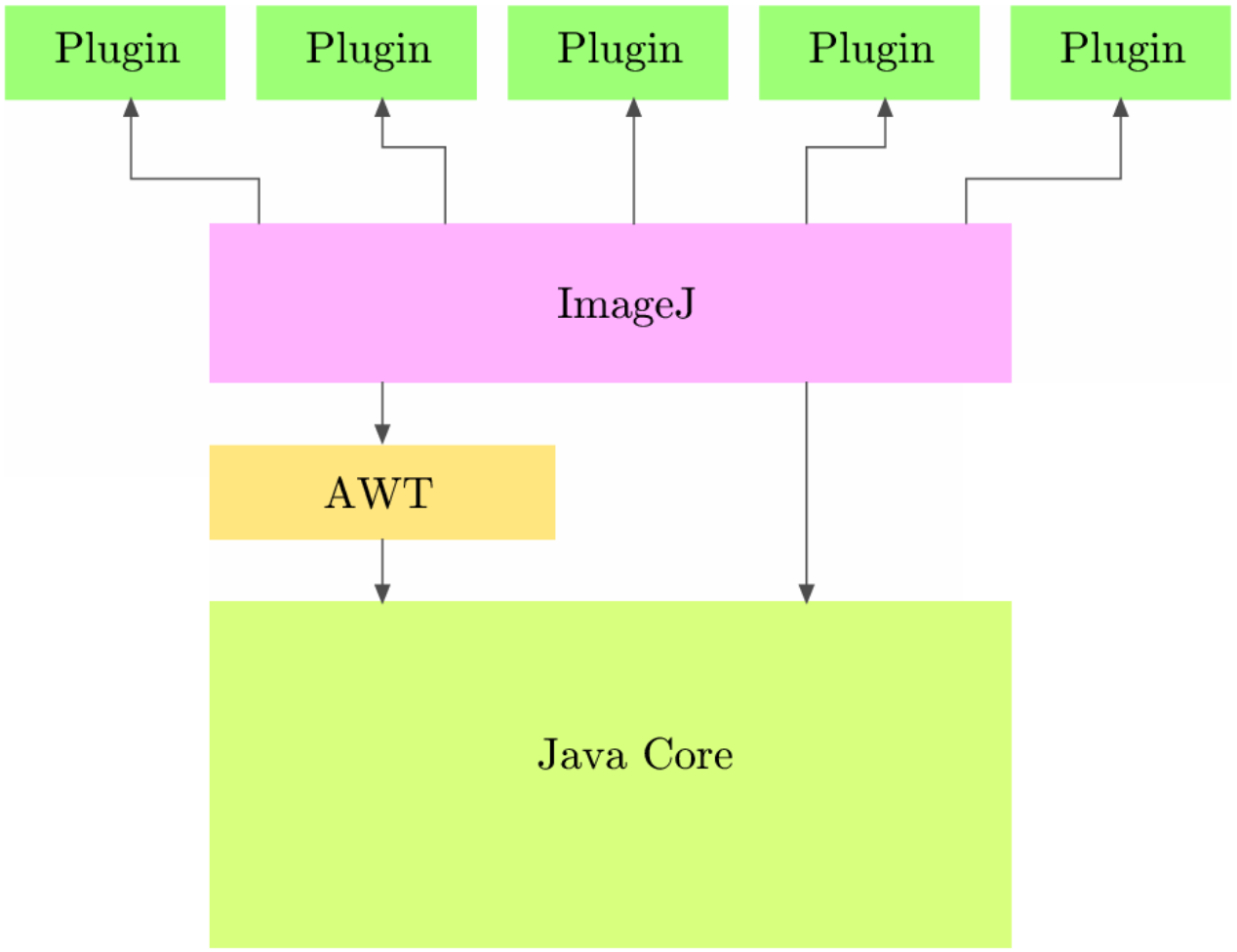
ImageJ

- Podpira TIFF (uncompressed), JPEG, GIF, PNG, BMP, DICOM, FITS
- Video: dodatki za branje in zapisovanje AVI, QuickTime (uncompressed)
- Undo

ImageJ vtičniki (Plugins)

- Java razred, ki implementira določen vmesnik
- `PlugIn`: na vhodu ne zahteva slike
- `PlugInFilter`: deluje nad trenutno aktivno sliko
  - `int Setup() (String arg, ImagePlus im)`
    - metoda klicana ob zagonu vtičnika. Preverja, ali slika *im* ustreza vtičniku. Vrne 32b opisnik z lastnostmi vtičnika.
  - `void Run() (ImageProcessor ip)`
    - izvede operacije nad sliko *ip*

ImageJ



Hello_World.java

```
import ij.*;
import ij.process.*;
import ij.gui.*;
import java.awt.*;
import ij.plugin.*;

public class Hello_World implements PlugIn {

    public void run(String arg) {
          IJ.showMessage("Hello_World","Hello
    world!");
    }
}
```

Plugins->New

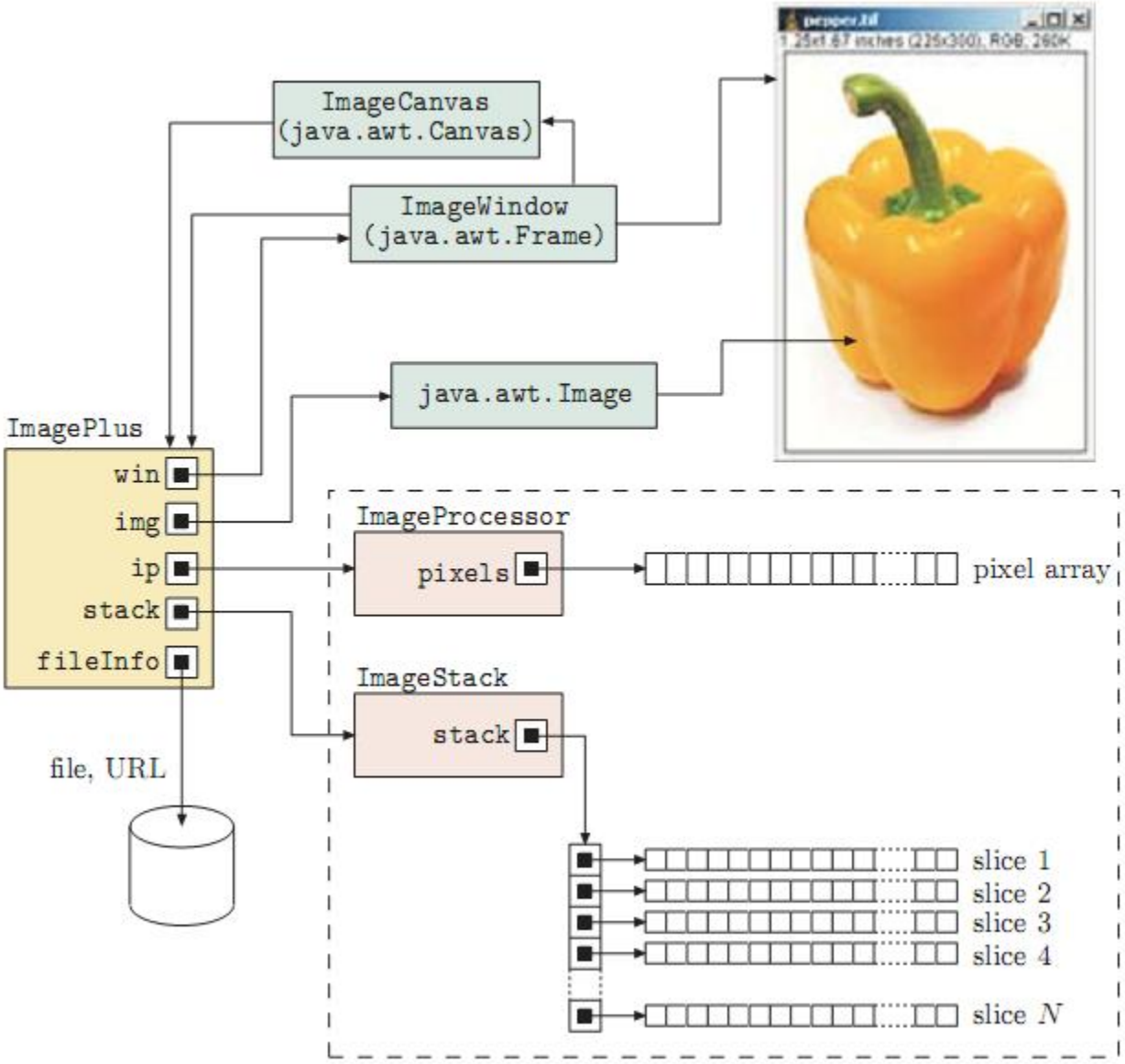
Type: Plugin

Name: Hello_World

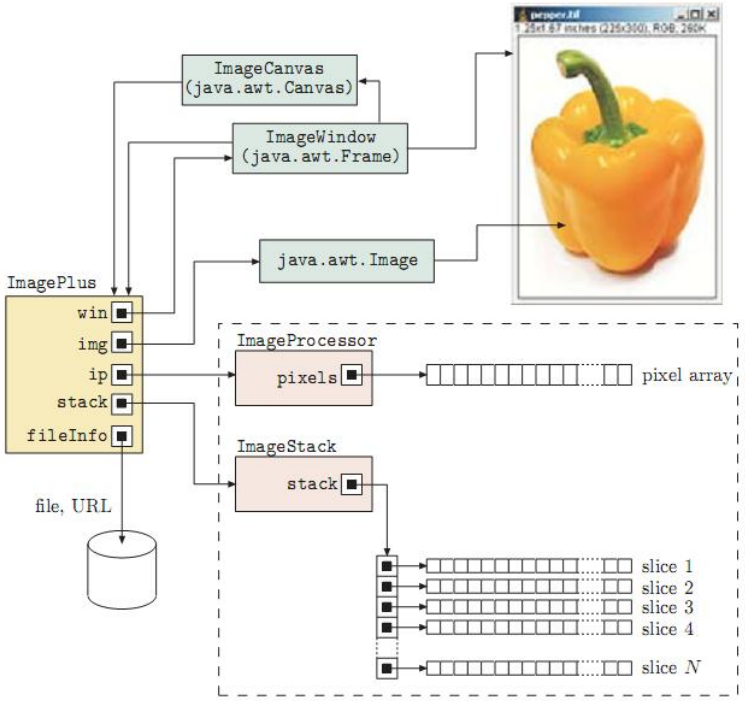
File->Compile and Run

Plugins->Shortcuts
->Install Plugin

ImageJ predstavitev slik



ImageJ API



`ImagePlus` (`ij`)

razširitev `java.awt` za prikaz slike

Slikovni elementi so shranjeni v `ImageProcessor` ali v `ImageStack` (sekvenca slik)

`ImageProcessor` (`ij.process`)

dostop in obdelava slikovnih elementov

podrazredi:

`ByteProcessor`,
`ShortProcessor`,
`FloatProcessor`,
`ColorProcessor`

My_Invert.java

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class My_Invert implements PlugInFilter {

  public int setup (String arg, ImagePlus im) {
    return DOES_8G;
  }

  public void run (ImageProcessor ip) {
    int w = ip.getWidth();
    int h = ip.getHeight();

    for (int u = 0; u < w; u++) {
        for (int v = 0; v < h; v++) {
            int p = ip.getPixel(u, v);
            ip.putPixel(u, v, 255 - p);
        }
    }
}
```

My_Invert.java

V mapi *plugins* ali podmapi

Znak _ v imenu pomeni vtičnik

Plugins ->
Compile and Run

Plugins->Shortcuts
->Install Plugin

My_Invert.java

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class My_Invert implements PlugInFilter {

  public int setup (String arg, ImagePlus im) {
    return DOES_8G;
  }

  public void run (ImageProcessor ip) {
    int w = ip.getWidth();
    int h = ip.getHeight();

    for (int u = 0; u < w; u++) {
      for (int v = 0; v < h; v++) {
        int p = ip.getPixel(u, v);
        ip.putPixel(u, v, 255 - p);
      }
    }
  }
```

DOES_16 – 16 bitne sivinske slike

DOES_32 – 32 bitne sivinske slike

DOES_8C – 8 bitne barvne slike

DOES_8G – 8 bitne sivinske slike

DOES_ALL

DOES_RGB

DOES_STACKS

DONE – `run()` se ne bo izvedel

NO_CHANGES

NO_IMAGE_REQUIRED

NO_UNDO

ROI_REQUIRED

STACK_REQUIRED

SUPPORTS_MASKING

Tipi slikovnih formatov v ImageJ

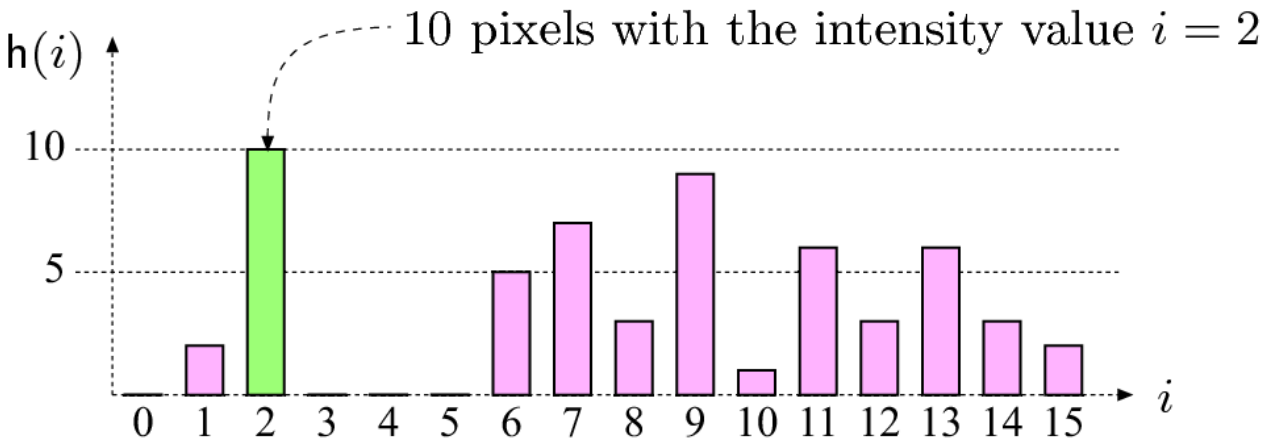
- 8 bitne sivinske slike `byte`
- 8 bitne barvne slike (indeks na tabelo LUT) `byte`
- 16 bitne sivinske slike `short`
- 32 bitne sivinske slike `float`
- RGB barvne slike `int`

- Java ne ponuja `unsigned byte`
- `byte: -128 ... 127`

```
int a = 200;                // a  = #000000C8
byte b = (byte) a;          // b  = #C8
int a1 = b;                 // a1 = -56
int a2 = (0xff & b);        // a2 = 200
```

Histogram

$$\mathsf{h}(i) = \mathrm{card}\{(u,v) \mid I(u,v) = i\}$$

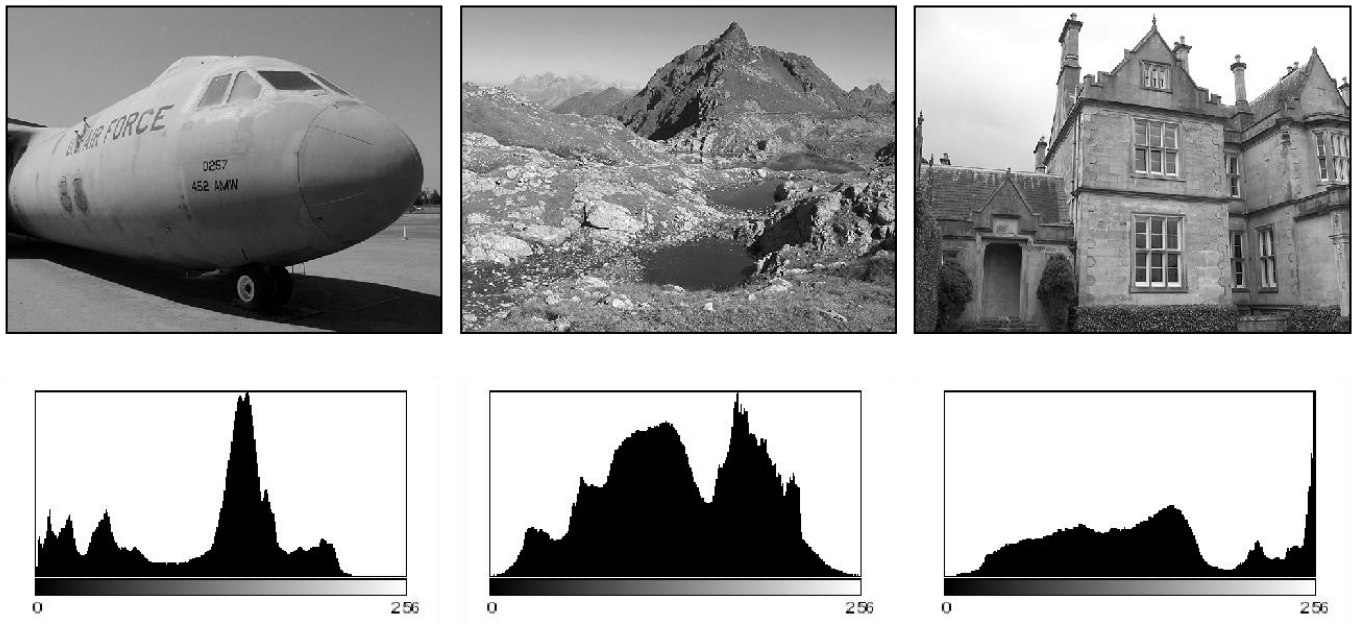


i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\mathsf{h}(i)$	0	2	10	0	0	0	5	7	3	9	1	6	3	6	3	2

Histogram

Kvalitativni atributi slike:

dinamični razpon
kontrast
segmentacija
saturacija



Histogram

```
for all levels i do
  H[i] = 0
for all pixel coordinates u and v do
  H[I(u, v)] ++
```

Compute_Histogram.java

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;
import ij.process.ByteProcessor;

public class Compute_Histogram implements
  PlugInFilter {
  String title = null;

  public int setup(String arg, ImagePlus im) {
    title = im.getTitle();
    return DOES_8G + NO_CHANGES;
  }

  public void run(ImageProcessor ip) {
    int[] H = new int[256];
    int w = ip.getWidth();
    int h = ip.getHeight();
    int hw = 256;
    int hh = 100;
    int maxH = 0;

    for (int v = 0; v < h; v++) {
        for (int u = 0; u < w; u++) {
                int i = ip.getPixel(u, v);
                H[i] = H[i] + 1;
        }
    }
...

    for (int i = 0; i<256; i++) {
        maxH = (H[i] > maxH) ? H[i] : maxH;
    }

    ImageProcessor histIp = new ByteProcessor(hw,
                                    hh);
    histIp.setValue(255);
    histIp.fill();
    for (int u = 0; u < hw; u++) {
        for (int v = 0; v < H[u]; v++) {
           histIp.putPixel(u,
                 hh-hh*v/maxH, 0);
        }
    }
    String hTitle = "Histogram of " + title;
    ImagePlus  histIm = new ImagePlus(hTitle,
                          histIp);
    histIm.show();
  }
}
```

Compute_Histogram_1.java

```
import ij.ImagePlus;
import ij.plugin.PlugIn;
import ij.process.ImageProcessor;
import ij.process.ByteProcessor;
import ij.io.Opener;
import ij.IJ;

public class Compute_Histogram_1 implements PlugIn {
    String title = null;

    public void run(String arg) {
           int[] H = new int[256];

           int hw = 256;
           int hh = 100;
           int maxH = 0;

           Opener op = new Opener();
           op.open();
           ImagePlus im = IJ.getImage();
           ImageProcessor ip = im.getProcessor();
```

```
...

           int w = im.getWidth();
           int h = im.getHeight();

           for (int v = 0; v < h; v++) {
                 for (int u = 0; u < w; u++) {
                       int i = ip.getPixel(u, v);
                       H[i] = H[i] + 1;
                 }
           }
    }
}
```

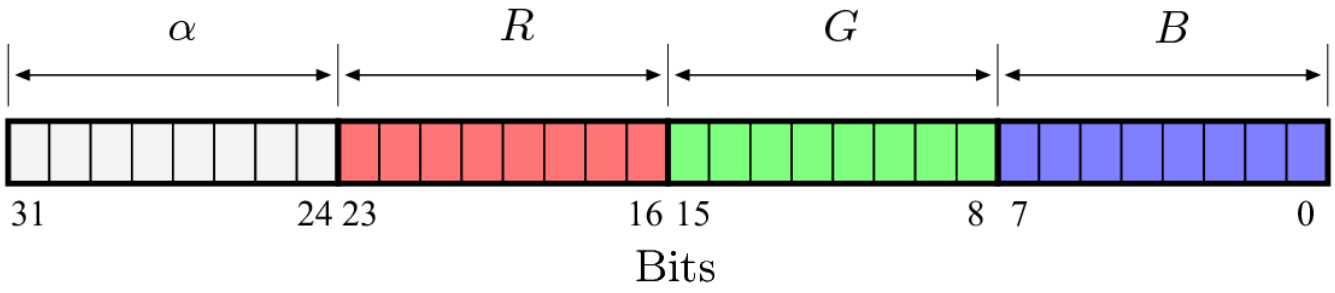
Histogram kot `PlugIn`

Sliko podamo preko dialoga `Opener`

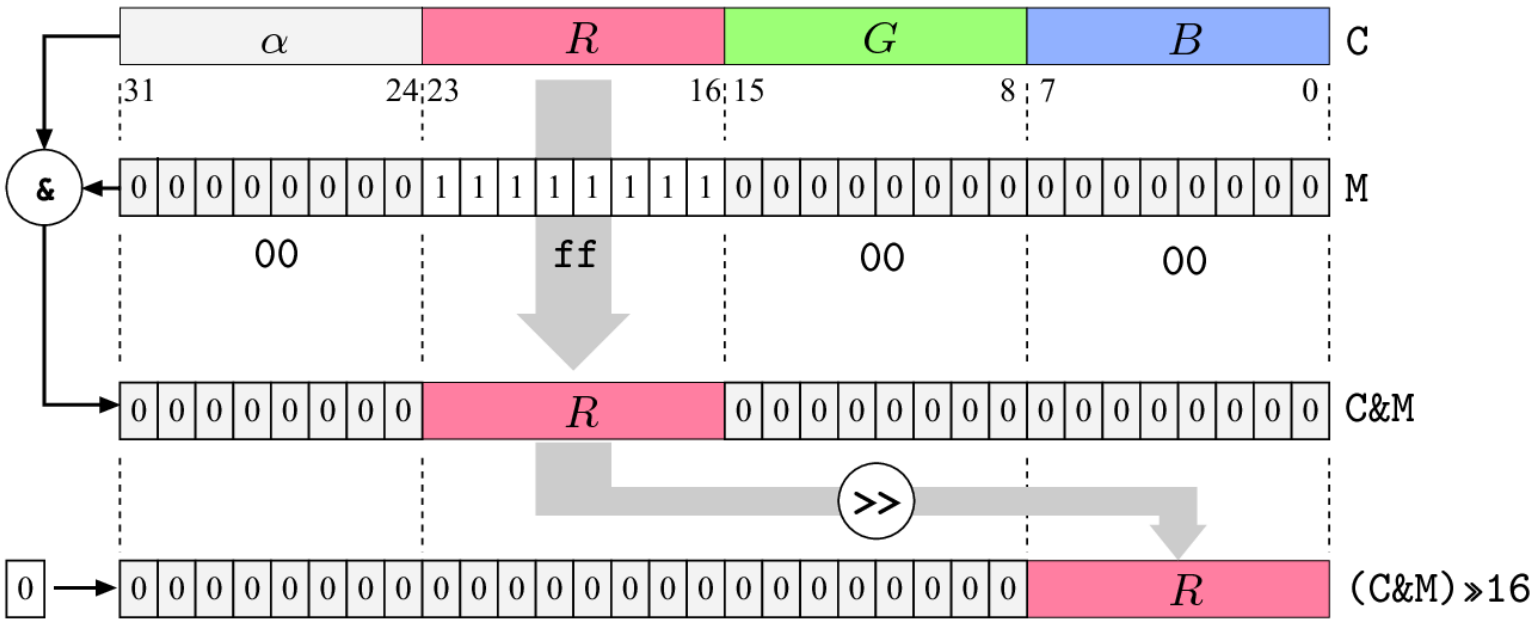
AppletDemo.java

```
import ij.*;
import java.applet.*;
import java.awt.*;
import ij.process.*;
import ij.gui.*;

public class HDemo extends Applet {
  String name;
  Image im;
  ImagePlus img;
  ImageProcessor ip = null;
  int[] h;
  HistogramWindow hw;

  public void init() {
    setLayout(new BorderLayout());
    Panel p = new Panel();
    p.setLayout(new GridLayout(1, 4));
    p.add(new Button("Restore"));
    p.add(new Button("Inv"));
    p.add(new Button("Ligh"));
    p.add(new Button("Dark"));
    add("South", p);
    img = new
        ImagePlus("c:\\ImageJ\\graybird.jpg");
    im = img.getImage();
    ip = img.getProcessor();
    ip.snapshot();
  }
...

public void update(Graphics g) {
    paint(g);
  }

  public void paint(Graphics g) {
    g.drawImage(im, 0, 0, this);
  }

  public boolean action(Event e, Object arg) {
    if (e.target instanceof Button) {
        String label = (String) arg;
        if (label.equals("Restore"))
             ip.reset();
        else if (label.equals("Inv"))
             ip.invert();
        else if (label.equals("Light"))
             ip.multiply(1.1);
        else if (label.equals("Dark"))
             ip.multiply(0.9);
        im = ip.createImage();
        repaint();
        return true;
    }
    return false;
  }
}
```

Barvne slike

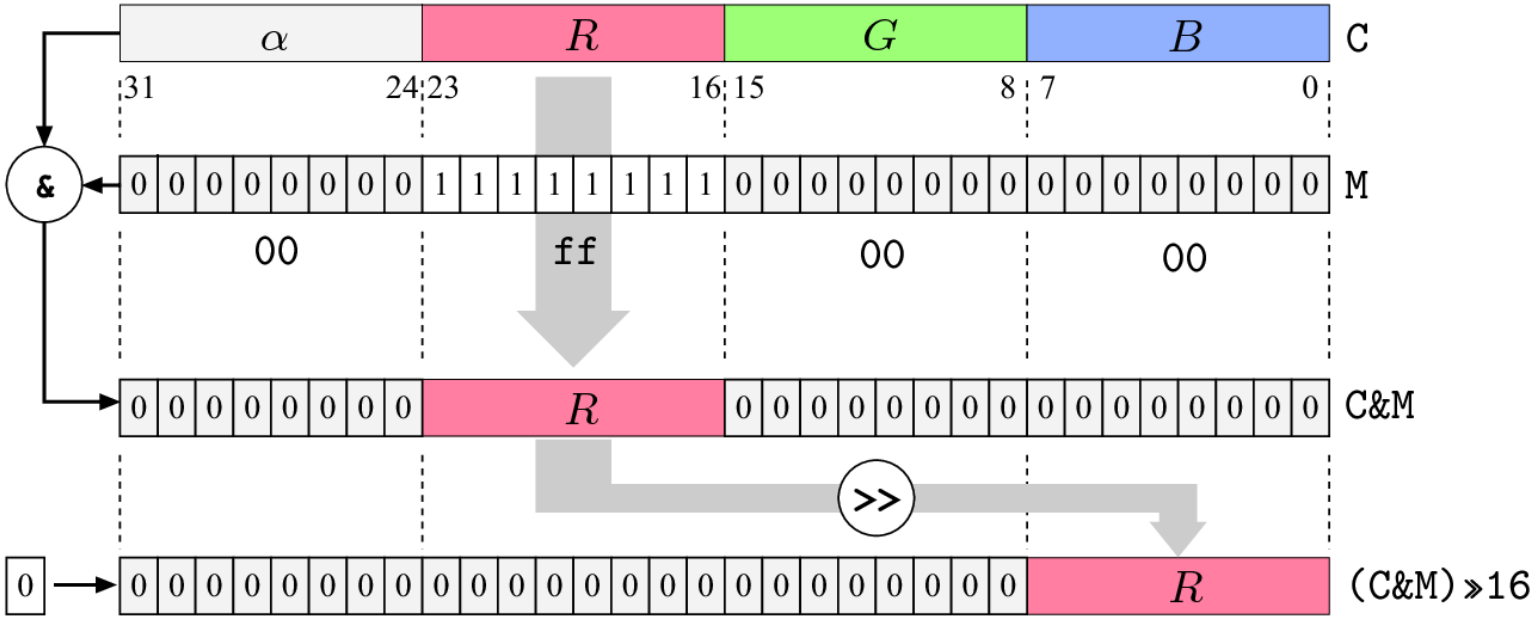


Barvne slike




```
1 int c = ip.getPixel(u,v); // a color pixel
2 int r = (c & 0xff0000) >> 16; // red value
3 int g = (c & 0x00ff00) >> 8;  // green value
4 int b = (c & 0x0000ff);       // blue value
```

Barvne slike




```
1 int r = 169; // red value
2 int g = 212; // green value
3 int b = 17; // blue value
4 int c = ((r & 0xff)<<16) | ((g & 0xff)<<8) | b & 0xff;
5 ip.putPixel(u,v,C);
```

Barvne slike `ColorProcessor`

```
1 int[] RGB = new int[3];
2 ...
3 RGB = ip.getPixel(u,v,RGB);
4 int r = RGB[0];
5 int g = RGB[1];
6 int b = RGB[2];
7 ...
8 ip.putPixel(u,v,RGB);
```

Brighten_Rgb.java

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ImageProcessor;

public class Brighten_Rgb implements PlugInFilter {

    public void run(ImageProcessor ip) {
        int[] pixels = (int[]) ip.getPixels();

        for (int i = 0; i < pixels.length; i++) {
            int c = pixels[i];

            int r = (c & 0xff0000) >> 16;
            int g = (c & 0x00ff00) >> 8;
            int b = (c & 0x0000ff);

            r = r + 10; if (r > 255) r = 255;
            g = g + 10; if (g > 255) g = 255;
            b = b + 10; if (b > 255) b = 255;

            pixels[i] = ((r & 0xff)<<16)
            | ((g & 0xff)<<8) | b & 0xff;
        }
    }

    public int setup(String arg, ImagePlus imp) {
        return DOES_RGB;
    }
}
```

Brighten_Rgb_1.java

```
import ij.ImagePlus;
import ij.plugin.filter.PlugInFilter;
import ij.process.ColorProcessor;
import ij.process.ImageProcessor;

public class Brighten_Rgb_1 implements PlugInFilter {

    public void run(ImageProcessor ip) {
      colorProcessor cp = (ColorProcessor) ip;
      int[] RGB = new int[3];

      for (int v = 0; v < cp.getHeight(); v++) {
            for (int u = 0; u < cp.getWidth(); u++) {
                  cp.getPixel(u, v, RGB);
                  RGB[0] = Math.min(RGB[0]+10, 255);
                  RGB[1] = Math.min(RGB[1]+10, 255);
                  RGB[2] = Math.min(RGB[2]+10, 255);
                  cp.putPixel(u, v, RGB);
            }
      }
    }

    public int setup(String arg, ImagePlus imp) {
      return DOES_RGB;
    }
}
```

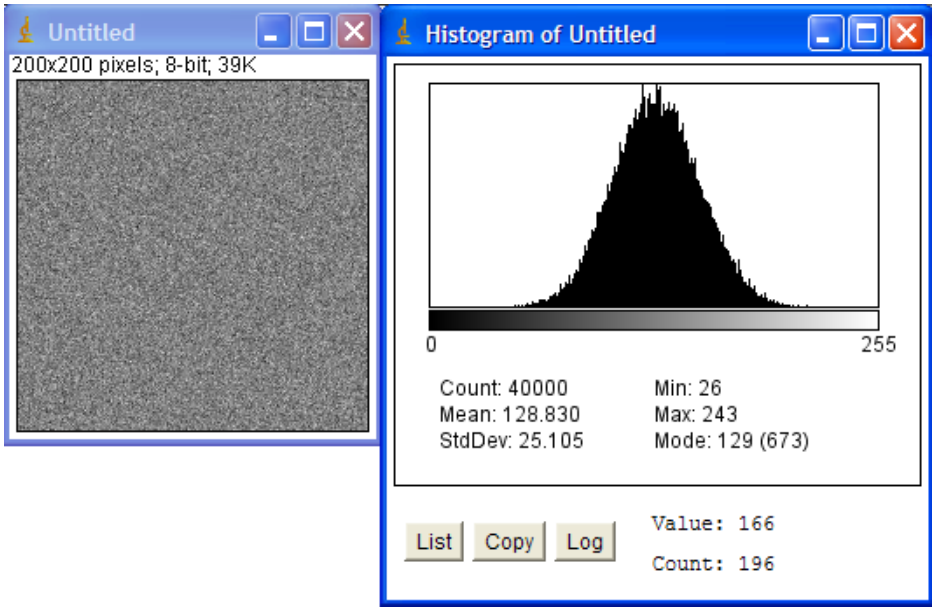
Šum v slikah

- Aditiven model šuma

$$I(x,y) = \overline{I(x,y)} + n(x,y)$$

- Normalno porazdeljen aditivni šum

$$n(x,y) \approx N(0,\sigma) = \sqrt{\frac{1}{2\Pi\sigma}} e^{-\frac{(t)^2}{2\sigma^2}}$$



Šum v slikah

- Modeliranje šuma

$$n(x,y) \cong I(x,y) - \overline{I(x,y)}$$

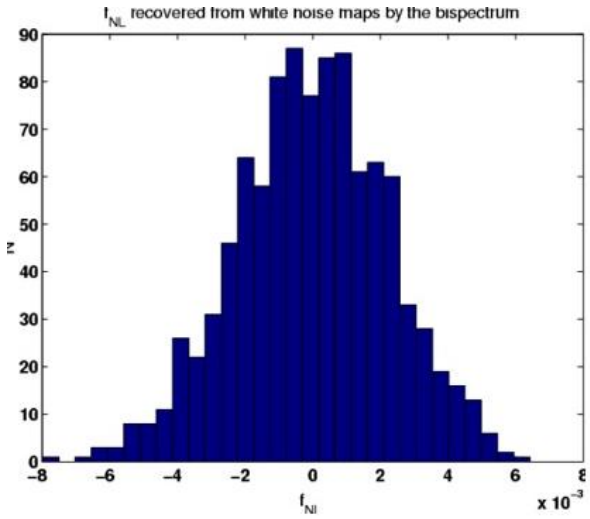
- sekvenca

$$I(x,y,t), \quad t = 1 \ldots n$$

$$\overline{I(x,y)} \cong \frac{1}{n} \sum_t I(x,y,t)$$

- srednja vrednost, standardna deviacija

$$SNR = \frac{\sigma_s}{\sigma_n}$$



Šum v slikah

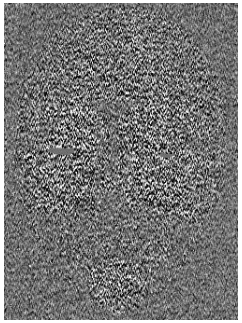
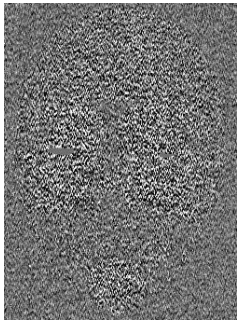
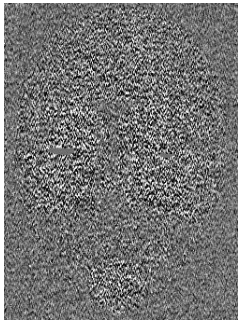
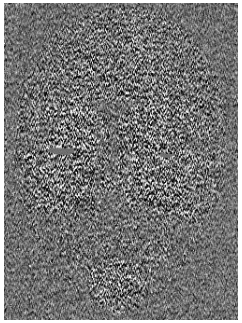
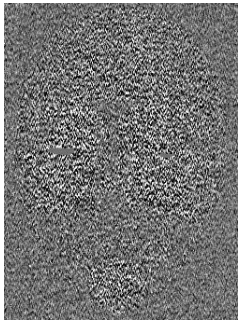




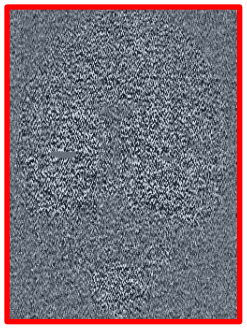
t

$$\overline{I(x,y)} \cong \frac{1}{n}\sum_{t} I(x,y,t)$$

$$n(x,y,t) \cong I(x,y,t) - \overline{I(x,y)}$$

Šum v slikah



$$SNR = \frac{\sigma_s}{\sigma_n}$$

*če SNR računamo za posamezen slikovni element, je odvisen od intenzitete

Histogrami s koši

- Slike z veliko zalogo vrednosti (npr. 32b sivinske slike)
- Slike z realnimi vrednostmi
- Histogram z B koši

$$\mathrm{h}(j) = \mathrm{card}\,\{(u,v) \mid a_j \leq I(u,v) < a_{j+1}\} \quad \text{for } \; 0 \leq j < B$$

- Primer: 14b sivinska slika, histogram z 256 koši
- 0 ... 2^{14} -1 vrednosti
- velikost koša: 2^{14} / 256 = 64
- $a_0 = 0$; $a_1 = 64$; $a_2 = 128$; ... $a_{255} = 16320$, $a_{256} = 2^{14}$

Histogrami s koši

$$
\begin{array}{lcrcl}
\mathsf{h}(0) & \leftarrow & 0 & \leq I(u,v) < & 64 \\
\mathsf{h}(1) & \leftarrow & 64 & \leq I(u,v) < & 128 \\
\mathsf{h}(2) & \leftarrow & 128 & \leq I(u,v) < & 192 \\
\vdots & & \vdots & \vdots & \vdots \\
\mathsf{h}(j) & \leftarrow & a_j & \leq I(u,v) < & a_{j+1} \\
\vdots & & \vdots & \vdots & \vdots \\
\mathsf{h}(255) & \leftarrow & 16320 & \leq I(u,v) < & 16384
\end{array}
$$

binnedHistogram(ImageProcessor ip)

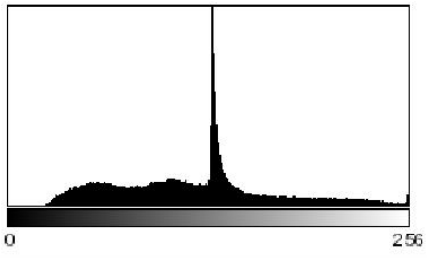
```
int[] binnedHistogram(ImageProcessor ip) {
    int K = 256; // število nivojev
    int B = 32;  // število košev
    int H = new int[B];
    int w = ip.getWidth();
    int h = ip.getHeight();

    for (int v = 0; v < h; v++) {
        for (int u = 0; u < w; u++) {
           int a = ip.getPixel(u.v);
           int i = a * B / K;
           H[i] ++;
        }
    }
    return H;
}
```

binnedHistogram(FloatProcessor ip)

```
int[] binnedHistogram(FloatProcessor ip) {
    double K = ip.getMax() - ip.getMin();
    int H[] = new int[B+1];
    int w = ip.getWidth();
    int h = ip.getHeight();
    double of = ip.getMin();

    for (int v = 0; v < h; v++) {
        for (int u = 0; u < w; u++) {
           double val = Math.floor((ip.getf(u,v)-
                                    of) * B / K);
           H[(int) val]++;
        }
    }
    return H;
}
```

Barvni RGB histogrami

(a)

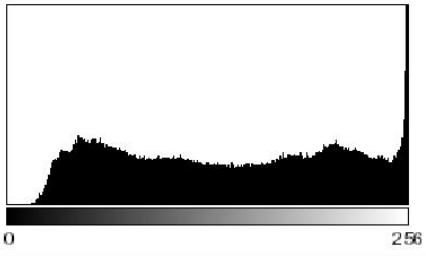


(b) h_{Lum}

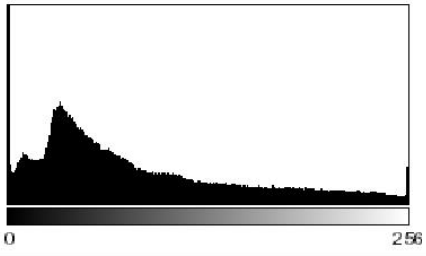
(c) R

(d) G

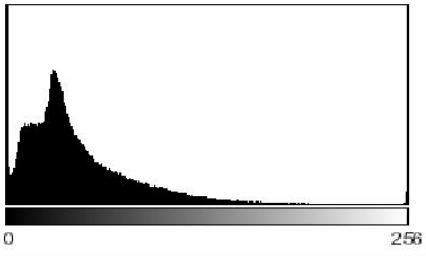
(e) B



(f) h_R



(g) h_G



(h) h_B

Barvni histogrami

$$\mathsf{h}_{\mathrm{R}}(200) = 24$$

$$(r, g, b) = (200, *, *)$$

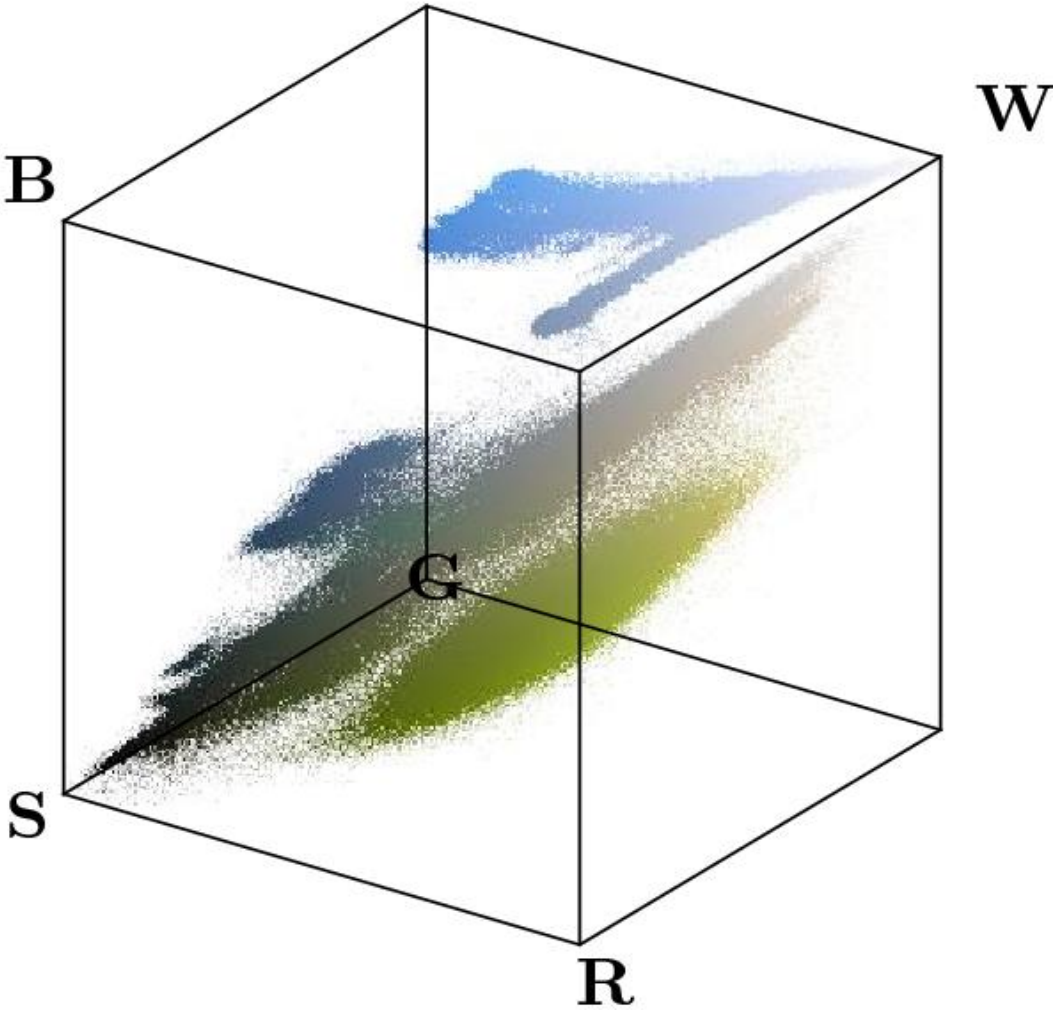
Barvni histogrami

$$\mathsf{h}_{\mathrm{R}}(50) = 100 \qquad \mathsf{h}_{\mathrm{G}}(50) = 100 \qquad \mathsf{h}_{\mathrm{B}}(50) = 100$$

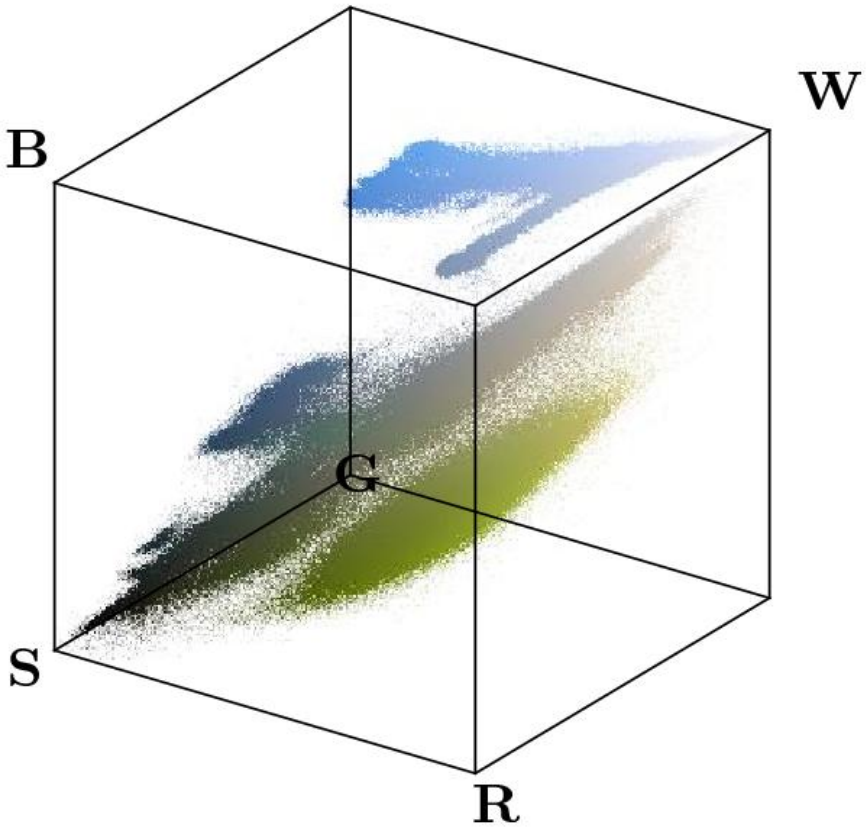
$$(r, g, b) = (50, 50, 50)$$

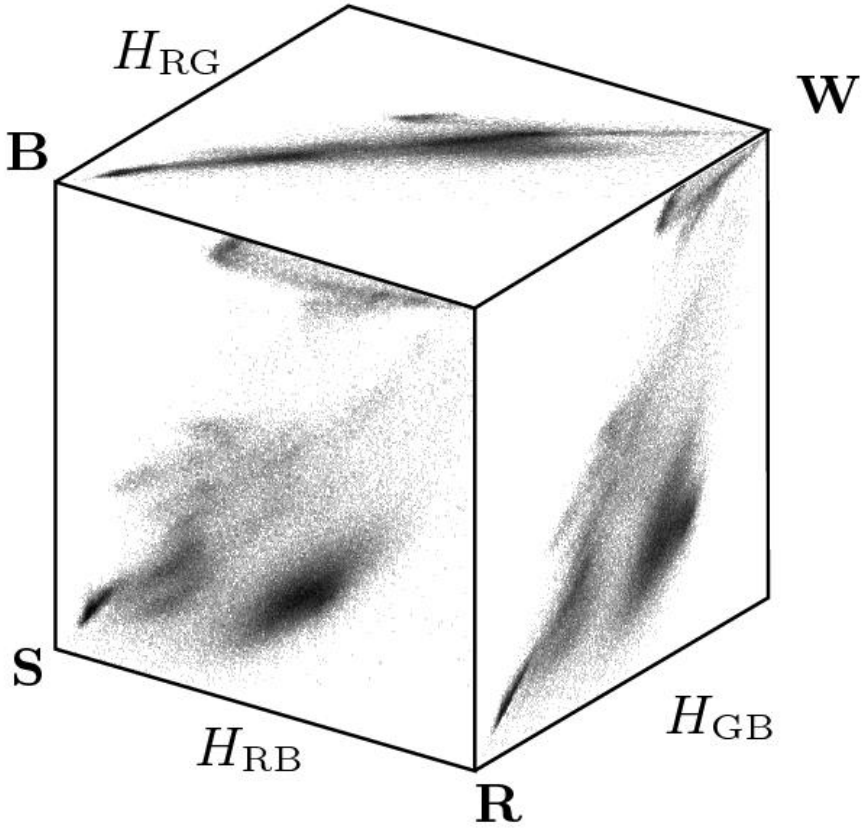
Polni RGB histogram

256 x 256 x 256 = 2^{24} košev tipa `int`



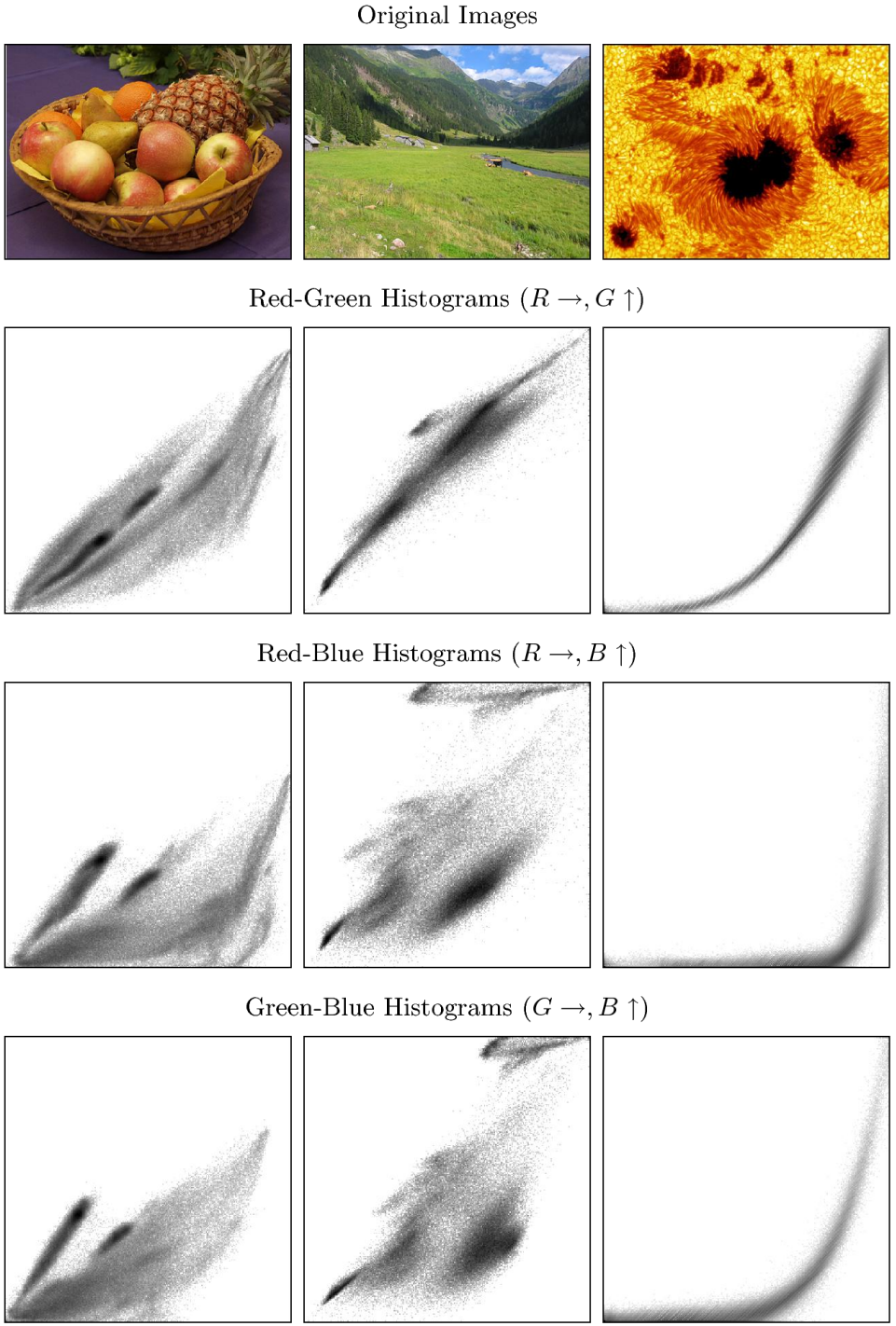
2D RGB histogrami





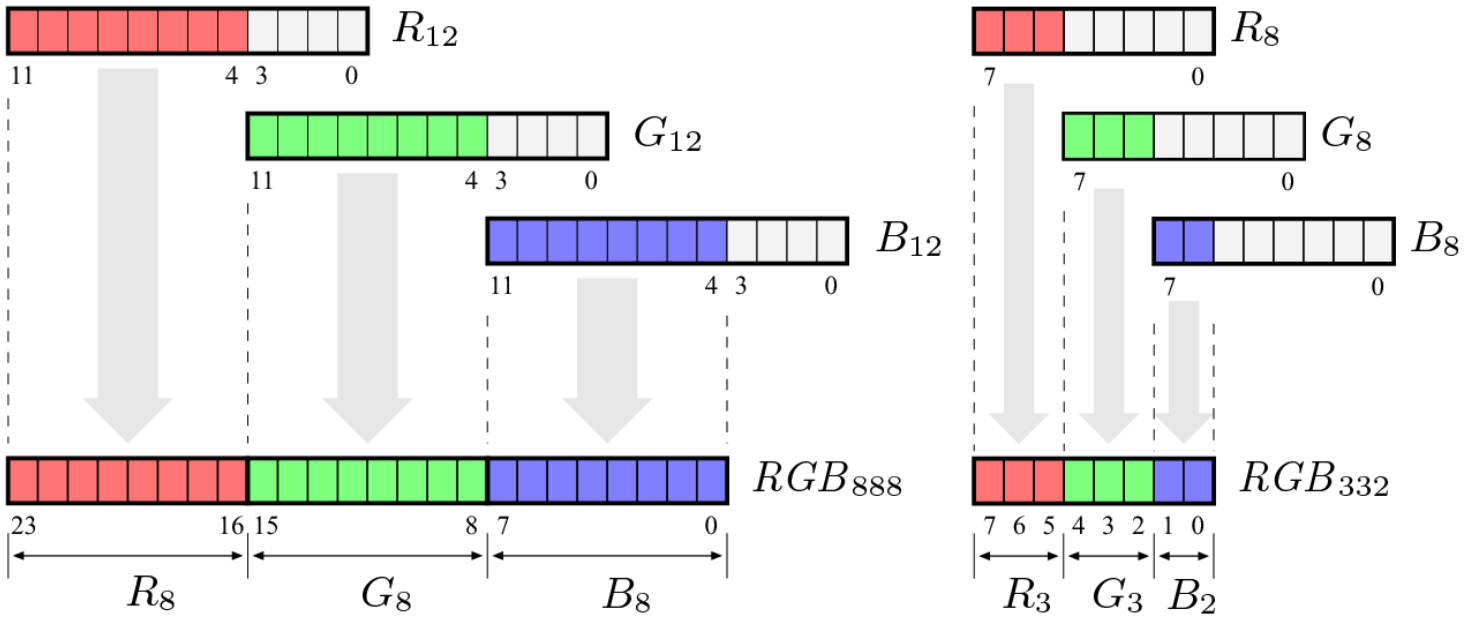
$$H_{\mathrm{RG}}(r,g) \leftarrow \text{number of pixels with } I_{\mathrm{RGB}}(u,v)=(r,g,*)$$
$$H_{\mathrm{RB}}(r,b) \leftarrow \text{number of pixels with } I_{\mathrm{RGB}}(u,v)=(r,*,b)$$
$$H_{\mathrm{GB}}(g,b) \leftarrow \text{number of pixels with } I_{\mathrm{RGB}}(u,v)=(*,g,b)$$

2D RGB histogrami



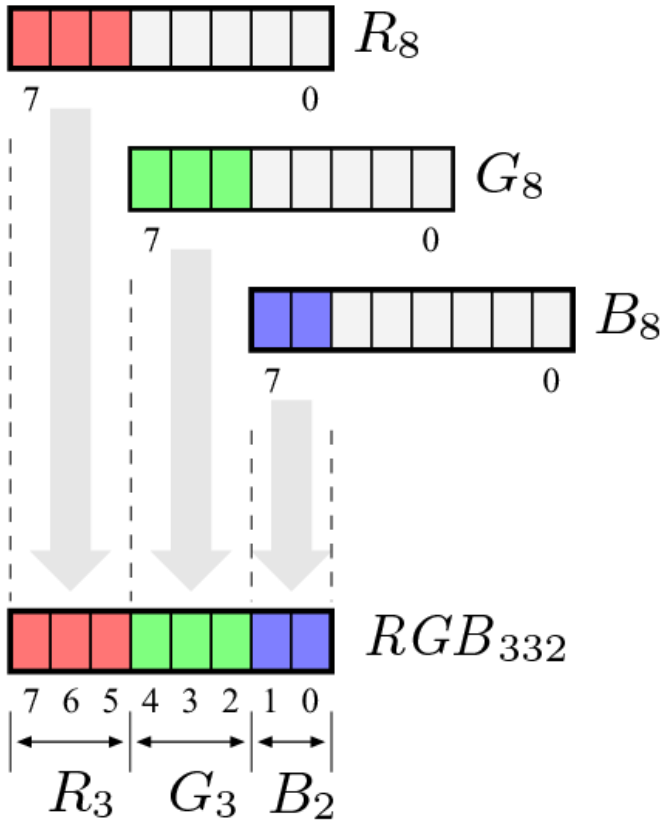
Linearna barvna kvantizacija

- zmanjšanje barvne palete, neodvisno od vsebine slike
- enakomerno vzorčenje barvnega prostora



3:3:2 quantization

```
ColorProcessor cp = (ColorProcessor) ip;
int C = cp.getPixel(u, v);
int R = (C & 0x00FF0000) >> 16;
int G = (C & 0x0000FF00) >> 8;
int B = (C & 0x000000FF) ;
byte RGB = (byte) (R & 0xE0 | (G & 0xE0) >> 3 |
                        (B & 0xC0) >> 6);
```



Vektorska barvna kvantizacija

- algoritmi, ki upoštevajo 3D barvno porazdelitev
- vektorska kvantizacija (r, g, b) trojic
- primer: **Median-cut**