Digital Image Processing Using Matlab

Haris Papasaika-Hanusch
Institute of Geodesy and Photogrammetry, ETH Zurich
haris@geod.baug.ethz.ch

Images and Digital Images

- A **digital image** differs from a photo in that the values are all **discrete**.
- Usually they take on only **integer** values.
- A digital image can be considered as a large array of discrete dots, each of which has a brightness associated with it. These dots are called picture elements, or more simply **pixels**.
- The pixels surrounding a given pixel constitute its **neighborhood** A neighborhood can be characterized by its shape in the same way as a matrix: we can speak of a 3x3 neighborhood, or of a 5x7 neighborhood.

48	219	168	145	244	188	120	58
49	218	87	94	133	35	17	148
174	151	74	179	224	3	252	194
77	127	87	139	44	228	149	135
138	229	136	113	250	51	108	163
38	210	185	177	69	76	131	53
178	164	79	158	64	169	85	97
96	209	214	203	223	73	110	200

Current pixel

3×5 neighbourhood

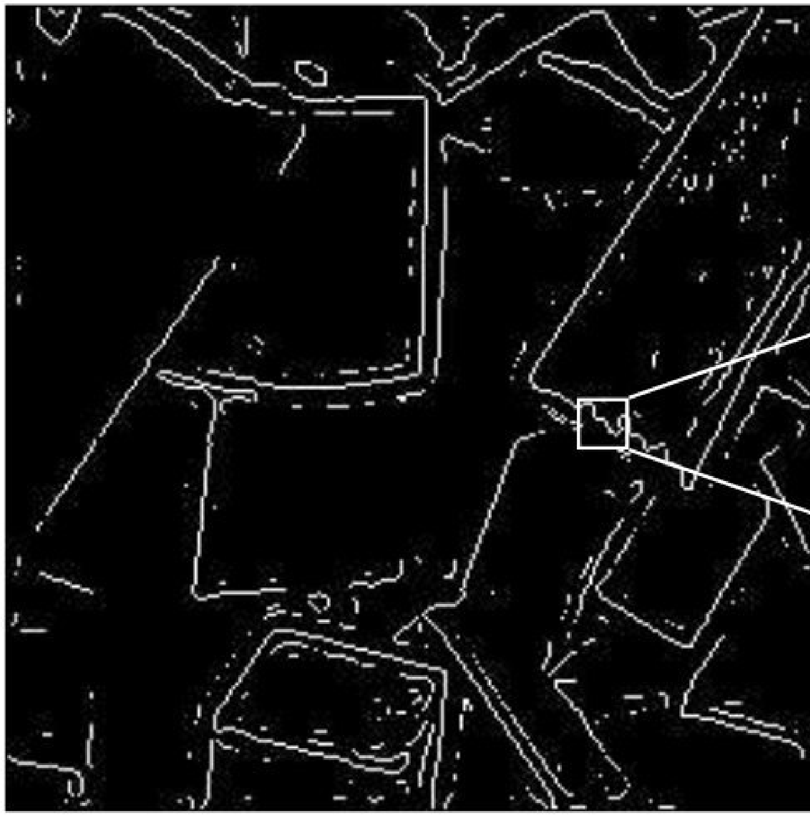
Aspects of Image Processing

- **Image Enhancement:** Processing an image so that the result is more suitable for a particular application. (sharpening or de-blurring an out of focus image, highlighting edges, improving image contrast, or brightening an image, removing noise)
- **Image Restoration:** This may be considered as reversing the damage done to an image by a known cause. (removing of blur caused by linear motion, removal of optical distortions)
- **Image Segmentation:** This involves subdividing an image into constituent parts, or isolating certain aspects of an image. (finding lines, circles, or particular shapes in an image, in an aerial photograph, identifying cars, trees, buildings, or roads.

Types of Digital Images

- **Binary:** Each pixel is just **black** or **white**. Since there are only two possible values for each pixel (0,1), we only need **one bit** per pixel.
- **Grayscale:** Each pixel is a shade of gray, normally from **0** (black) to **255** (white). This range means that each pixel can be represented by **eight bits**, or exactly **one byte**. Other greyscale ranges are used, but generally they are a power of **2**.
- **True Color**, or **RGB**: Each pixel has a particular color; that color is described by the amount of **red**, **green** and **blue** in it. If each of these components has a range 0–255, this gives a total of $\mathbf{256^3}$ different possible colors. Such an image is a “stack” of **three matrices**; representing the **red**, **green** and **blue** values for each pixel. This means that for every pixel there correspond 3 values.

Binary Image



1	1	0	0	0	0
0	0	1	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	1	1	0
0	0	0	0	0	1

Grayscale Image

230	229	232	234	235	232	148
237	236	236	234	233	234	152
255	255	255	251	230	236	161
99	90	67	37	94	247	130
222	152	255	129	129	246	132
154	199	255	150	189	241	147
216	132	162	163	170	239	122

Color Image

49	55	56	57	52	53
58	60	60	58	55	57
58	58	54	53	55	56
83	78	72	69	68	69
88	91	91	84	83	82
69	76	83	78	76	75
61	69	73	78	76	76

64	76	82	79	78	78
93	93	91	91	86	86
88	82	88	90	88	89
125	119	113	108	111	110
137	136	132	128	126	120
105	108	114	114	118	113
96	103	112	108	111	107

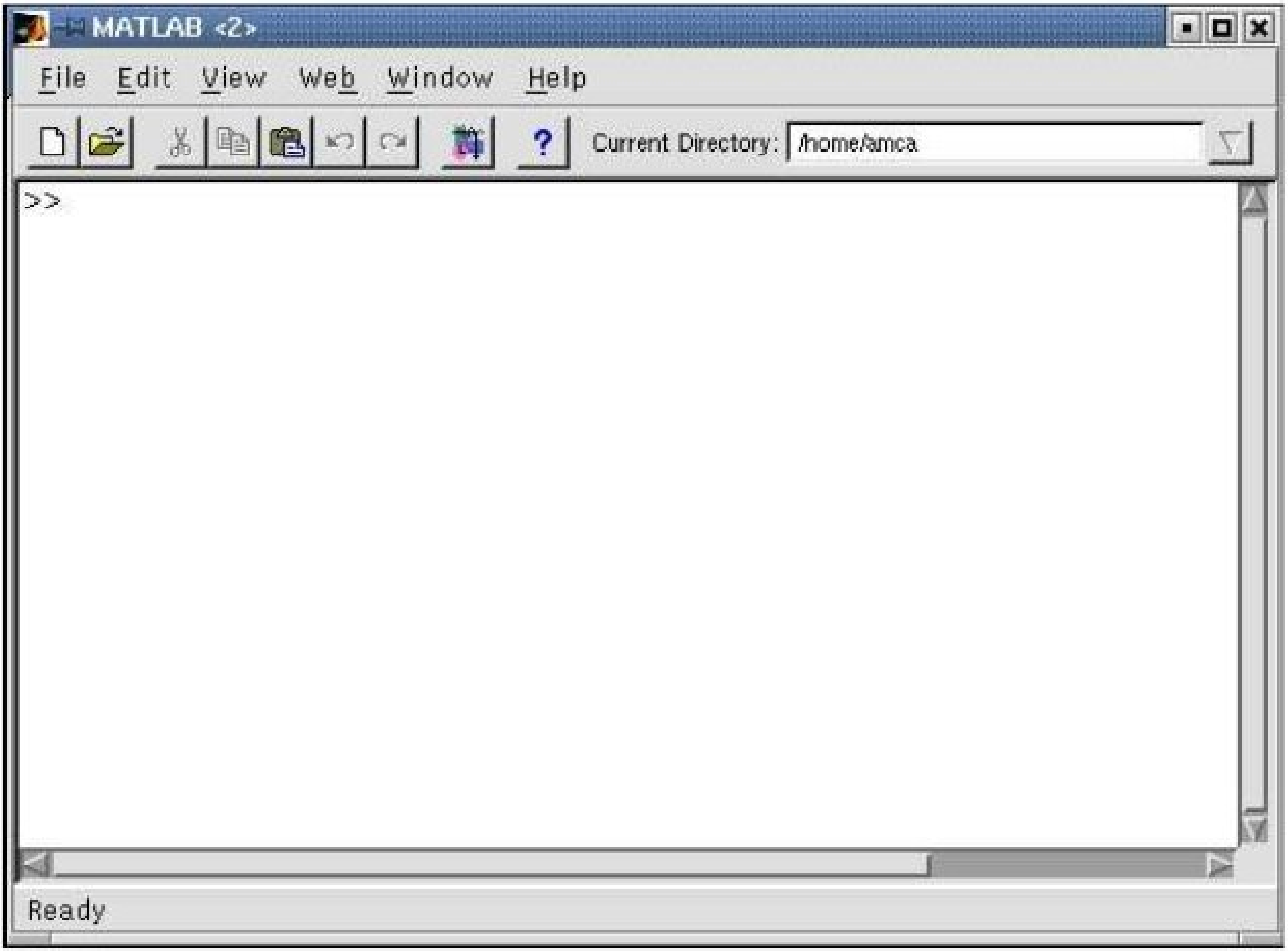
66	80	77	80	87	77
81	93	96	99	86	85
83	83	91	94	92	88
135	128	126	112	107	106
141	129	129	117	115	101
95	99	109	108	112	109
84	93	107	101	105	102

Red Green Blue

General Commands

- **imread**: Read an image
- **figure**: creates a figure on the screen.
- **imshow(g)**: which displays the matrix g as an image.
- **pixval on**: turns on the pixel values in our figure.
- **impixel(i,j)**: the command returns the value of the pixel (i,j)
- **iminfo**: Information about the image.

Command Window



Data Types

Data type	Description	Range
int8	8-bit integer	−128 — 127
uint8	8-bit unsigned integer	0 — 255
int16	16-bit integer	−32768 — 32767
uint16	16-bit unsigned integer	0 — 65535
double	Double precision real number	Machine specific

Image Information

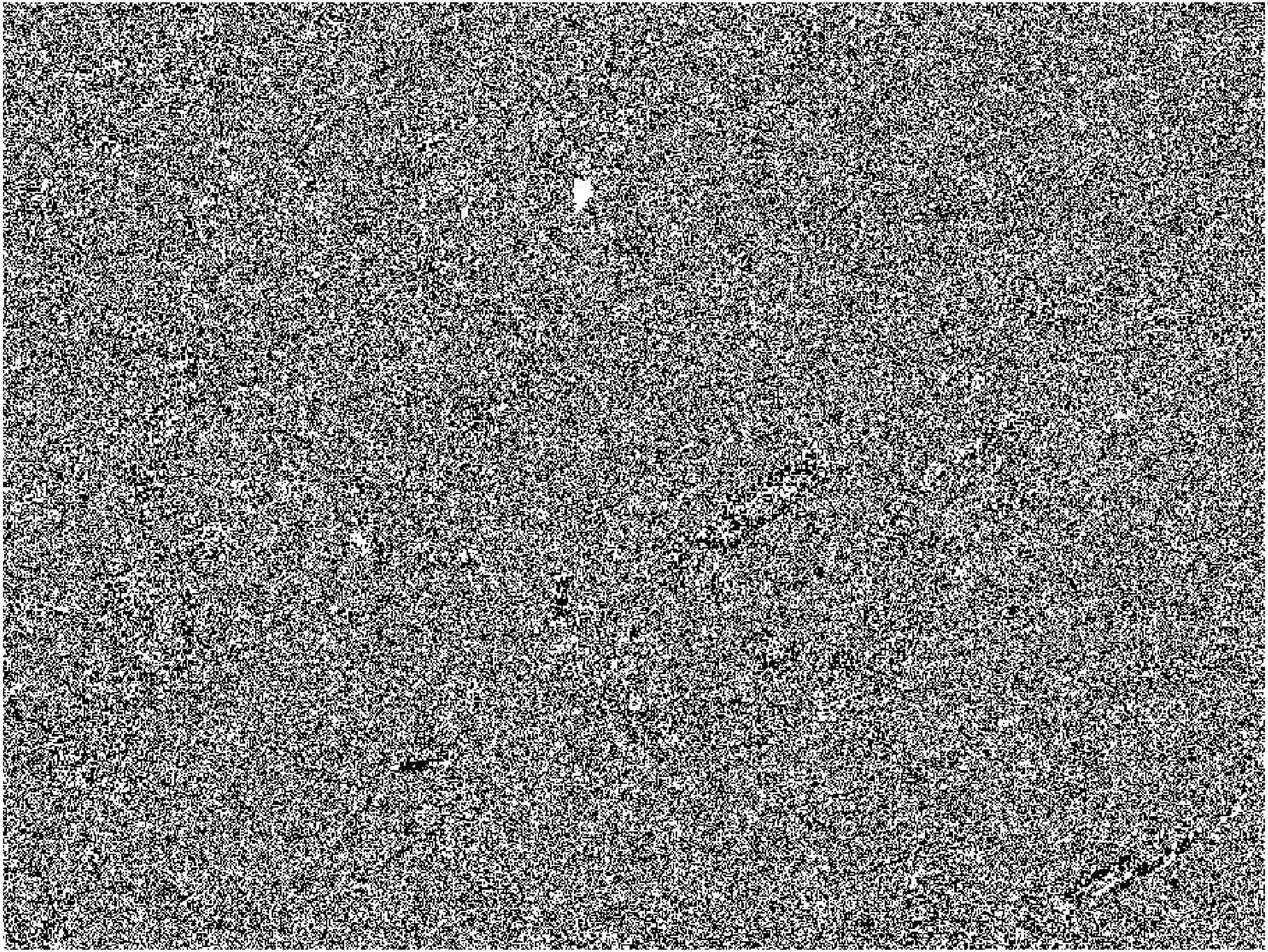
```
                     Filename: 'aster.tif'
                  FileModDate: '13-Mar-2008 16:54:26'
                     FileSize: 17224424.00
                       Format: 'tif'
                FormatVersion: []
                        Width: 4100.00
                       Height: 4200.00
                     BitDepth: 8.00
                    ColorType: 'grayscale'
              FormatSignature: [77.00 77.00 0 42.00]
                    ByteOrder: 'big-endian'
               NewSubFileType: 0
                BitsPerSample: 8.00
                  Compression: 'Uncompressed'
    PhotometricInterpretation: 'BlackIsZero'
                 StripOffsets: [525x1 double]
              SamplesPerPixel: 1.00
                 RowsPerStrip: 8.00
              StripByteCounts: [525x1 double]
                  XResolution: 1.00
                  YResolution: 1.00
               ResolutionUnit: 'None'
                     Colormap: []
          PlanarConfiguration: 'Chunky'
                    TileWidth: []
                   TileLength: []
                  TileOffsets: []
               TileByteCounts: []
                  Orientation: 1.00
                    FillOrder: 1.00
              GrayResponseUnit: 0.01
               MaxSampleValue: 255.00
               MinSampleValue: 0
                 Thresholding: 1.00
                     Software: 'ERDAS IMAGINE '
                 SampleFormat: 'Unsigned integer'
```

Bit Planes

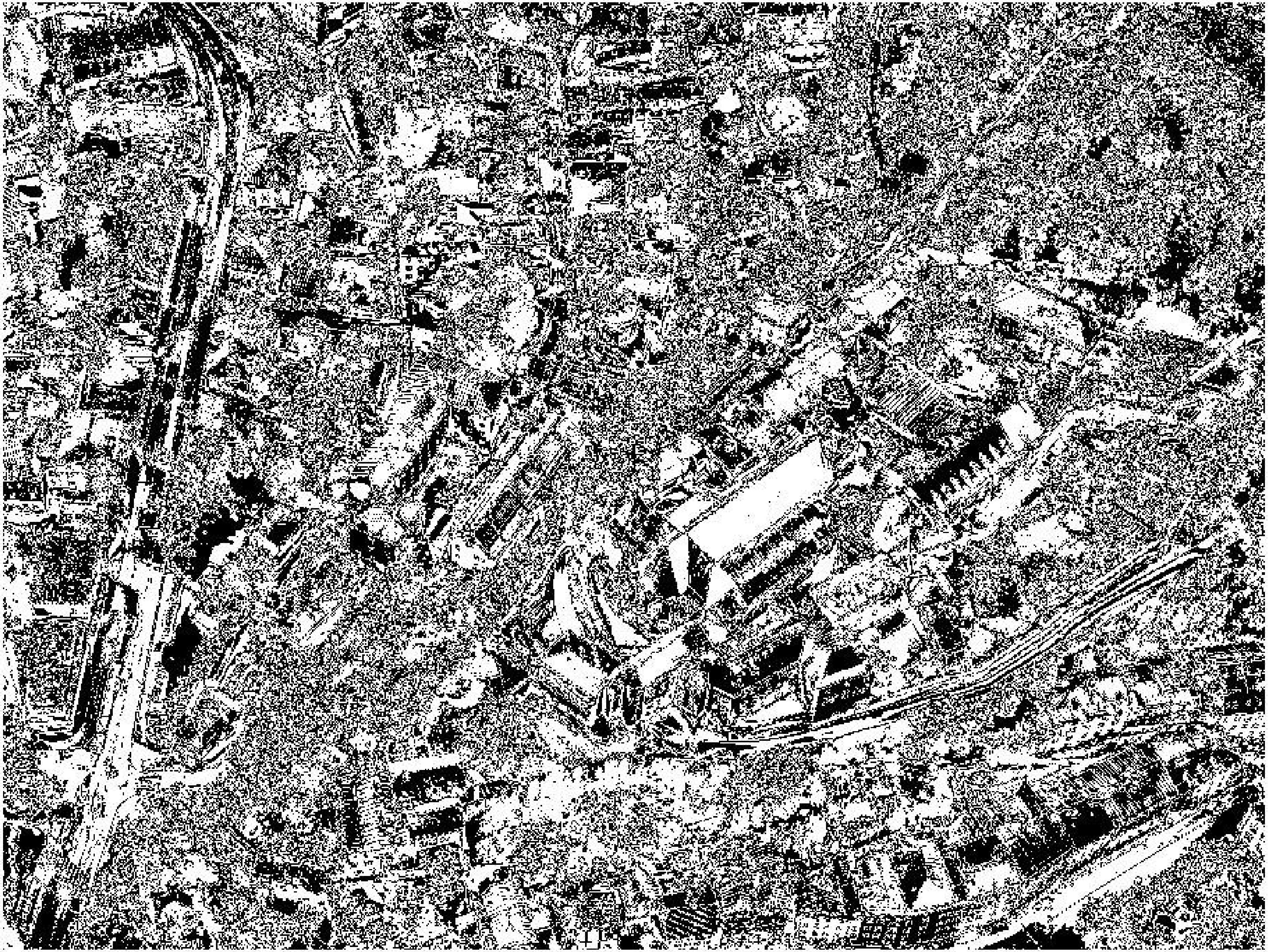
- Greyscale images can be transformed into a sequence of binary images by breaking them up into their **bit-planes**.
- We consider the grey value of each pixel of an 8-bit image as an 8-bit binary word.
- The **0th bit plane** consists of the **last bit** of each grey value. Since this bit has the least effect (**least significant bit plane**).
- The **7th bit plane** consists of the first bit in each value (**most significant bit plane**.

Initial Image

Bit Plane 0



Bit Plane 4



Bit Plane 7

Spatial Resolution

- **Spatial resolution** is the density of pixels over the image: the greater the spatial resolution, the more pixels are used to display the image.
- **Halve** the size of the image: It does this by taking out every other row and every other column, thus leaving only those matrix elements whose row and column indices are even.
- **Double** the size of the image: all the pixels are repeated to produce an image with the same size as the original, but with half the resolution in each direction.

Interpolation

$$
\begin{array}{ccccccc}
x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} & \cdots \\
x_{21} & \boxed{x_{22}} & x_{23} & \boxed{x_{24}} & x_{25} & \boxed{x_{26}} & \cdots \\
x_{31} & x_{32} & x_{33} & x_{34} & x_{35} & x_{36} & \cdots \\
x_{41} & \boxed{x_{42}} & x_{43} & \boxed{x_{44}} & x_{45} & \boxed{x_{46}} & \cdots \\
x_{51} & x_{52} & x_{53} & x_{54} & x_{55} & x_{56} & \cdots \\
x_{61} & \boxed{x_{62}} & x_{63} & \boxed{x_{64}} & x_{65} & \boxed{x_{66}} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
\longrightarrow \boxed{\texttt{imresize(x,1/2)}} \longrightarrow
\begin{array}{cccc}
x_{22} & x_{24} & x_{26} & \cdots \\
x_{42} & x_{44} & x_{46} & \cdots \\
x_{62} & x_{64} & x_{66} & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{array}
$$

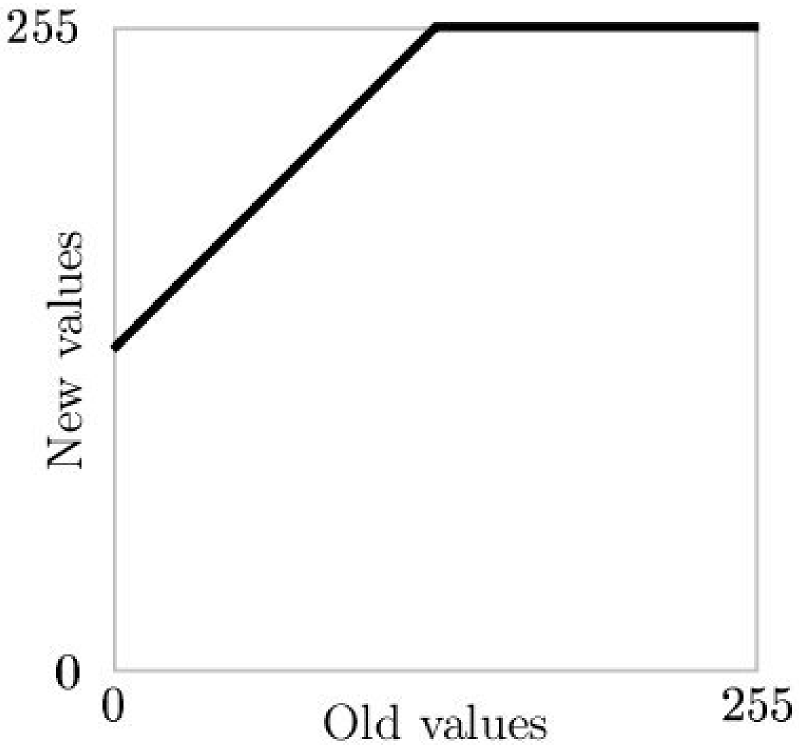
Extrapolation

$$\begin{matrix}
\boxed{\begin{matrix} x_{22} & x_{22} \\ x_{22} & x_{22} \end{matrix}} & \boxed{\begin{matrix} x_{24} & x_{24} \\ x_{24} & x_{24} \end{matrix}} & \boxed{\begin{matrix} x_{26} & x_{26} \\ x_{26} & x_{26} \end{matrix}} & \cdots \\
\boxed{\begin{matrix} x_{42} & x_{42} \\ x_{42} & x_{42} \end{matrix}} & \boxed{\begin{matrix} x_{44} & x_{44} \\ x_{44} & x_{44} \end{matrix}} & \boxed{\begin{matrix} x_{46} & x_{46} \\ x_{46} & x_{46} \end{matrix}} & \cdots \\
\boxed{\begin{matrix} x_{62} & x_{62} \\ x_{62} & x_{62} \end{matrix}} & \boxed{\begin{matrix} x_{64} & x_{64} \\ x_{64} & x_{64} \end{matrix}} & \boxed{\begin{matrix} x_{66} & x_{66} \\ x_{66} & x_{66} \end{matrix}} & \cdots \\
\vdots & \vdots & \vdots & \ddots
\end{matrix}$$

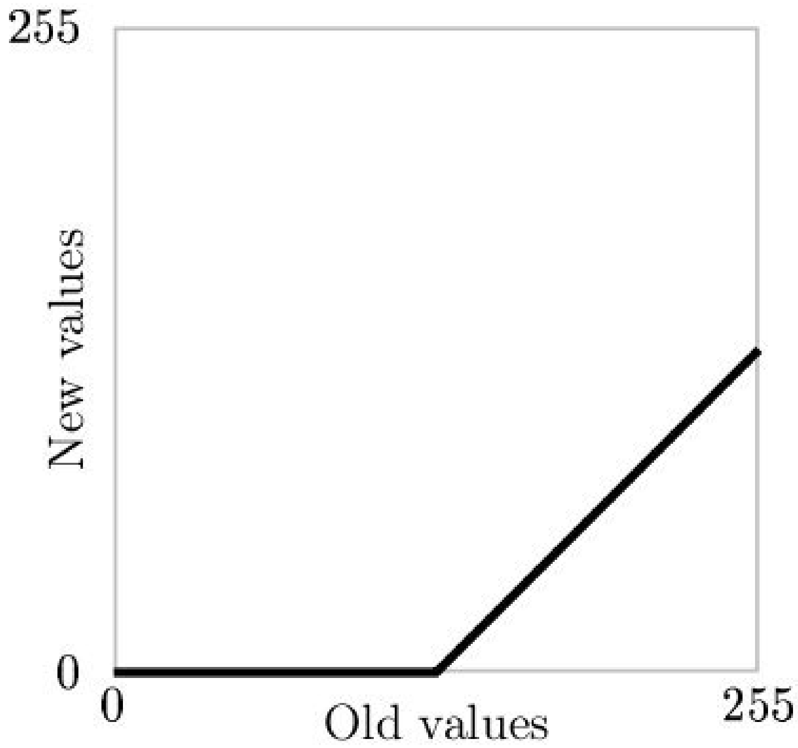
Arithmetic Operations

- These operations act by applying a simple function y=f(x) to each gray value in the image.
- Simple functions include **adding** or **subtract** a constant value to each pixel: $y = x \pm C$ (imadd, imsubtract)
- **Multiplying** each pixel by a constant: $y = C \cdot x$ (immultiply, imdivide)
- **Complement**: For a grayscale image is its photographic negative.

Addition - Subtraction

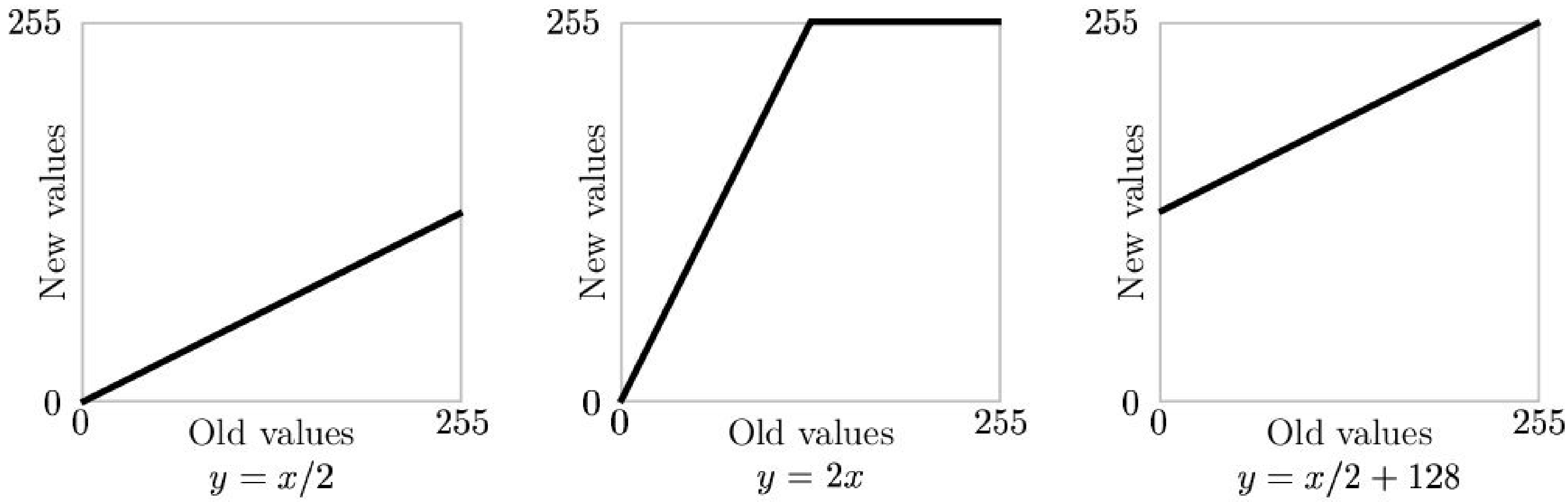


Adding 128 to each pixel

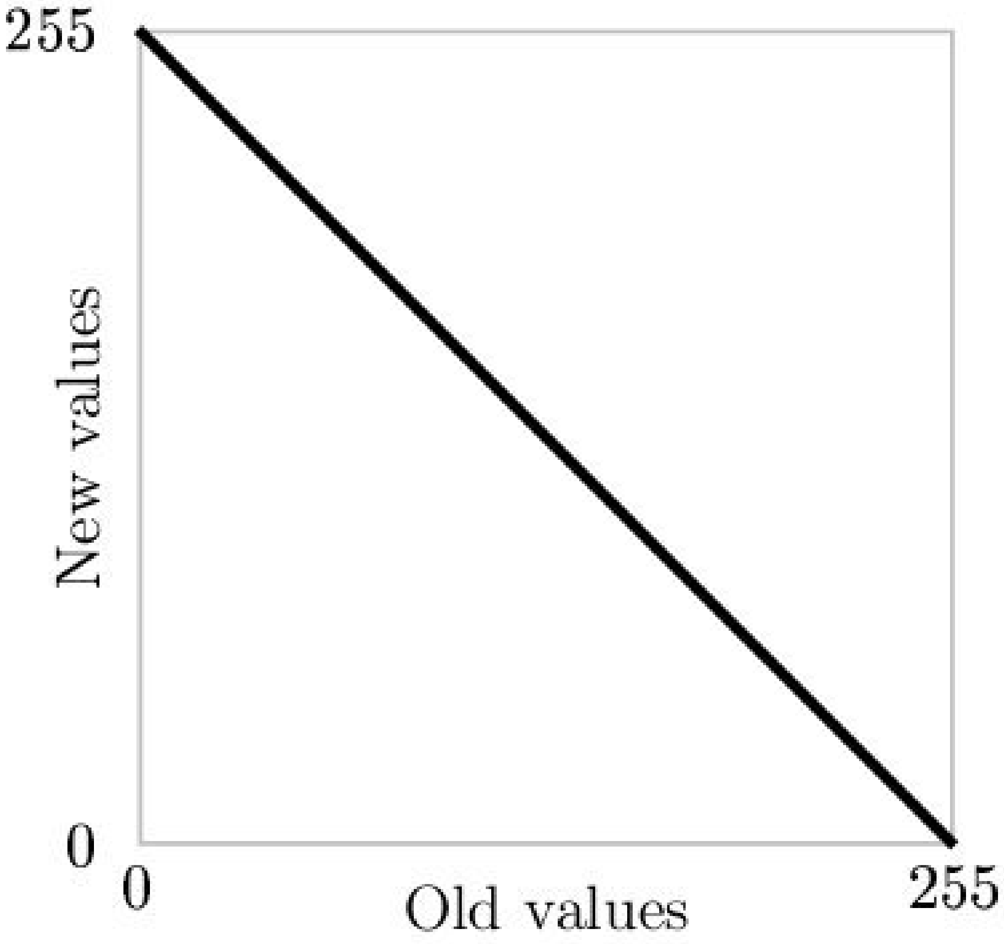


Subtracting 128 from each pixel

Multiplication-Division



Complement



$$y = 255 - x$$

Addition

Image: J

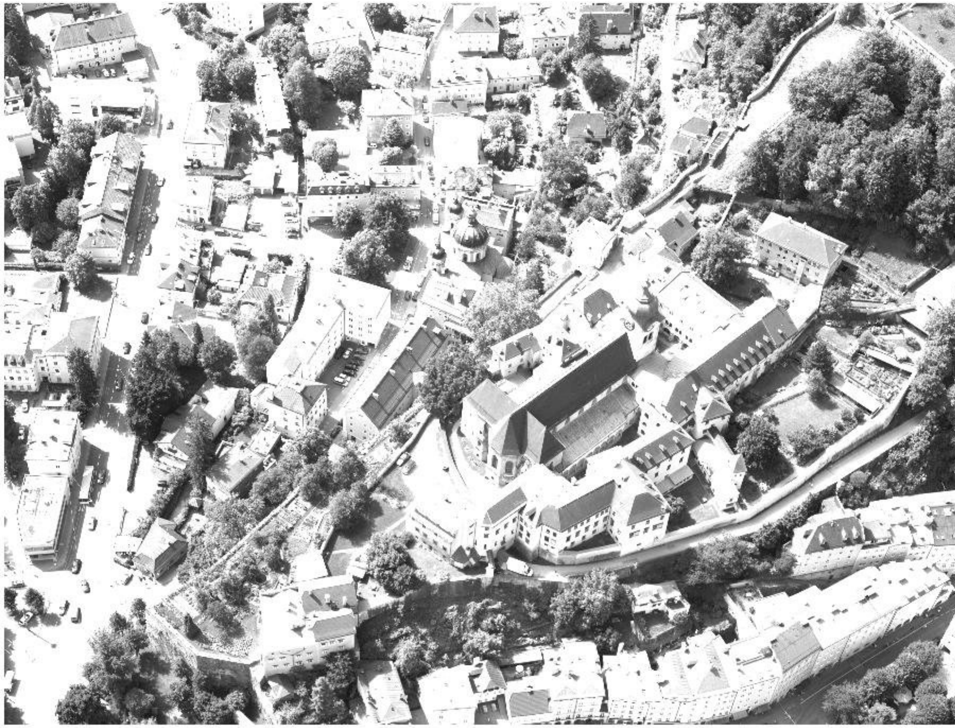


Image: J+20

Subtraction

Image: J

Image: J-50

Multiplication

Image: J

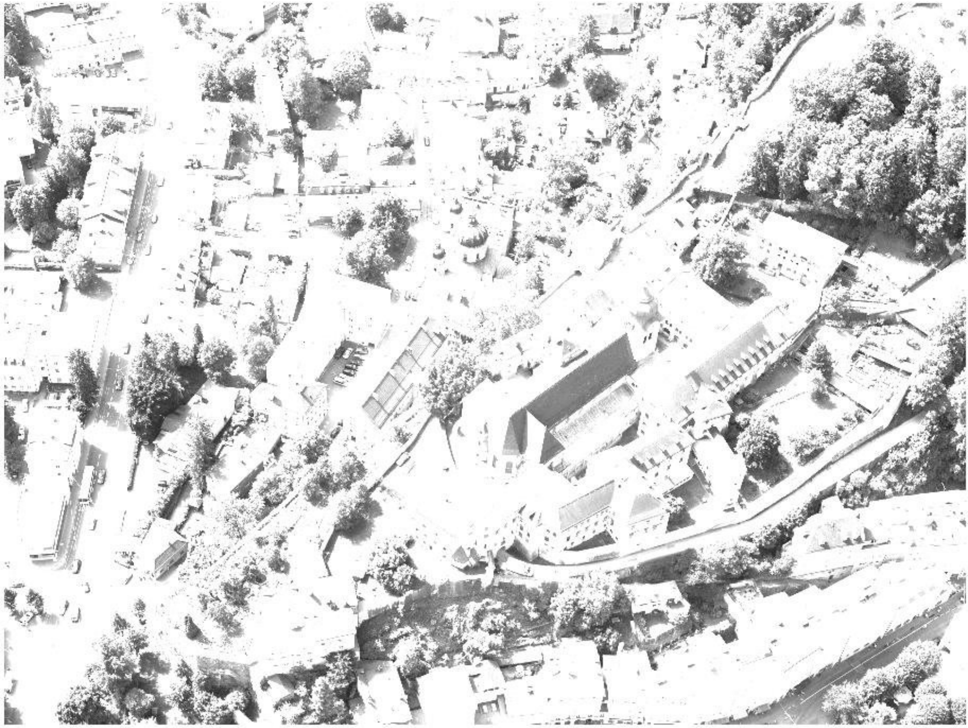


Image: J*3

Division

Image: J

Image: J/2

Complement

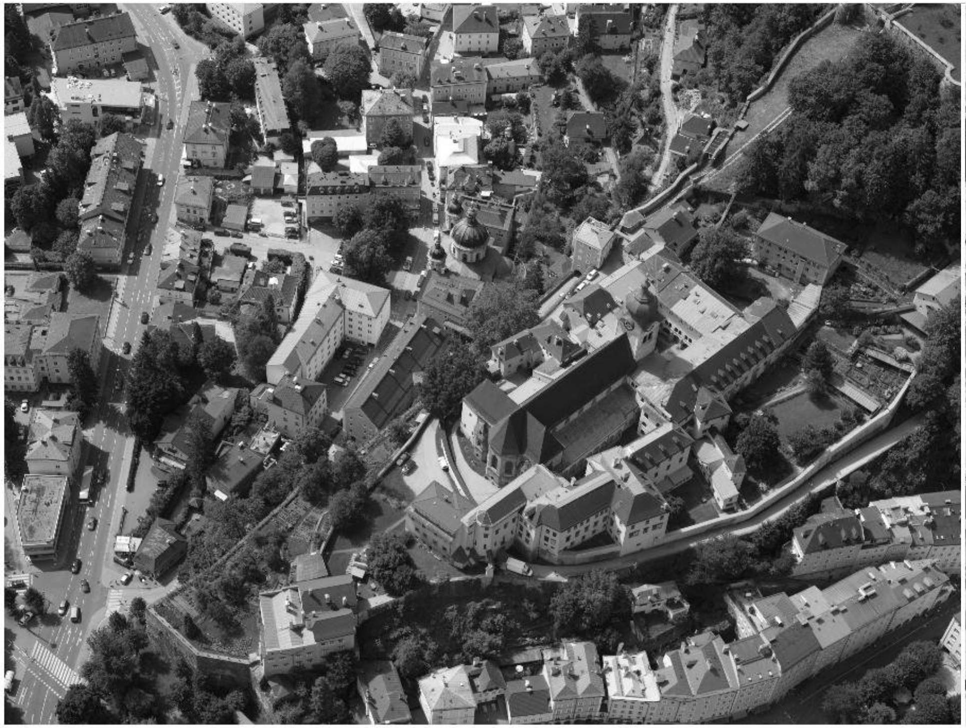


Image: J

Image: 255-J

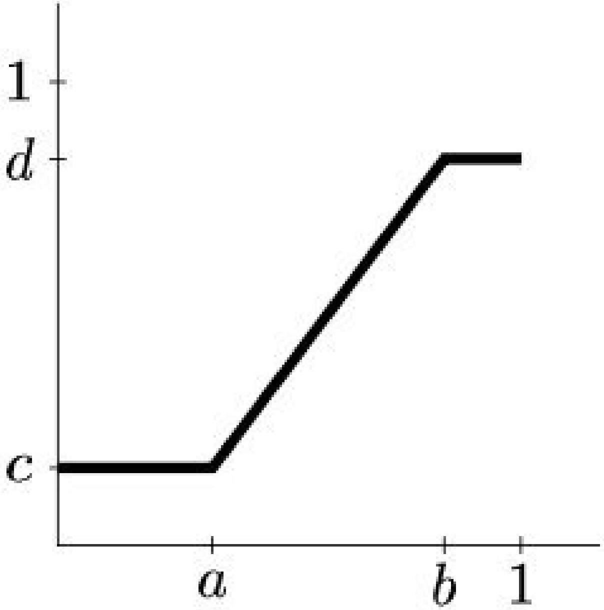
Histograms

- Given a grayscale image, its histogram consists of the histogram of its gray levels; that is, a graph indicating the number of times each gray level occurs in the image.
- We can infer a great deal about the appearance of an image from its histogram.
  - In a **dark** image, the gray levels would be clustered at the lower end
  - In a **uniformly bright** image, the gray levels would be clustered at the upper end.
  - In a **well contrasted** image, the gray levels would be well spread out over much of the range.
- **Problem**: Given a poorly contrasted image, we would like to enhance its contrast, by spreading out its histogram. There are **two** ways of doing this.

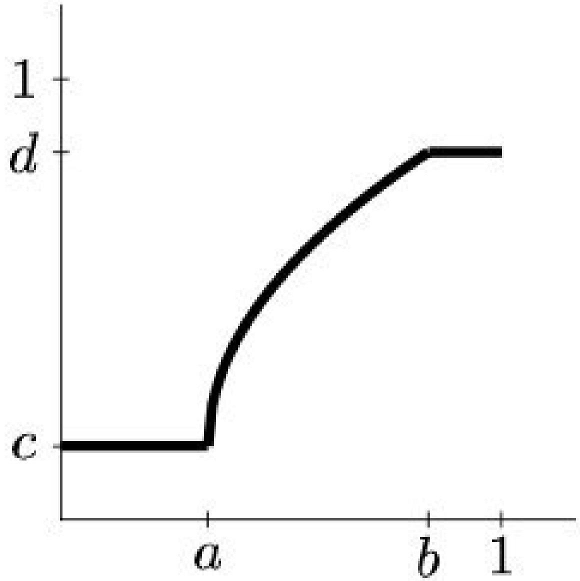
Histogram Stretching (Contrast Stretching)

- Poorly contrasted image of range [a,b]
- We can stretch the gray levels in the center of the range out by applying a piecewise linear function
- This function has the effect of stretching the gray levels [a,b] to gray levels [c,d], where a<c and d>b according to the equation:
- $$j = \frac{(c-d)}{(b-a)} \cdot (i-a) + c$$
- imadjust(I,[a,b],[c,d])
- Pixel values less than c are all converted to c, and pixel values greater than d are all converted to d.

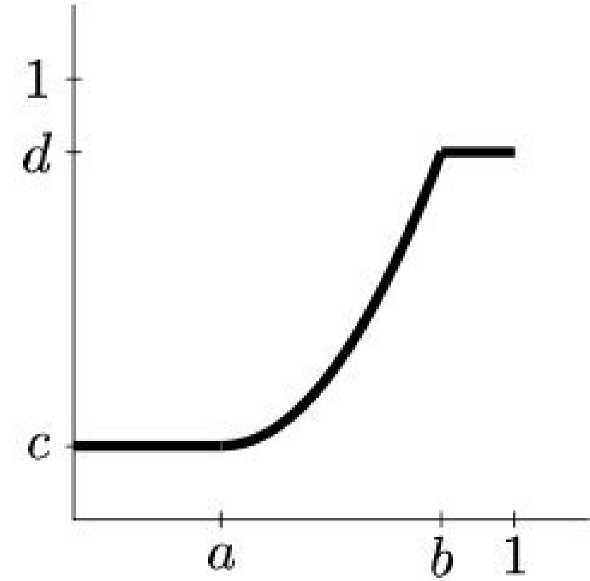
Histogram Stretching



$$y = \left(\frac{x-a}{b-a}\right)^{\gamma}(d-c)+c.$$

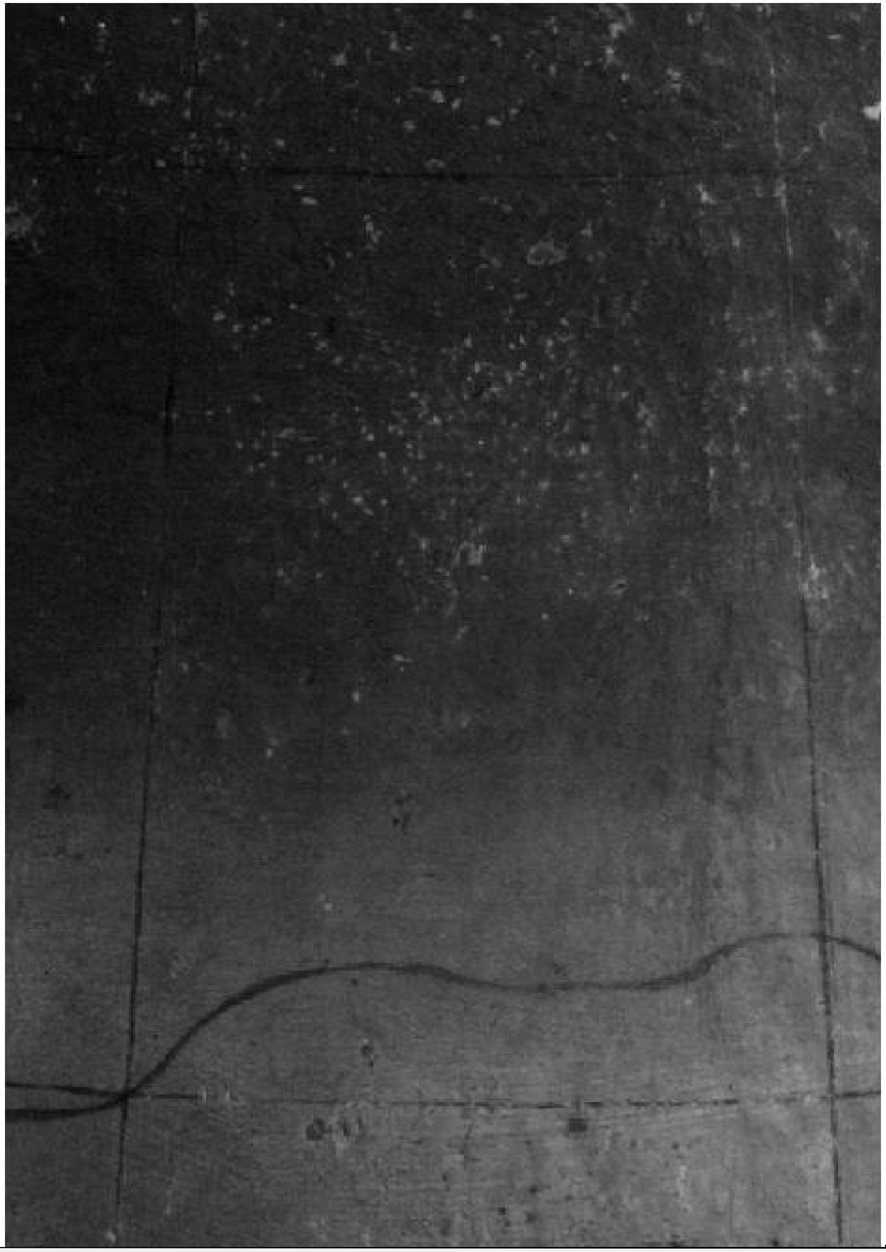


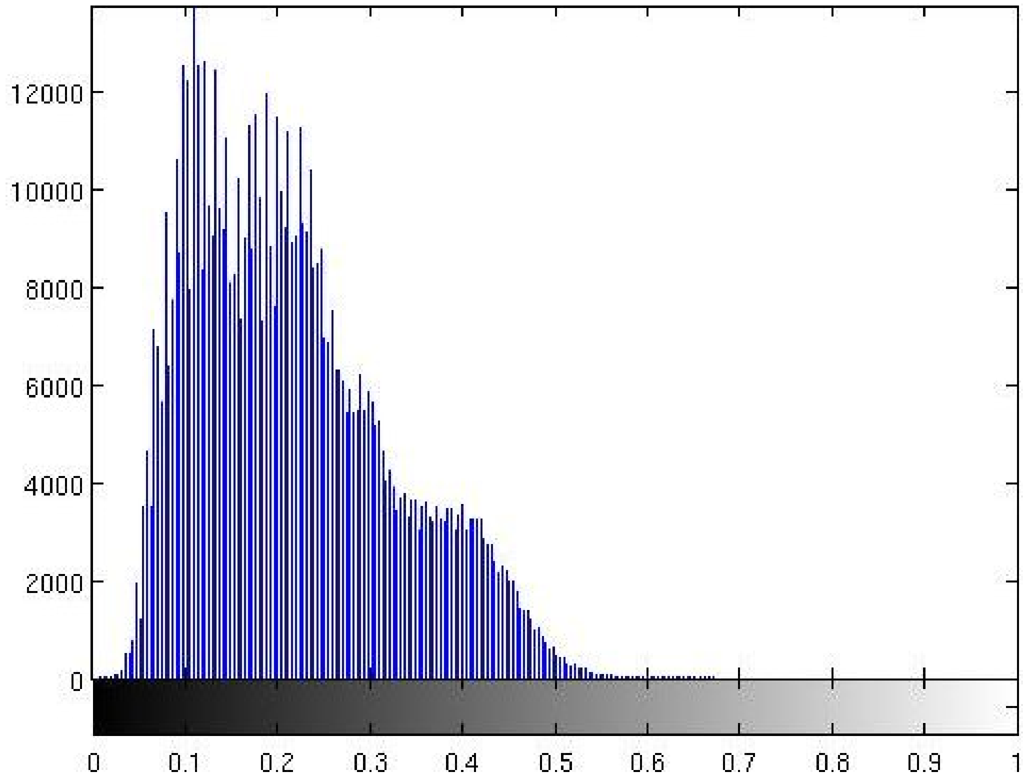
gamma < 1



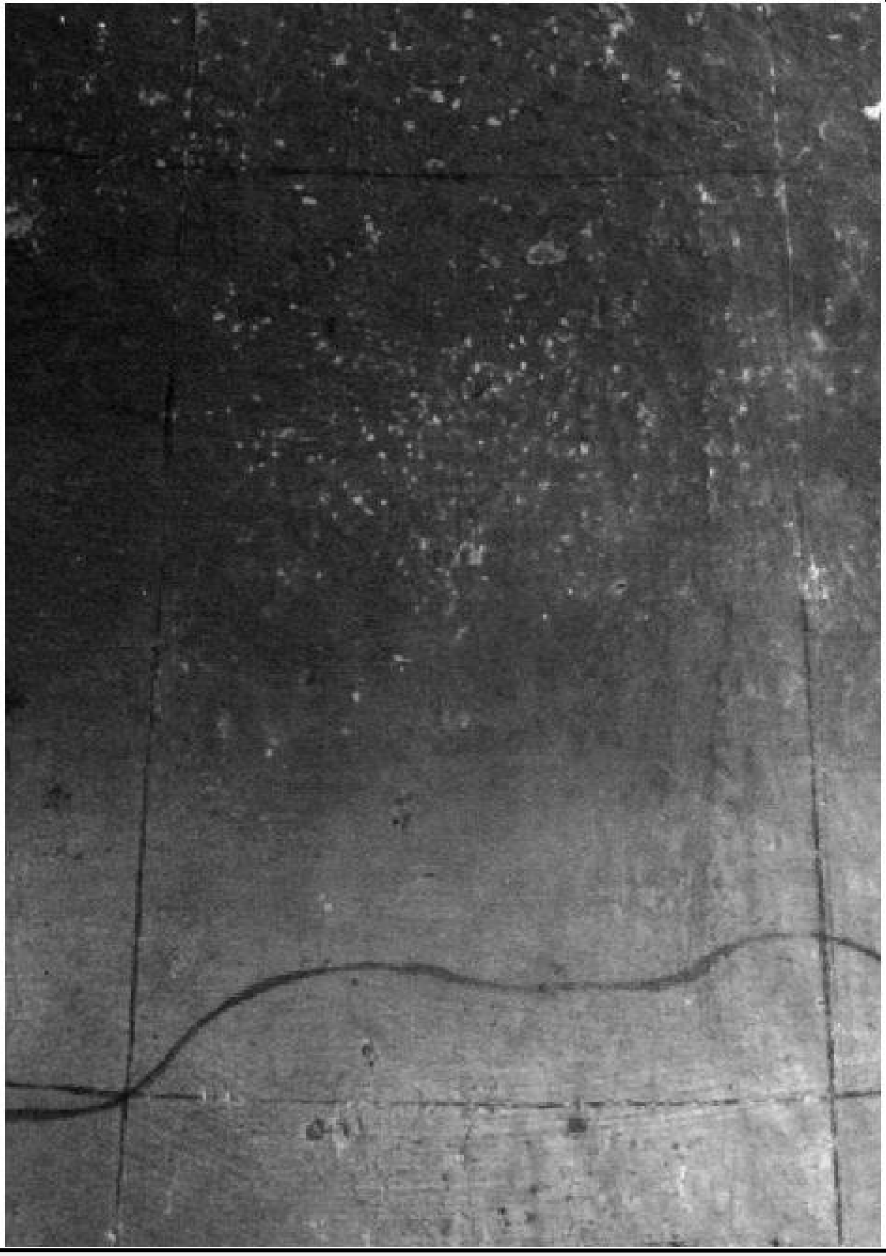
gamma > 1

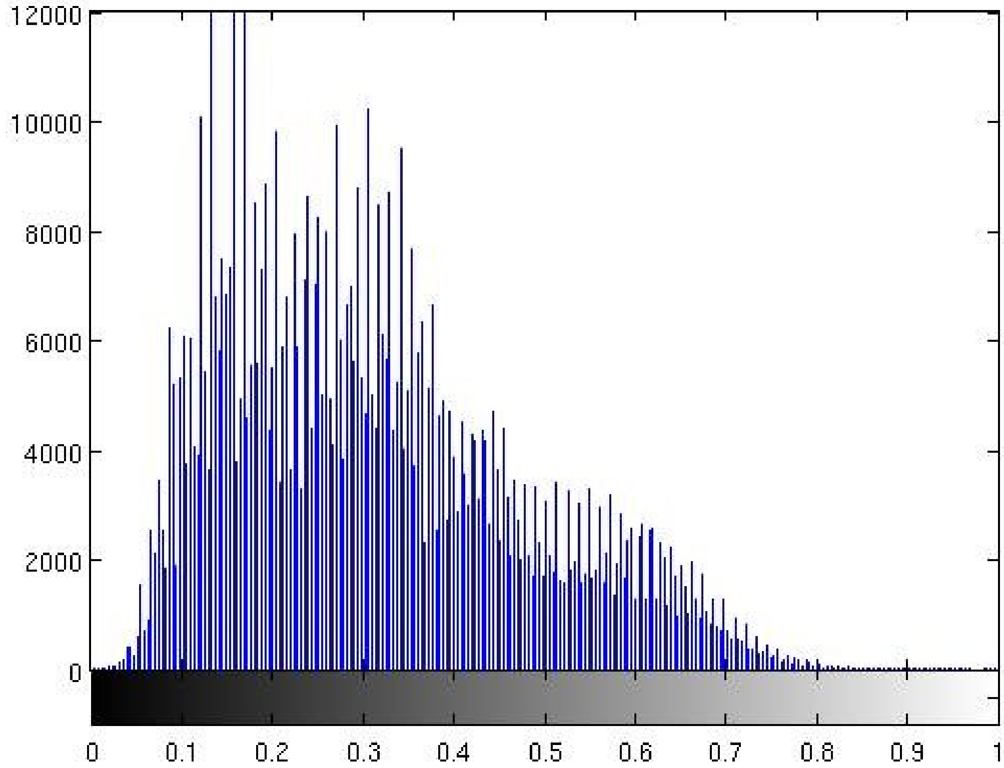
Before Histogram Stretching





After Histogram Stretching





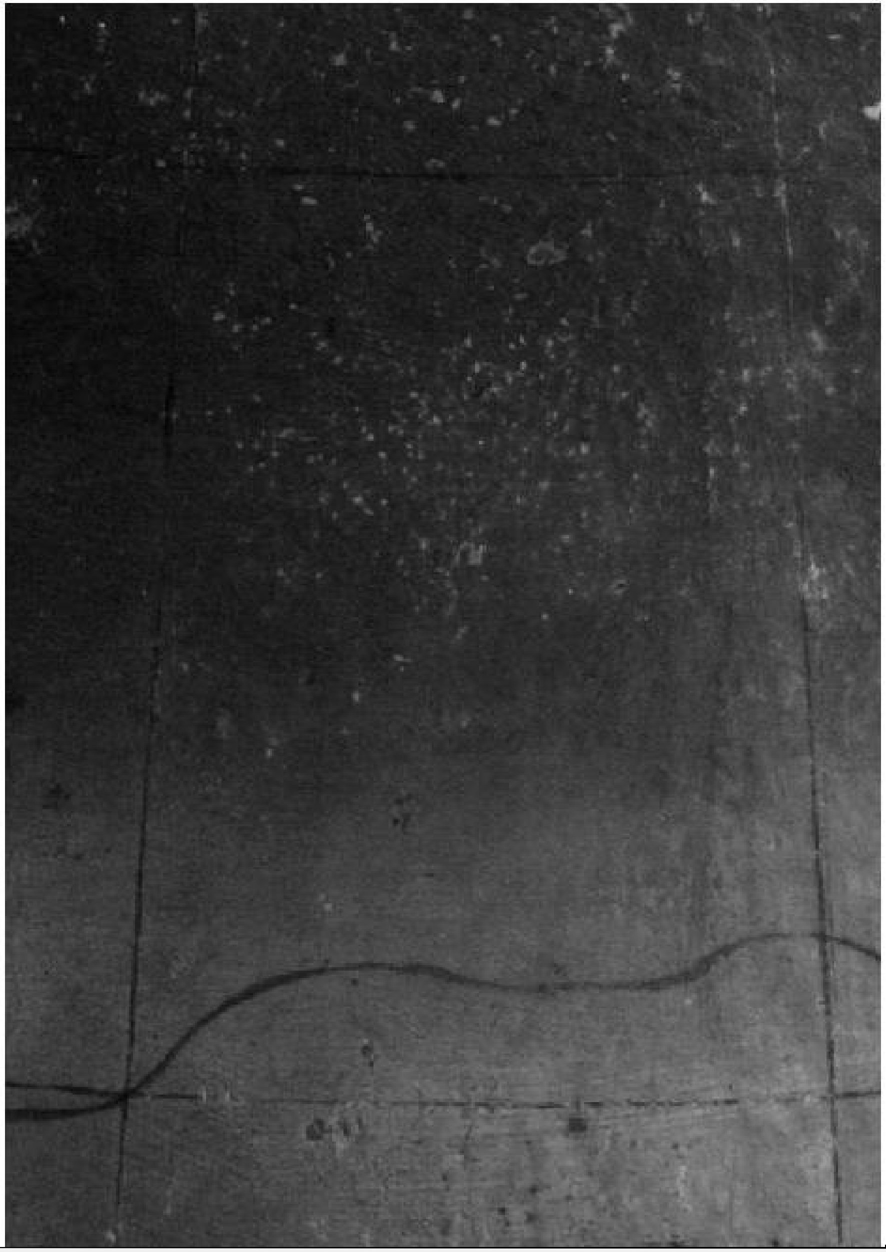
Histogram Equalization

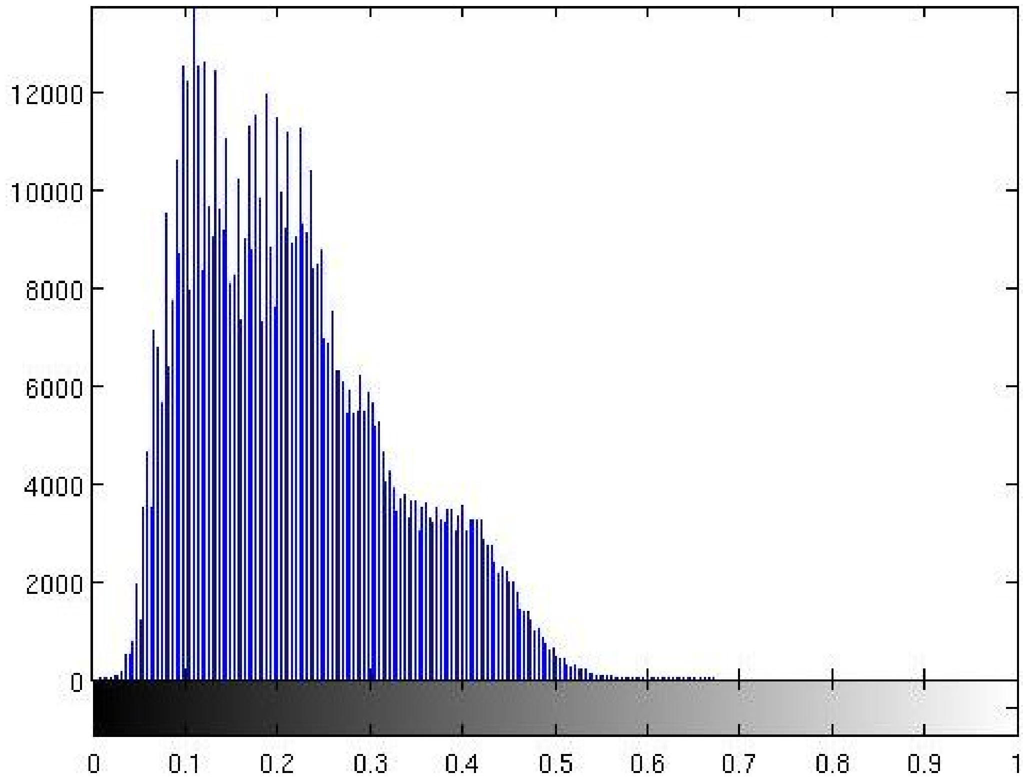
- The trouble with the previous method of histogram stretching is that they require user input.
- Histogram equalization, is an entirely automatic procedure.
- Suppose an image has ***L*** different gray levels ***0,1,2,...,1-L*** and that gray level ***i*** occurs n_i times in the image. Suppose also that the total number of pixels in the image is ***n*** so that $n_0+n_1+n_2+\dots n_L=n$. To transform the gray levels to obtain a better contrast el ***i*** to:

$$\left(\frac{n_0+n_1+\cdots+n_i}{n}\right)(L-1).$$

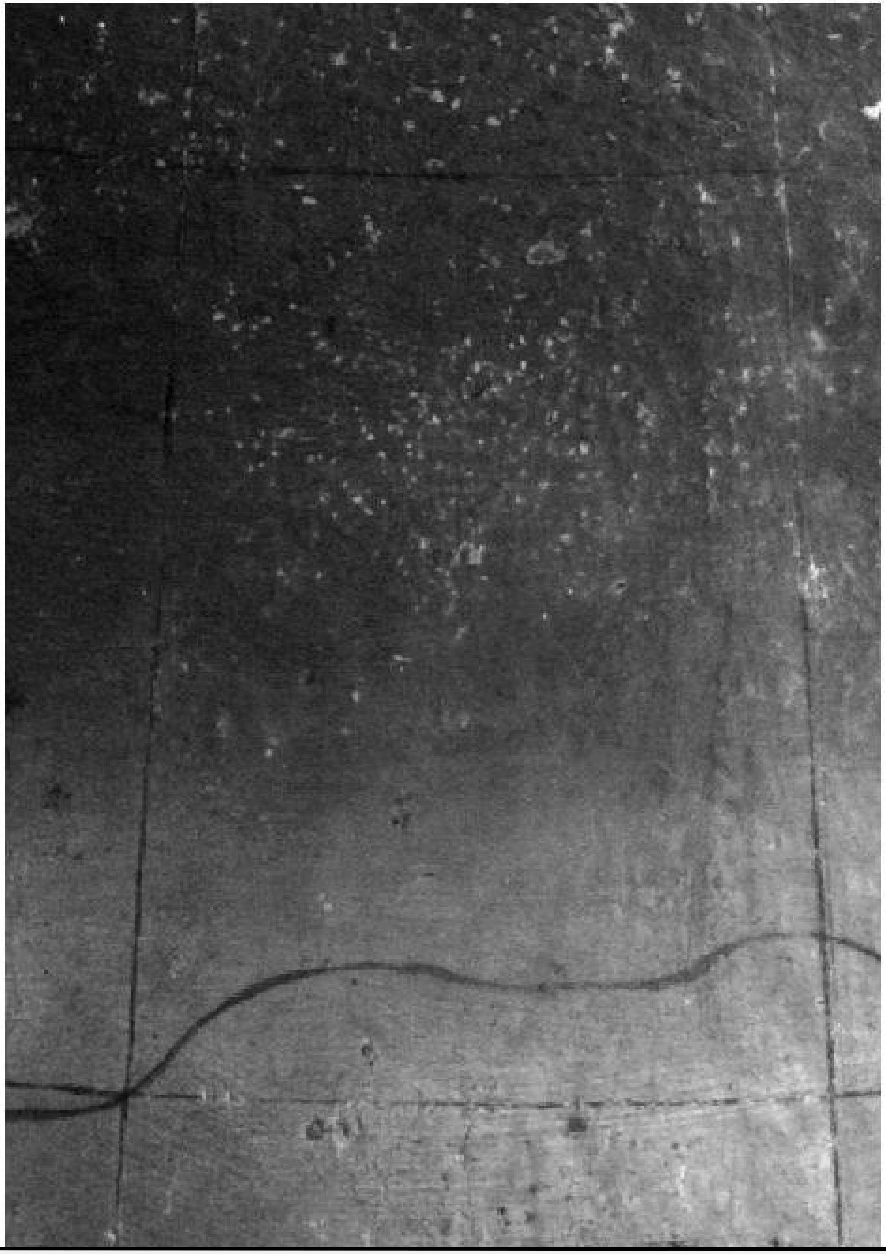
- and this number is rounded to the nearest integer.
- A roughly equal number of pixels is mapped to each of the ***L*** levels, so that the histogram of the output image is approximately flat.

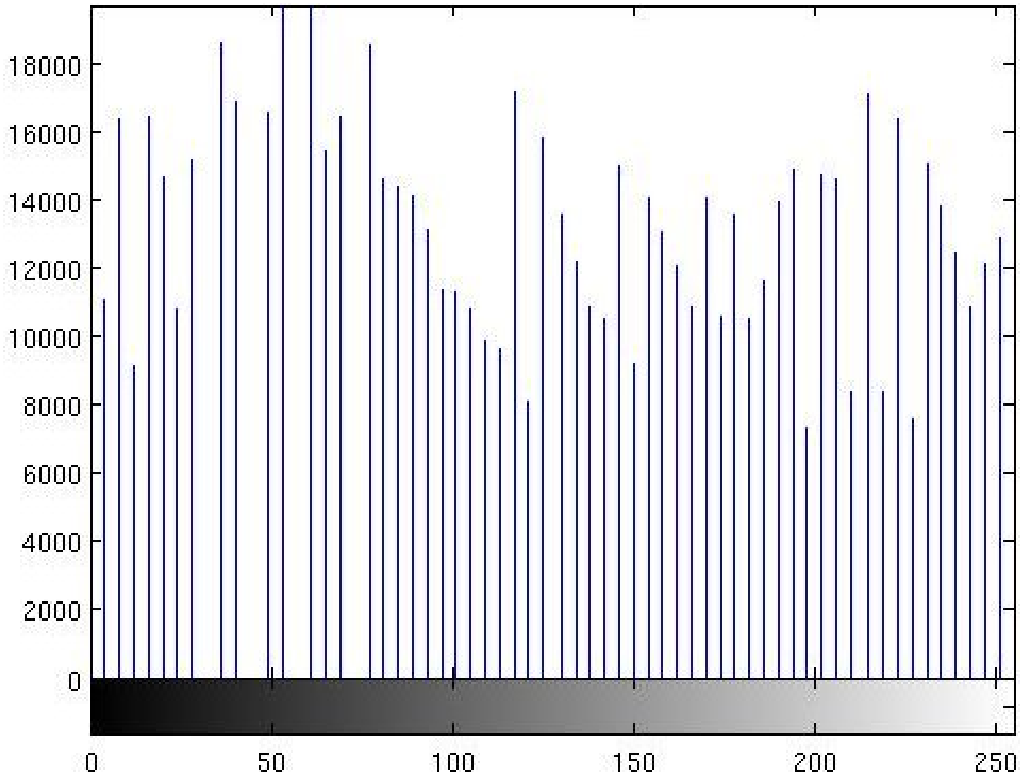
Before Histogram Equalization





After Histogram Equalization





Thresholding

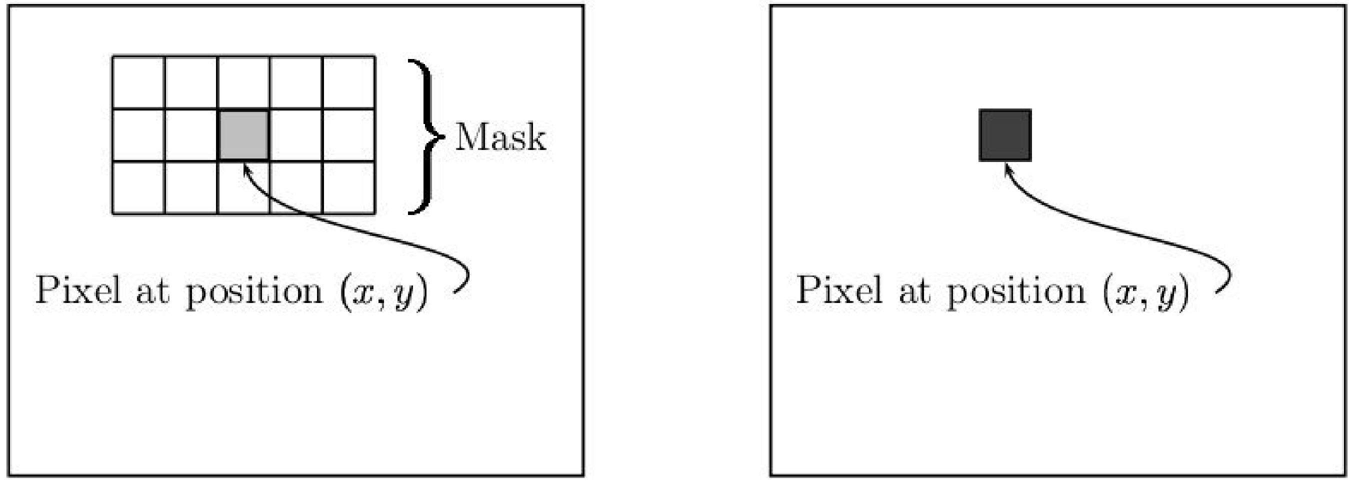
- **Single thresholding**: A grayscale image is turned into a binary image by first choosing a gray level ***T*** in the original image, and then turning every pixel black or white according to whether its gray value is greater than or less than ***T***.
  - A pixel becomes white if its gray level is **> *T***
  - A pixel becomes black if its gray level is **<= *T***
- Double thresholding: Here we choose two values T1 and T2 and apply a thresholding operation as:
  - A pixel becomes white if its gray level between ***T1*** and ***T2***
  - A pixel becomes black if its gray level is otherwise

Thresholding

Spatial Filtering

- Move a "**mask**": a rectangle (usually with sides of odd length) or other shape over the given image.
- A new image whose pixels have gray values calculated from the gray values under the mask.
- The combination of mask and function is called **filter**.
- Linear function of all the gray values in the mask, then the filter is called a **linear filter**.
- Spatial filtering requires 3 steps:
  1. position the mask over the current pixel,
  2. form all products of filter elements with the corresponding elements of the neighborhood,
  3. add up all the products.
- This must be repeated for every pixel in the image.
- filter2(filter,image,shape)

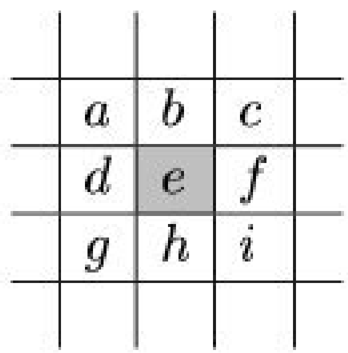
Masks



Original image

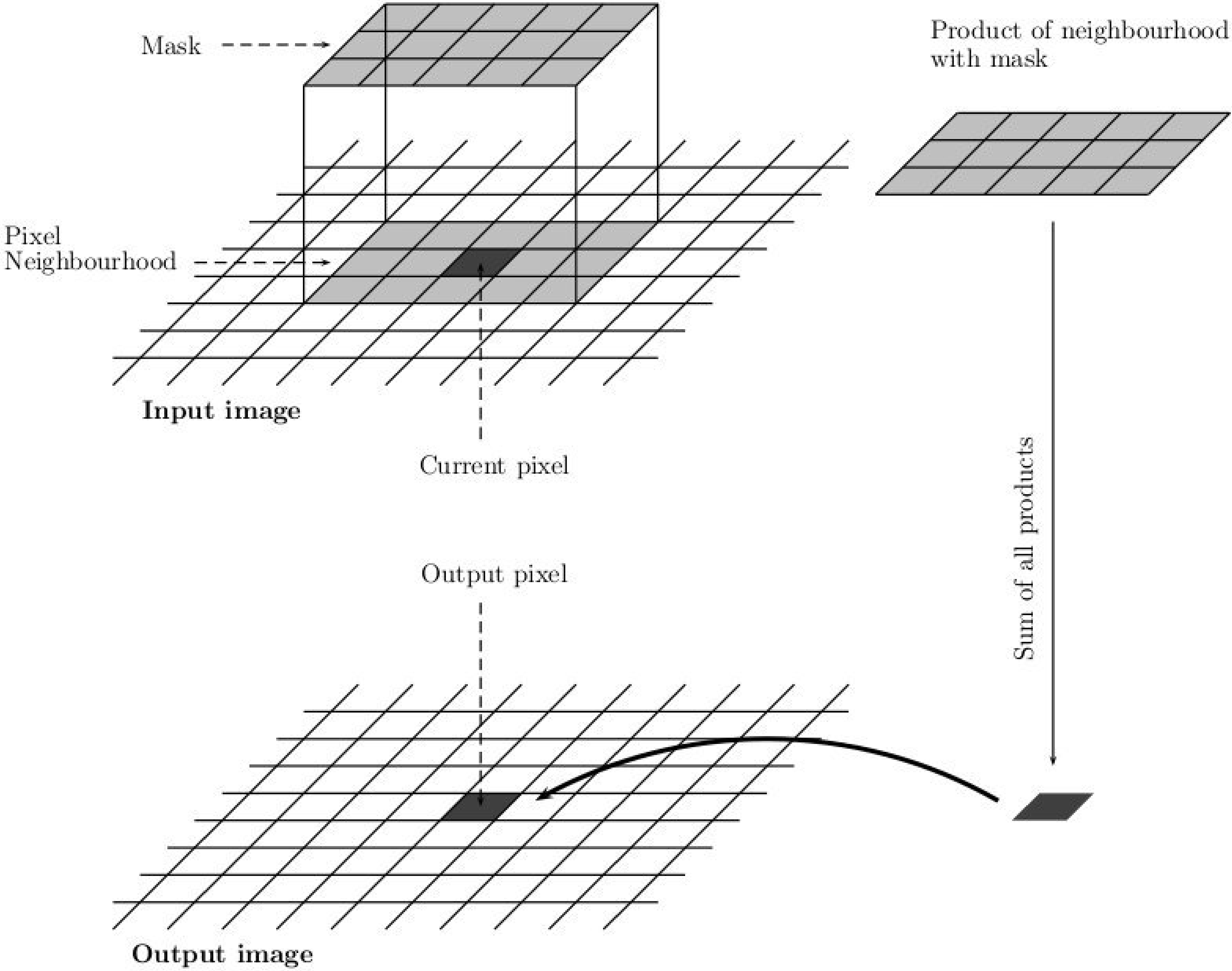
Image after filtering

$$\sum_{s=-1}^{1} \sum_{t=-2}^{2} m(s,t)p(i+s, j+t).$$



$$\longrightarrow \frac{1}{9}(a+b+c+d+e+f+g+h+i)$$

Filtering Working Flow



Frequencies; Low and High Pass Filters

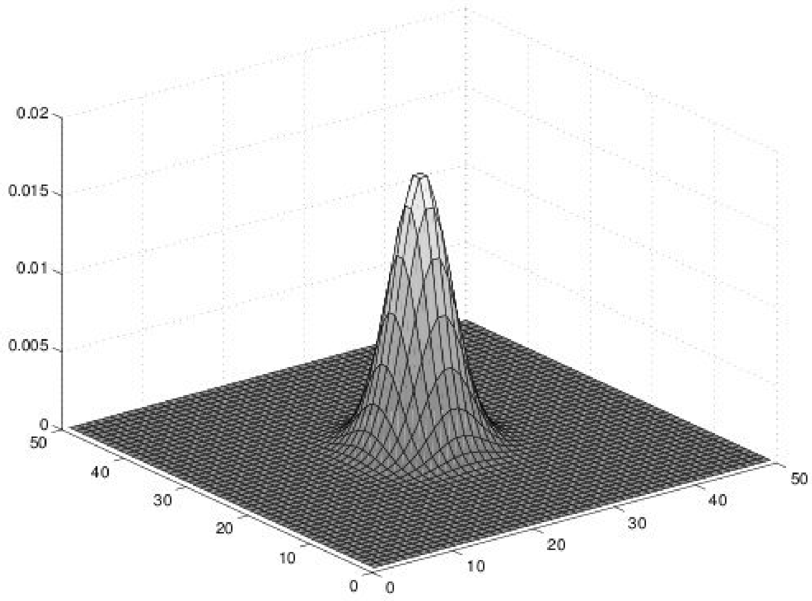
- **Frequencies** are the amount by which grey values change with distance.
- **High frequency components** are characterized by large changes in grey values over small distances; (edges and noise)
- **Low frequency components** are parts characterized by little change in the gray values. (backgrounds, skin textures)
- **High pass filter**: if it “passes over” the high frequency components, and reduces or eliminates low frequency components.
- **Low pass filter**: if it “passes over” the low frequency components, and reduces or eliminates high frequency components.

Gaussian Filters

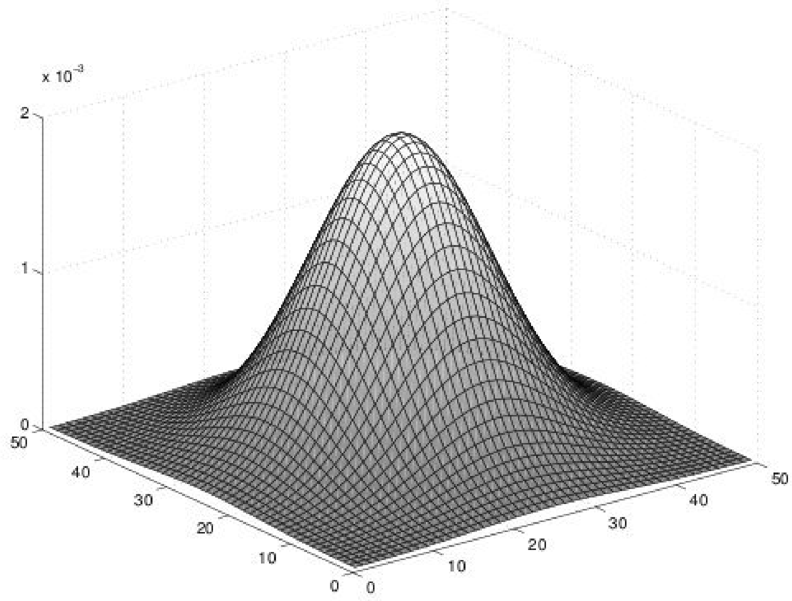
- Gaussian filters are a class of low-pass filters, all based on the Gaussian probability distribution

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

- where ***σ*** is the standard deviation: a large value ***σ*** of produces to a flatter curve, and a small value ***σ*** leads to a "pointier" curve.
- Blurrin

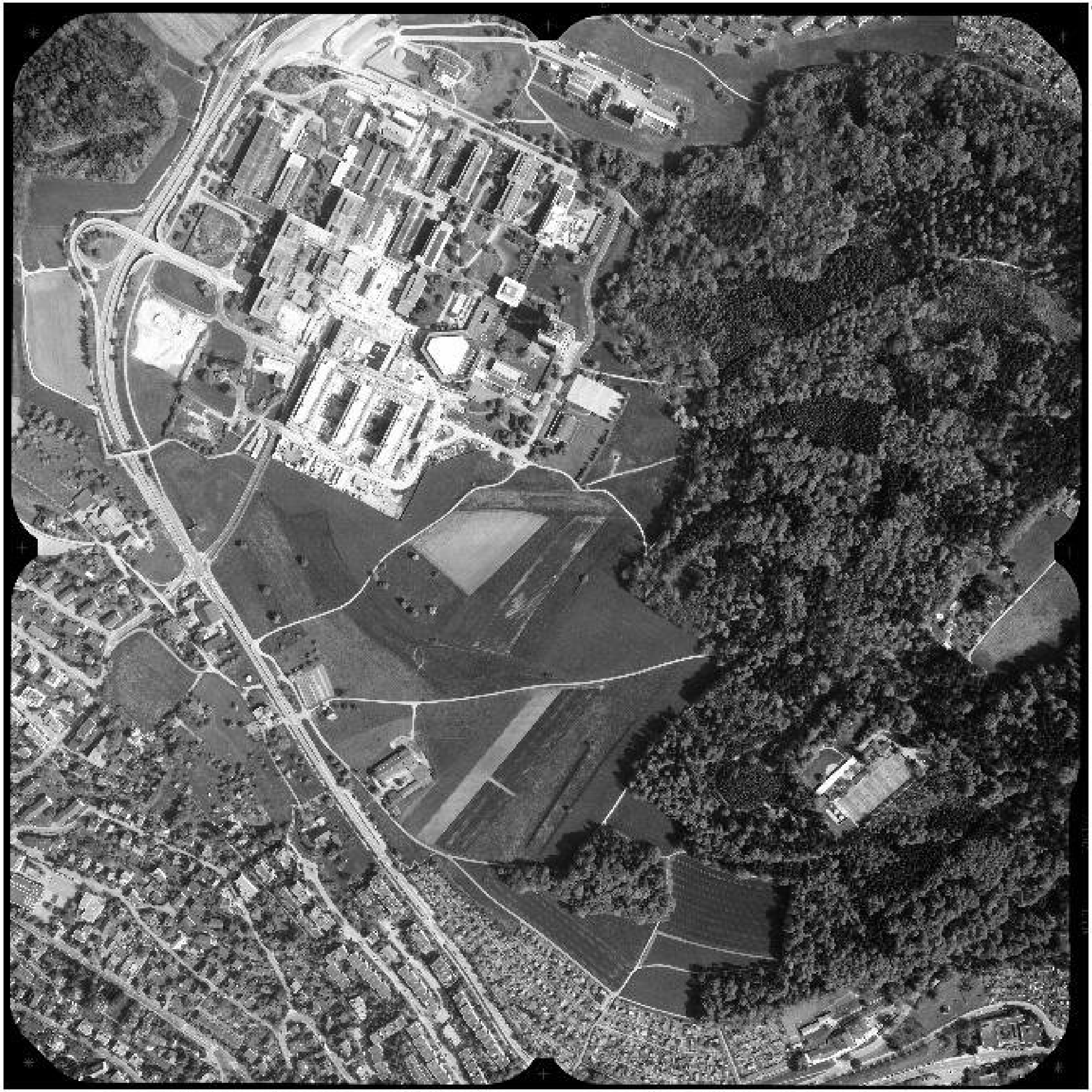


$\sigma = 3$



$\sigma = 9$

Gaussian Filters

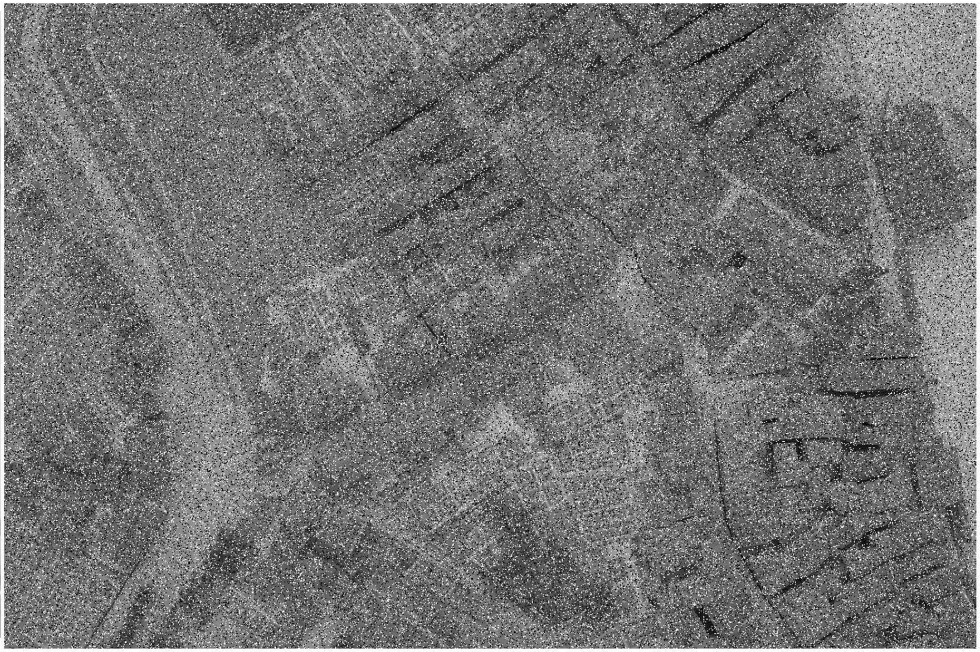


Gaussian Filters

Noise

- Noise is any degradation in the image signal, caused by external disturbance.
- **Salt and pepper noise**: It is caused by sharp, sudden disturbances in the image signal; it is randomly scattered white or black (or both) pixels. It can be modeled by random values **added** to an image
- **Gaussian noise**: is an idealized form of white noise, which is caused by random fluctuations in the signal.
- **Speckle noise**: It is a major problem in some radar applications. It can be modeled by random values multiplied by pixel values.
-

Salt & Pepper Noise



Gaussian Noise

Speckle Noise

Edge Detection

- Motivation: detect changes

change in the pixel value $\longrightarrow$ large gradient

image $x(m,n)$ $\longrightarrow$ $\boxed{\text{Gradient operator}}$ $\xrightarrow{g(m,n)}$ $\boxed{\text{Thresholding}}$ $\longrightarrow$ edge map $I(m,n)$

$$I(m,n) = \begin{cases} 1 & |g(m,n)| > th \\ 0 & otherwise \end{cases}$$

We can implement those two steps by basic MATLAB functions.

Common Edge Operators

1. Prewitt operator

2. Sobel operator

vertical

$$\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

horizontal

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Canny Edge Detector

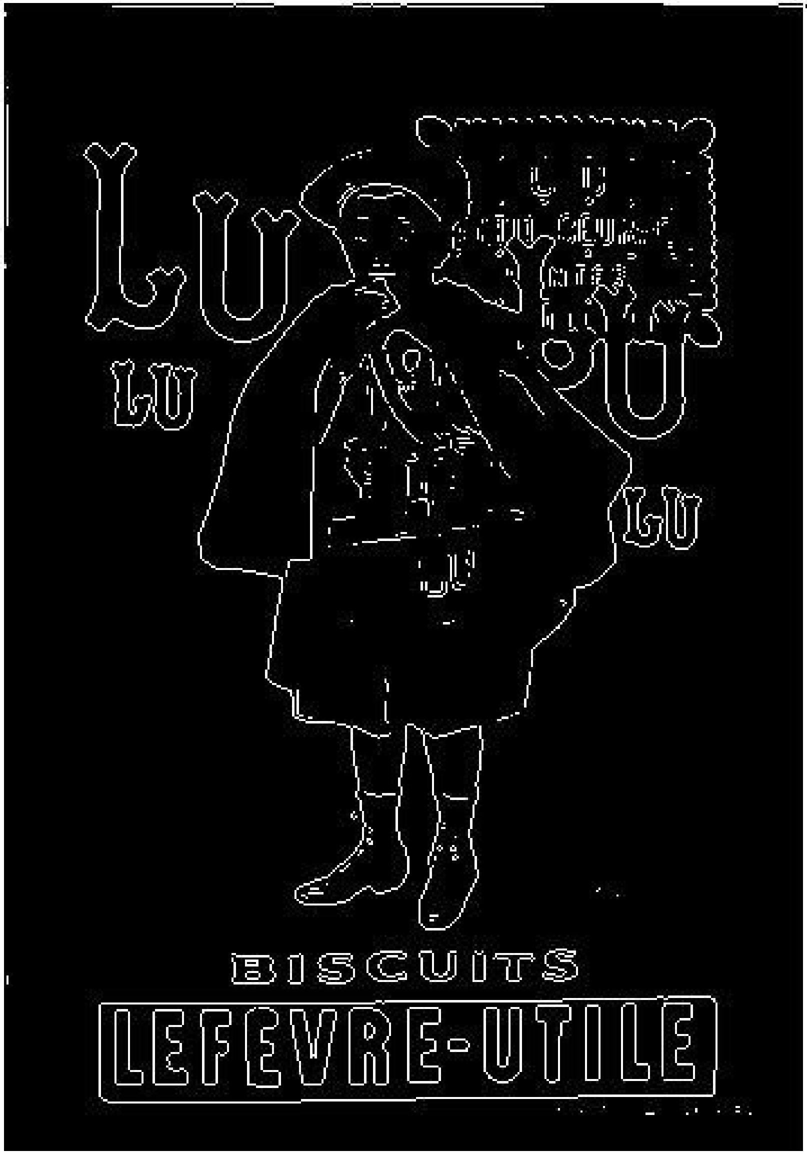
- **Low error rate of detection**
  - Well match human perception results
- **Good localization of edges**
  - The distance between actual edges in an image and the edges found by a computational algorithm should be minimized
- **Single response**
  - The algorithm should not return multiple edges pixels when only a single one exists

Edge Detectors

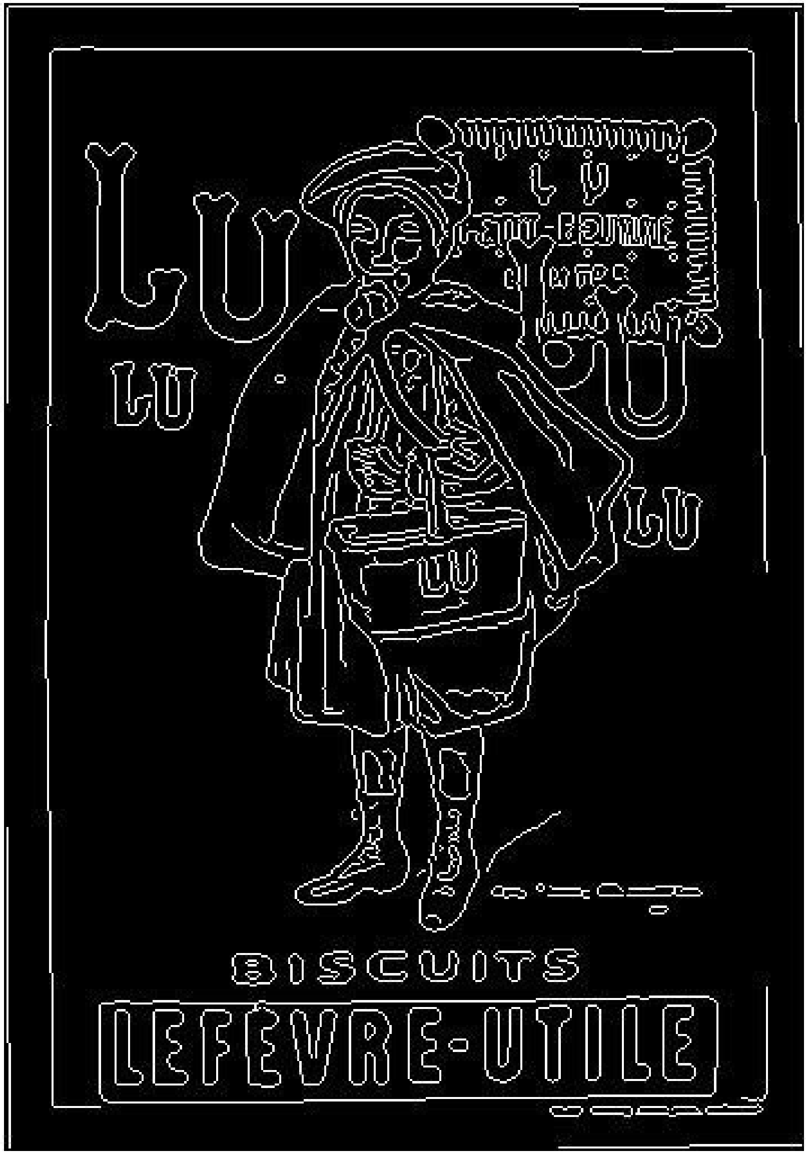
Color Image

Grayscale Image

Edge Detectors



Sobel

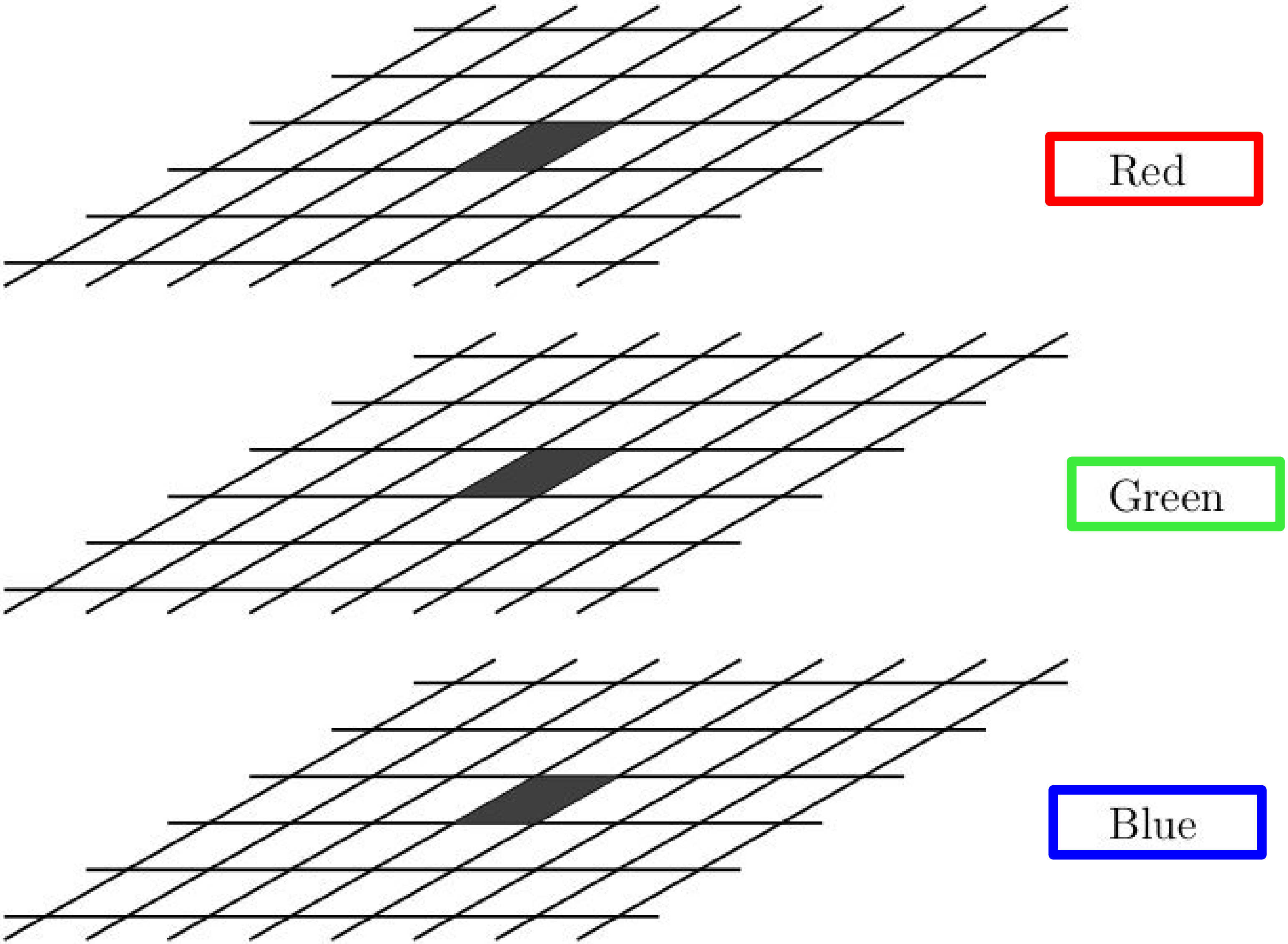


Canny

Color Images

- A **color model** is a method for specifying colors in some standard way. It generally consists of a 3D coordinate system and a subspace of that system in which each color is represented by a single point.
- **RGB**: In this model, each color is represented as 3 values ***R***, ***G*** and ***B***, indicating the amounts of red, green and blue which make up the color.
- **HSV**:
  - Hue: The “true color” attribute (red, green, blue, orange, yellow, and so on).
  - Saturation: The amount by which the color as been diluted with white. The more white in the color, the lower the saturation.
  - Value: The degree of brightness: a well lit color has high intensity; a dark color has low intensity.

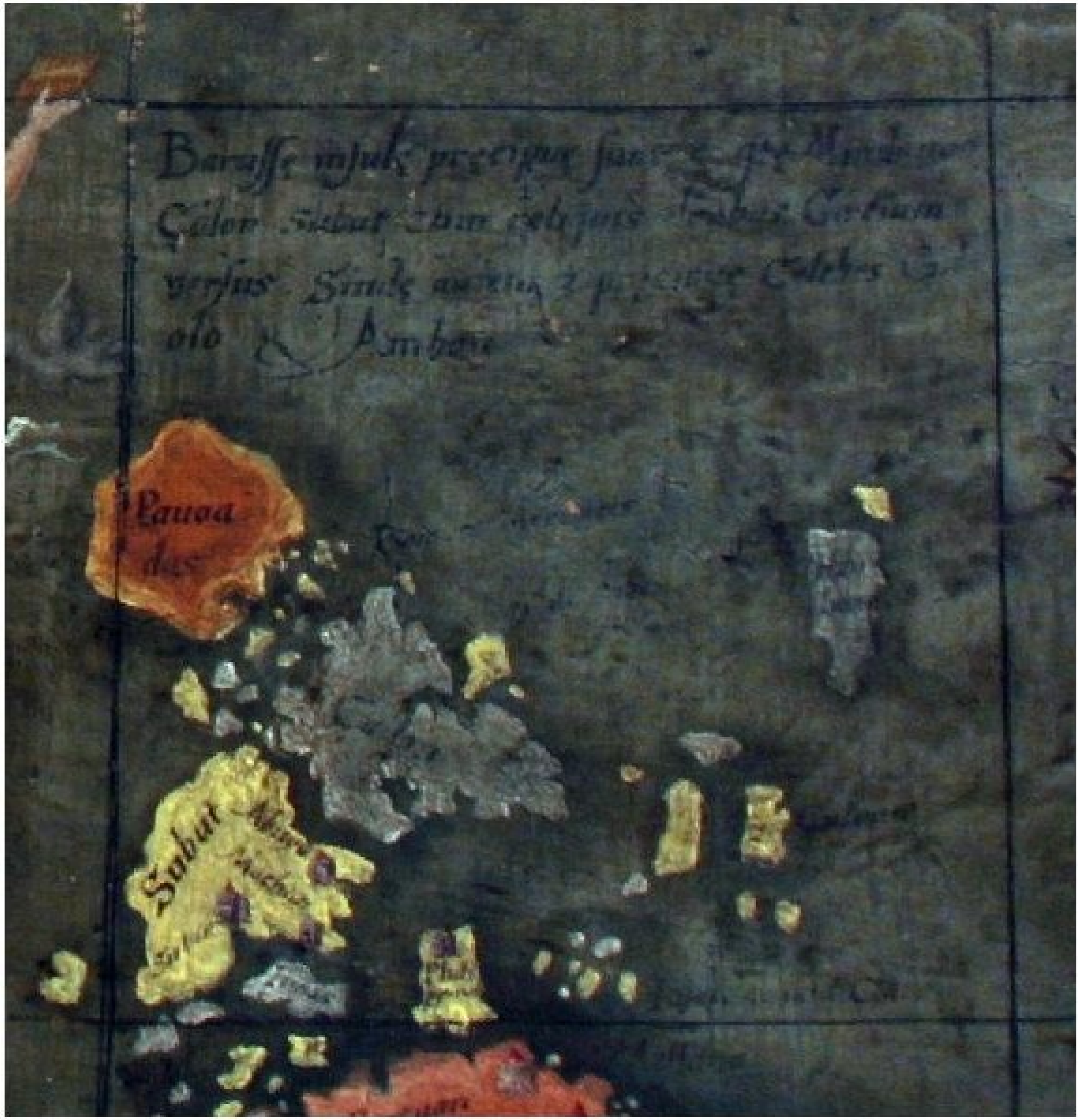
Color Image



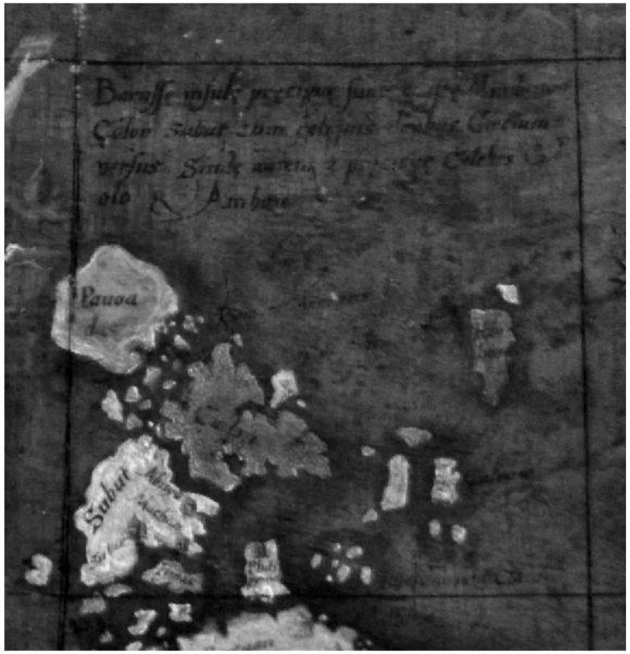
Color Conversion

Function	Use	Format
ind2gray	Indexed to Greyscale	y=ind2gray(x,map);
gray2ind	Greyscale to indexed	[y,map]=gray2ind(x);
rgb2gray	RGB to greyscale	y=rgb2gray(x);
gray2rgb	Greyscale to RGB	y=gray2rgb(x);
rgb2ind	RGB to indexed	[y,map]=rgb2ind;
ind2rgb	Indexed to RGB	y=ind2rgb(x,map);

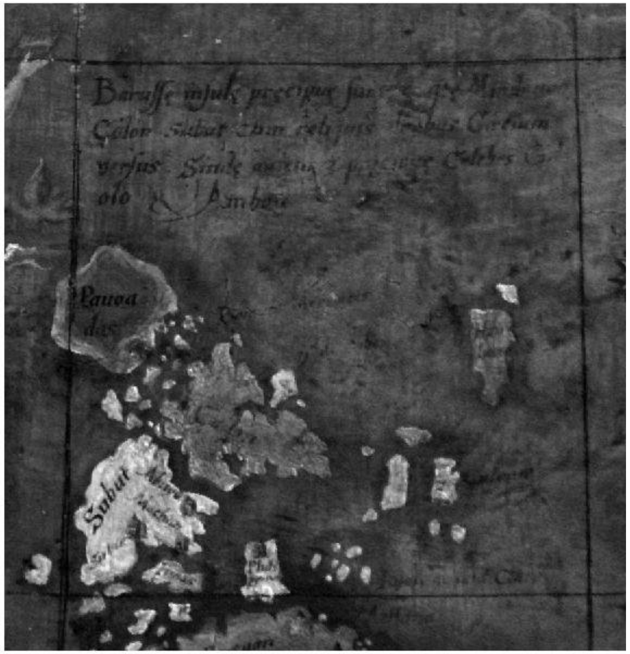
Initial Color Image



RGB Color Space



RED

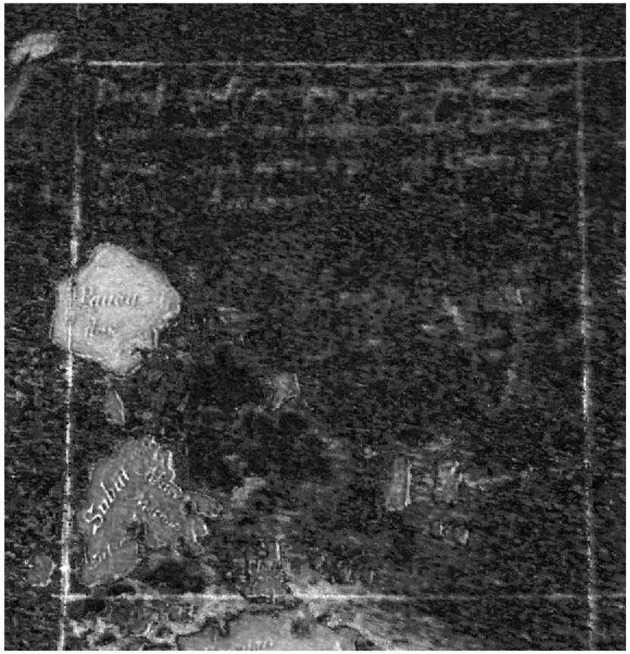


GREEN

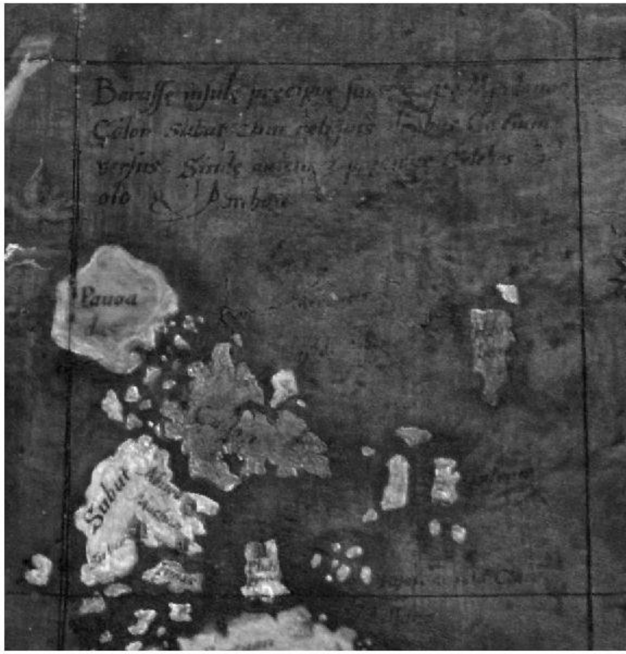
BLUE

HSV Color Space

HUE



SATURATION

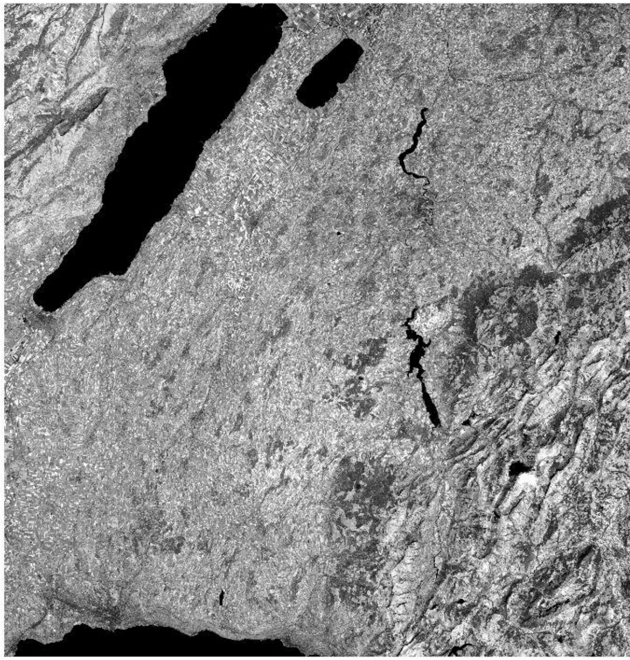


VALUE

COLOR COMPOSITE

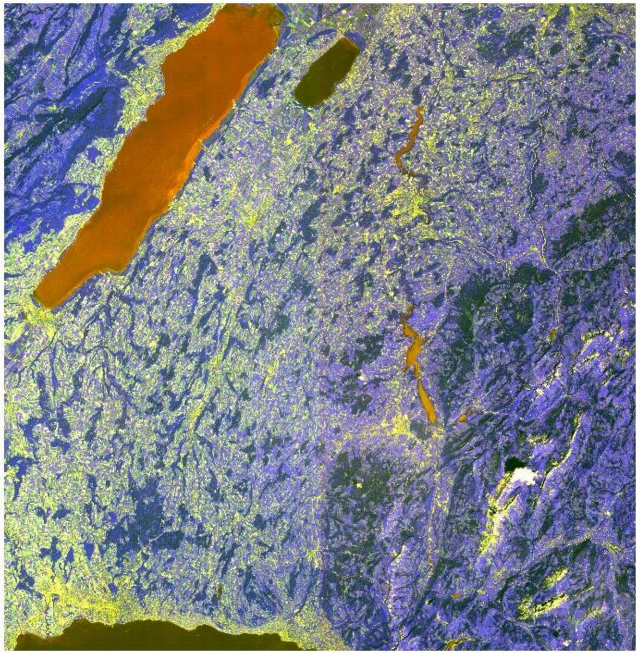
Aster 1

Aster 2

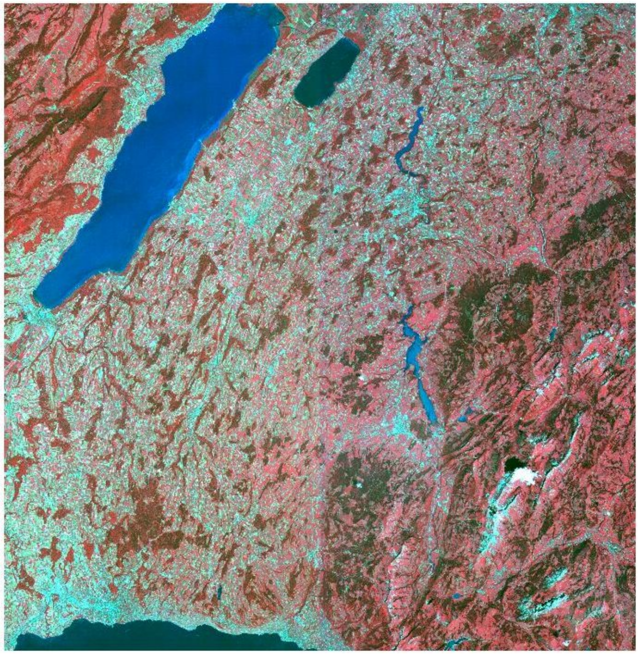


Aster 3

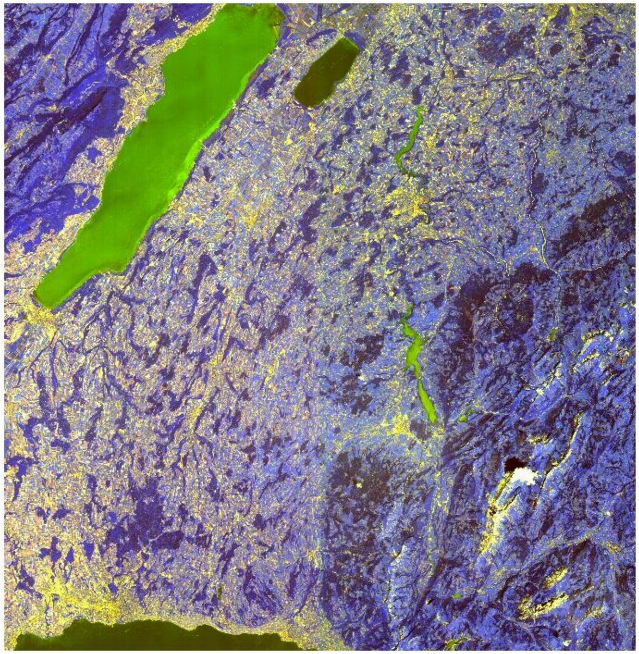
COLOR COMPOSITE



Aster 1-2-3



Aster 3-2-1



Aster 2-1-3