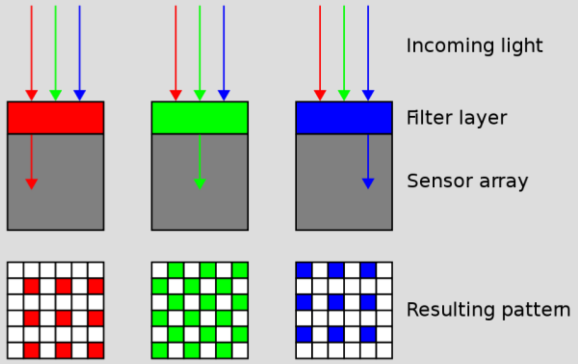
Debayering-Demosaicing

Bayer filter on a single sensor



The very first step in most cameras is an upsampling operation:

- the true values in the red and blue channels are only known at 25% of the pixels
- the true values in the green channel are known at 50% of all positions

Usually one estimates the missing values in the green channel first and then one computes the missing red and blue values since the green values often contain information about the red and the blue values

Interpolation

How do we determine the pixel values in the new image (zoomed or shrunk)?

Interpolation is the process of using known data to estimate values in new locations.

Interpolation methods:

- Nearest neighbour
- Bilinear interpolation
- Cubic spline interpolation

Bilinear and Bicubic Interpolation

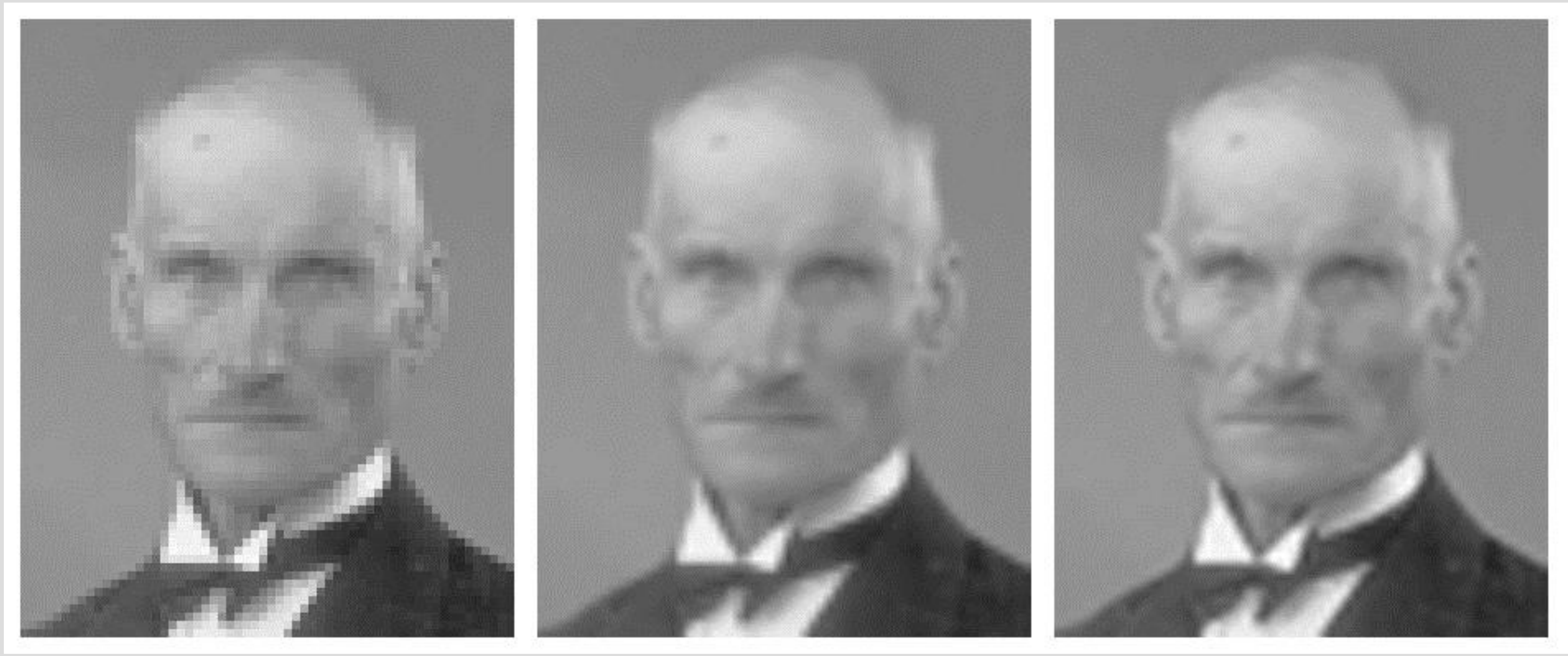
(x,y) is the pixel position and v(x,y) the value to be computed

Bilinear: $v(x, y) = ax + by + cxy + d$

Bicubic: $v(x, y) = \sum_{i=0}^{3} \sum_{j=0}^{3} a_{ij} x^i y^j$

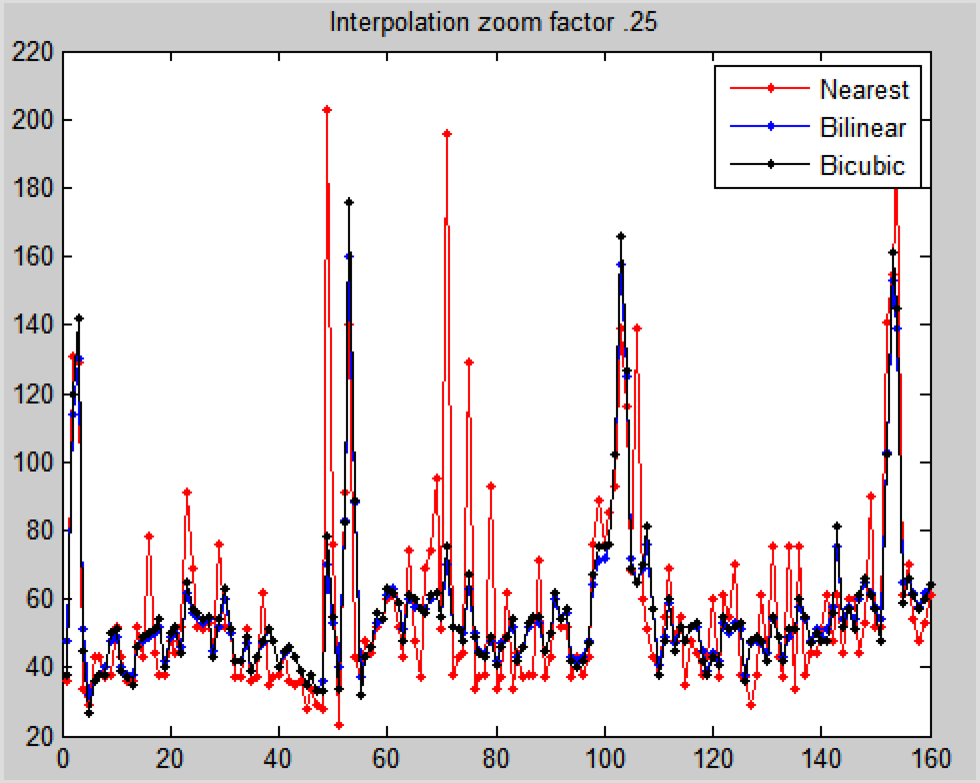
4 unknowns a,b,c,d & 4 equations from the 4 neighbors
16 unknowns from 16 nearest neighbors

Interpolation

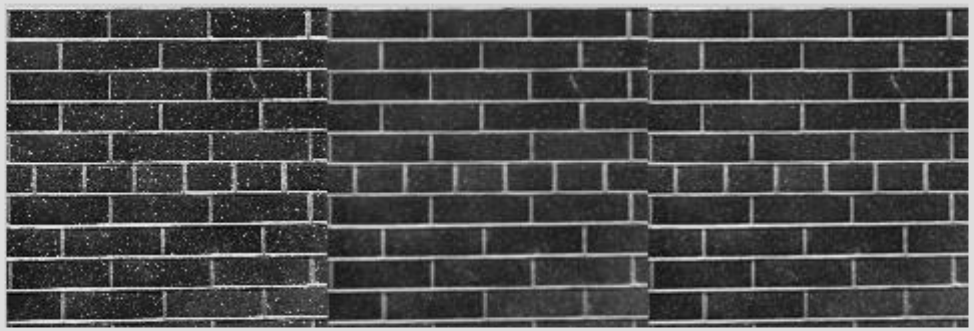


Nearest neighbour Bilinear Cubic spline

Matlab, Resize 1/4



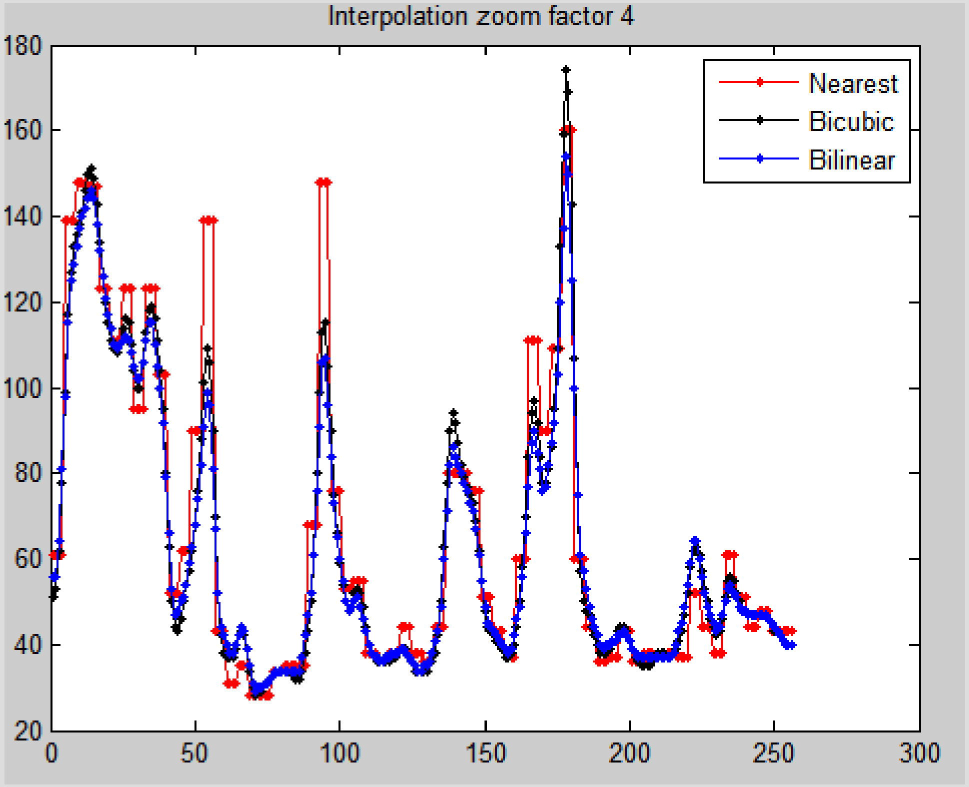
```
JB = imresize(I, 0.25,'bilinear');
JC = imresize(I, 0.25,'bicubic');
JN = imresize(I, 0.25,'nearest');
jj = cat(2,JN,JB,JC);
figure(1)
imshow(jj,'InitialMagnification',100)
figure(2)
plot(JN(128,:),'r.-')
hold on
plot(JB(128,:),'b.-')
plot(JC(128,:),'k.-')
legend('Nearest','Bilinear','Bicubic')
title('Interpolation zoom factor .25')
```



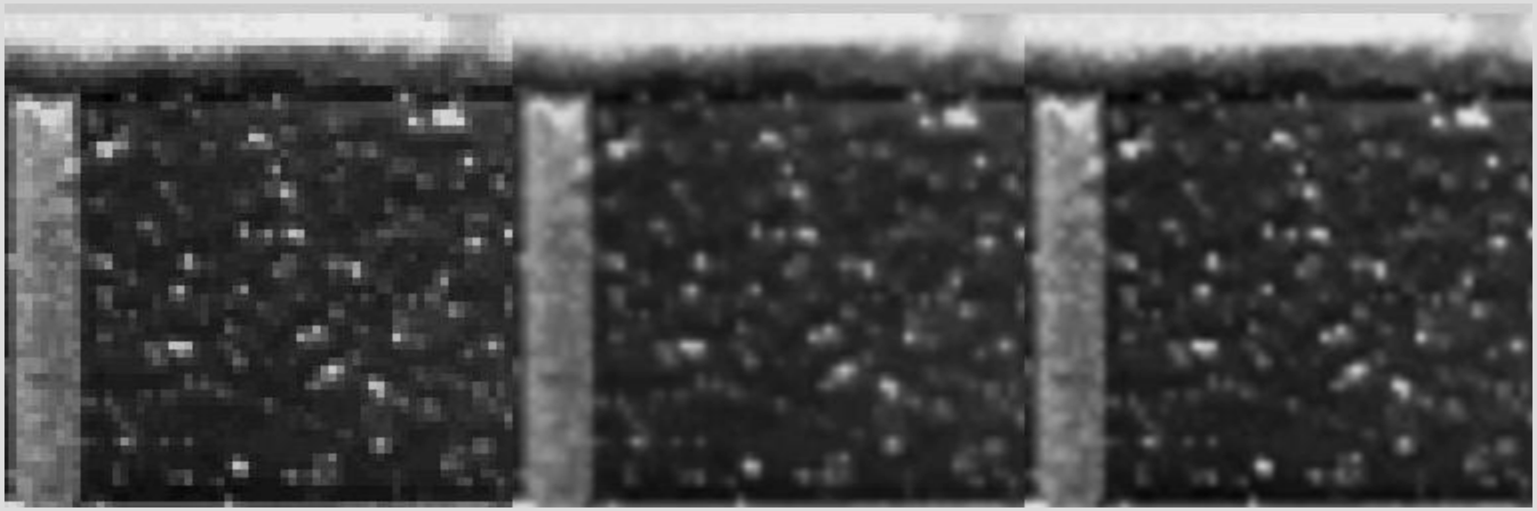
..\Matlab\ResizeDemo.m

..\Matlab\html\ResizeDemo.html

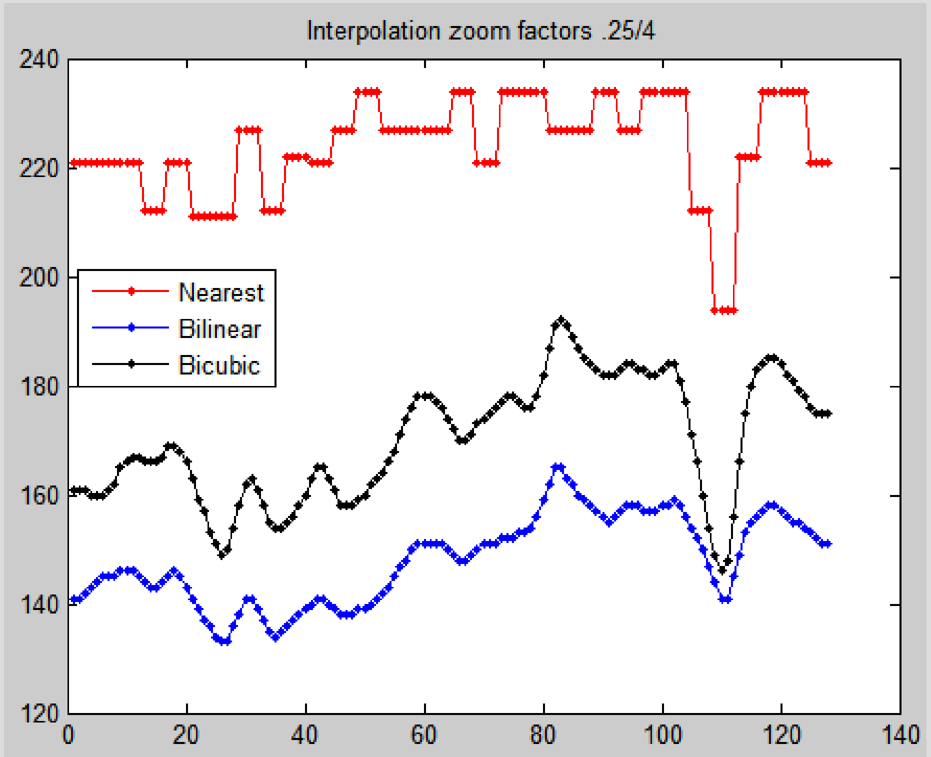
Matlab Resize 4



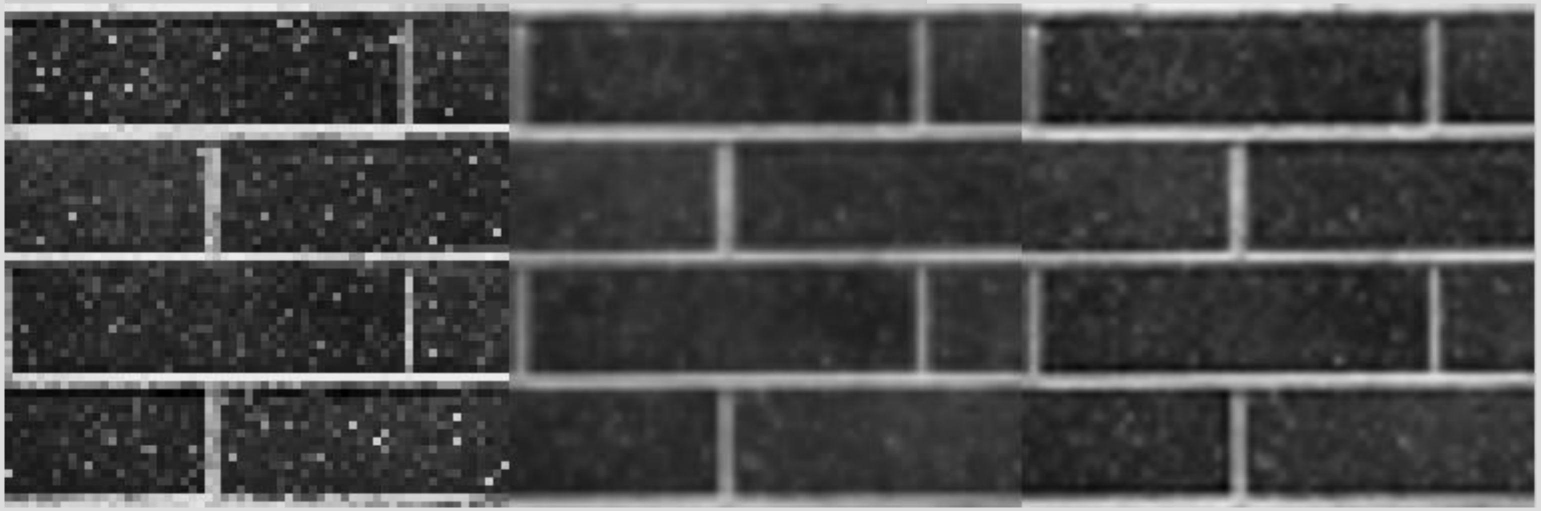
```
I = imread('D26.gif');
BN = imresize(I(1:64,1:64), 4,'nearest');
BB = imresize(I(1:64,1:64), 4,'bilinear');
BC = imresize(I(1:64,1:64), 4,'bicubic');
kk = cat(2,BN,BB,BC);
imshow(kk,'InitialMagnification',100)
figure
plot(BN(128,:),'r.-')
hold on
plot(BC(128,:),'k.-')
plot(BB(128,:),'b.-')
legend('Nearest','Bicubic','Bilinear')
title('Interpolation zoom factor 4')
```



Matlab, Resize 0.25/4



```
LN = imresize(imresize(I,0.25,'nearest'),4,'nearest');
LB = imresize(imresize(I,0.25,'bilinear'),4,'bilinear');
LC = imresize(imresize(I,0.25,'bicubic'),4,'bicubic');
S = (1:256);
ll = cat(2,LN(S ,S),LB S ,S),LC(S ,S));
```



Geometry/Intensity Transformations

Reiner Lenz

2015

Mappings and Homogeneous Coordinates

Usually a point in the plane is describe by a coordinate vector $\binom{x}{y}$

Linear mappings preserving the origin are given by matrix: $\mathrm{M}\binom{x}{y}$

Shifting the origin with a vector Δ gives: $\binom{u}{v} = M\binom{x}{y} + \binom{\Delta_x}{\Delta_y}$

The mapping is given by the pair (M,Δ)

Inhomogeneous Coordinates

A coordinate vector $\begin{pmatrix} x \\ y \end{pmatrix}$

Defines the inhomogeneous coordinate vector $\begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

The inhomogeneous coordinate vector $\begin{pmatrix} u \\ v \\ w \end{pmatrix}$

Defines the homogeneous coordinate vector $\begin{pmatrix} u/w \\ v/w \end{pmatrix}$

Mapping

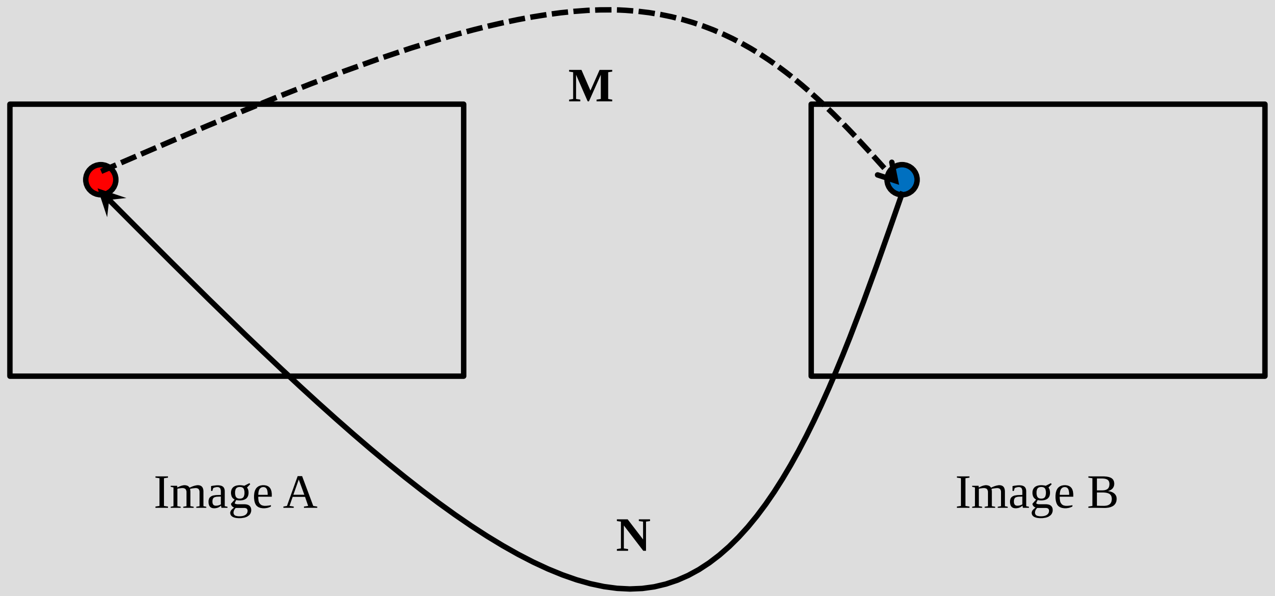
The mapping: $\begin{pmatrix} u \\ v \end{pmatrix} = M \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} \Delta_x \\ \Delta_y \end{pmatrix}$

can be written as: $\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} m_{11} & m_{12} & \Delta_x \\ m_{21} & m_{22} & \Delta_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \widehat{M} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}$

Collecting N points in matrices U and X of size (3xN)
gives the matrix equation

$$U = \widehat{M}\, X$$

Mapping direction



Comments:

Use M if you want to move a single point from A to B
Use N if you want to build a new image B from given image A

Registration

Previously we assumed that we know X and $\hat{M}$:

Collecting N points in matrices U and X of size (3xN)
gives the matrix equation

$$U = \hat{M} X$$

In many cases we know point pairs (common rows in U and X)
and we want to estimate the best solution $\hat{M}$.

This is registration and will be used in one of the labs

Solving Linear Equations

Very often one has to solve linear equations of the type

$$B = AX$$

or

$$B = XA$$

where A and B are known matrices and X is unknown.

These are two different cases since AB is in general different from BA

There are three cases:

- There is exactly one solution
- There is no exact solution
- There are many different solutions

If A was invertible then

$$A^{-1}B \text{ or } BA^{-1}$$

are solutions

Many solutions/Approximations

Equation

$$b = Ax$$

with vectors b and x and matrix A

rows(A) < columns(A)

more than 1 solution

rows(A) > columns(A)

NO exact solution

Inverting non-invertible matrices

Example: $A = (1,2)$
$b = 2;$
$x = (\xi,\eta)'$
$b = Ax = \xi+2\eta = 2$
possible solutions: $\eta = 1-\xi/2$

Matlab provides several methods to solve equations $b = Ax$

backslash:	`x = A\b`
pinv:	`iA = pinv(A);    x = iA*b`

pinv & backslash

Consider this data:

```
X = rand(10,2);
A = X(:,[1 2 2]);
C0 = [2;3;4];
b = A*C0; (corresponding to B = AX)
```

A is clearly rank deficient, since the second and third columns are replicated. Can we recover C0 using backslash or pinv?

```
C1 = A\b
Warning: Rank deficient, rank = 2, tol = 3.8e-15.
C1 =
          2
          7
          0
```

```
C2 = pinv(A)*b
C2 =
          2
        3.5
        3.5
```

The pinv solution is the solution with the minimum possible norm of all solutions.

```
norm(C1)
ans =
     7.2801

norm(C2)
ans =
     5.3385
```

Solutions

From

C0 = [2;3;4];

and

A = X(:,[1 2 2]);

we can guess that the general solution (v,w) should satisfy

v+w=7

therefore

w = 7-v

and the squared norm of

(v,7-v)

is

$$v^2 + (7 - v)^2 = 2v^2 - 14v + 49$$

Differentiate and get

$$4v - 14$$

Extrema at v = 0, v = 7 and v = 3.5

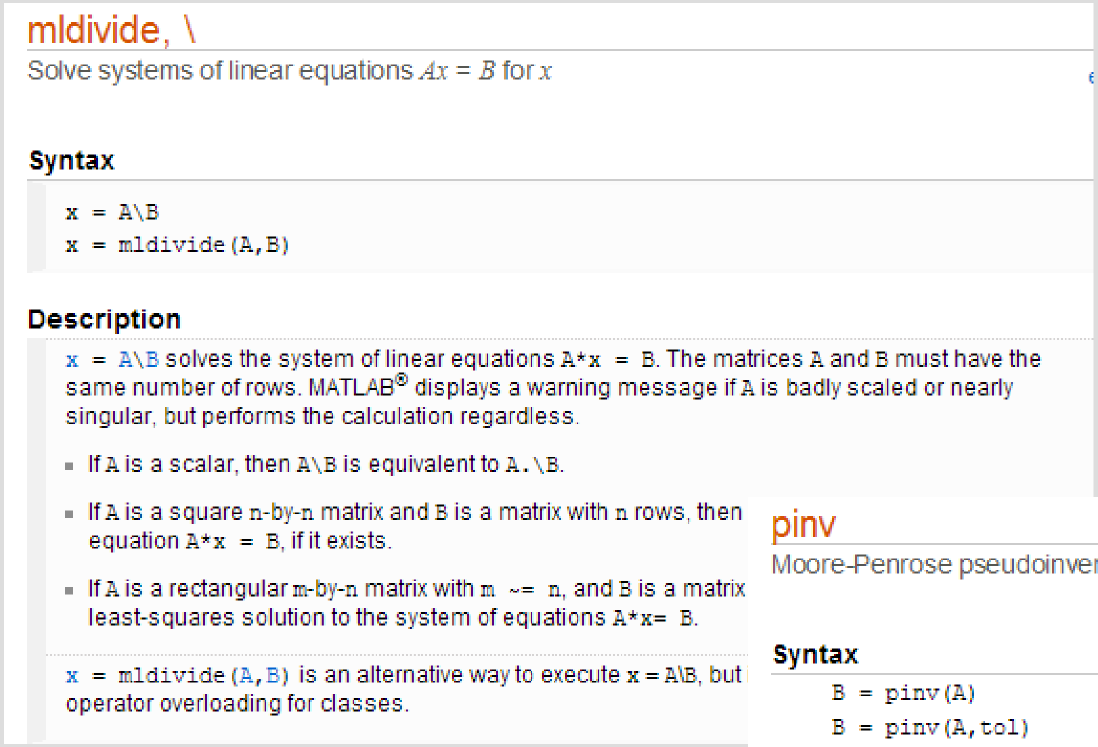
Solving Equations

Vector of unknowns: x
Known variables A, b
Equations: $Ax = b$
Solving for A if A is not-square
Find x such that $\|Ax-b\|$ is minimum

Leads to
$Ax \approx b \rightarrow A'Ax = A'b \rightarrow x = (A'A)^{-1}A'b$

x = A\b

mldivide,backslash,pinv



mldivide, \

Solve systems of linear equations $Ax = B$ for x

Syntax

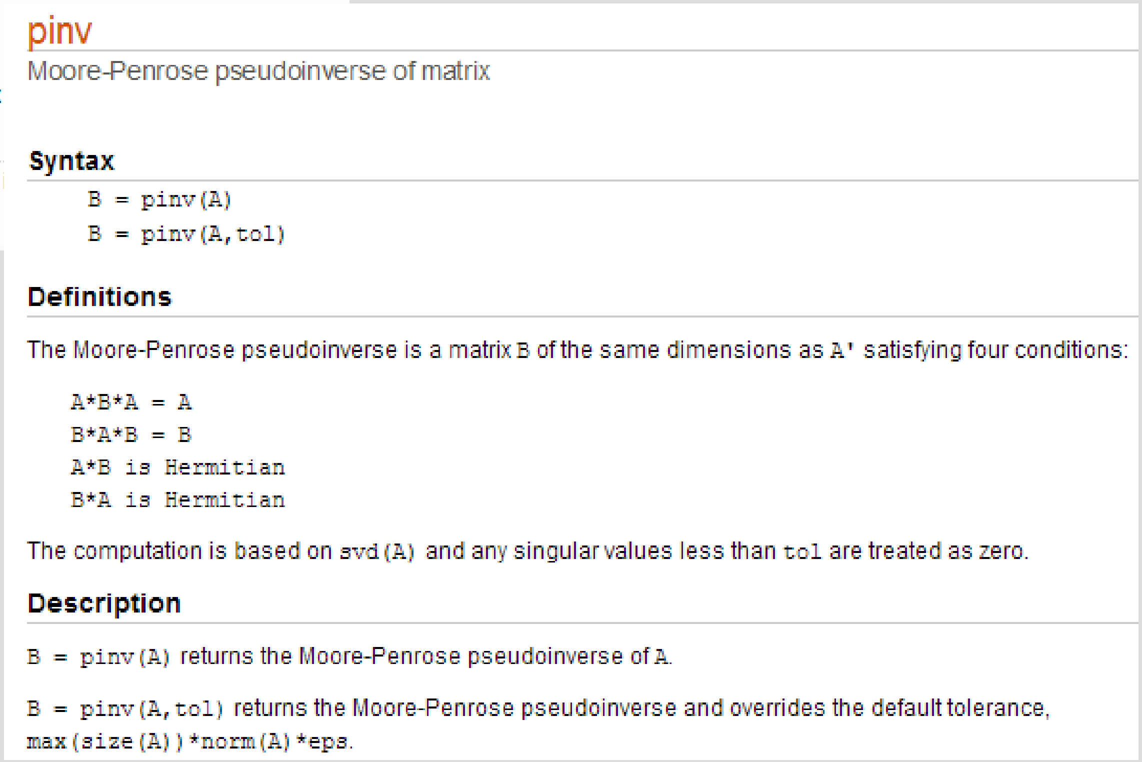
```
x = A\B
x = mldivide(A,B)
```

Description

`x = A\B` solves the system of linear equations `A*x = B`. The matrices `A` and `B` must have the same number of rows. MATLAB® displays a warning message if `A` is badly scaled or nearly singular, but performs the calculation regardless.

- If `A` is a scalar, then `A\B` is equivalent to `A.\B`.
- If `A` is a square `n`-by-`n` matrix and `B` is a matrix with `n` rows, then
  equation `A*x = B`, if it exists.
- If `A` is a rectangular `m`-by-`n` matrix with `m ~= n`, and `B` is a matrix
  least-squares solution to the system of equations `A*x= B`.

`x = mldivide(A,B)` is an alternative way to execute x = A\B, but
operator overloading for classes.



pinv

Moore-Penrose pseudoinverse of matrix

Syntax

```
B = pinv(A)
B = pinv(A,tol)
```

Definitions

The Moore-Penrose pseudoinverse is a matrix `B` of the same dimensions as `A'` satisfying four conditions:

```
A*B*A = A
B*A*B = B
A*B is Hermitian
B*A is Hermitian
```

The computation is based on `svd(A)` and any singular values less than `tol` are treated as zero.

Description

`B = pinv(A)` returns the Moore-Penrose pseudoinverse of `A`.

`B = pinv(A,tol)` returns the Moore-Penrose pseudoinverse and overrides the default tolerance, `max(size(A))*norm(A)*eps`.

Approximation

Equation: b = Ax (p,R given)
backslash: bs = A\b;
pinv: iA = pinv(A); iAb = iA*b;

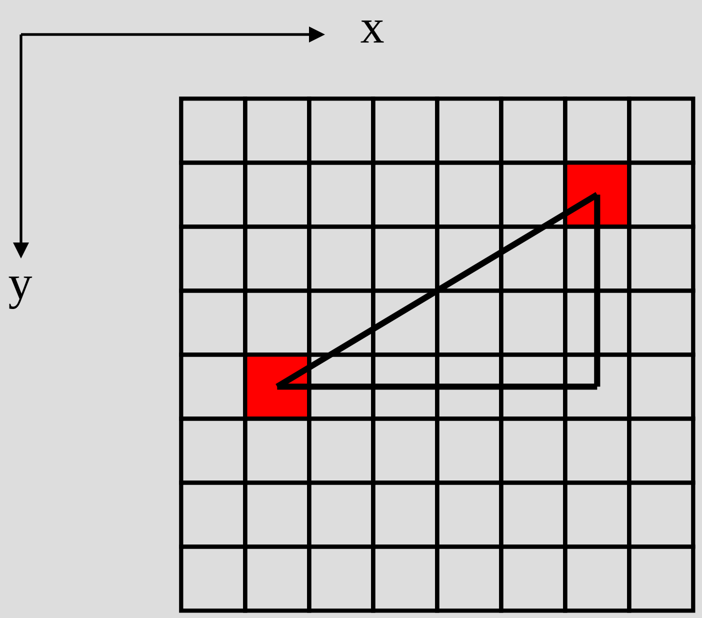
```
%%
% Generate equation b = AX
npts = 100;
A = [1,2]
x = rand(2,npts);
b = A*x;

%% solutions with pinv and \
bs = A\b;
iA = pinv(A);
iAb = iA*b;

%% between solution-ground truth
diffbs = (x-bs);
diffIab = (x-iAb);
```

```
%%
bsnorm2 = diffbs(1,:).^2+diffbs(2,:).^2;
Iabnorm2 = diffIab(1,:).^2+diffIab(2,:).^2;

%% mean and plot differences
disp([mean(bsnorm2),mean(Iabnorm2)])
figure
plot((1:npts),bsnorm2,'r.',(1:npts),Iabnorm2,'ko')
```

..\Matlab\SolveEqs.m

..\Matlab\html\SolveEqs.html

Distance measures (metric) in digital images



- Euclidean distance:

$$D_E = (\Delta x^2 + \Delta y^2)^{1/2}$$

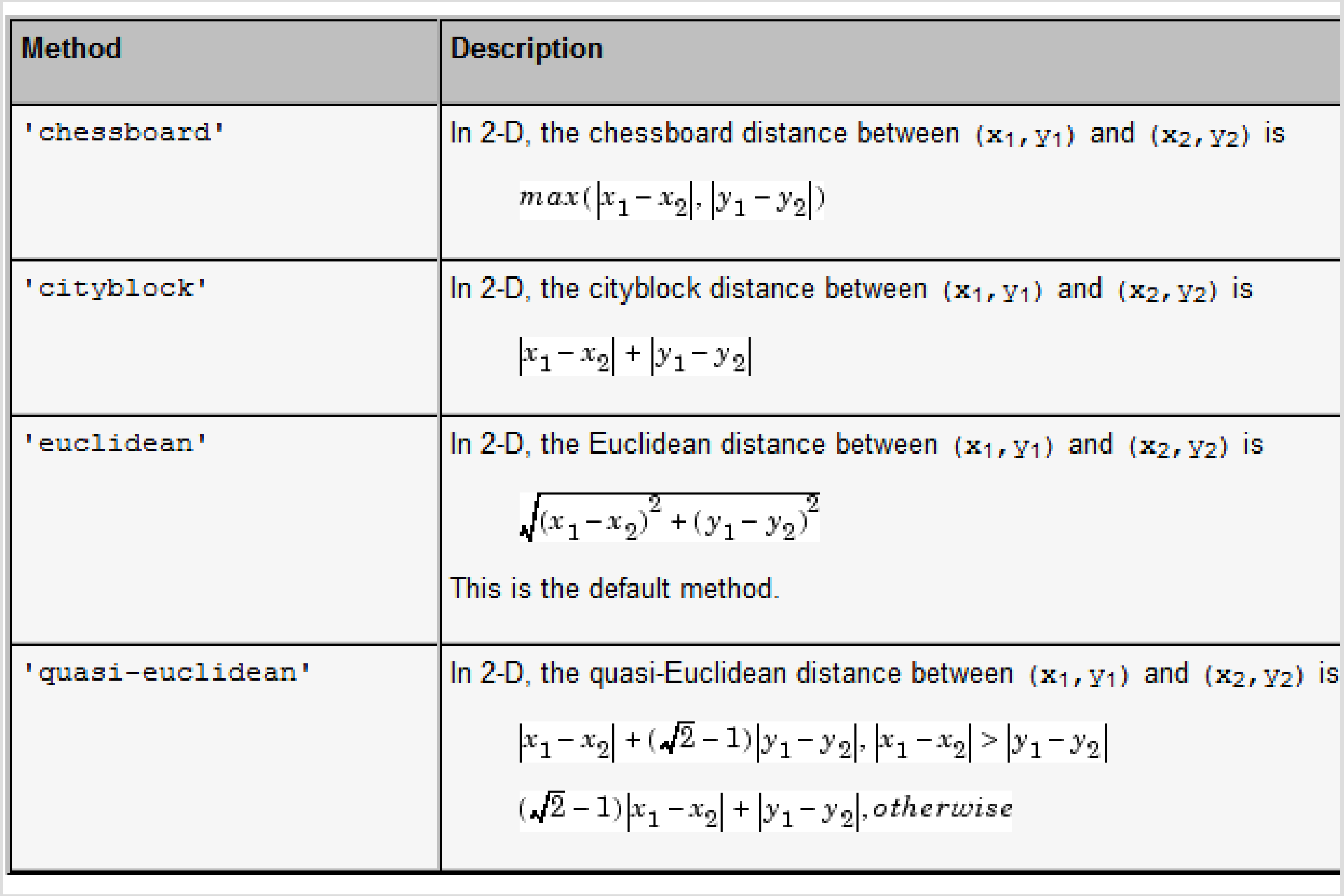
- Cityblock distance:

$$D_4 = |\Delta x| + |\Delta y|$$

- Chessboard distance:

$$D_8 = \max(|\Delta x|, |\Delta y|)$$

Distance metrics in MATLAB:



Method	Description
`'chessboard'`	In 2-D, the chessboard distance between `(x1,y1)` and `(x2,y2)` is $max(\|x_1-x_2\|, \|y_1-y_2\|)$
`'cityblock'`	In 2-D, the cityblock distance between `(x1,y1)` and `(x2,y2)` is $\|x_1-x_2\| + \|y_1-y_2\|$
`'euclidean'`	In 2-D, the Euclidean distance between `(x1,y1)` and `(x2,y2)` is $\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}$ This is the default method.
`'quasi-euclidean'`	In 2-D, the quasi-Euclidean distance between `(x1,y1)` and `(x2,y2)` is $\|x_1-x_2\| + (\sqrt{2}-1)\|y_1-y_2\|, \|x_1-x_2\| > \|y_1-y_2\|$ $(\sqrt{2}-1)\|x_1-x_2\| + \|y_1-y_2\|, otherwise$

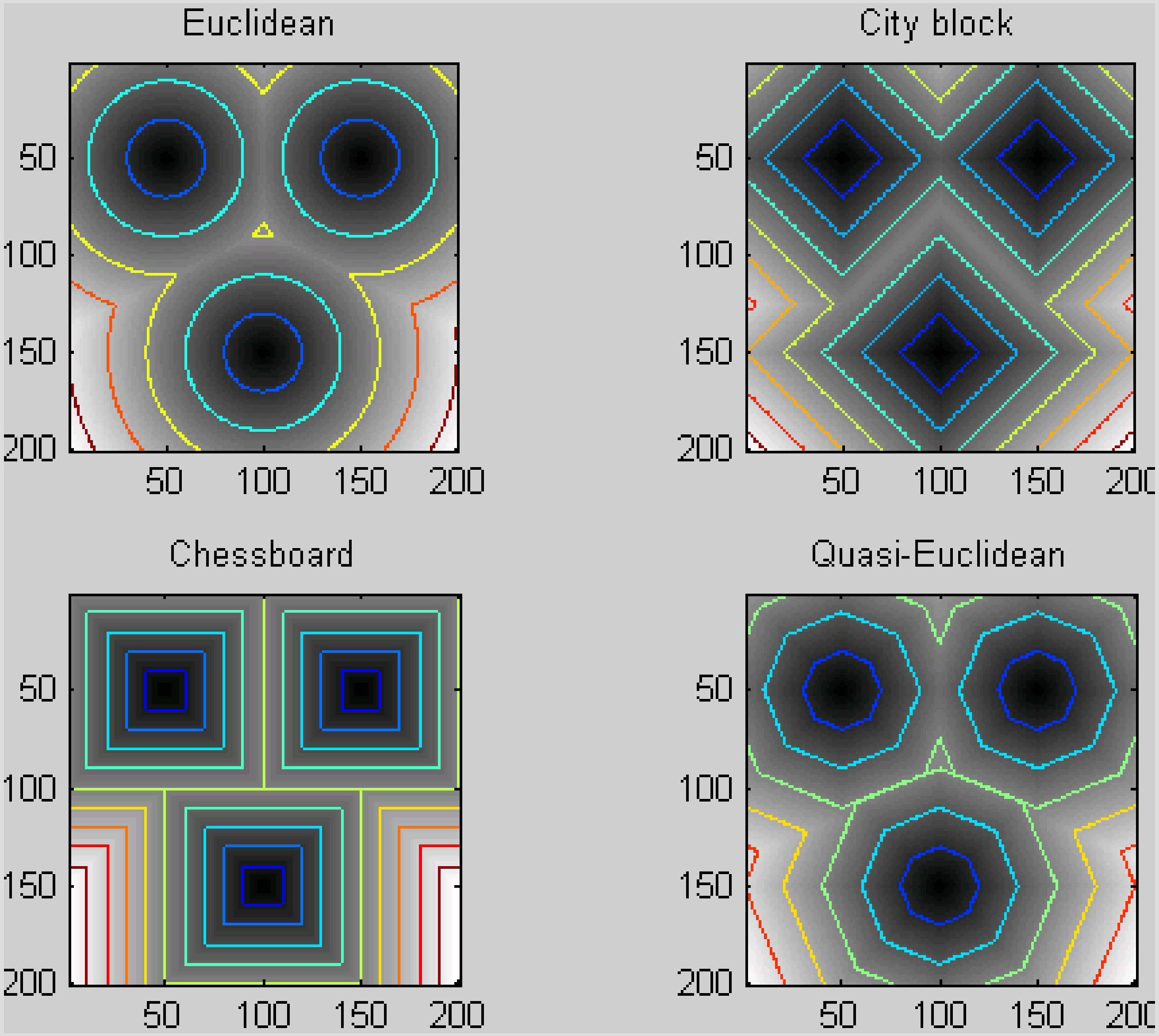
Distance Map

Binary image => 2 pixel values (p_1 and p_2)
A *distance map* indicates the distance from every pixel with value p_1 to the nearest pixel with value p_2

Ex. Cityblock,
$p_2 = 0$

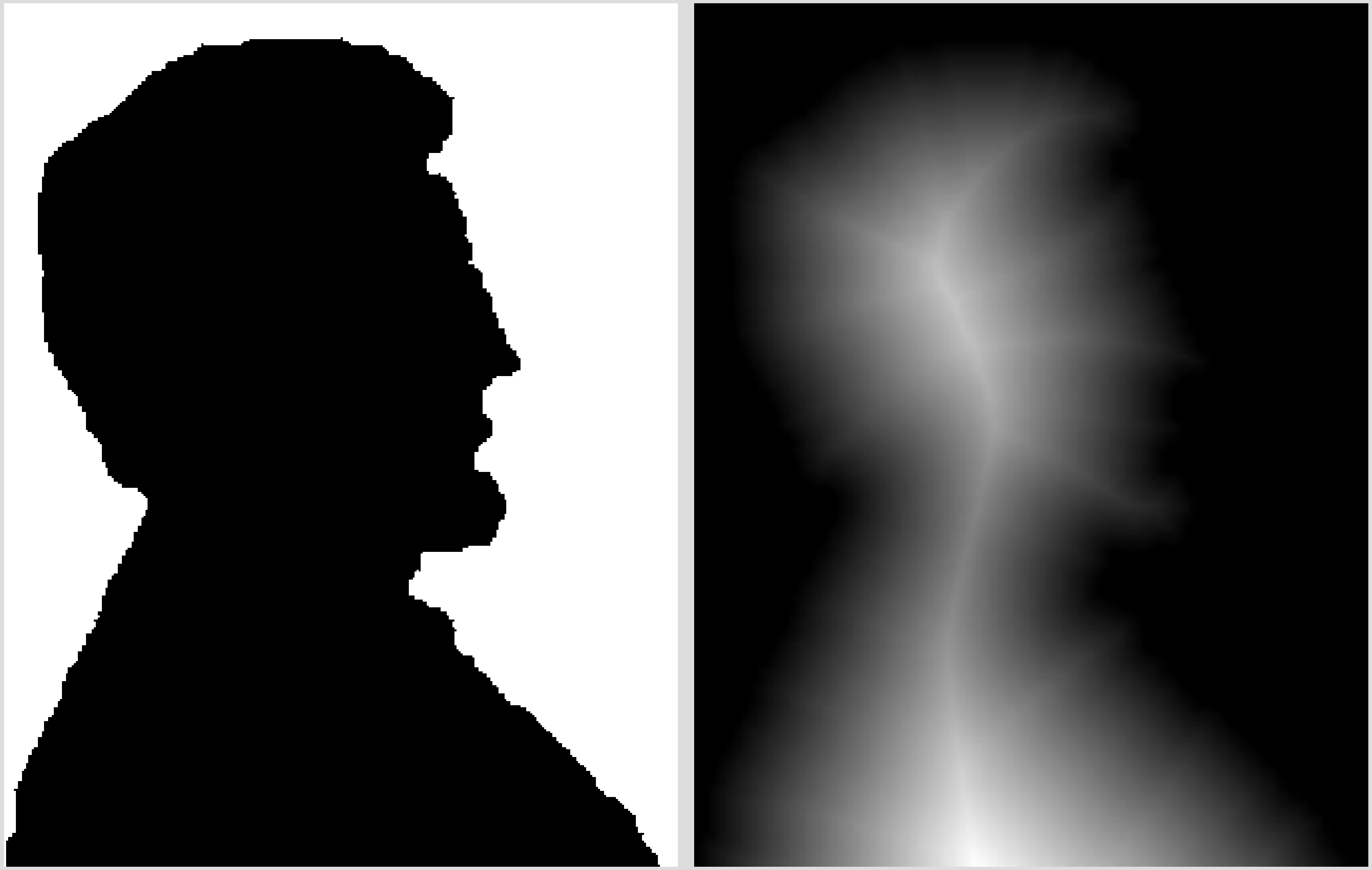
2	1	2	3
1	0	1	2
2	1	2	3
3	2	3	4

Distance maps:



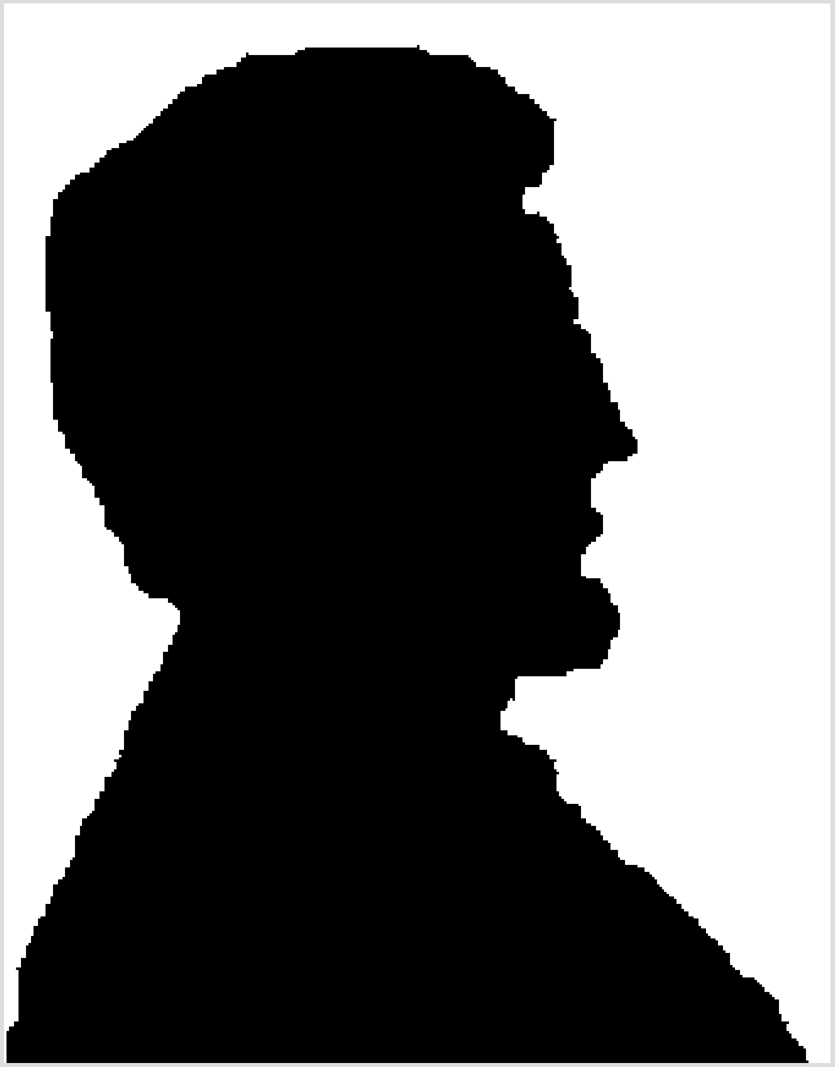
Minimum distance of a point to one of three given points

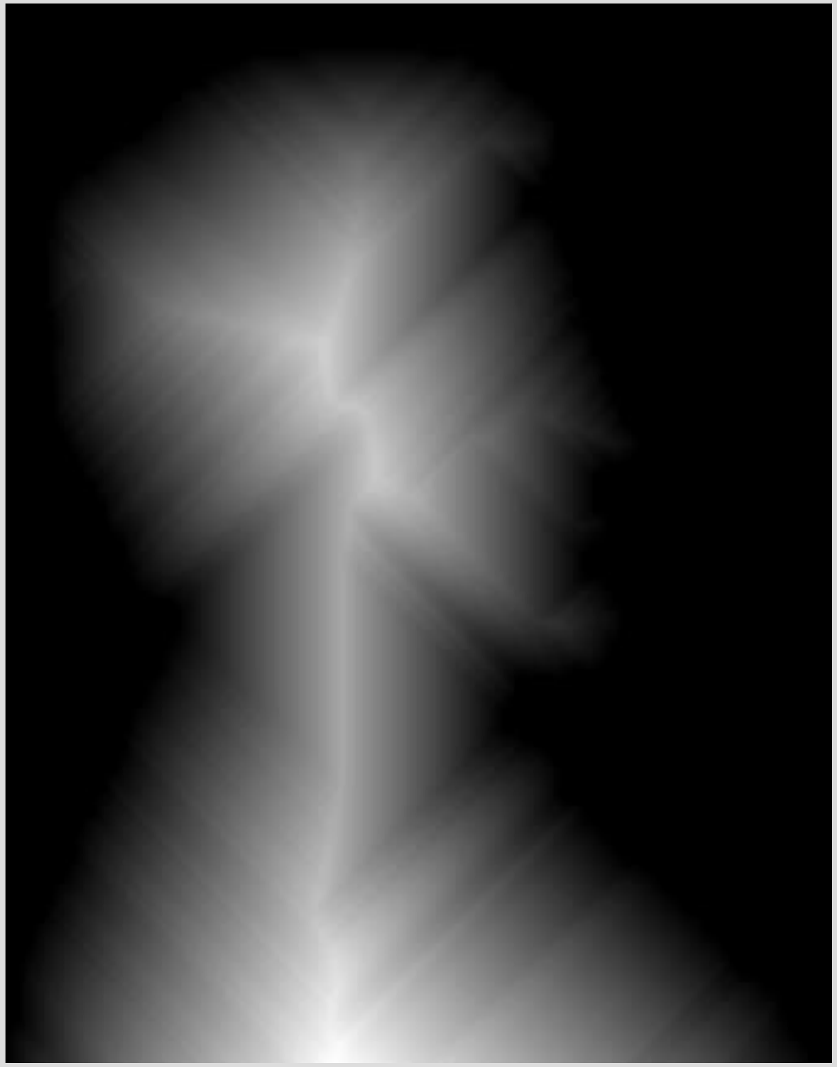
Euclidian



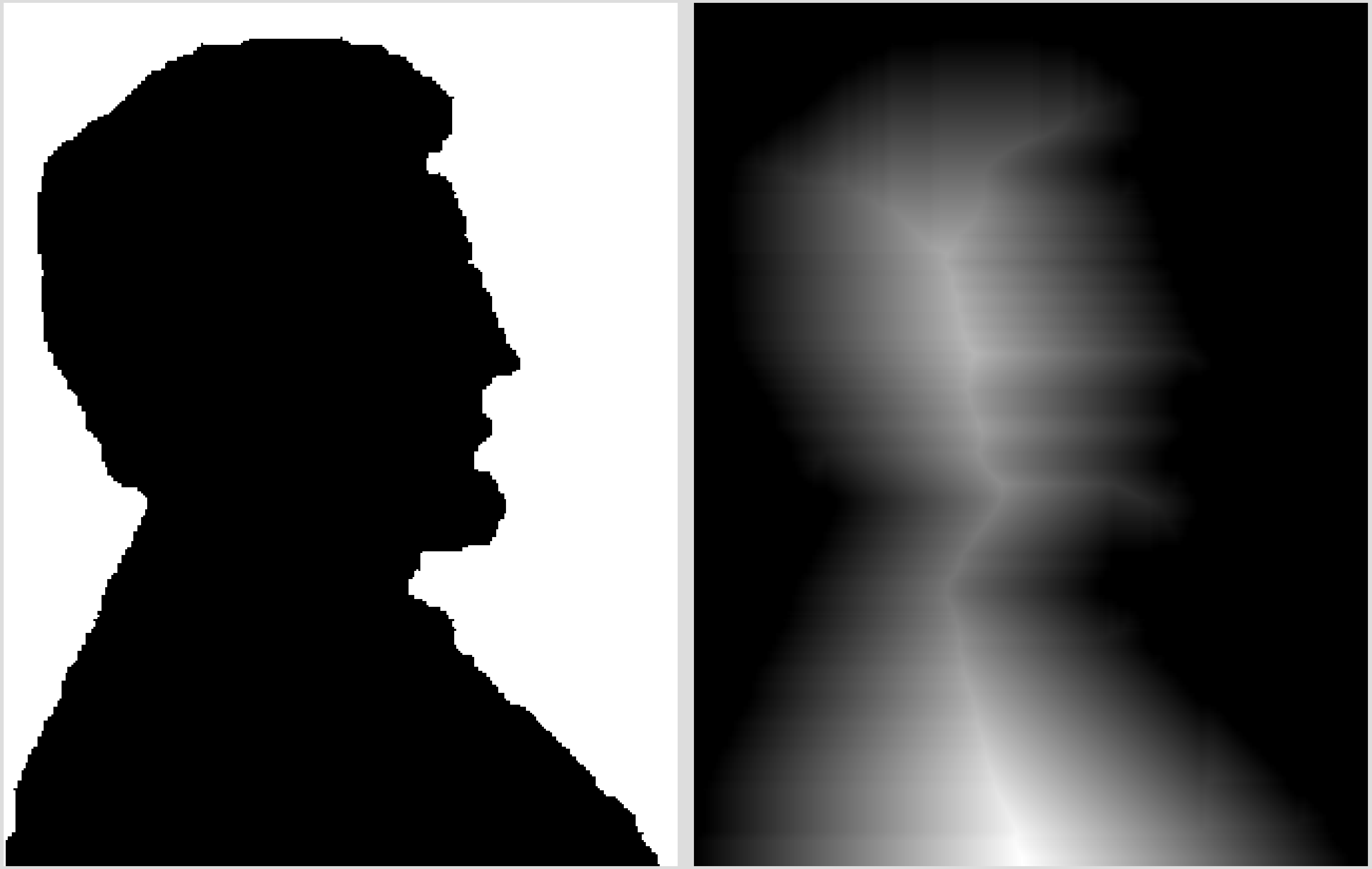
For every point compute the distance to the nearest boundary point

Chessboard

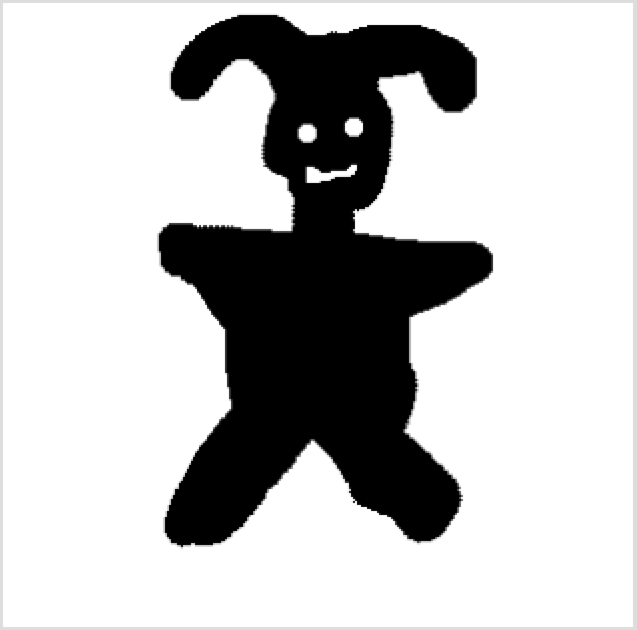


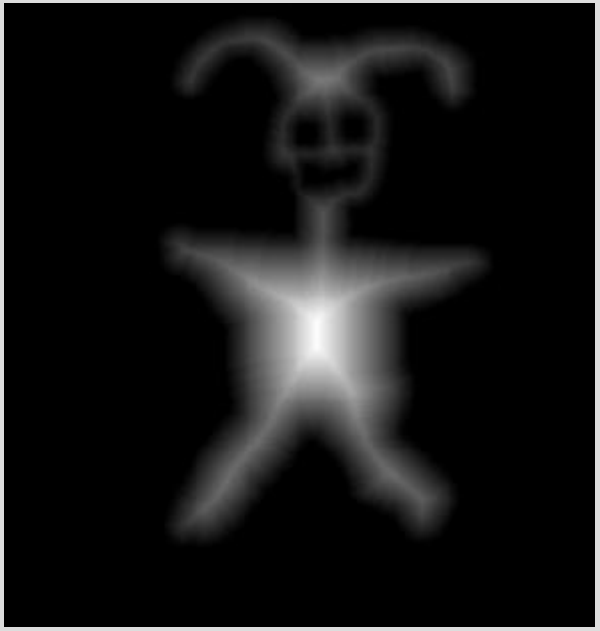


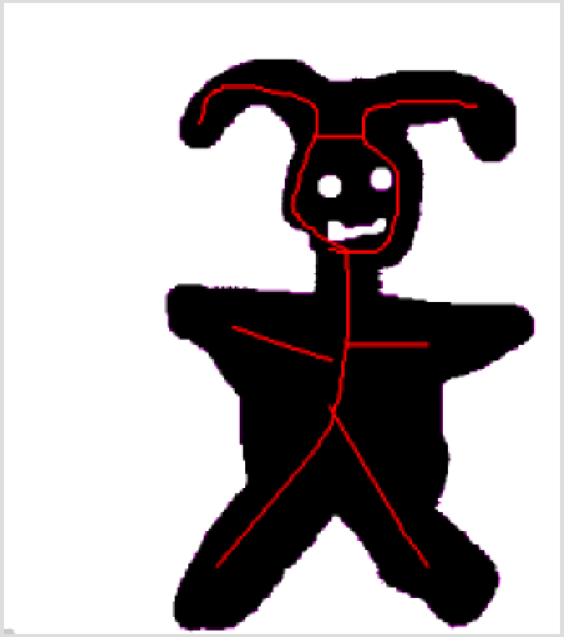
Cityblock



Distance Transform (DT)


Binary image


DM


Medial Axis Transf.

(local maxima in DM)

Distance transform (cityblock)

- *Initialization:* set the background pixels to N (N>largest possible distance) and the object pixels to 0.
- For every pixel d(x,y):

$$d_n\,(x,y) = min\begin{bmatrix} d_{n-1}(x,y), \\ d_{n-1}(x+1,y)+1, \\ d_{n-1}(x-1,y)+1, \\ d_{n-1}(x,y+1)+1, \\ d_{n-1}(x,y-1)+1 \end{bmatrix}$$

$$n = 1,2,\ldots\ldots,m$$

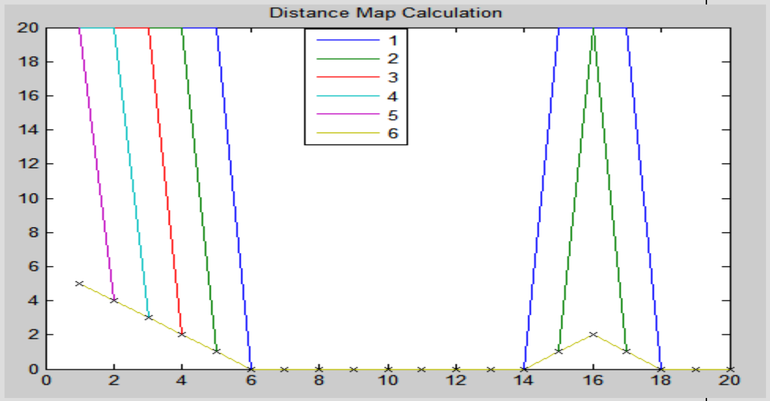
Repeat until no change ($d_m = d_{m-1}$)

Matlab 1-D Example

..\Matlab\html\SimpleDistmap.html

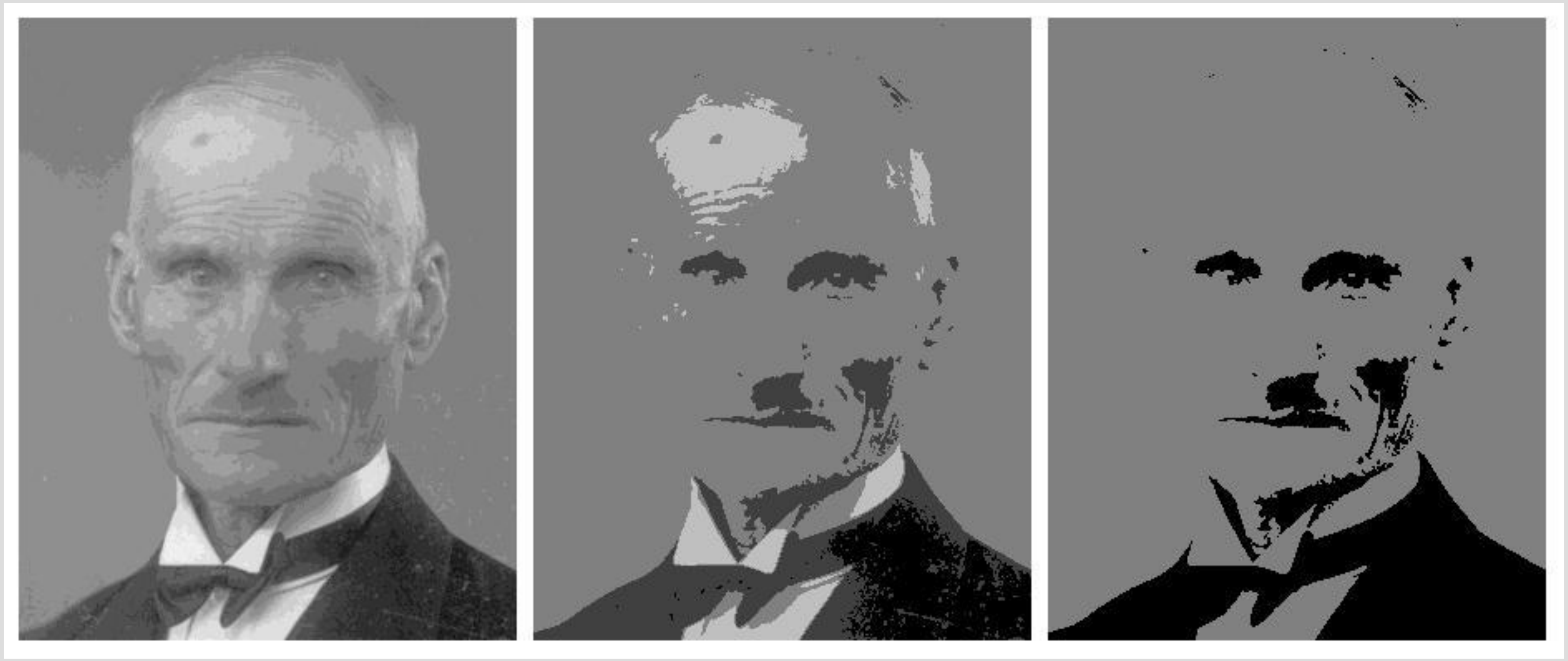
```
%% Demo simple distance map
%% Init

clear all
close all

nopts = 20; %length of vector

%% Create object and init dm

vector = zeros(1,nopts);
vector([(1:5),(15:17)])=nopts;
distmap = zeros(nopts,nopts);
distmap(1,:) = vector;
```

```
%% Distance map calculation
for k = 1:nopts-1
    for l = 2:nopts-1
        distmap(k+1,l) = …
                min([distmap(k,l-1)+1,distmap(k,l),distmap(k,l+1)+1]);
    end
    distmap(k+1,1) = min([distmap(k,1),distmap(k,2)+1]);
    distmap(k+1,end) = min([distmap(k,end-1)+1,distmap(k,end)]);
end

%% Plot result

figure

plot(distmap(1:6,:)')
legend('1','2','3','4','5','6')
title('Distance Map Calculation')
hold on
plot(distmap(end,:)','kx')
```



Photometric resolution

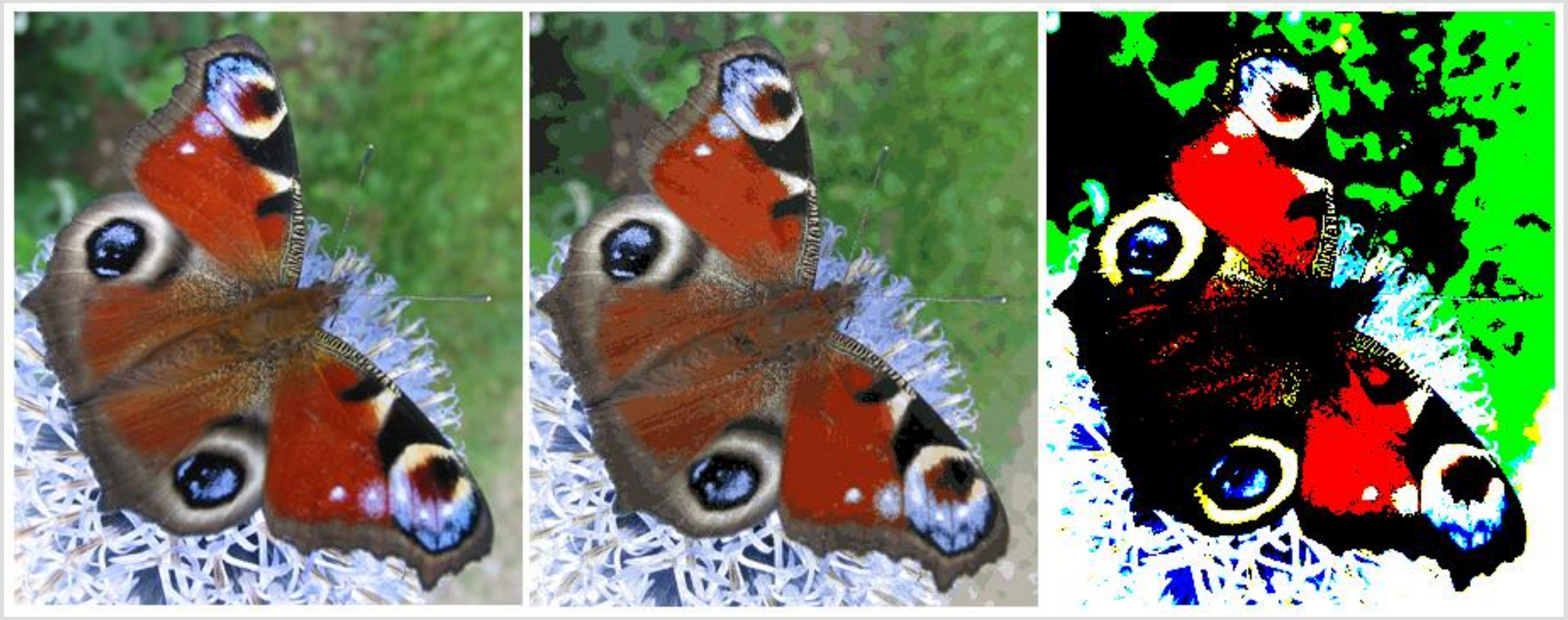
The number of gray values (tone, intensity values) is given by 2^n where n is the number of bits/pixel. Common resolution for grayscale images is 8 bits/pixel (1 byte), but 16 (or more) bits/pixel are used for raw-images and HDR, in radiology etc.



5 bits 2 bits 1 bit

Photometric resolution

For color images it is common practice to use 3 channels (RGB) with 1 byte per channel, i.e. 3 bytes per pixel.



24 bits 6 bits 3 bits

Tone Mapping

Typical devices can display 2 (ink/no-ink) or 256 intensity levels

*“**Tone mapping** is a technique used in image processing and computer graphics to map one set of colors to another in order to approximate the appearance of high dynamic range images in a medium that has a more limited dynamic range.*

…

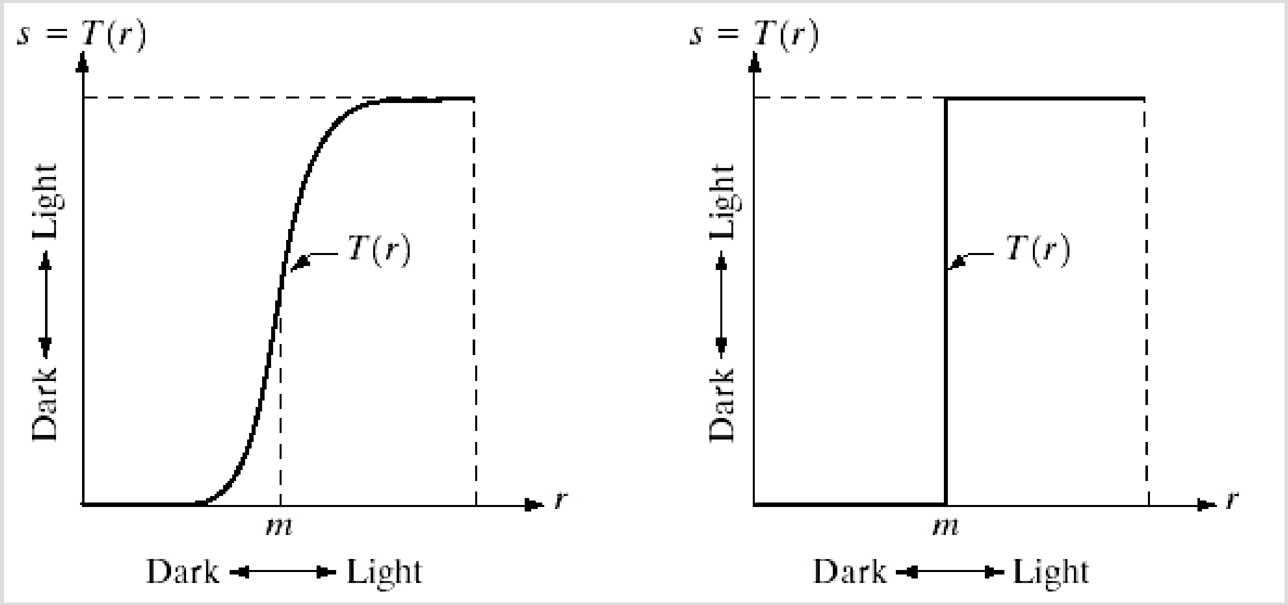
Tone mapping addresses the problem of strong contrast reduction from the scene radiance to the displayable range while preserving the image details and color appearance important to appreciate the original scene content.”

Wikipedia

More later on HDR

Intensity transformations

The pixel values (intensities) are transformed by a transformation function.



Intensity transformations

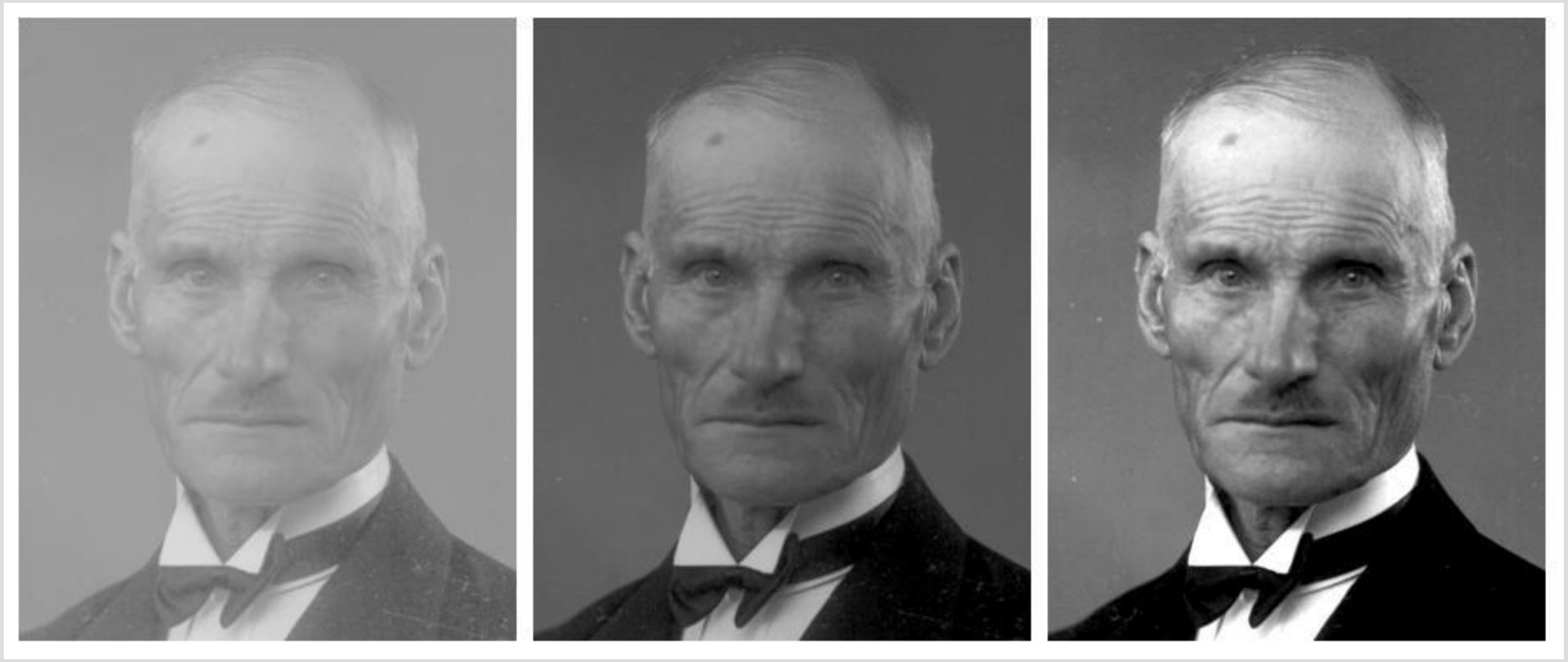
The following function implements a linear intensity transformation.

```
function Q = LToneMap(P,T,G)
%LToneMap Tone map with a linear function
%   P = input image
%   T = offset
%   G = gain
%
% Simple version assumes P is in 0,1

if (T < 0) | (T>1)
    error('LTM','Wrong T')
else
    Q = 0.5 * G*(P-T);
    Q(Q<0)=0;
    Q(Q>1)=1;
end
return
end
```

..\Matlab\LToneMap.m

Intensity transformations

Three examples:



T=0.4, G=0.75 T=0.7, G=1.0 T=0.6, G=2.0