Image Processing and Data Visualization with MATLAB

Image Processing

(based on MATLAB Help)

Hansrudi Noser

June 28-29, 2010

UZH, Multimedia and Robotics Summer School

Product overview

- The image processing toolbox is a collection of functions extending MATLAB with special image processing operations concerning
  - Spatial image transformations
  - Morphological operations
  - Neighborhood and block operations
  - Linear filtering and filter design
  - Transforms
  - Image analysis and enhancement
  - Image registration
  - Deblurring
  - Region of interest operations

Contents

- Introduction
- Reading and Writing Image data
- Spatial Transforms
- Image Registration
- Image Filters
- Transforms
- Morphological Operations

Digital Images in MATLAB

- The basic data structure in MATLAB is the array which is the container for the most common discrete (digital) image types such as
  - Gray value images
  - True color images
  - Movies
  - …
- Therefore the full power of MATLAB is available for digital image processing applications

Image coordinate systems: Pixel Coordinates

- In 2D digital images the location of a pixel (picture element) in the pixel coordinate system is given by a pair of integer indices ranging from 1 to the length of the row or column.
- Pixel coordinates: (row, column)

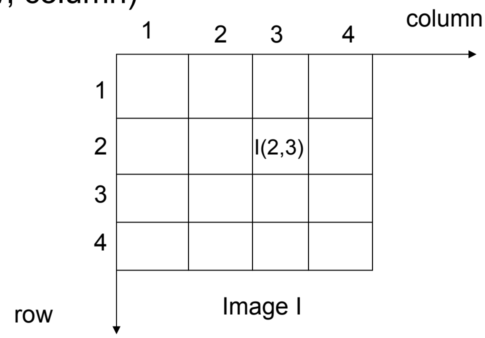
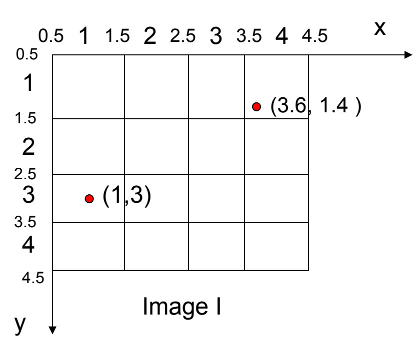


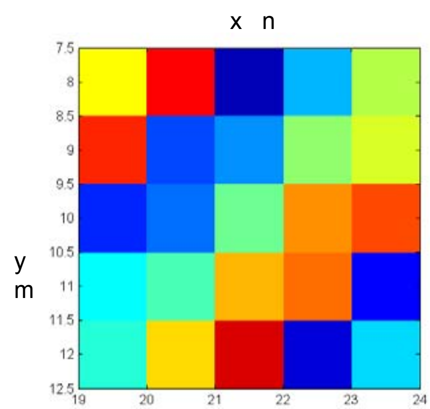
Image coordinate systems: Spatial Coordinates

- A pixel is a square patch with continuous coordinates
- The center point of a pixel corresponds to the pixel coordinate system
- Spatial coordinates: (x, y)
- Attention: In spatial coordinate systems the horizontal and vertical order is reversed with respect to the pixel coordinate system.
  - (row, column) / (x, y)
  - (3, 1) / (1, 3)



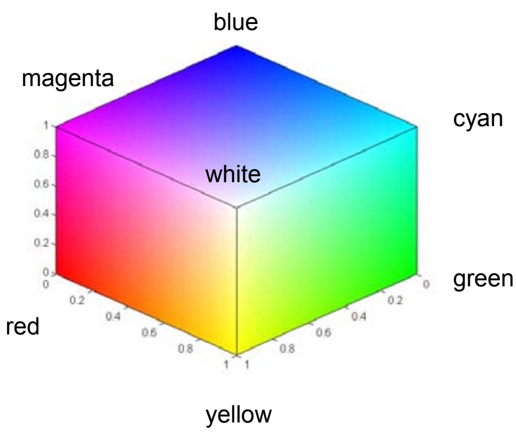
Non-Default Spatial Coordinates

- Non-default spatial coordinate systems can be defined by setting image properties
  - m x n image
  - XDATA: [x1 x2]
  - YDATA: [y1 y2]
- Pixel width: (x2 – x1) / (n -1)
- Pixel height: (y2 – y1) / (m – 1)



A = magic(5);
x = [19.5 23.5];
y = [8.0 12.0];
image(A,'XData',x,'YData',y), axis image, colormap(jet(25))

RGB Colors



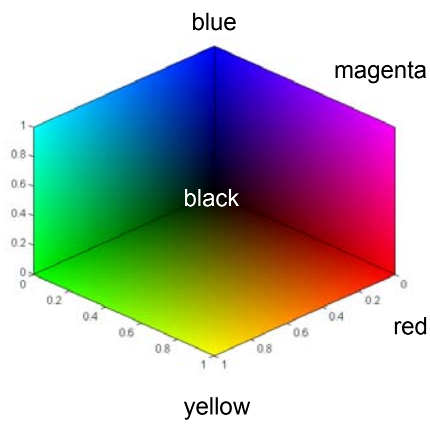
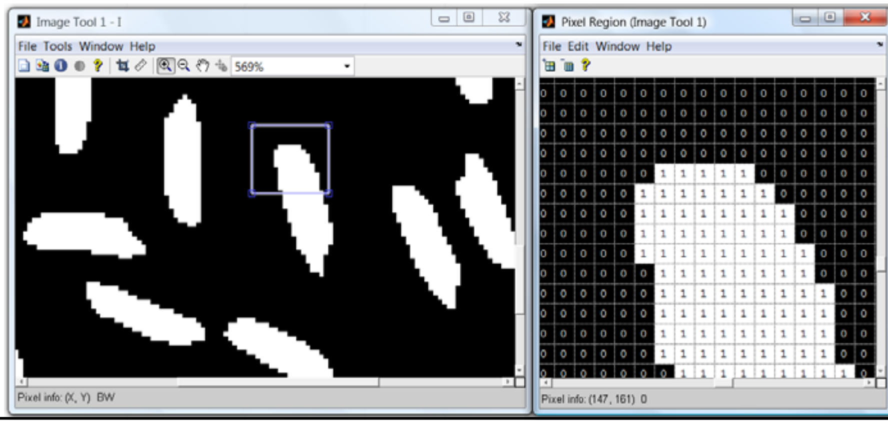


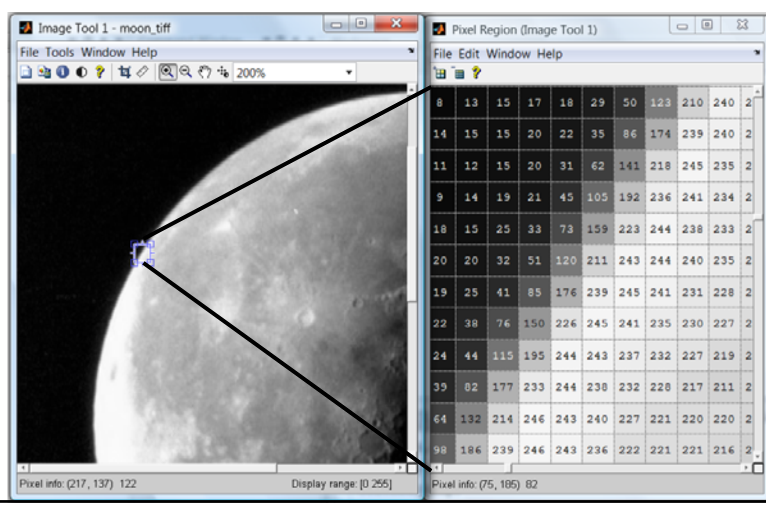
Image Types: Binary Images

- The value of a pixel has only the values 0 or 1



Grayscale Images

- Pixel values define gray levels

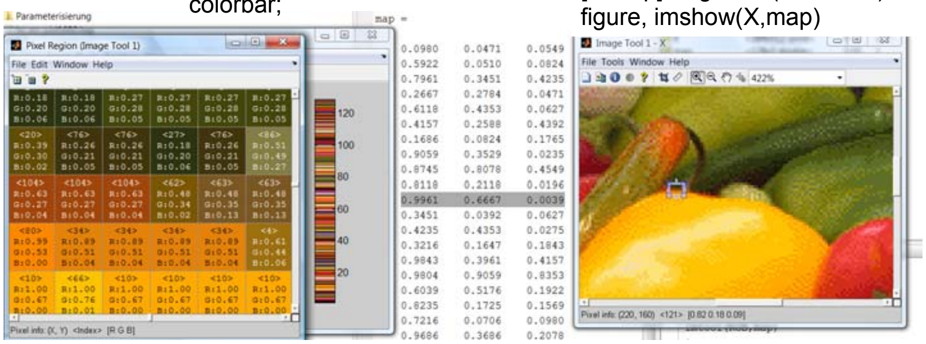


Indexed Images

- In indexed images pixel values are indices to colormap entries

colormap(map);
colorbar;

RGB = imread('peppers.png');
[X,map] = rgb2ind(RGB,128);
figure, imshow(X,map)



Truecolor Images

- In a truecolor image each pixel color is defined by its red-green-blue component, a triple of 3 values.
- The image is given by an m x n x 3 array.

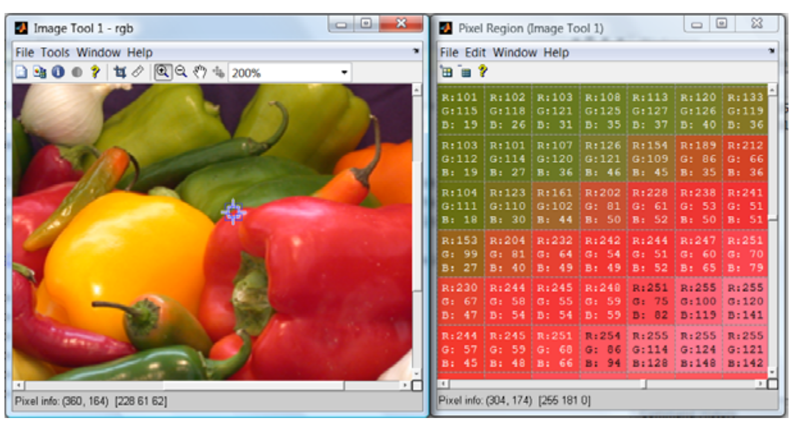


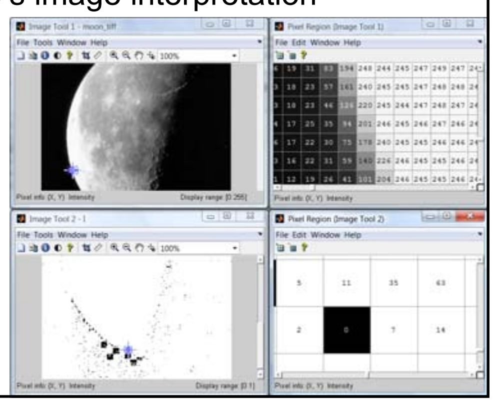
Image Classes

- Image classes or storage classes. Pixel values can be of the following types:
  - logical: 0,1
  - uint8: [0..255]
  - unit16: [0..65536]
  - int16: [-32768..32767]
  - single: [0.0 … 1.0]
  - double: [0.0 … 1.0]
- Binary
  - logical
- Indexed
  - logical, unit8, uint16: [0..p-1]
  - single, double:[1..p]
- Grayscale
  - uint8, unit16, int16, single, double
- Truecolor
  - uint8, uint16, single, double

Converting Between Image Classes

- When converting between image classes we need to rescale and/or offset the data
- Instead of type casting use specialized functions which take into account MATLAB's image interpretation
  - Im2uint8, im2unit16,
  - im2int16, im2single,
  - im2double

```
moon_tiff = imread('moon.tif');
imtool(moon_tiff)
I=single(moon_tiff)
imtool(I)
```



Converting between Image Types

- Sometimes type conversions are necessary
  - How to filter the intensity values of an indexed truecolor Image?
    - Convert it to truecolor format, filter it, and convert it back to indexed format
  - For publications often image type conversions are necessary
  - For creating animated GIFs you need indexed images

Image Type Conversions

- Attention: The image type conversions can modify your image
  - Example: Truecolor to grayscale
- Conversion is possible by standard MATLAB commands
  - Example: Grayscale image I to truecolor
    - RGB = cat(3,I,I,I);
- Better: conversion by specialized image toolbox functions

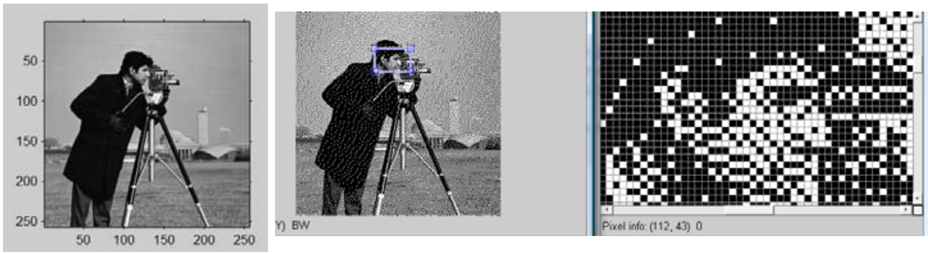
Conversion Functions

- dither
  - grayscale to binary or truecoloer to indexed
- gray2ind
- grayslice
  - grayscale to indexed by multilevel thresholding
- im2bw
  - grayscale, indexed, truecolor to binary by luminance threshold
- ind2gray
- ind2rgb
- rgb2gray
- rgb2ind

Dithering

- Increases the apparent number of colors
- Changes the colors of pixels in a neighborhood so that the average color in each neighborhood approximates the original RGB color
- Increase of color resolution decreases spatial resolution
- Dithering is used by printers

Dithering: grayscale to binary

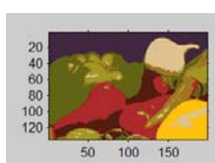


I = imread('cameraman.tif');
BW = dither(I);
imshow(I), figure, imshow(BW), figure imtool(BW)

Dithering: Color reduction

rgb=imread('onion.png');
imshow(rgb);

[X_no_dither,map]= rgb2ind(rgb,8,'nodither');
imshow(X_no_dither,map);



[X_dither,map]=rgb2ind(rgb,8,'dither');
imshow(X_dither,map);

1. Read an rgb image
2. Convert it to indexed with only 8 colors without dithering
3. Convert it to indexed with only 8 colors with dithering

Image Sequences

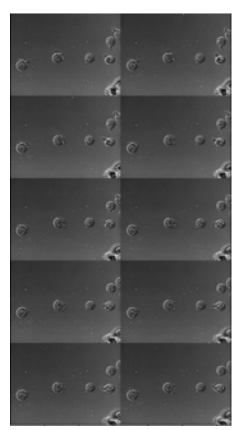
imabsdiff	m-by-n-by-p or m-by-n-by-3-by-p	Image sequences must be the same size.
imadd	m-by-n-by-p or m-by-n-by-3-by-p	Image sequences must be the same size. Cannot add scalar to image sequence.
imbothat	m-by-n-by-p only	SE argument must be 2-D.
imclose	m-by-n-by-p only	SE argument must be 2-D.
imdilate	m-by-n-by-p only	SE argument must be 2-D.
imdivide	m-by-n-by-p or m-by-n-by-3-by-p	Image sequences must be the same size.
imerode	m-by-n-by-p only	SE argument must be 2-D.

…

- Collection of images
  - Images related by time:
    - Frames of movies
  - Images related by spatial location:
    - MRI (magnetic resonance imaging)
    - CT (computed tomography)
    - Also called: image stacks, image sequences, image slices
- Image sequences can be stored in multidimensional arrays
  - m x n x p array for p two dimensional (m x n) grayscale images
  - m x n x 3 x p array for p truecolor images
- Many toolbox functions accept multi-dimensional arrays

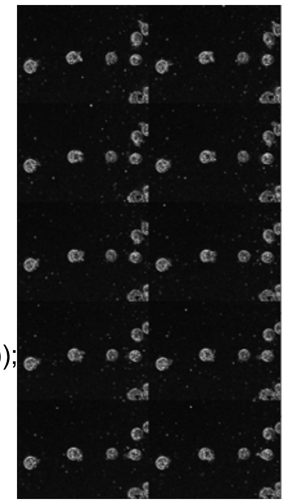
Example: Filtering of Image Sequence (1)

```
% Create an array of filenames that make up the image
sequence

fileFolder = fullfile(matlabroot,'toolbox','images','imdemos');
dirOutput = dir(fullfile(fileFolder,'AT3_1m4_*.tif'));
fileNames = {dirOutput.name}';
numFrames = numel(fileNames);
I = imread(fileNames{1});

% Preallocate the array

sequence = zeros([size(I) numFrames],class(I));
sequence(:,:,1) = I;

% Create image sequence array

for p = 2:numFrames
        sequence(:,:,p) = imread(fileNames{p});
end
```



Example: Filtering of Image Sequence (2)

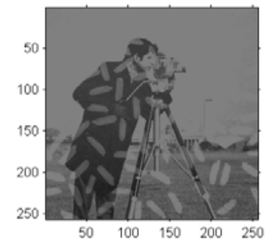
```
% Process sequence  (imSequenceProcessing.m)

sequenceNew = stdfilt(sequence,ones(3));

% View results

figure;
for k = 1:numFrames
        imshow(sequence(:,:,k));
        title(sprintf('Original Image # %d',k));
        pause(1);
        imshow(sequenceNew(:,:,k),[]);
        title(sprintf('Processed Image # %d',k));
        pause(1);
end
```



Multi-Frame Image Arrays

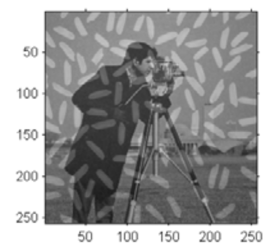
- immovie, montage use multi-frame arrays
- p m x n frames are stored in the following arrays
  - m x n x 1 x p : binary, grayscale, indexed
  - m x n x 3 x p : true color
- Creation of multi-frame arrays with cat
  - A=cat(4,A1,A2,A3)
  - Squeeze for removing the singleton dimension

Image Arithmetic



- Image Arithmetic is possible
  - Addition, subtraction, multiplication, …
- Attention: overflow is possible
  - Values exceeding the range of a type are saturated to that range

```
I = imread('rice.png');
I2 = imread('cameraman.tif');
K = imdivide(imadd(I,I2), 2); % bad

K = imlincomb(.5,I,.5,I2); % good
```



Contents

- Introduction
- Reading and Writing Image data
- Spatial Transforms
- Image Registration
- Image Filters
- Transforms
- Morphological Operations

Reading and Writing Images

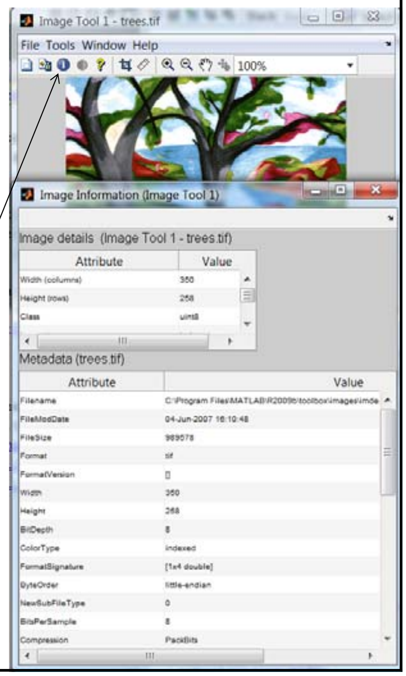
- Images can be stored in many file formats on storage devices
- MATLAB supports many standard graphics and medical file formats
  - Getting information (imfinfo)
  - Reading files (imread)
  - Writing files (imwrite)
- imformats lists image file format supported by imfinfo, imread, imwrite
  - bmp, gif, jpg, tif, …

File Formats

- Common image file formats are
  - Microsoft® Windows® Bitmap (BMP)
  - Graphics Interchange Format (GIF)
  - Joint Photographic Experts Group (JPEG)
  - Portable Network Graphics (PNG)
  - Tagged Image File Format (TIFF) formats
- The image data can be
  - Raw
  - Lossless compressed (RLE)
  - Lossy compressed

Getting Image Information

- Imfinfo: information according to file format but always
  - Name of file
  - File format
  - Version number
  - Modification data
  - File size in bytes
  - Image width and height
  - Number of bits per pixel
  - Image type (rgb, grayscale, indexed)
- Image Information Tool imtool('trees.tif')



Reading Images

- Use imread for importing (loading) images
  - Many file formats are supported
  - Examples:
    - RGB = imread('football.jpg');
    - [X,map] = imread('trees.tif');
- Image files can also contain multiple images (tif, gif, dcm)
  - Imread only reads single images, but it can be specified which one
    - Example: imread('mri.tif',frame);

Reading of multi-frame images

- Example of reading 27 images in a TIFF file

```
% get number of images in the file
finfo = imfinfo('mri.tif');
[nImages, m] = size(finfo);

% preallocate 4-D array
mri = zeros([128 128 1 nImages],'uint8');

% read the images
for frame=1:nImages
        [mri(:,:,:,frame),map] = imread('mri.tif',frame);
end
```

Writing Image Data to a File

- Images can be exported with imwrite
  - Format defined by filename extension or an explicite argument
  - There exist format-specific parameters
- Write examples
  - imwrite(X,map,'clown.bmp')
  - imwrite(I,'clown.png','BitDepth',4);
  - imwrite(A, 'myfile.jpg', 'Quality', 100);
- imwrite uses internal rules to determine the storage class used in the output image

Converting Between Graphics File Formats

- Conversion with imread and imwrite
- Example of tif – jpg conversion:
  - moon_tiff = imread('moon.tif');
    imwrite(moon_tiff,'moon.jpg');
- Details on format specific parameters can be found on the reference pages of imread and imwrite

GIF - Graphics Interchange Format

- GIF Files
  - Indexed images
  - No compression
  - Supported bitdepths
    - 1 bit: logical
    - 2-8 bit: uint8
  - Multiframe (animated) GIF files are possible
  - Used on web-pages, as icons, buttons, …
- Format specific syntax of imread
  - [...] = imread(..., idx), when animated gif, reads one or more frames
    - Idx is integer scalar or vector

GIF - Writing

- imwrite(X, map, filename, Param1, val1, …)
- When writing multiframe GIF images
  - X should be a 4 dimensional m x n x 1 x p array where p is the number of frames to write
- Some GIF specific parameters
  - DelayTime: [0..655]   frame time
  - LoopCount: [0..65535] number of loops in animation

Example: Animated GIF Production

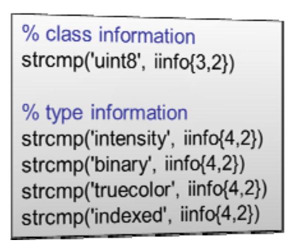
- Use of animated GIF files
  - Presentations (delayTime = 3 to 7 sec)
  - Movies (delayTime = 0.04 sec)
  - Animated icons, banners on web-pages
  - …
- Goal
  - Write an M-File that produces an animated GIF file from grayscale or rgb images placed in a given folder

Animated GIF: Input images

```
% input data
movieName = 'testAnimGif';
numberLoops = 3;
imageTime = 0.2;

% Create an array of filenames that make up the image sequence
fileFolder = fullfile(matlabroot,'toolbox','images','imdemos');
dirOutput = dir(fullfile(fileFolder,'AT3_1m4_*.tif'));

fileNames = {dirOutput.name}';
numFrames = numel(fileNames);
```

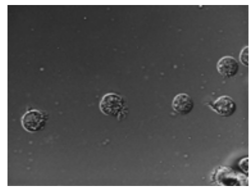
Animated GIF: Initialization

```
% read first image, display it, and get class and type
information
fname = fullfile(fileFolder, fileNames{1});
I = imread(fname);
h = imshow(I);
iinfo = imattributes(h);

% Preallocate the array
if strcmp('truecolor', iinfo{4, 2})
   nChannels = 3
   [I1, myMap] = rgb2ind(I, 256);
   sequence = zeros([size(I1) 1 numFrames], class(I1));
   sequence(:,:,:,1) = I1;
else
   nChannels = 1
   myMap = colormap('gray');
   sequence = zeros([size(I) 1 numFrames], class(I));
   sequence(:,:,:,1) = I;
end
```



Animated GIF: Creation and Saving

```
% Create image sequence array
for p = 2:numFrames
    if strcmp('truecolor', iinfo{4, 2})
        fname = fullfile(fileFolder, fileNames{p});
        I = imread(fname);
        [I1] = rgb2ind(I, myMap);
        sequence(:,:,:,p) = I1;
    else
        fname = fullfile(fileFolder, fileNames{p});
        sequence(:,:,:,p) = imread(fname);
    end
end

% Save animated GIF
imwrite(sequence, myMap, movieName,…
            'gif',…
            'LoopCount', numberLoops,…
            'DelayTime', imageTime);
```



Contents

- Introduction
- Reading and Writing Image data
- Spatial Transforms
- Image Registration
- Image Filters
- Transforms
- Morphological Operations

Spatial Transformations

- A spatial transformation is a geometric operation
- It modifies the spatial relationship between pixels in an image, mapping pixel locations in an input image to new locations in an output image
- Supported image transformations:
  - Resizing
  - Rotating
  - Cropping
  - General 2D spatial transformations
  - N dimensional spatial transformations

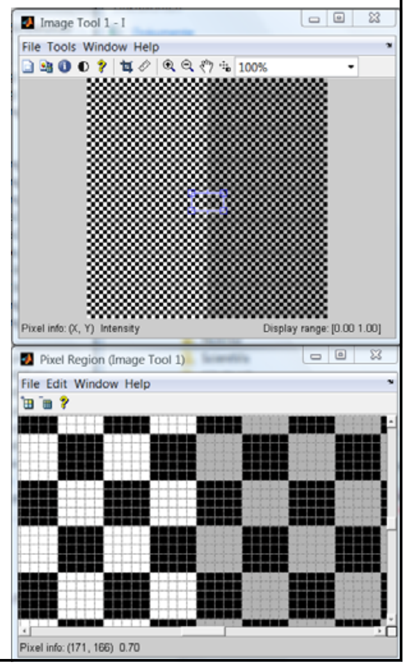
Resizing

- Images can be enlarged or reduced with imresize
  - J = imresize(I, 1.25);
  - J = imresize(I, [100 150]; aspect ratio is adjusted
  - J = imresize(I, [100 NaN]; aspect ratio is preserved
- Interpolation can be used when enlarging images to improve the result
- Antialiasing improves the result when reducing images – artifacts due to loss of information
  - Stair-step
  - Moiré patterns (ripple effect)

Resizing Artifacts

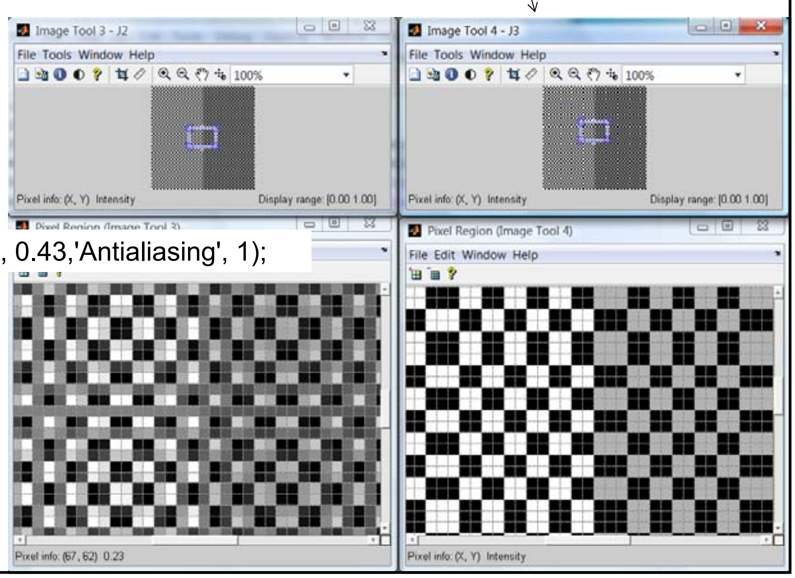
I = checkerboard(5, 30, 30);
imtool(I)

Original image



Ripple Effect

J3=imresize(I, 0.43,'Nearest', 'Antialiasing', 0);



J2=imresize(I, 0.43,'Antialiasing', 1);

Rotating

- To rotate an image, use the imrotate function.
- By default, imrotate creates an output image large enough to include the entire original image
- imrotate uses nearest-neighbor interpolation by default to determine the value of pixels in the output image
- This example rotates an image 35° counterclockwise and specifies bilinear interpolation.
- I = imread('circuit.tif'); J = imrotate(I,35,'bilinear'); imshow(I) figure, imshow(J)

Rotating

- To rotate an image, use the imrotate function.
- By default, imrotate creates an output image large enough to include the entire original image
- Interpolation methods are
  - Nearest neighbor (default)
  - Bilinear
  - Bicubic

Rotation examples
Stair-step effect

```
I = checkerboard(20,4,4);
J1 = imrotate(I,35,'nearest');
J1 = imrotate(I,35,'bilinear');
J1 = imrotate(I,35,'bicubic');
```

Nearest neighbor bilinear bicubic

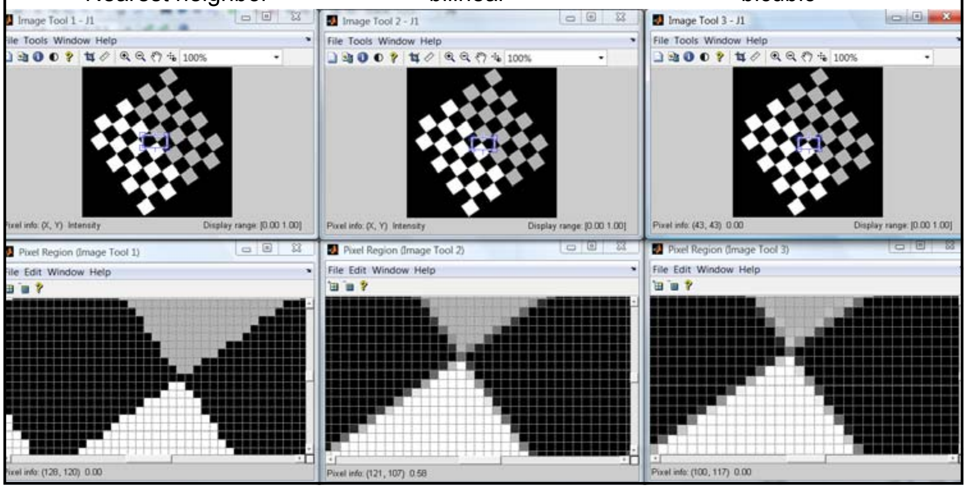
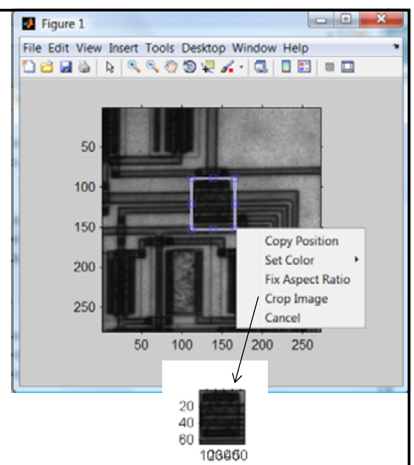
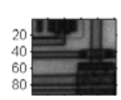


Image Cropping

- Extraction of a rectangular portion of an image
  - Interactively
    - I = imread('circuit.tif');
    - J = imcrop(I);



  - programmatically by specifying the size and position of the crop region
    - J = imcrop(I,[60 40 100 90]);

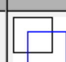
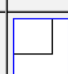
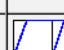
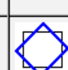


General 2D Spatial Transformations

- A three-step process in MATLAB
  1. Define the transformation parameters
  2. Create a transformation structure (TFORM, maketform) that defines the type of transformation you want to perform
  3. Perform the transform with imtransform

Transformation Matrices (3x3)

- Affine transformations
  - Rigid
    - Translation
    - Rotation
  - Scale
  - Shear
- Using sets of non-collinear points in input and output images
  - 3 points for affine, 4 points for perspective transformations

Affine Transform	Example	Transformation Matrix	
Translation		$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ t_x & t_y & 1 \end{bmatrix}$	t_x specifies the displacement along the *x* axis t_y specifies the displacement along the *y* axis.
Scale		$\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$	s_x specifies the scale factor along the *x* axis s_y specifies the scale factor along the *y* axis.
Shear		$\begin{bmatrix} 1 & sh_y & 0 \\ sh_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	sh_x specifies the shear factor along the *x* axis sh_y specifies the shear factor along the *y* axis.
Rotation		$\begin{bmatrix} \cos(q) & \sin(q) & 0 \\ -\sin(q) & \cos(q) & 0 \\ 0 & 0 & 1 \end{bmatrix}$	q specifies the angle of rotation.

MATLAB help

TFORM Structure

- Creation of a TFORM structure to specify the spatial transformation with
  - T = maketform(*transformationtype*, …*transformationData*)
  - Transformation types are
    - Affine
    - Projective
    - Box
    - Custom
    - composite

Transformation Types (1)

- Affine
  - Translation, rotation, scaling, shearing
  - Straight and parallel lines remain, rectangles might become parallelograms
- Projective
  - Straight lines remain, parallel lines converge toward vanishing points
- Box
  - Each dimension is shifted and scaled independently

Transformation Types (2)

- Custom
  - User defined, providing the forward and/or inverse functions
- Composite
  - Composition of two or more transformations

Transformation from control points

- With the function TFORM=cp2tform(in-points, base-points, transfType) spatial transformation can be inferred from control point pairs
- Transformation types
  - Nonreflective similarity (2 pairs)
  - Similarity (3 pairs)
  - Affine (3 pairs)
  - Projective (4 pairs)
  - Polynomial
  - Piecewise linear
  - Lwm (local weighted mean)
  - … (see help)

Performing the Spatial Transformation

- Finally, the image is transformed with the imtransform function and the specified TFORM structure
- J = imtransform(Image, tform);

Contents

- Introduction
- Reading and Writing Image data
- Spatial Transforms
- Image Registration
- Image Filters
- Transforms
- Morphological Operations

Image Registration

- Image registration is the process of aligning two or more images of the same scene
- Typically, an input image is brought into alignment with a base or reference image by applying spatial transformations
- Typical image differences are
  - Different viewpoints
  - Changes in perspective
  - Lens or sensor distortion

Image Registration Examples

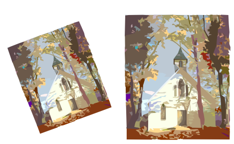
- Aligning of satellite images taken at different times to see how a river has migrated
- Aligning pictures taken from flying aeroplanes to create large maps
- Medical pre- and postop CT-images
- Aligning and comparing medical images created by different diagnostic modalities (MRI, CT)

Point Mapping

- Tools are provided by image processing toolbox which support point mapping
- Homologous point pairs (landmarks) in the base image and input image are manually selected
- Then, a spatial mapping is inferred from these control points
- This is often an iterative process experimenting with different types of transformations, before a satisfactory result is achieved

Illustration of point mapping process

Input image



Base image

⇩

Control point selection
cpselect

⇩

Fine tuning
cpcorr (optional)

⇩

Create spatial transformation structure (TFORM)
cp2tform

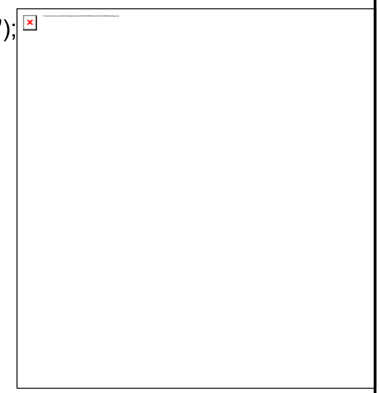
⇩

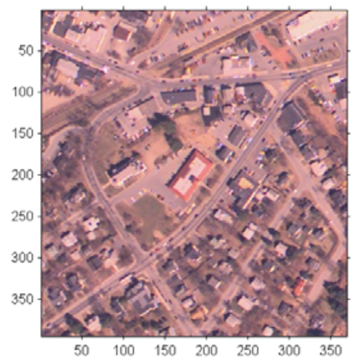
Perform spatial transformation
imtransform

1. Read the images

```
orthophoto = imread('westconcordorthophoto.png');
figure, imshow(orthophoto)
unregistered = imread('westconcordaerial.png');
figure, imshow(unregistered)
```

Base image
orthophoto

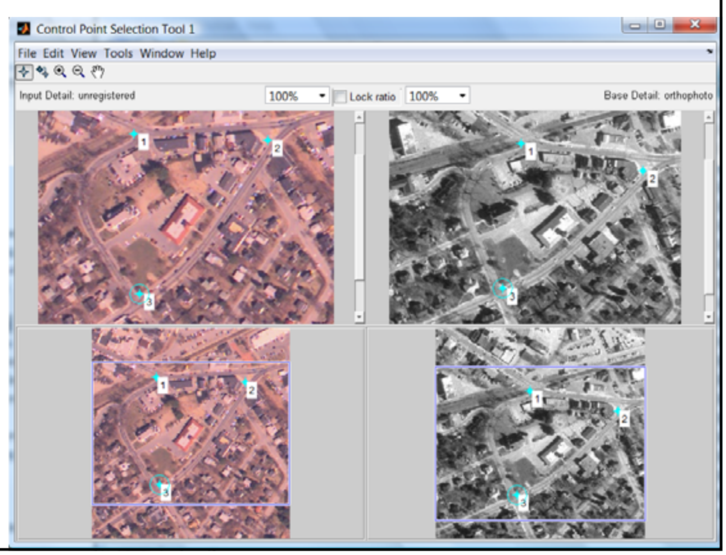




Input image
unregistered

2. Select Control Points

```
cpselect(unregistered, orthophoto)
```



3. Save Controlpoints to Workspace

input_points =

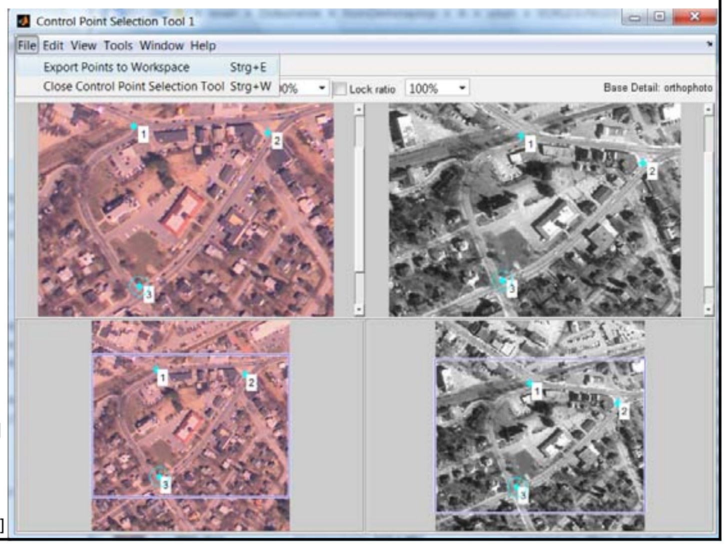
```
120.0000   93.0000
287.0000  101.0000
127.0000  293.0000
```

base_points =

```
165.0000  110.0000
317.0000  144.0000
142.0000  291.0000
```

cpstruct =

```
        inputPoints: [3x2 double]
         basePoints: [3x2 double]
     inputBasePairs: [3x2 double]
                ids: [3x1 double]
       inputIdPairs: [3x2 double]
        baseIdPairs: [3x2 double]
   isInputPredicted: [3x1 double]
    isBasePredicted: [3x1 double]
```



4. Specifiy and Compute TFORM

mytform = cp2tform(input_points, base_points, 'affine');

mytform

mytform <1x1 struct>

Field	Value	Min	Max
ndims_in	2	2	2
ndims_out	2	2	2
forward_fcn	@fwd_affine		
inverse_fcn	@inv_affine		
tdata	<1x1 struct>		

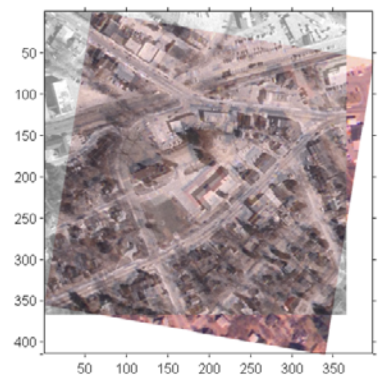
mytform.tdata

mytform.tdata <1x1 struct>

Field	Value	Min	Max
T	[0.9172,0.1605,0;-0.1471,0.8994,0;68.6134,7.0964,1]	-0.1471	68.6134
Tinv	[1.0599,-0.1892,0;0.1734,1.0809,0;-73.9540,5.3079,1]	-73.9540	5.3079

5. Transform the Input Image

```
registered = imtransform(unregistered, mytform, 'FillValues', 255);

figure; imshow(registered);

hold on

h = imshow(orthophoto, gray(256));

set(h, 'AlphaData', 0.6)
```



Contents

- Introduction
- Reading and Writing Image data
- Spatial Transforms
- Image Registration
- Image Filters
- Transforms
- Morphological Operations

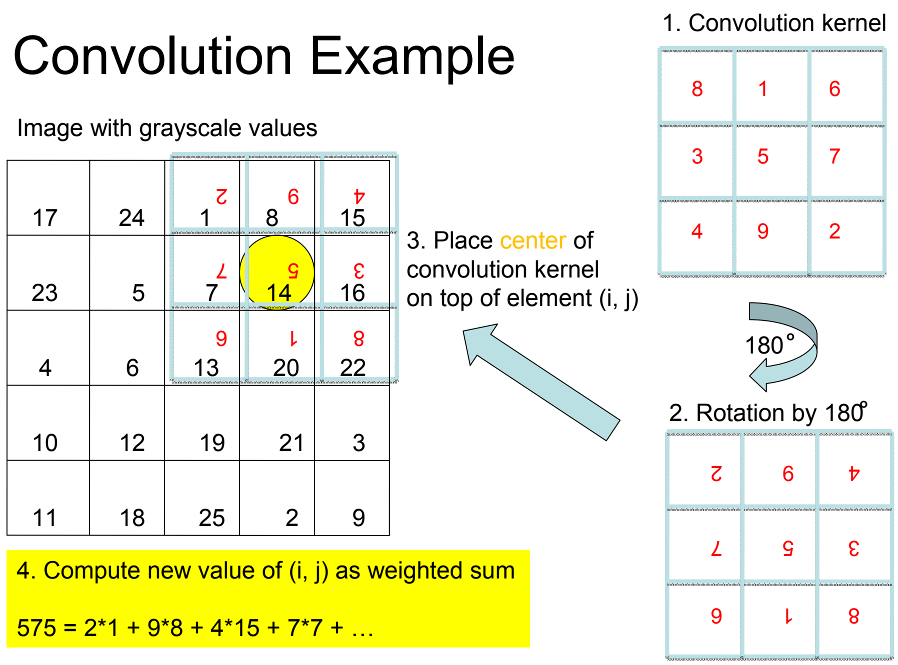
Linear Filters in the Spatial Domain

- Image filtering is a technique for modifying or enhancing images such as
  - Smoothing
  - Sharpening
  - Edge enhancements
- Filtering is neighborhood operation
  - The value of a given pixel in the output image is a function of the pixels in the neighborhood of the corresponding input pixel
- Linear filtering is an operation in which the value of an output pixel is linear combination of the its neighborhood pixels.

Convolution

- Linear filtering of an image is accomplished through an operation called *convolution*.
- Convolution is a neighborhood operation in which each output pixel is the weighted sum of neighboring input pixels.
- The matrix of weights is called the *convolution kernel*, also known as the *filter*.
- A convolution kernel is a correlation kernel that has been rotated 180 degrees.

Convolution Example

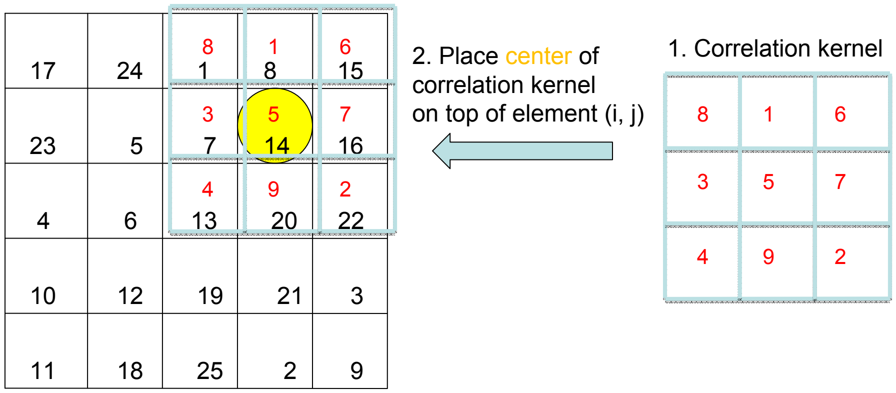


Correlation

- The operation called *correlation* is closely related to convolution
- In correlation, the value of an output pixel is also computed as a weighted sum of neighboring pixels.
- The difference is that the matrix of weights, in this case called the *correlation kernel*, is not rotated during the computation.
- The Image Processing Toolbox filter design functions return correlation kernels.

Correlation Example

Image with grayscale values



3. Compute new value of (i, j) as weighted sum

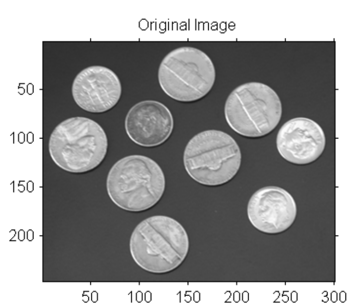
585 = 8*1 + 1*8 + 6*15 + 3*7 + …

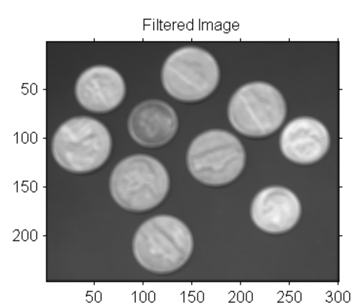
Example: Averaging Filter

```
I = imread('coins.png');
h = ones(5,5) / 25;

I2 = imfilter(I, h);

imshow(I), title('Original Image');
figure, imshow(I2), title('Filtered Image')
```

h =

0.0400	0.0400	0.0400	0.0400	0.0400
0.0400	0.0400	0.0400	0.0400	0.0400
0.0400	0.0400	0.0400	0.0400	0.0400
0.0400	0.0400	0.0400	0.0400	0.0400
0.0400	0.0400	0.0400	0.0400	0.0400





Options of imfilter

- imfilter(A,h): filter using correlation
- imfilter(A,h,'conv'): filter using convolution
- What happens if the kernel border falls outside the image?
  - Zero padding
    - outside image values are supposed to be zero
  - Replicated boundary pixels
    - outside image values are replicated boundary pixels
  - Symmetric
    - mirror-reflecting the array across the array border.
  - Circular:
    - assuming the input array is periodic

Zero Padding / Replicated

```
I = imread('eight.tif');
h = ones(5,5) / 25;
```

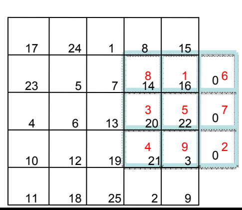
```
I2 = imfilter(I,h);
```

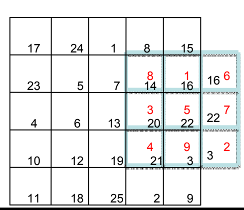
```
I3 = imfilter(I,h,'replicate');
```

Original Image

Filtered Image with Black Border

Filtered Image with Border Replication





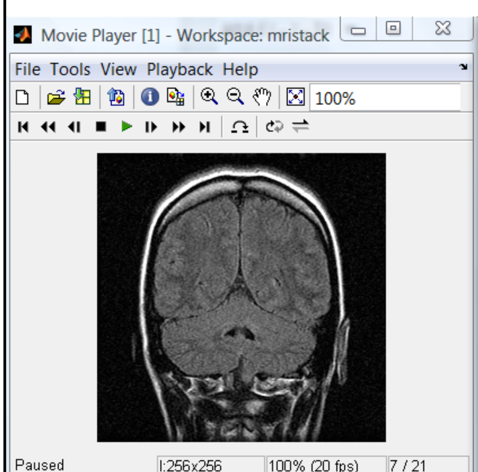
Multidimensional Filtering

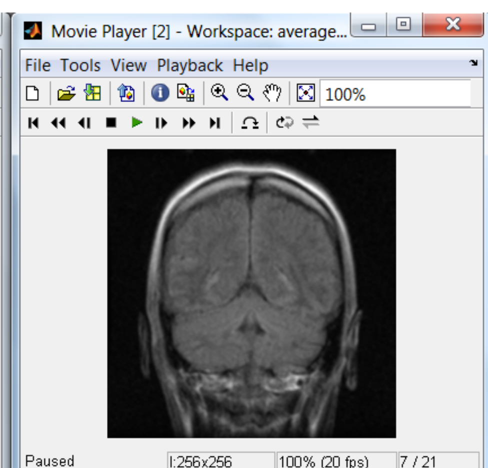
- Imfilter also handles multidimensional images with multidimensional filters
- Example of filtering an rgb image with 2D averaging kernel
  - Each color plane is averaged with 2D filter

```
h = ones(5,5)/25;
rgb2 = imfilter(rgb,h);
```

3D Filtering of MRI Image Stack

```
load mristack

h=ones(3,3,3) / 27;

averageStack=imfilter(mristack,h);
```





Predefined Filters

- h = fspecial(*type*) creates a 2D filter h of the specified type
- fspecial returns h as a correlation kernel, which is the appropriate form to use with imfilter

- *type* is a string having one of these values
  - Avarage
  - Disk
  - Gaussian
  - Laplacian
  - Log
  - Motion
  - Prewitt
  - Sobel
  - unsharp

Contrast Enhancement

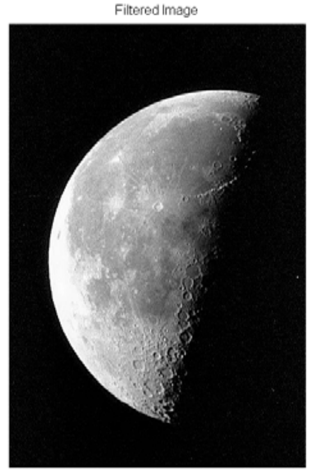
Original Image

```
I = imread('moon.tif');
h = fspecial('unsharp');
I2 = imfilter(I,h);
```

h =

-0.1667	-0.6667	-0.1667
-0.6667	4.3333	-0.6667
-0.1667	-0.6667	-0.1667

Filtered Image



Contents

Transforms

- Normally, an image is mathematically represented as an intensity function f(x,y) of two spatial variables (x,y): spatial domain
- The term transform refers to an alternative mathematical representation of an image
- For example, in the frequency domain, an image is represented by a sum of complex exponentials of varying magnitudes, frequencies and phases
- Transforms can be useful for a wide range of purposes such as
  - convolution, enhancement, feature detection, and compression

Examples of Transforms

- Fourier Transform
- Discrete Cosine Transform
- Radon Transform
- The inverse Radon Transform
- Fan-Beam Projection

FT: The Fourier Transform

- The Fourier transform is a representation of an image as a sum of complex exponentials of varying magnitudes, frequencies, and phases
- The Fourier transform plays a critical role in a broad range of image processing applications, including image
  - Enhancement
  - Analysis
  - Restoration
  - Compression.

Definition of Fourier Transform

- f(m,n) is a function of two discrete spatial variables m and n
- The 2D Fourier transform of f(m, n) is given by

$$F(\omega_1,\omega_2)=\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} f(m,n)\cdot e^{-i\omega_1 m}\cdot e^{-i\omega_2 n} \qquad \omega=\frac{2\pi}{T}$$

- ω_1,ω_2 are frequency variables (radians/sample)
- Called the frequency domain representation of f(m,n)
- In ω_1,ω_2 periodic complex valued function with period 2π
- DC (direct current) or constant component

$$F(0,0)=\sum_{m=-\infty}^{\infty}\sum_{n=-\infty}^{\infty} f(m,n)$$

The inverse Fourier Transform

- The inverse two-dimensional Fourier transform is given by

$$f(m,n)=\frac{1}{4\pi^2}\int_{\omega_1=-\pi}^{\pi}\int_{\omega_2=-\pi}^{\pi} F(\omega_1,\omega_2)\cdot e^{i\omega_1 m}\cdot e^{i\omega_2 n}\,d\omega_1 d\omega_2$$

- f(m,n) can be represented as a
  - sum of an infinite number of complex exponentials (sinusoids) with different frequencies ω_1,ω_2
  - The magnitude and phase of the contribution at the frequencies are given by $F(\omega_1,\omega_2)$

DFT: Discrete Fourier Transform

- Input and output values are discrete
- Values are nonzero only over a finite region
- There exists an algorithm for computing efficiently the DFT, also called FFT (fast Fourier transform)

$$p, m = 0, 1, \ldots, M-1$$
$$q, n = 0, 1, \ldots, N-1$$

- DFT $$F(p,q) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) \cdot e^{-i\left(\frac{2\pi}{M}\right)pm} \cdot e^{-i\left(\frac{2\pi}{N}\right)qn}$$
- Inverse DFT $$f(m,n) = \frac{1}{MN} \sum_{p=0}^{M-1} \sum_{q=0}^{N-1} F(p,q) \cdot e^{i\left(\frac{2\pi}{M}\right)pm} \cdot e^{i\left(\frac{2\pi}{N}\right)qn}$$

Relationship between FT and DFT

- The DFT coefficients F(p,q) are discrete samples of the Fourier transform $F(\omega_1, \omega_2)$

$$p = 0, 1, \ldots, M-1$$
$$q = 0, 1, \ldots, N-1$$
$$F(p,q) = F(\omega_1, \omega_2)\Big|_{\substack{\omega_1 = 2\pi p/M \\ \omega_2 = 2\pi q/N}}$$

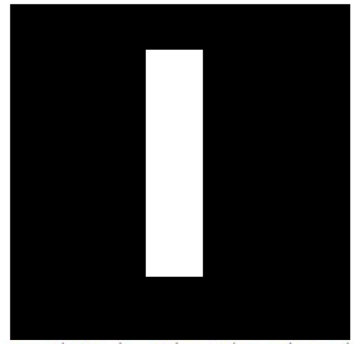
DFT in MATLAB

- MATLAB supports the computation of the DFT by the FFT algorithm in one, two, and N-dimensions
- FFT
  - fft, fft2, fftn
- Inverse FFT
  - ifft, ifft2, ifftn
- Rearangement / centering of output
  - Shift zero-frequency component to center of spectrum
  - fftshift, ifftshift

Visualizing the FT

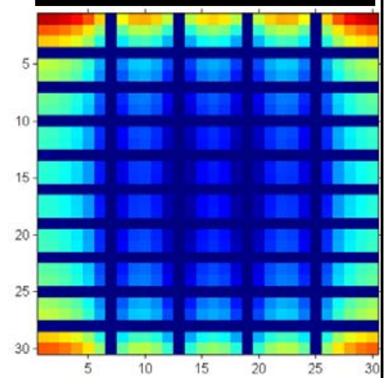
- Ways to visualize the DFT
  - Mesh plot of the magnitude $\left|F\left(p,q\right)\right|$
  - 2D image with colormap of $\log\left(\left|F\left(p,q\right)\right|\right)$

Ex: DFT of rectangular region (1)



- Construction of image

```
f = zeros(30,30);
f(5:24,13:17) = 1;
imshow(f,'InitialMagnification','fit')
```



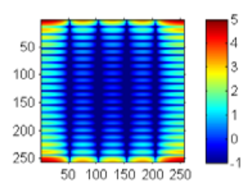
- Compute and visualize the 30-by-30 DFT

```
F = fft2(f);
F2 = log(abs(F));
imshow(F2,[-1 5],'InitialMagnification','fit');
colormap(jet);
```

Ex: DFT of rectangular region (2)

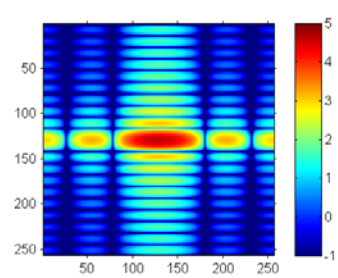
- Finer sampling by zero padding

```
F = fft2(f,256,256);
imshow(log(abs(F)),[-1 5]);
colormap(jet);
colorbar
```



- Centering of zero-frequency

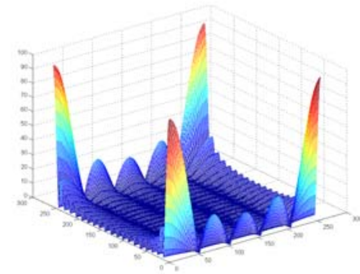
```
F2 = fftshift(F);
imshow(log(abs(F2)),[-1 5]);
colormap(jet);
colorbar
```



Ex: DFT of rectangular region (3)

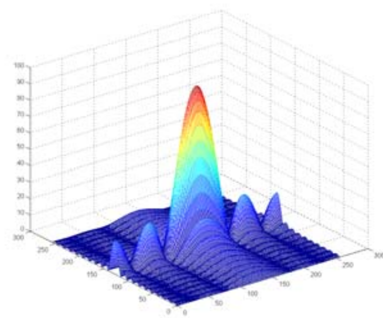
- Visualization as mesh plot of magnitude

```
[X,Y] = meshgrid(0:255);
mesh(X,Y,abs(F));
```



- Centered zero-frequency

```
mesh(X,Y,abs(F2));
```



Fast Convolution

- Key property of the Fourier transform:
  - The multiplication of two Fourier transforms corresponds to the convolution of the associated spatial functions
- The FFT-based convolution method is most often used for large inputs. For small inputs it is generally faster to use imfilter

Example of Fast Convolution

- Create 2 matrices and zero pad them
- Fast convolution
  - Compute the DFTs of both matrices
  - Multiply both DFTs
  - compute the inverse 2D DFT of the result
- (Verify with conv2)

```
A = magic(3);
B = ones(3);
A(8,8) = 0;
B(8,8) = 0;
```

```
C = ifft2( fft2(A) .* fft2(B) );
```

```
C = C(1:5,1:5);
C = real(C)
```

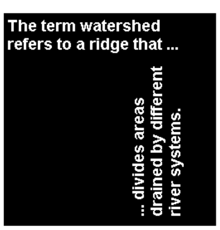
```
C =
    8.0000    9.0000   15.0000    7.0000    6.0000
   11.0000   17.0000   30.0000   19.0000   13.0000
   15.0000   30.0000   45.0000   30.0000   15.0000
    7.0000   21.0000   30.0000   23.0000    9.0000
    4.0000   13.0000   15.0000   11.0000    2.0000
```

```
A = magic(3);
B = ones(3);
conv2(A,B)
```

```
ans =
     8     9    15     7     6
    11    17    30    19    13
    15    30    45    30    15
     7    21    30    23     9
     4    13    15    11     2
```

Locating Image Features with Correlation (1)

- Correlation by using the Fourier transform
- Correlation is also called template matching
- Problem:
  - locate occurrences of the letter *"a"* in an image containing text



1. Read image
2. Make template of letter “a”


```
bw = imread('text.png');
```

```
a = bw(32:45,88:98);
```

Locating Image Features with Correlation (2)

- Compute the correlation of the template image with the original image by rotating the template image by 180^o and then using the FFT-based convolution technique
  - Convolution is equivalent to correlation if you rotate the convolution kernel by 180^o

```
C = real( ifft2( fft2(bw)  .*  fft2( rot90(a,2), 256, 256 ) ) );
```

Locating Image Features with Correlation (3)

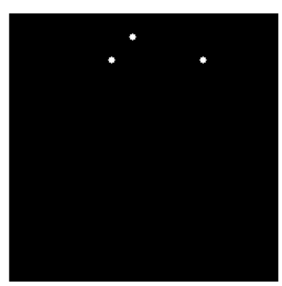
- Determine the locations of the template

```
thresh = max(C(:))*0.88

I = (C > thresh);
```

thresh =
59.84

```
se = strel('disk',3,0)

I2 = imdilate(I, se);
```

Image dilation →



The Discrete Cosine Transform DCT

- The DCT represents an image as a sum of sinusoids of varying magnitudes and frequencies
- The dct2 function computes the 2D DCT of an image
- The DCT has the property that, for a typical image, most of the visually significant information about the image is concentrated in just a few coefficients of the DCT
- For this reason, the DCT is often used in image compression applications
- For example, the DCT is at the heart of the international standard lossy image compression algorithm known as JPEG.
  - The name comes from the working group that developed the standard: the Joint Photographic Experts Group

The 2D DCT of an M-by-N matrix A

$$B_{pq} = \alpha_p \alpha_q \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} A_{mn} \cos\left(\frac{\pi(2m+1)p}{2M}\right) \cos\left(\frac{\pi(2n+1)p}{2N}\right), \quad \begin{matrix} 0 \le p \le M-1 \\ 0 \le p \le N-1 \end{matrix}$$

$$\alpha_p = \begin{cases} 1/\sqrt{M}, & p=0 \\ \sqrt{2/M}, & 1 \le p \le M-1 \end{cases}, \qquad \alpha_q = \begin{cases} 1/\sqrt{N}, & q=0 \\ \sqrt{2/N}, & 1 \le q \le N-1 \end{cases}$$

The B_{pq} are called DCT coefficients of the image A

The inverse 2D DCT

$$A_{mn} = \sum_{p=0}^{M-1}\sum_{q=0}^{N-1} B_{pq}\alpha_p\alpha_q \cos\left(\frac{\pi(2m+1)p}{2M}\right)\cos\left(\frac{\pi(2n+1)p}{2N}\right), \quad \begin{matrix} 0 \le m \le M-1 \\ 0 \le n \le N-1 \end{matrix}$$

$$\alpha_p = \begin{cases} 1/\sqrt{M}, & p=0 \\ \sqrt{2/M}, & 1 \le p \le M-1 \end{cases}, \qquad \alpha_q = \begin{cases} 1/\sqrt{N}, & q=0 \\ \sqrt{2/N}, & 1 \le q \le N-1 \end{cases}$$

The basis functions of the DCT

$$\alpha_p\alpha_q \cos\left(\frac{\pi(2m+1)p}{2M}\right)\cos\left(\frac{\pi(2n+1)p}{2N}\right), \quad \begin{matrix} 0 \le p \le M-1 \\ 0 \le q \le N-1 \end{matrix}$$

The DCT coefficients B_{pq}, then, can be regarded as the *weights* applied to each basis function

The DCT in MATLAB

- There are two ways to compute the DCT using Image Processing Toolbox functions
- dct2
  - An FFT-based algorithm for speedy computation with large inputs
- dctmtx

$$T_{pq} = \begin{cases} 1/\sqrt{M}, & p=0, \quad 0 \le q \le M-1 \\ \sqrt{2/M}\cos\left(\frac{\pi(2q+1)p}{2M}\right), & 1 \le p \le M-1, \quad 0 \le q \le M-1 \end{cases}$$

  - returns the square orthonormal DCT transform matrix to be used for transforming efficiently small square images
- The 2D DCT of the matrix A is computed as
  - B = T * A * T'
- And the inverse 2D DCT of the matrix A as
  - A = T' * B * T

Image Compression with DCT

- JPEG image compression algorithm uses DCT
- Input image is divided into 8-by-8 or 16-by-16 blocks for which the 2D DCT is computed
- The DCT coefficients are then quantized, coded, and transmitted (saved)
- The JPEG receiver (or JPEG file reader) decodes the quantized DCT coefficients, computes the inverse two-dimensional DCT of each block, and then puts the blocks back together into a single image.
- For typical images, many of the DCT coefficients have values close to zero; these coefficients can be discarded without seriously affecting the quality of the reconstructed image.

Example Code for JPEG compression

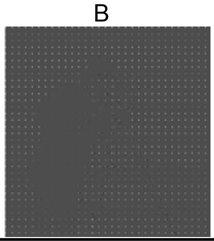
```
I = imread('cameraman.tif');
I = im2double(I);

T = dctmtx(8);
dct = @(block_struct) T * block_struct.data * T';
B = blockproc(I,[8 8],dct);
```

```
mask = [
1 1 1 1 0 0 0 0
1 1 1 0 0 0 0 0
1 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0];
```

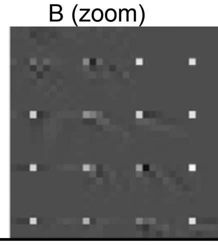
```
B2 = blockproc(B,[8 8],@(block_struct) mask .* block_struct.data);
invdct = @(block_struct) T' * block_struct.data * T;
I2 = blockproc(B2,[8 8],invdct);
```

I

B



B (zoom)

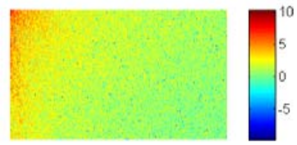


I2

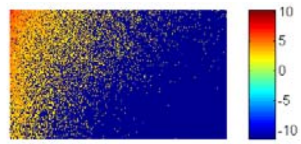
Image Compression with dct2

```
RGB = imread('autumn.tif');
I = rgb2gray(RGB);
```

```
J = dct2(I);
imshow(log(abs(J)),[]), colormap(jet(64)), colorbar;
```




```
J(abs(J) < 10) = 0.00001;
imshow(log(abs(J)),[]), colormap(jet(64)), colorbar;
```




```
K = idct2(J);
imshow(K, [0 255])
```

Contents

- Introduction
- Reading and Writing Image data
- Spatial Transforms
- Image Registration
- Image Filters
- Transforms
- Morphological Operations

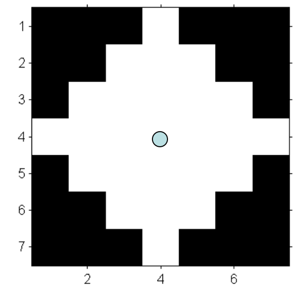
Morphological Operations

- In image processing morphological operations are used for
  - Contrast enhancement
  - Noise removal
  - Thinning
  - Skeletonization,
  - Filling
  - Segmentation

Morphology

- *Morphology* is a broad set of image processing operations processing images based on shapes
- Morphological operations apply a structuring element to an input image, creating an output image of the same size
- In a morphological operation, the value of each pixel in the output image is based on a comparison of the corresponding pixel in the input image with its neighbors defined by a structuring element
  - By choosing the size and shape of the neighborhood, you can construct a morphological operation that is sensitive to specific shapes in the input image.

Structuring Element

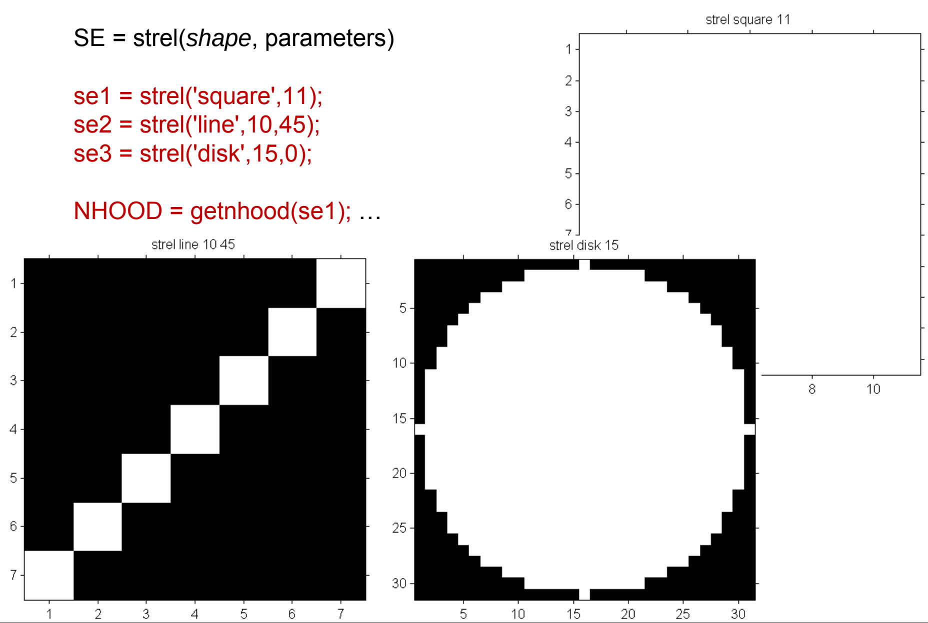

Diamond like structuring Element with origin

- A structuring element is a matrix consisting of only 0's and 1's that can have any arbitrary shape and size
- Pixel values of 1 define the neighborhood of a processed pixel
- Structuring Elements can be 1D, 2D or 3D
- The center or the origin of the structuring element identifies the pixel being processed
- The origin is given by:
  - origin = floor((size(nhood)+1)/2)

Examples of Structuring Elements

SE = strel(*shape*, parameters)

```
se1 = strel('square',11);
se2 = strel('line',10,45);
se3 = strel('disk',15,0);

NHOOD = getnhood(se1); …
```



Dilation and Erosion

- The most basic morphological operations are dilation and erosion.
- Dilation adds pixels to the boundaries of objects in an image
- Erosion removes pixels on object boundaries
- The number of pixels added or removed from the objects in an image depends on the size and shape of the *structuring element* used to process the image
- The value of given pixel in the output image is determined by applying a rule to the corresponding pixel and its neighbors in the input image.

Rules of Dilation and Erosion

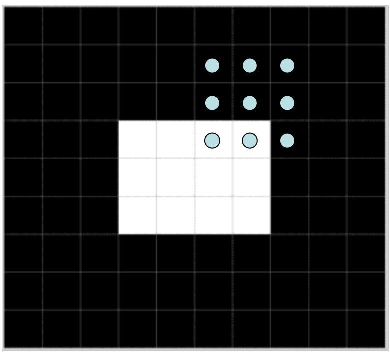
- Dilation
  - The value of the output pixel is the *maximum* value of all the pixels in the input pixel's neighborhood
  - In a binary image, if any of the pixels is set to the value 1, the output pixel is set to 1
- Erosion
  - The value of the output pixel is the *minimum* value of all the pixels in the input pixel's neighborhood
  - In a binary image, if any of the pixels is set to 0, the output pixel is set to 0.

Dilation Example

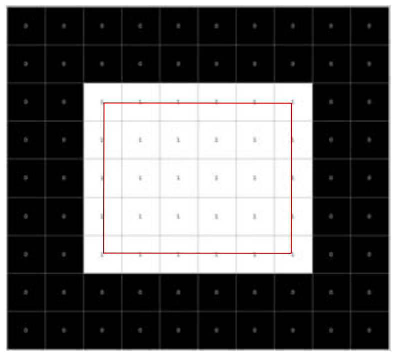
SE =
Flat STREL object containing 9 neighbors.
Neighborhood:

```
1 1 1
1 1 1
1 1 1
```

```
BW = zeros(9,10);
BW(4:6,4:7) = 1
```



```
SE = strel('square',3);
BW2 = imdilate(BW,SE)
```

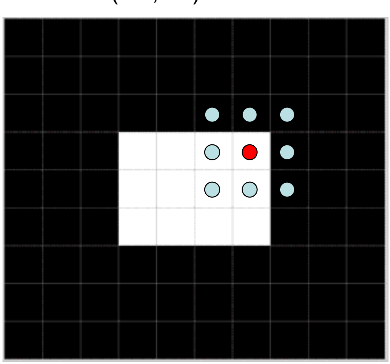


Erosion Example

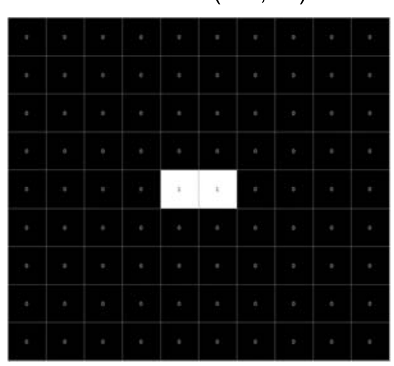
SE =
Flat STREL object containing 9 neighbors.
Neighborhood:

```
1 1 1
1 1 1
1 1 1
```

```
BW = zeros(9,10);
BW(4:6,4:7) = 1
```



```
SE = strel('square',3);
BW2 = imerode(BW,SE)
```

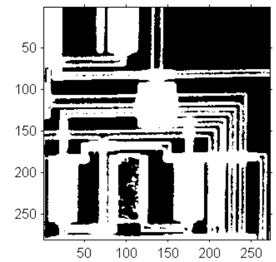


Morphological Opening

- Morphological *opening* of an image is an erosion followed by a dilation, using the same structuring element for both operations
  - imopen or equivalent
  - imerode and imopen
- Use morphological opening to remove small objects from an image while preserving the shape and size of larger objects in the image

Example of Morphological Opening

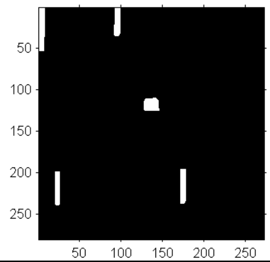
Problem: Remove small thin lines of
BW1 = imread('circbw.tif');



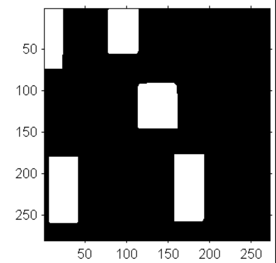
The structuring element should be large enough to remove the lines when you erode the image, but not large enough to remove the rectangles.
SE = strel('rectangle',[40 30]);

BW2 = imerode(BW1,SE);

BW3 = imopen(BW1,SE);



BW3 = imdilate(BW2,SE);

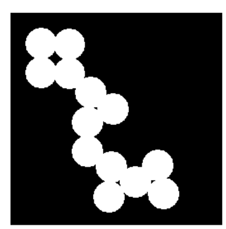


Morphological Closing

- Morphological *closing* of an image consists of dilation followed by an erosion with the same structuring element
  - imclose
  - imdilate and imerode
- Fills holes and gaps

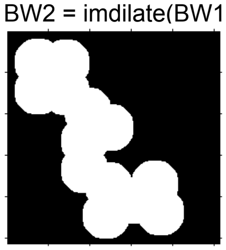
Example of Morphological Closing

Problem: fill holes and gaps of
BW1 = imread('circbw.tif');



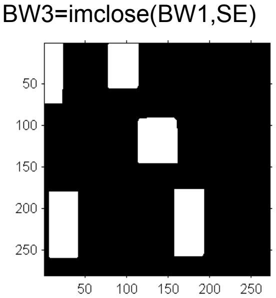
The structuring element should be large enough to fill the holes and gaps
SE = strel('circle',10);

BW2 = imdilate(BW1,SE);



BW3 = imerode(BW2,SE);

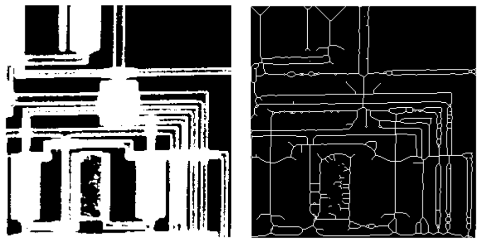
BW3=imclose(BW1,SE)



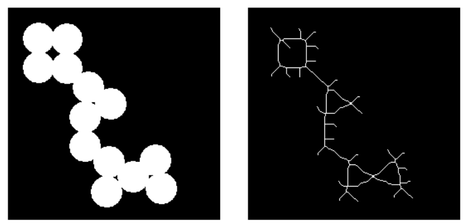
Skeletonization

Reduces all objects in an image to lines,
without changing the essential structure of the image

```
BW1 = imread('circbw.tif');
BW2 = bwmorph(BW1,'skel',Inf);
```



```
BW = imread('circles.png');
BW3 = bwmorph(BW,'skel',Inf);
```



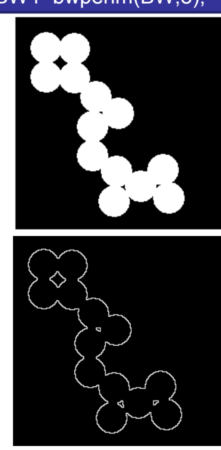
Determination of Perimeter

bwperim returns a binary image containing only the perimeter pixels of objects
in the input image.

A pixel is part of the perimeter if it is nonzero
and it is connected to at least one zero-valued pixel.

The default connectivity is 4 for two dimensions,
6 for three dimensions.

```
BW = imread('circles.png');
BW4=bwperim(BW,8);
```



```
BW1 = imread('circbw.tif');
BW2 = bwperim(BW1,8);
```