Numerical Methods in TEM

Christoph T. Koch

Max Planck Institut für Metallforschung

http://hrem.mpi-stuttgart.mpg.de/koch/Vorlesung

Outline of this second lecture

- Fourier Filtering
- Aligning images (cross-correlation, phase correlation, sub-pixel precision)

Fourier Filtering of Images and Diffractograms

Fourier-processing of images can be used to:

- reduce/enhance features of a certain range of spatial frequencies (e.g. low/high frequency noise, lattice information)
- smoothen images
- interpolate images (increase the sampling)
- correlate two images (cross-correlation, phase correlation, etc.)
- determine relative image shift with sub-pixel precision

Image filters in the DM Script Database

http://www.felmi-zfe.tugraz.at/dm_scripts/welcome.html

TU Graz Graz University of Technology

FELMI
back

home
policy
contact
submission

search

last update:
04/18/2007

[rss-feed]

freeware scripts grouped by category

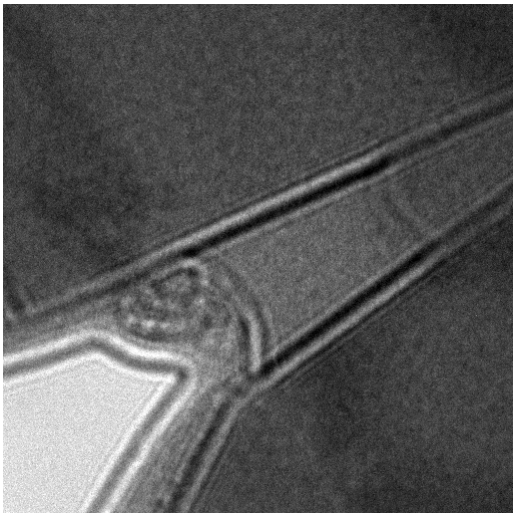
—Category—

Image Filters

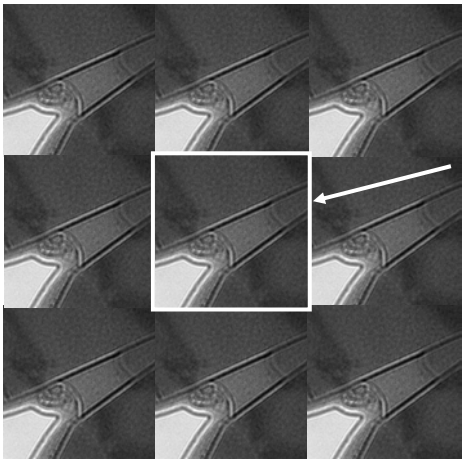
- 2nd Derivative
- Butterworth Filter
- Emboss Filter
- Extremum Filter
- Filter-GaussHat
- Frei and Chen Filter
- GS Closing Filter 3x3
- GS Closing Filter 3x5
- GS Closing Filter 5x5
- GS Opening Filter 3x3
- GS Opening Filter 3x5
- GS Opening Filter 5x5
- Hilditch Skeletonization
- Horizontal Derivatives Filter
- HRTEM Filter
- Hurst Texture Filter
- Hybrid Median Filter
- Kirsch Filter
- Laplacian Filter
- Marr Hildreth (DoG) Filter
- Maximum Filter
- Median Filter
- Mid Point Filter
- Minimum Filter
- Mexican Hat Filter (7x7)
- Olympic Filter
- Prewitt Filter
- Quick Threshold
- Range Filter
- Remove FEG Fluctuations
- Remove X Ray Filter
- Roberts Cross Filter
- Rolling Ball Filter
- Rolling Ball Treshold Filter
- Scale Image Intensity
- Sobel Filter
- Top Hat Filter
- Top Hat Threshold Filter
- Variance Filter
- Vertical Derivatives Filter

Artefacts in Fourier-processing

Origin of horizontal/vertical lines in FFT



Origin of horizontal/vertical lines in FFT



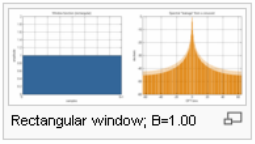
Very sharp boundaries produce streaks in the FFT containing all spatial frequencies.

How to avoid FFT streaks

- Multiplication with a Mask that reduces the edges
  - Hamming window
  - Hann window [also called Hanning-, or raised cosine window],
  - Gauss window,
  - Triangular window,
  - etc.
- Edge smoothing

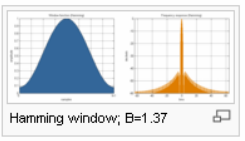
Window functions

Rectangular window



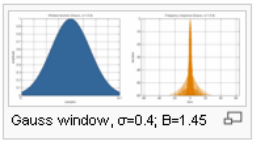
$$w(n) = 1$$

Hamming window



$$w(n) = 0.53836 - 0.46164 \cos\left(\frac{2\pi n}{N-1}\right)$$

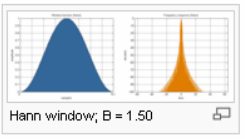
Gauss windows



$$w(n) = e^{-\frac{1}{2}\left(\frac{n-(N-1)/2}{\sigma(N-1)/2}\right)^2}$$

$$\sigma \leq 0.5$$

Hann window



$$w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right)$$

Usage: Multiply image with these windows before performing FFT

The command ExprSize

Image ExprSize(width,height,expression)

This function can be used to define the range of a 2-dimensional loop by defining the limits of irow and icol in a mathematical expression.
For example:

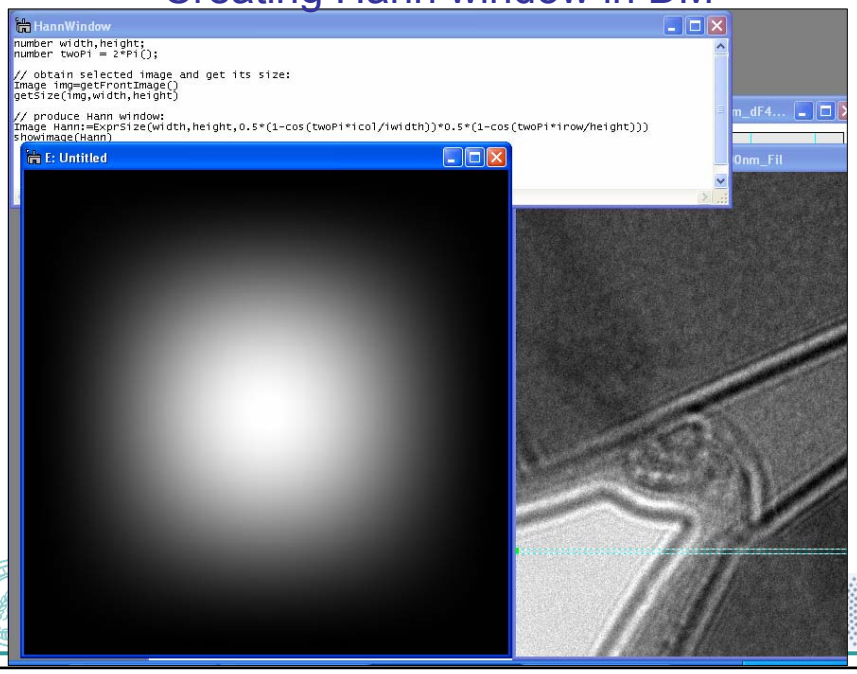
```
Image img:=ExprSize(width,height,cos(0.3*icol)*cos(0.1*irow))
```

Executes in less than 1sec for a 500 x 500 image and replaces:

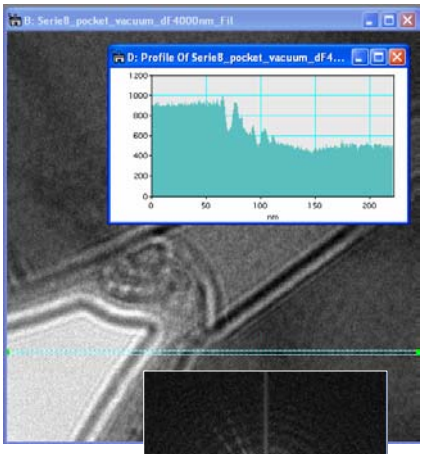
```
number ix,iy
img:= CreateFloatImage("test image",width,height);
for (ix=0;ix<width;ix++) {
  for (iy=0;iy<height;iy++) {
    SetPixel(img,ix,iy,cos(0.3*ix)*cos(0.1*iy))
  }
}
```

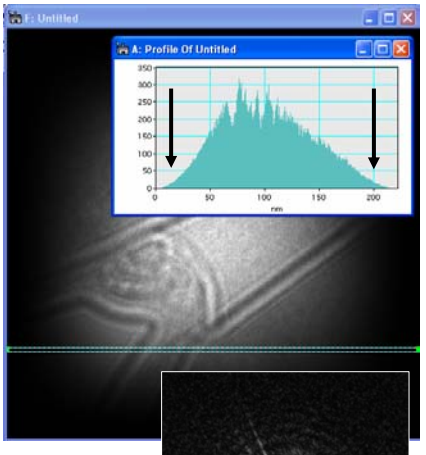
which takes 38 seconds to compute for 500 x 500 pixels.

Creating Hann window in DM



Problem with Window multiplication





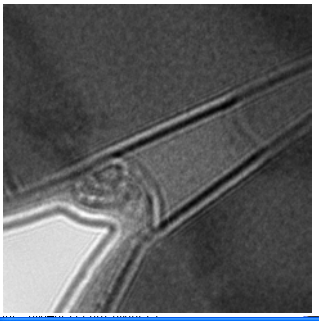
Alternative: Edge Smoothing

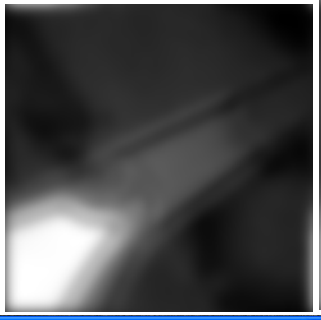
Instead of multiplying the whole image with a window function, one may also smoothen the image (this will produce leakage across image boundaries) and interpolate between the smoothed and the exact image close to the boundary.

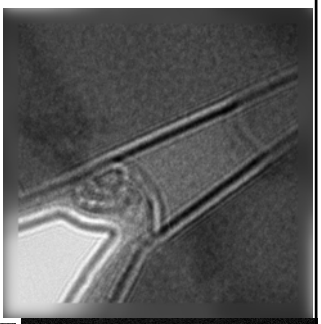
$$I_{new}(\vec{r}) = \frac{(1-\Delta)}{N_{edge}}\left[I(\vec{r}) \otimes Gauss(\vec{r})\right] + \frac{\Delta}{N_{edge}} I(\vec{r})$$

Δ: the number of pixels away from the edge
N_{edge}: the width of the edge

Edge smoothing

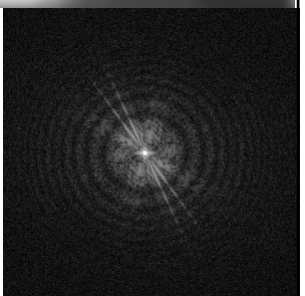






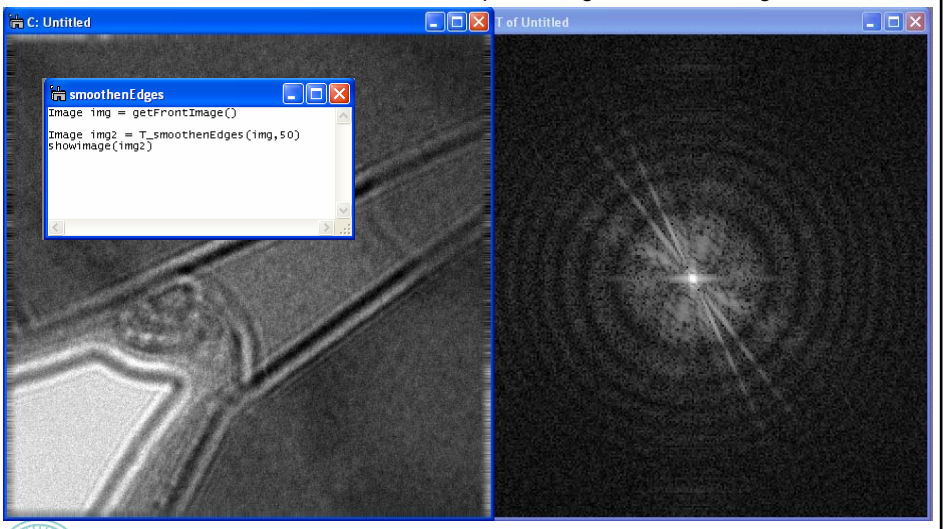
GaussEdgeSmoothing

```
number width, height, h2,w2
number Gwidth = 10;  // width of Gaussian in pixels in reciprocal space
number Nedge  = 100; // width of smoothed edge in pixels

// obtain selected image and get its size:
Image img=getFrontImage()
getSize(img,width,height)
h2 = height/2;
w2 = width/2;

// produce a Gaussian of equal size
ComplexImage Gauss:=ExprSize(width,height,exp(-((irow-h2)**2+(icol-w2)**2)/(Gwidth**2)))

// convolute image with Gaussian
Image img2 = realifft(realfft(img)*Gauss)

// replace edge of original image with smoothed image:
Image img3 = tert((abs(icol-w2)<w2-Nedge)&&(abs(irow-h2)<h2-Nedge),img,img2)

showimage(img3)
```



Fast Edge Smoothing

Instead of convolution with a Gaussian, a simple average across the edges is assumed



T_smoothenEdges(img,Nedge) is provided by Transforms.dll

Image Interpolation

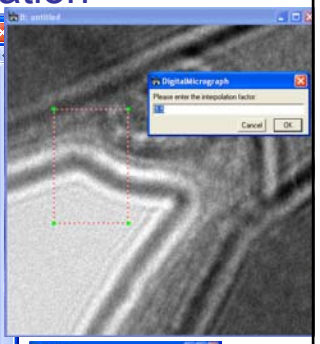
There are many ways to interpolate between the pixels of an image.
They differ in the function they assume for image intensity value between pixels.

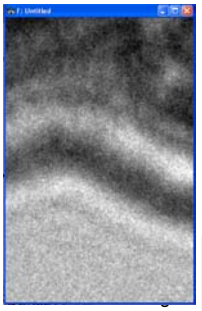
- Linear interpolation: I[x] = (1-dx) ·I[floor(x)] + dx · I[ceil(x)]
- Cubic interpolation: includes second derivative. Be careful when interpolating noisy data with a cubic spline!!!
- Fourier interpolation: simply add arbitrarily many zeros in reciprocal space

Fourier Image Interpolation

FourierInterpolate

```
number width, height, h2,w2
number widthNew,heightNew,hn2,wn2
number interpFactor = 1.5;
number top,left,bottom,right

// Obtain user input, exit, if the user pressed "Cancel"
if (!GetNumber("Please enter the interpolation factor: ",interpFactor,interpFactor)) {
    exit(0)
}

// obtain selected image and get its size:
ComplexImage img=getFrontImage()[]
GetSelection(img, top, left, bottom, right)
width  = right - left
height = bottom - top

// compute the center of the selected image:
h2 = height/2;
w2 = width/2;

// generate size of even sized new image
widthNew  = 2*floor(0.5*interpFactor*width)
heightNew = 2*floor(0.5*interpFactor*height)
hn2 = heightNew/2;
wn2 = widthNew/2;

// generate the new image
ComplexImage img2:=CreateComplexImage("Interpolated Image",widthNew,heightNew);

// Compute FFT of current image (selection)
T_fft_C2C(img,img)
img = T_shiftImageCenterComplex(img)

// place the FFT of the current image into the new image
img2[hn2-h2,wn2-w2,hn2+h2,wn2+w2] = img
img2 = T_shiftImageCenterComplex(img2)

// inverse FFT the bigger image:
T_ifft_C2C(img2,img2);

Image img3=real(img2)
showimage(img3)

// clean up memory:
deleteImage(img2);
```





See also script for rebinning in database

Image Alignment

Cross correlation

$$\mathrm{XCF}(\mathbf{x}) = \mathrm{FT}^{-1}[c_1^* c_2]$$

[coef=1]

c1 = fft(img1)
c2 = fft(img2)

Mutual correlation

$$\mathrm{MCF}(\mathbf{x}) = \mathrm{FT}^{-1}\left[\frac{c_1^* c_2}{\sqrt{|c_1^* c_2|}}\right]$$

[coef=0.5]

Phase correlation

$$\mathrm{PCF}(\mathbf{x}) = \mathrm{FT}^{-1}\left[F(k)\frac{c_1^* c_2}{|c_1^* c_2|}\right]$$

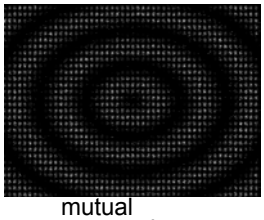
[coef=0]

Implementation in DM (if Transforms.dll is installed):

T_XCorrelateGeneral(img2,img1,xcorr,coef,damping)

(if damping > 0, a Gaussian function is also multiplied to the product c1*·c2)

Different correlation functions



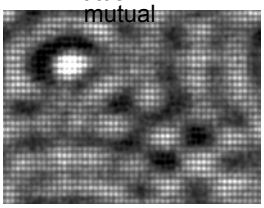
(Script testfftw.s on website)

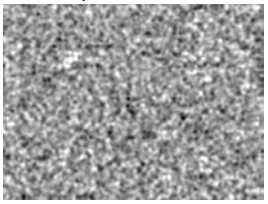
mutual

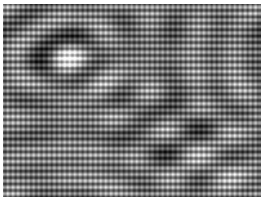
phase

cross correlation

mutual







Script for Testing different image alignment schemes
Input shift: 103, 84
Regular (damped) Cross-correlation has maximum value of 4618.22 at (103, 84)
Phase correlation has maximum value of 279.992 at (95, 84)
Mutual correlation has maximum value of 521.172 at (103, 84)

Image Alignment with Defocus-Correction

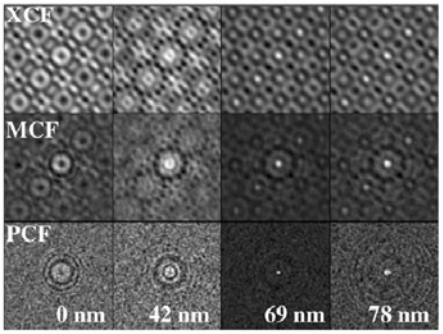


Fig. 2. Comparison of cross- mutual- and phase correlation functions (XCF, MCF and PCF) between two images from the focal series, using the region marked as B in Fig. 1 for different compensated focus differences marked. Due to the periodicity of the crystalline specimen, the XCF peak repeats periodically. The PCF does not show this repetitive pattern, but without phase compensation (left column) the peak is widened into concentric rings due to the focus difference. When the phase compensation for the actual focus difference of 69 nm is applied, the PCF collapses into a single sharp peak.

R.R. Meyer, Ultramicroscopy 92 (2002) 89–109

Correct Alignment

$$\chi^2(\vec{r}) = \sum \left[I_1(\vec{r}) - I_2(\vec{r}) \right]^2$$
$$= \sum I_1(\vec{r})^2 - 2\sum I_1(\vec{r}) I_2(\vec{r}) + \sum I_2(\vec{r})^2$$

Fourier transform edge effects may be removed by computing this this chi^2-difference for only a small portion of the second image, i.e. by moving a window containing a sub-region of the second image across the first image and computing above expression at every position. This can be computed by the following function:

```
//////////////////////////////////////////////////////////////////
image findSmallInBig(image &big,image &small) {

    number Nx,Ny,Nx2,Ny2;
    complexImage chi2Map;

    GetSize(big,Nx,Ny);
    GetSize(small,Nx2,Ny2);

    // create the mask, which has the size of the bigger image and will hold the
    // smaller one, padding the remaining area with zeros:
    Image mask = RealImage("Mask", 4, Nx,Ny);

    // create just the empty mask with ones in the region where it contained the
    // smaller image:
    mask[]           = ExprSize(Nx,Ny,0);
    mask[0,0,Nx2,Ny2] = ExprSize(Nx2,Ny2,1);

    // Compute the sum over the edge around the small image:
    chi2Map = conjugate(RealFFT(mask))*RealFFT(big*big);

    //
    mask[0,0,Nx2,Ny2] = small;
    image chi2 = real(RealIFFT(chi2Map-2*conjugate(RealFFT(mask))*RealFFT(big)));

    // to be exact, one would have to add also the sum over all pixels squared
    // of the smaller image, but this only produces an offset
    return chi2;
}
//////////////////////////////////////////////////////////////////
```