# Digital Image Processing

- The improvement of pictorial information for human interpretation
- The processing of image data for storage, transmission, and autonomous machine perception

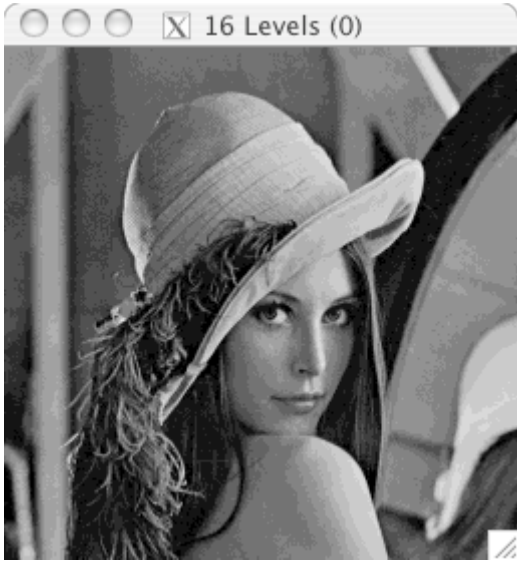
# Digital Image Processing

## Basic Steps

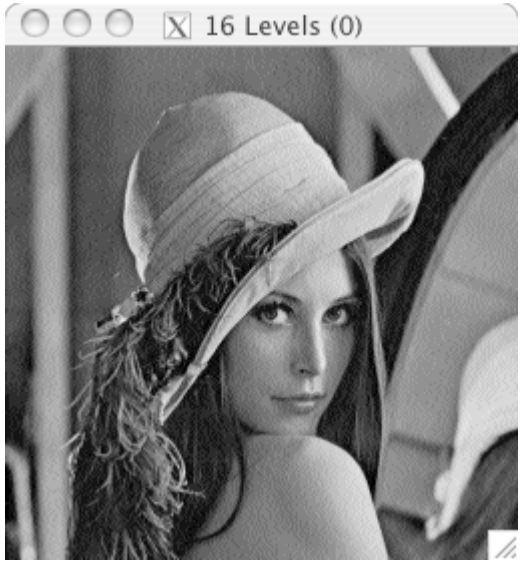
- Image Acquisition
- Image Enhancement
- Image Restoration
- Compression
- Segmentation
- Representation and description
- Recognition

# IGS Quantization


Original Image




Quantization
16 Levels



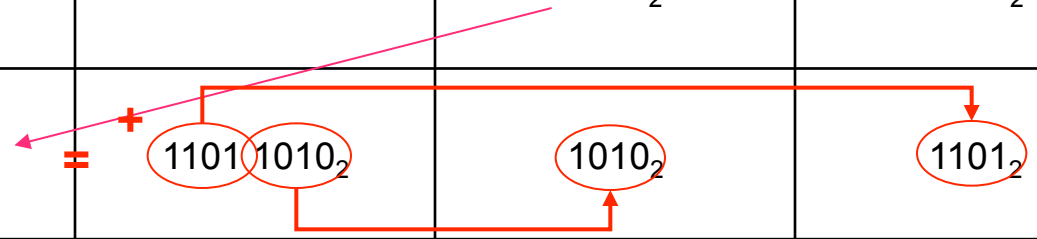

IGS Quantization
16 Levels

The human visual system is very sensitive to edges (contrast)

If we can eliminate edges (or false contours) we can improve the appearance of grey level quantized imagery

# IGS Quantization

| DC (within a row of the image) | Sum (current DC + lower bits of previous sum) | Lower 4 bits (for 16 grey levels) | IGS Code |
|---|---|---|---|
| $1001\ 1100_2$ | | $1100_2$ | $1001_2$ |
| $1100\ 1110_2$ | $1101\ 1010_2$ | $1010_2$ | $1101_2$ |
| $0100\ 0001_2$ | $0100\ 1011_2$ | $1011_2$ | $0100_2$ |
| $1111\ 0010_2$ | $1111\ 0010_2$ | | $1111_2$ |




NOTE: If the upper bits are all set high, the lower bits are not added to this DC

# Imaging Spectrometers

## What are They?

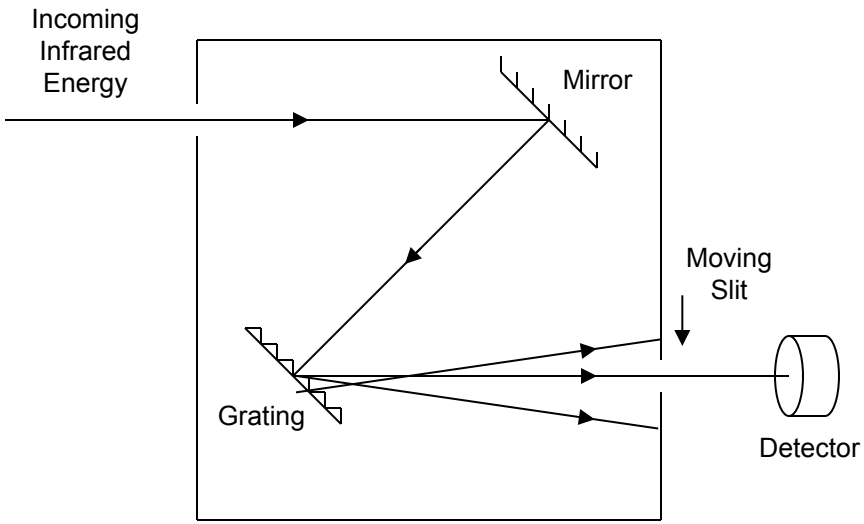
*Spectrometer* - A device that collects incident radiation, separates that radiation into its spectral components, and measures the relative or absolute intensity of these individual spectral components

A device that can accomplish this at many different spatial points simultaneously or nearly simultaneously and record the response at all wavelength values is known as an *imaging spectrometer*

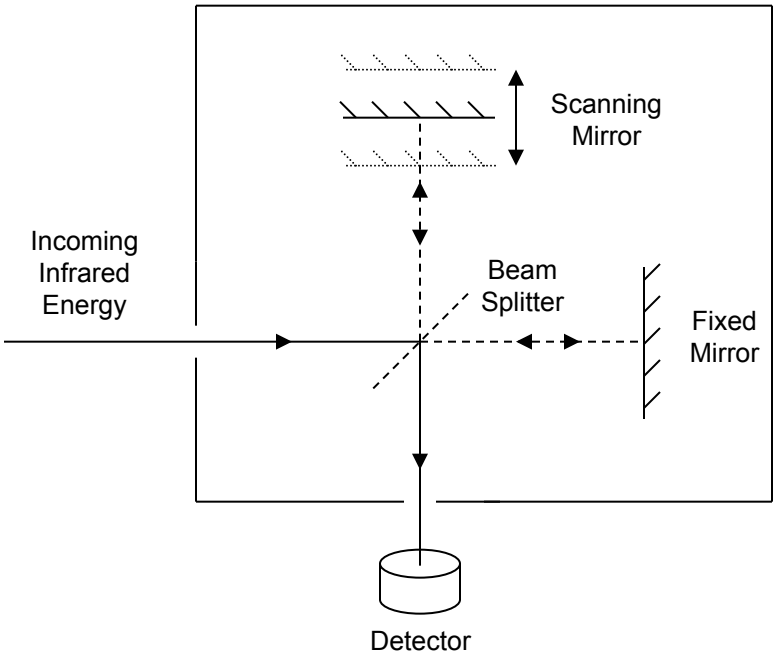
# Imaging Spectrometers

## Designs

Dispersive

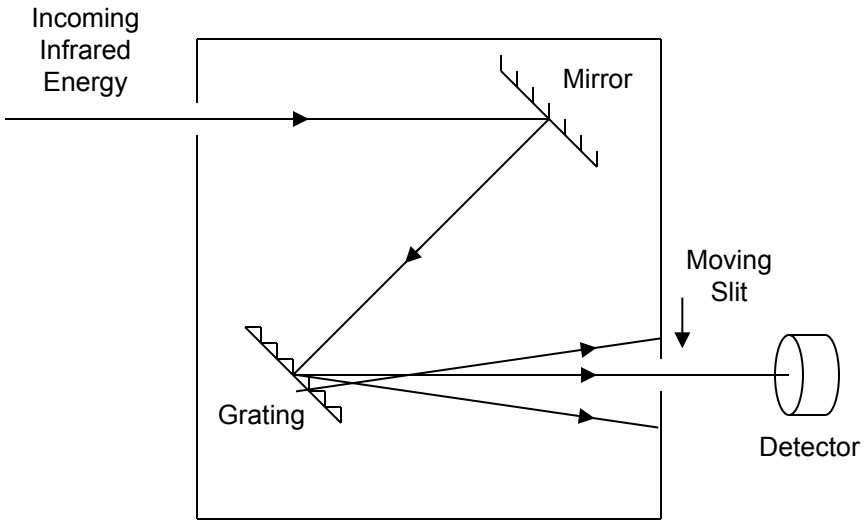


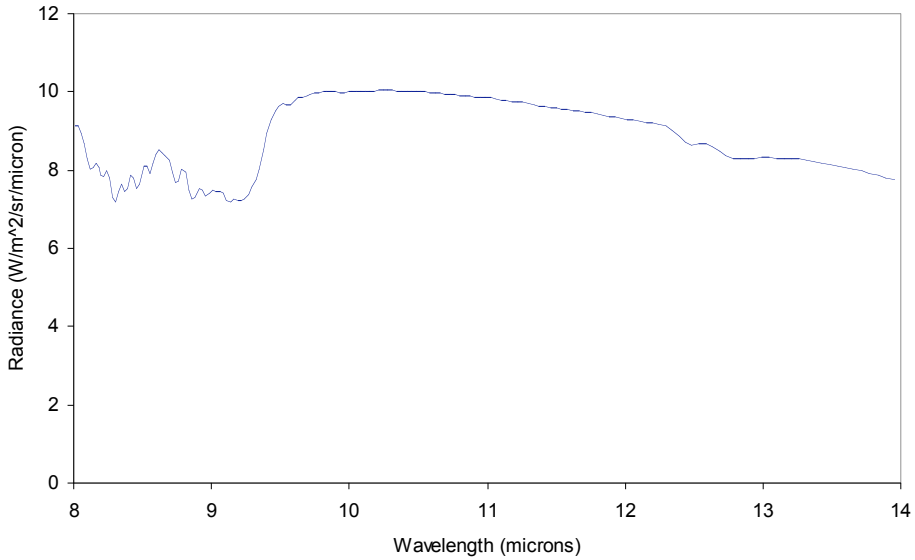

Michelson
Interferometer



# Imaging Spectrometers

## Dispersive Spectrometer Output

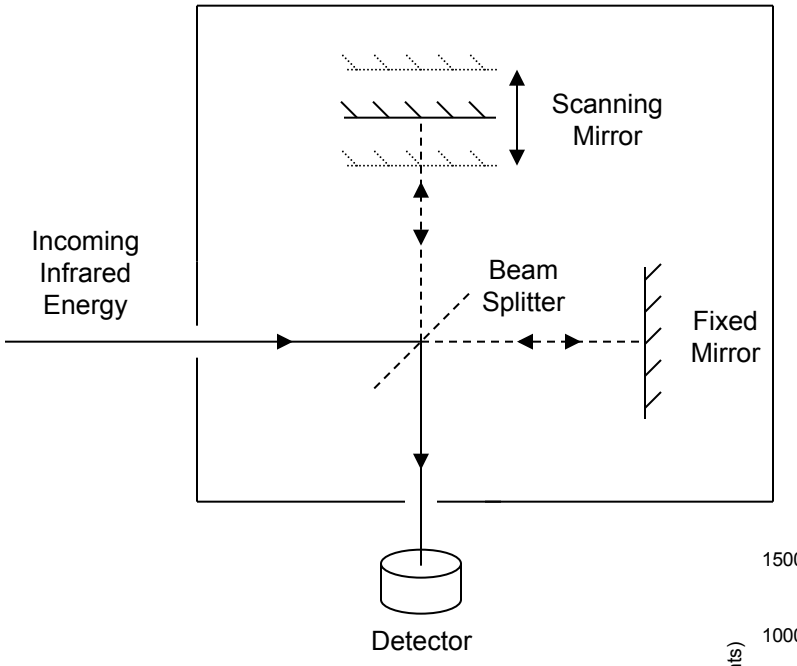


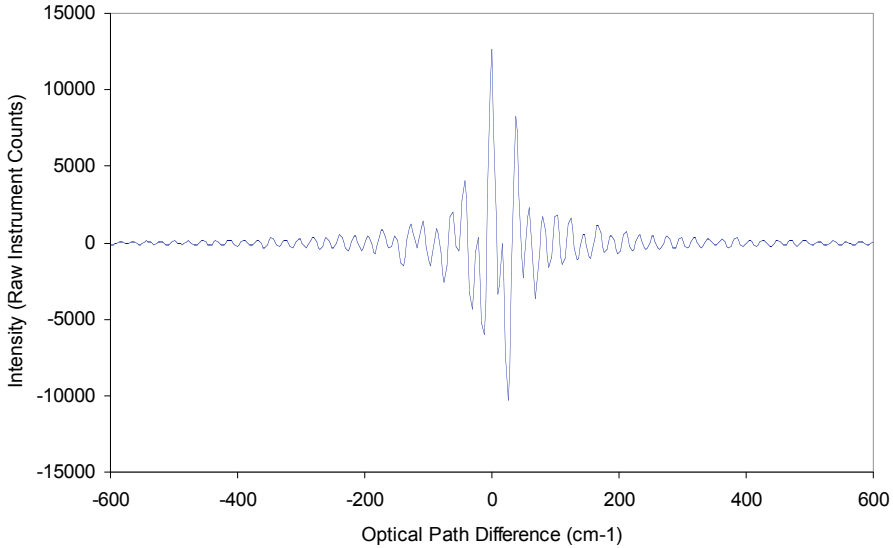




Quartz Sand at 3 $cm^{-1}$resolution

# Imaging Spectrometers

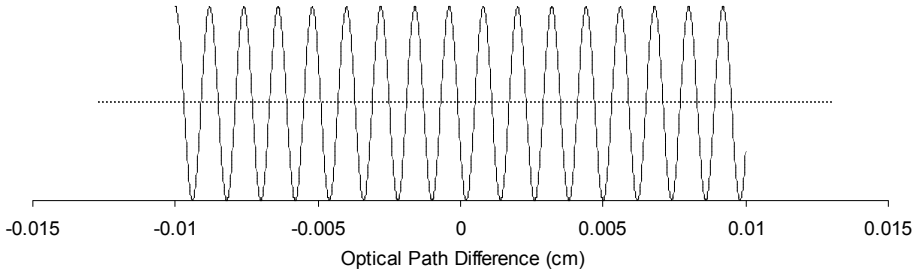
## Michelson Interferometer Output

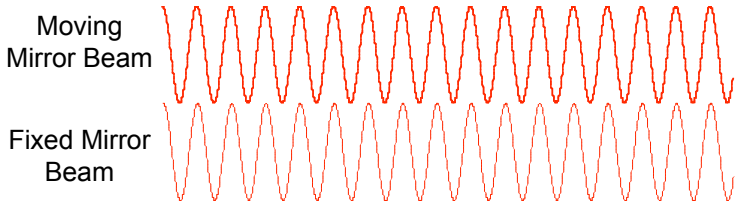


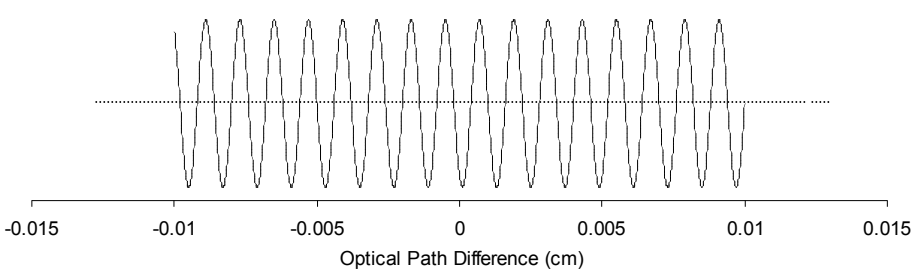
# Imaging Spectrometers

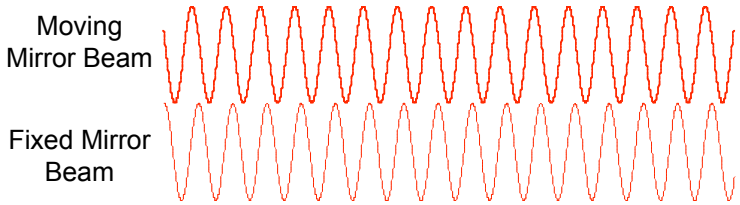
## Single Frequency Modulation



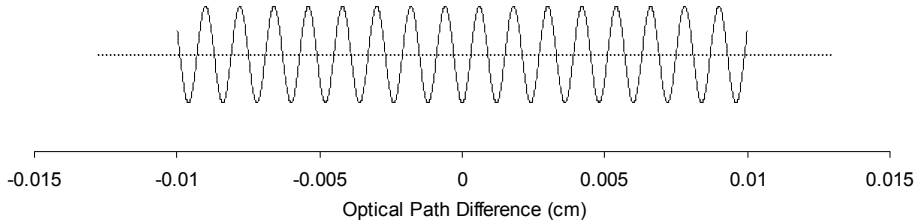



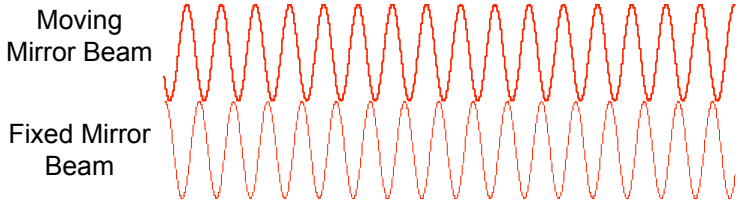

OPD = 0λ



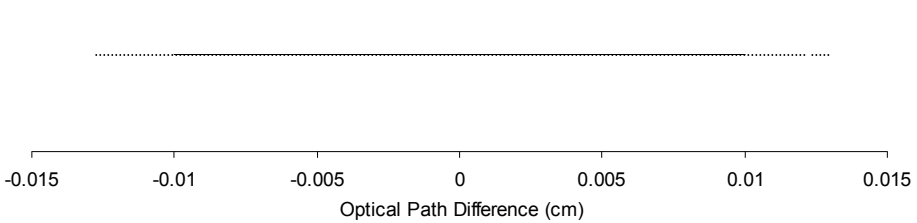



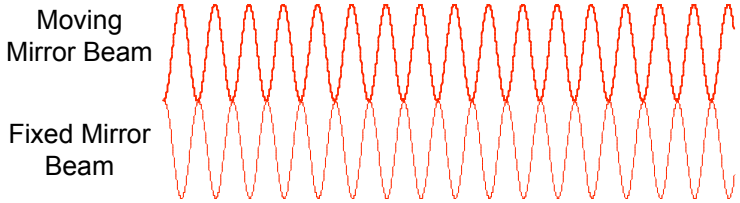

OPD = 1/6λ




OPD = 1/3λ

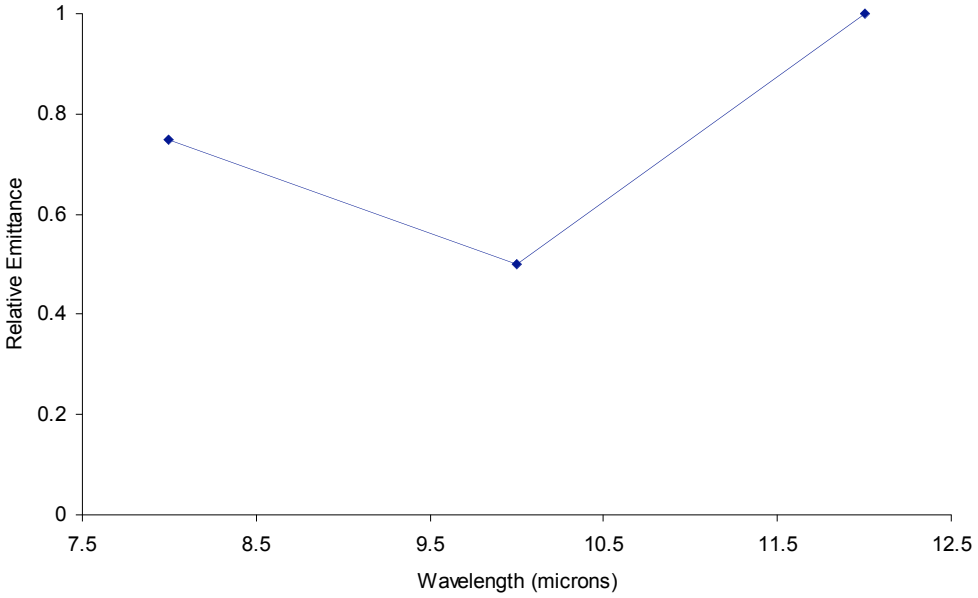


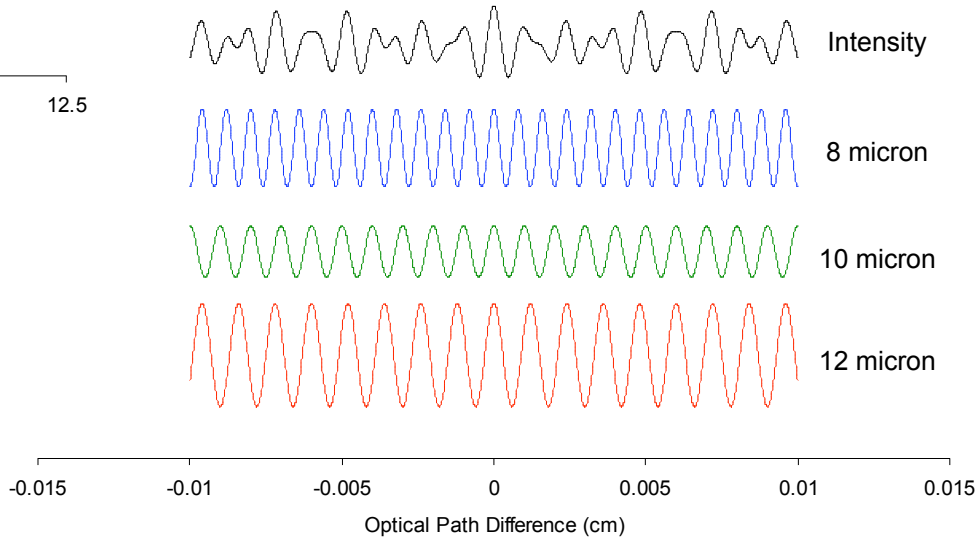




OPD = 1/2λ

# Imaging Spectrometers

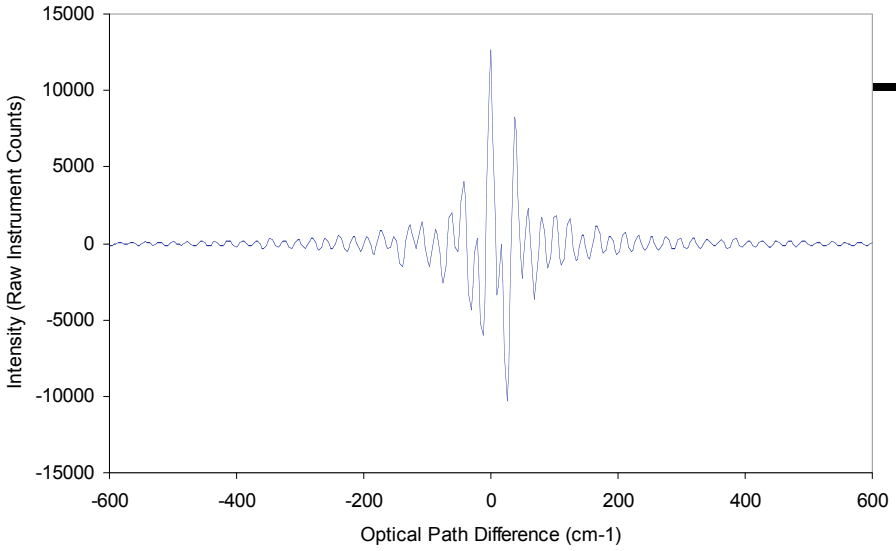
## Multiple Frequency Interferogram



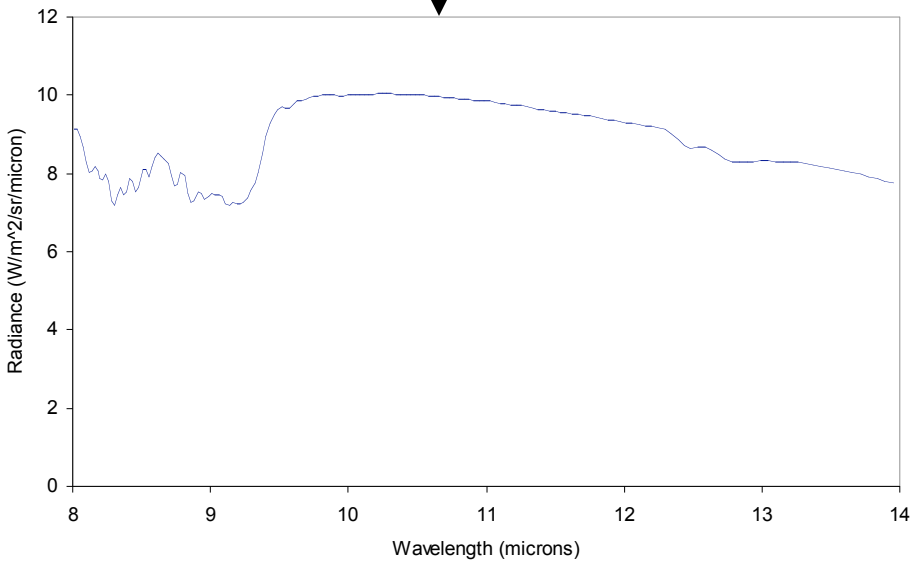



# Imaging Spectrometers

## Computation of Radiance Spectra




Apodization Window
Inverse Fourier Transform

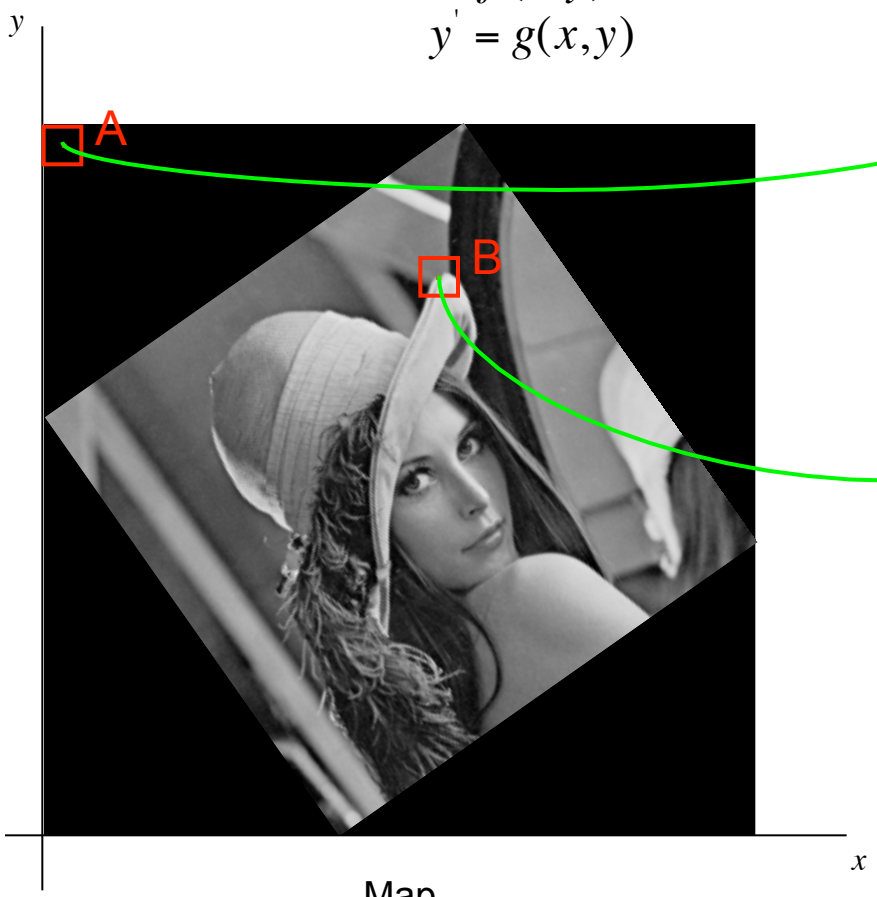


# Geometric Manipulation

## Image Mapping Concept

Mapping Polynomials / Affine Transforms

$$x' = f(x,y)$$
$$y' = g(x,y)$$

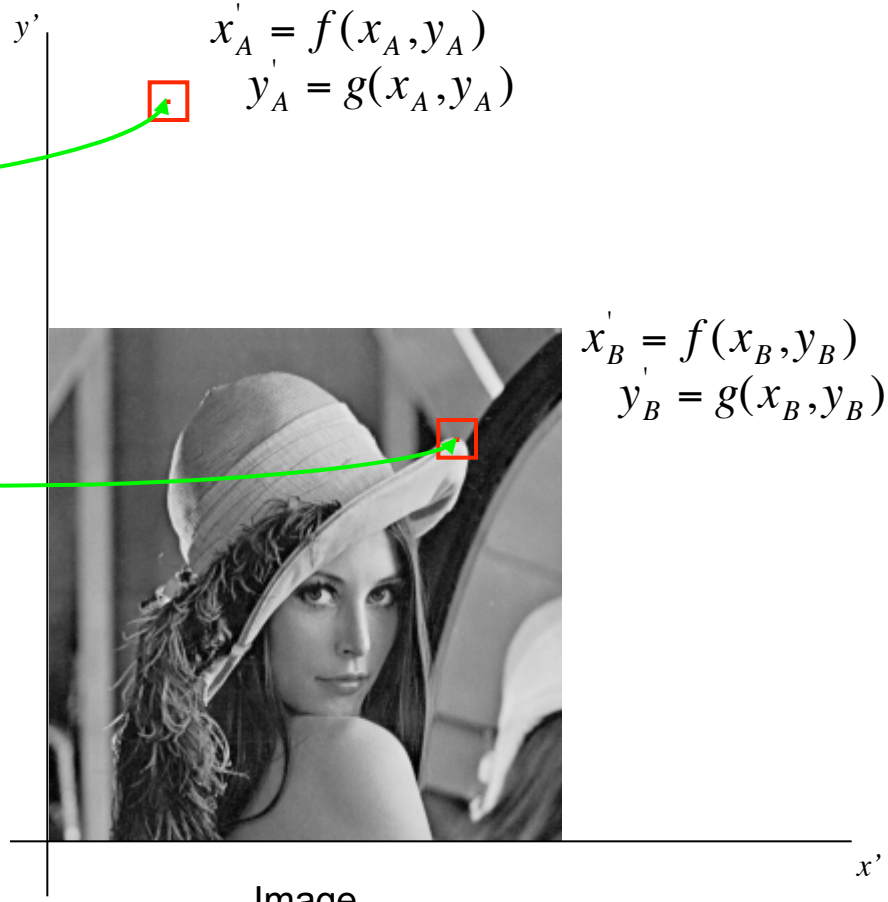



Map

What you want the new image to look like

$$x'_A = f(x_A,y_A)$$
$$y'_A = g(x_A,y_A)$$

$$x'_B = f(x_B,y_B)$$
$$y'_B = g(x_B,y_B)$$

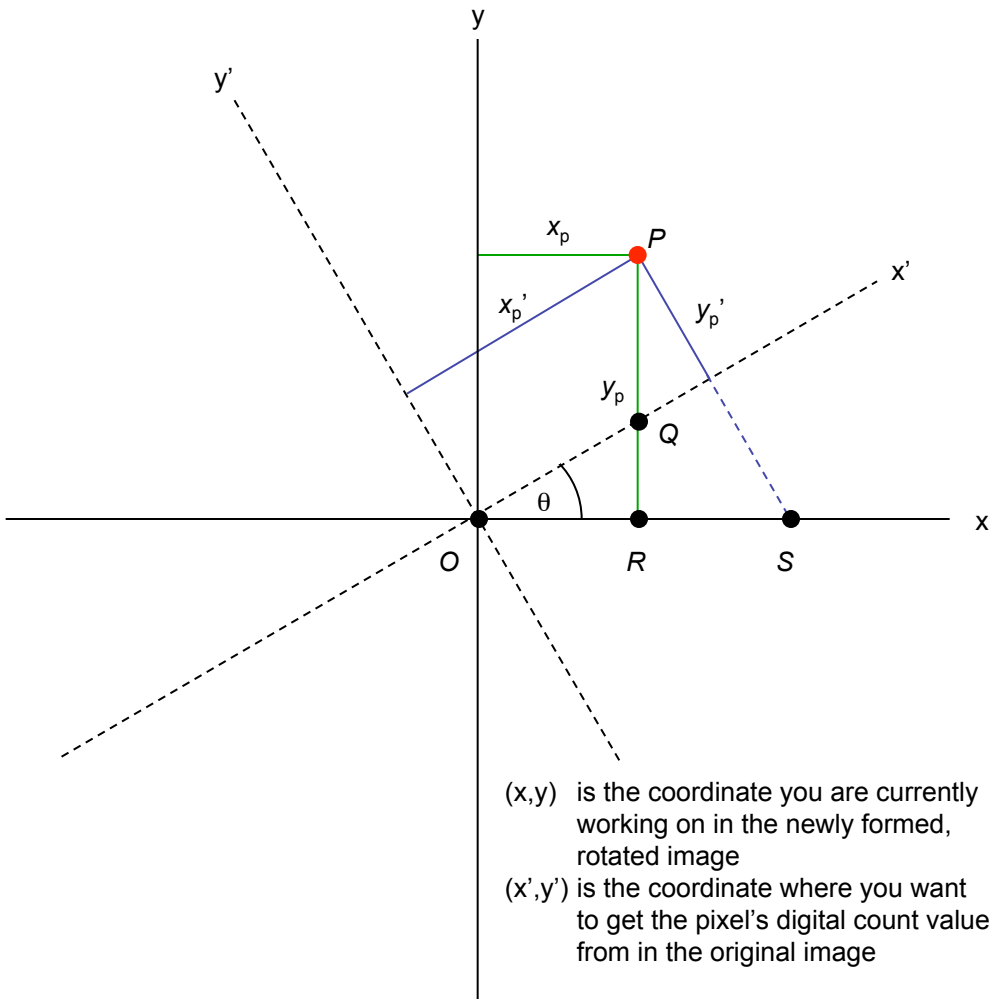



Image

Where you will get the digital count values from

# Geometric Manipulation

## Rotational Mapping Function




(x,y) is the coordinate you are currently working on in the newly formed, rotated image

(x',y') is the coordinate where you want to get the pixel's digital count value from in the original image

$$
\begin{aligned}
x_p' &= \overline{OS}\cos\theta && \text{where } \overline{OS} = x_p + \overline{RS} \\
&= (x_p + \overline{RS})\cos\theta && \text{where } \overline{RS} = y_p \tan\theta \\
&= (x_p + y_p \tan\theta)\cos\theta \\
&= x_p \cos\theta + y_p \tan\theta \cos\theta \\
&= x_p \cos\theta + y_p \sin\theta
\end{aligned}
$$

$$
\begin{aligned}
y_p' &= \overline{PQ}\cos\theta && \text{where } \overline{PQ} = y_p - \overline{QR} \\
&= (y_p - \overline{QR})\cos\theta && \text{where } \overline{QR} = x_p \tan\theta \\
&= (y_p - x_p \tan\theta)\cos\theta \\
&= y_p \cos\theta - x_p \tan\theta \cos\theta \\
&= y_p \cos\theta - x_p \sin\theta
\end{aligned}
$$

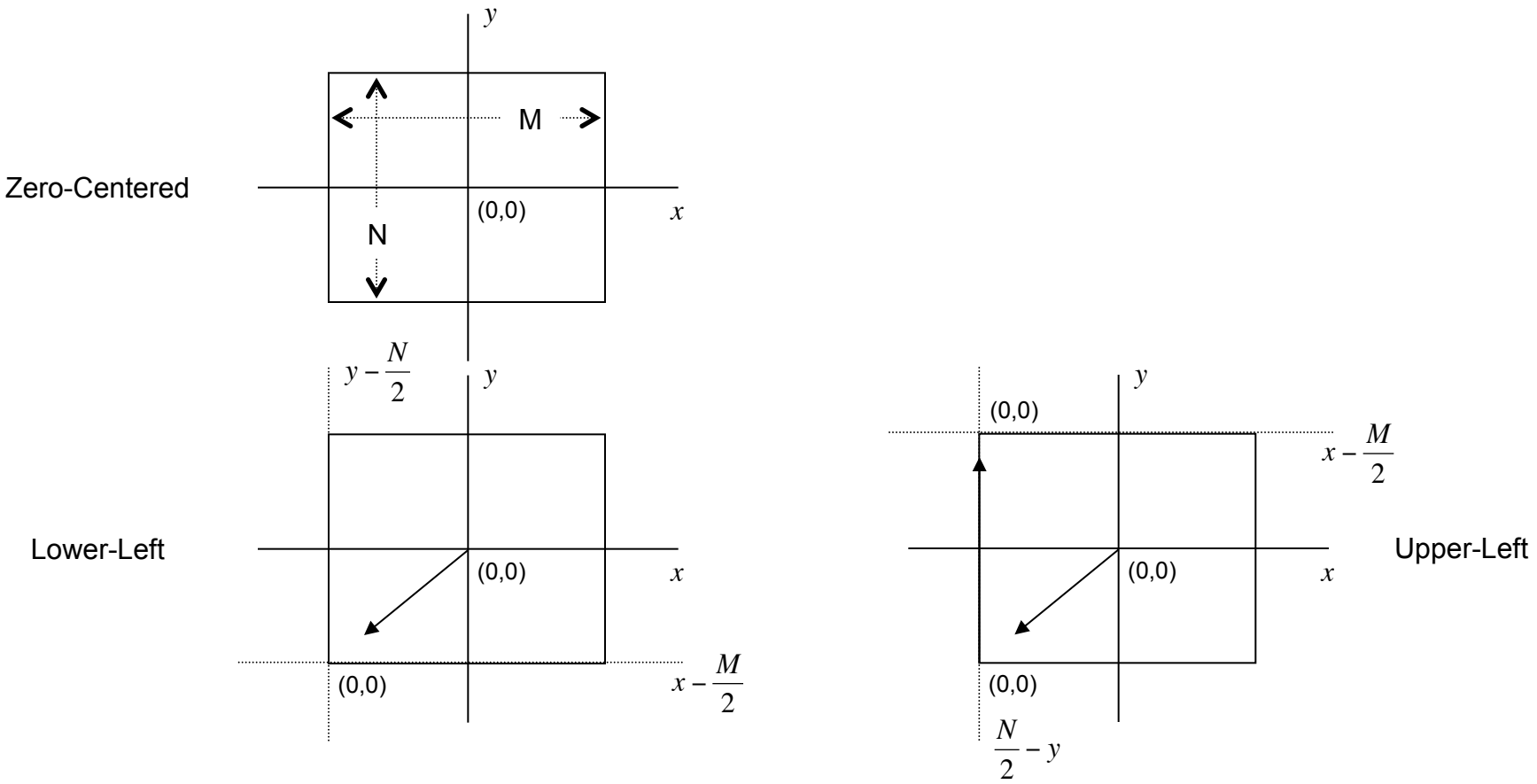
which can be represented in matrix notation as

$$
\begin{bmatrix} x_p' \\ y_p' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_p \\ y_p \end{bmatrix}
$$

# Geometric Manipulation

## Rotational Mapping Function - Implementation

The previous mapping function assumes you have a zero-centered coordinate system. In most computational application, the arrays that hold the image data have the origin in the upper left corner of that array.



# Geometric Manipulation

## Rotational Mapping Function - Implementation

So, in order to create an $M_{column}\,x\,N_{row}$ rotated image from a $K_{column}\,x\,L_{row}$ original image, you must translate the coordinates both before and after the rotation has occurred

$$\begin{bmatrix} x_p' - \frac{K}{2} \\ \frac{L}{2} - y_p' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_p - \frac{M}{2} \\ \frac{N}{2} - y_p \end{bmatrix}$$

$$\left(x_p' - \frac{K}{2}\right) = \left(x_p - \frac{M}{2}\right)\cos\theta + \left(\frac{N}{2} - y_p\right)\sin\theta$$

$$x_p' = x_p\cos\theta - y_p\sin\theta - \frac{M}{2}\cos\theta + \frac{N}{2}\sin\theta + \frac{K}{2}$$

$$\left(\frac{L}{2} - y_p'\right) = -\left(x_p - \frac{M}{2}\right)\sin\theta + \left(\frac{N}{2} - y_p\right)\cos\theta$$

$$-y_p' = -y_p\cos\theta - x_p\sin\theta + \frac{M}{2}\sin\theta + \frac{N}{2}\cos\theta - \frac{L}{2}$$

$$y_p' = y_p\cos\theta + x_p\sin\theta - \frac{M}{2}\sin\theta - \frac{N}{2}\cos\theta + \frac{L}{2}$$

For computational ease

$$\hat{x}_p = x_p - \frac{M}{2}$$

$$\hat{y}_p = \frac{N}{2} - y_p$$

$$\begin{bmatrix} \hat{x}_p' \\ \hat{y}_p' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} \hat{x}_p \\ \hat{y}_p \end{bmatrix}$$

$$\hat{x}_p' = x_p' - \frac{K}{2}$$

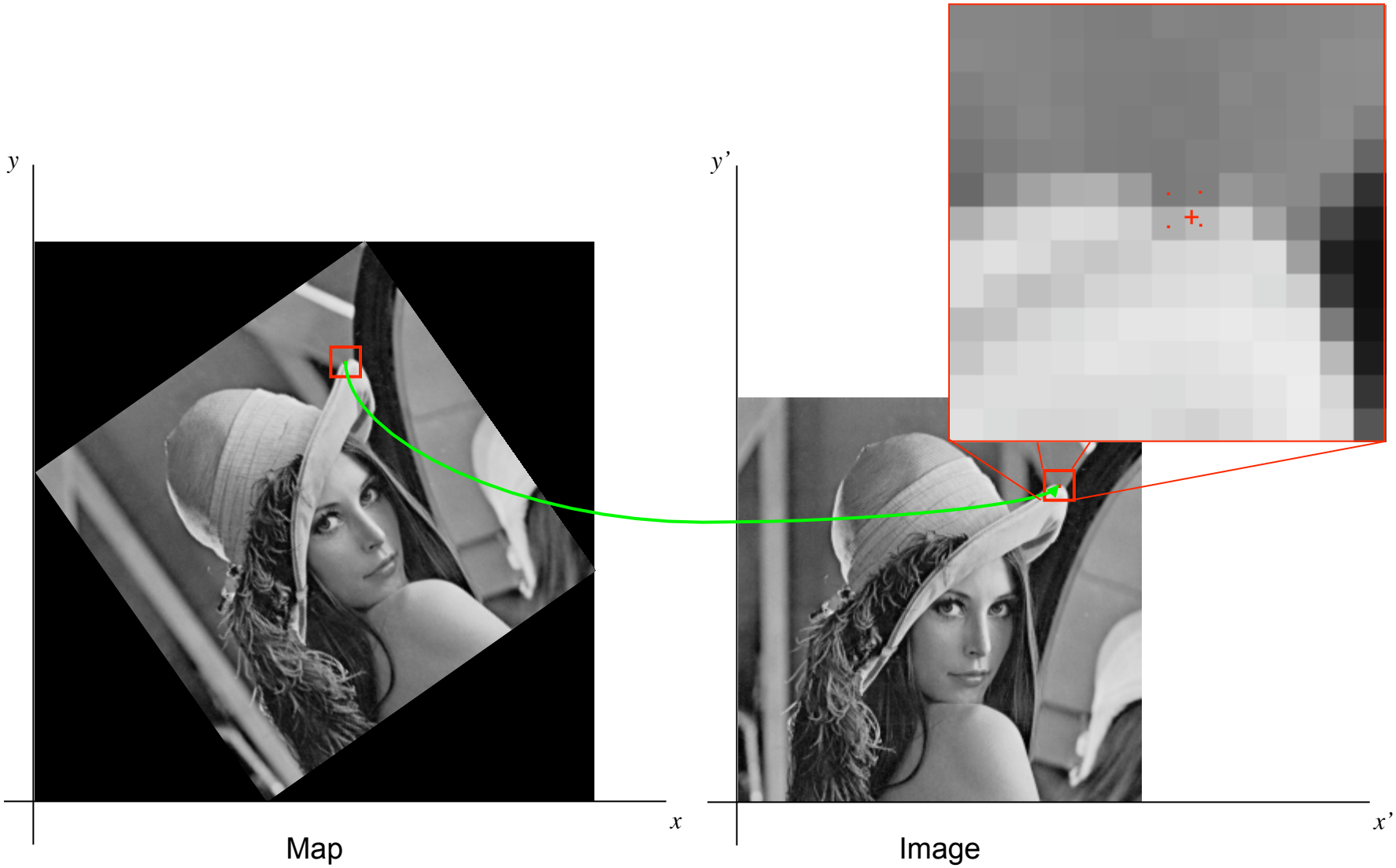
$$x_p' = \hat{x}_p' + \frac{K}{2}$$

$$\hat{y}_p' = \frac{L}{2} - y_p'$$

$$y_p' = \frac{L}{2} - \hat{y}_p'$$

# Geometric Manipulation

## Which Digital Count Do You Choose?



# Geometric Manipulation

## Resampling Choices

Choose the closest pixel's digital count value

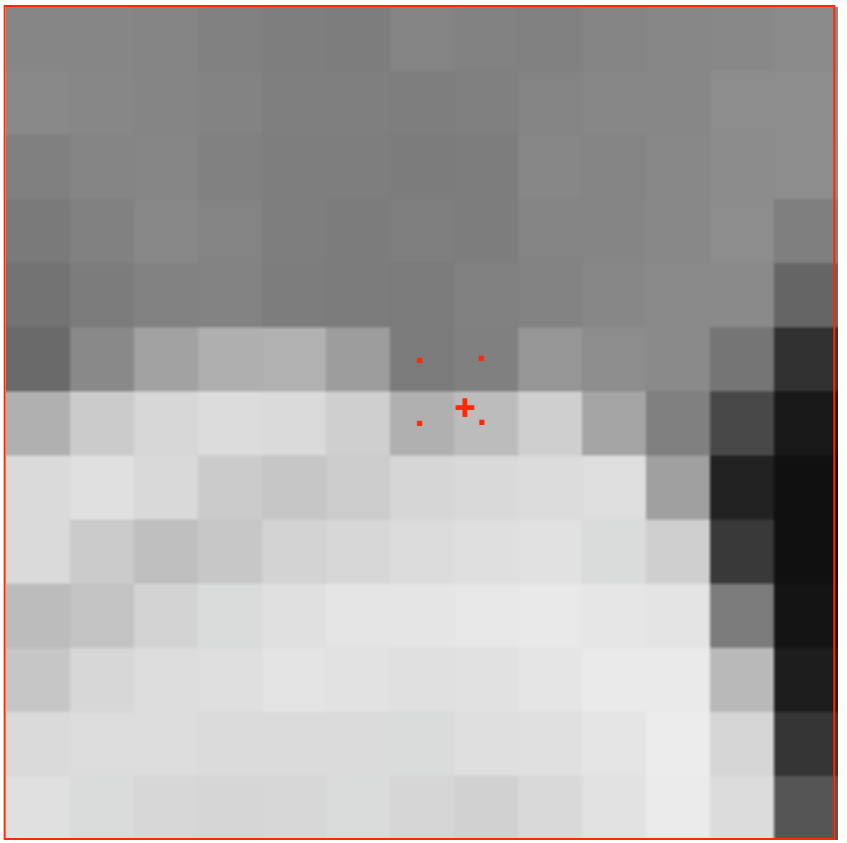
- Nearest Neighbor Resampling

Compute a "weighted average" of the digital count values of the four surrounding pixels

- Bilinear Interpolation Resampling

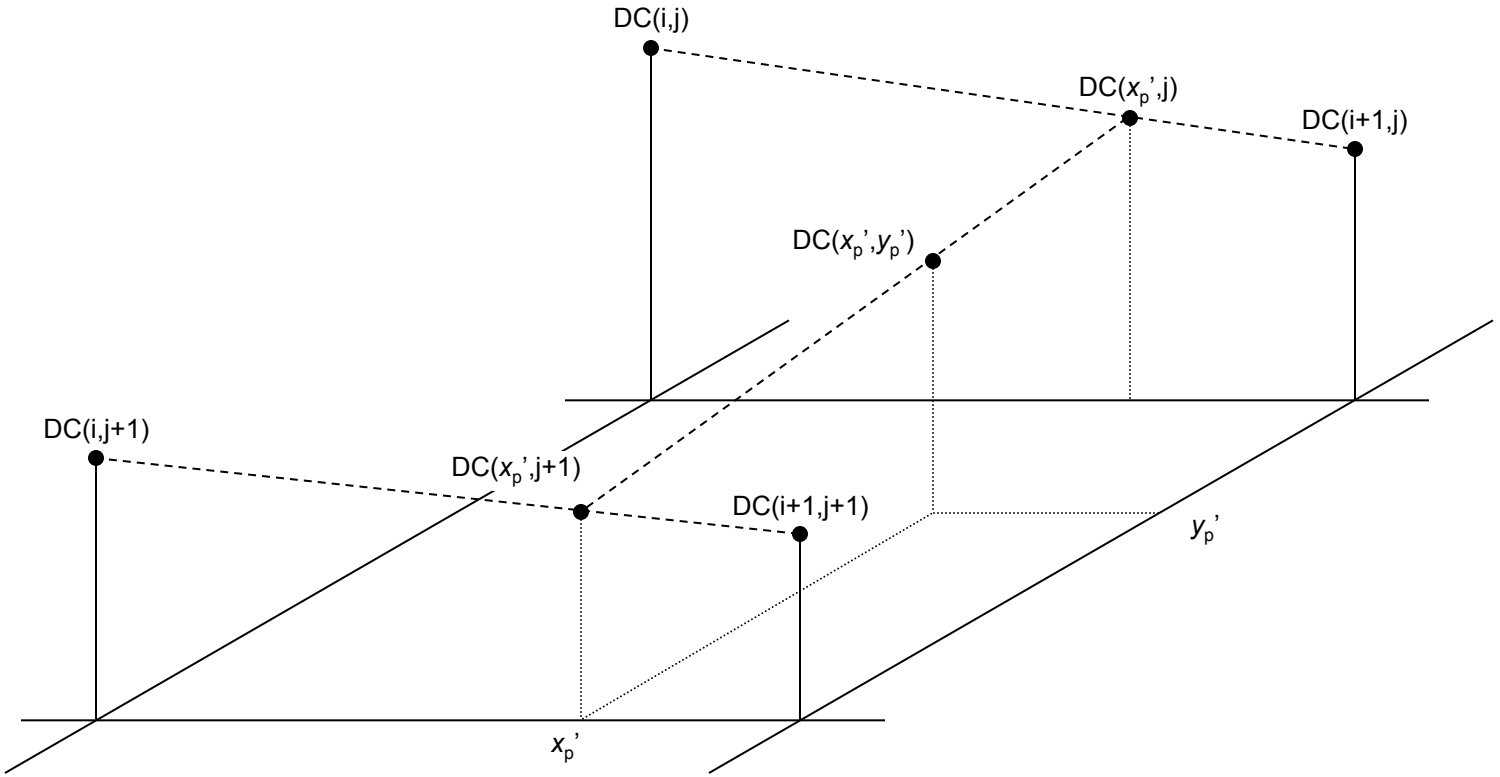
Look at the sixteen closest neighbors and fit a cubic polynomial through an intersecting line (sharpening while we're at it)

- Cubic Convolution Resampling



# Geometric Manipulation

## Bilinear Interpolation Resampling Function



# Geometric Manipulation

## Bilinear Interpolation Resampling Function

$$\frac{DC(i+1,j)-DC(i,j)}{(i+1)-i}=\frac{DC(x_p',j)-DC(i,j)}{x_p'-i}$$

$$DC(i+1,j)-DC(i,j)=\frac{DC(x_p',j)-DC(i,j)}{x_p'-i}$$

$$DC(x_p',j)-DC(i,j)=\left(DC(i+1,j)-DC(i,j)\right)\left(x_p'-i\right)$$

$$\boxed{DC(x_p',j)=\left(DC(i+1,j)-DC(i,j)\right)\left(x_p'-i\right)+DC(i,j)}$$

$$\frac{DC(i+1,j+1)-DC(i,j+1)}{(i+1)-i}=\frac{DC(x_p',j+1)-DC(i,j+1)}{x_p'-i}$$

$$DC(i+1,j+1)-DC(i,j+1)=\frac{DC(x_p',j+1)-DC(i,j+1)}{x_p'-i}$$

$$DC(x_p',j+1)-DC(i,j+1)=\left(DC(i+1,j+1)-DC(i,j+1)\right)\left(x_p'-i\right)$$

$$\boxed{DC(x_p',j+1)=\left(DC(i+1,j+1)-DC(i,j+1)\right)\left(x_p'-i\right)+DC(i,j+1)}$$

$$\frac{DC(x_p',j+1)-DC(x_p',j)}{(j+1)-j}=\frac{DC(x_p',y_p')-DC(x_p',j)}{y_p'-j}$$

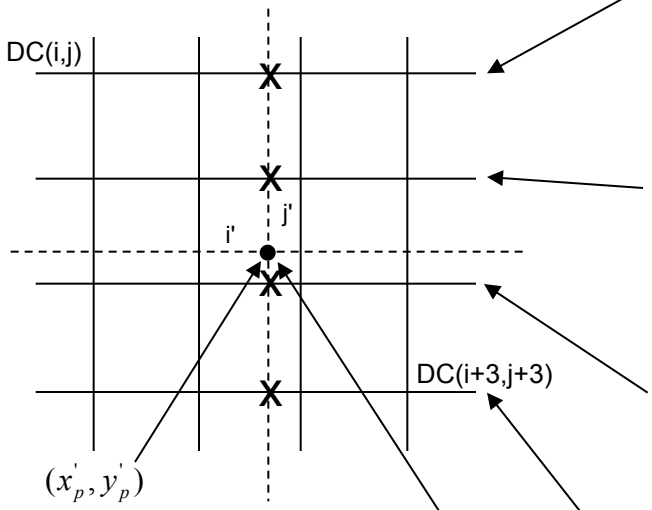
$$DC(x_p',j+1)-DC(x_p',j)=\frac{DC(x_p',y_p')-DC(x_p',j)}{y_p'-j}$$

$$DC(x_p',y_p')-DC(x_p',j)=\left(DC(x_p',j+1)-DC(x_p',j)\right)\left(y_p'-j\right)$$

$$\boxed{DC(x_p',y_p')=\left(DC(x_p',j+1)-DC(x_p',j)\right)\left(y_p'-j\right)+DC(x_p',j)}$$

# Geometric Manipulation

## Cubic Interpolation Resampling Function




$$i = FIX(x_p')$$
$$j = FIX(y_p')$$
$$i' = x_p' - i$$
$$j' = y_p' - j$$

$$DC(i, j') = j'\{j'(j'[DC(i, j+3) - DC(i, j+2) + DC(i, j+1) - DC(i, j)] + [DC(i, j+2) - DC(i, j+3) - 2DC(i, j+1) + 2DC(i, j)]) + [DC(i, j+2) - DC(i, j)]\} + DC(i, j+1)$$

$$DC(i+1, j') = j'\{j'(j'[DC(i+1, j+3) - DC(i+1, j+2) + DC(i+1, j+1) - DC(i+1, j)] + [DC(i+1, j+2) - DC(i+1, j+3) - 2DC(i+1, j+1) + 2DC(i+1, j)]) + [DC(i+1, j+2) - DC(i+1, j)]\} + DC(i+1, j+1)$$

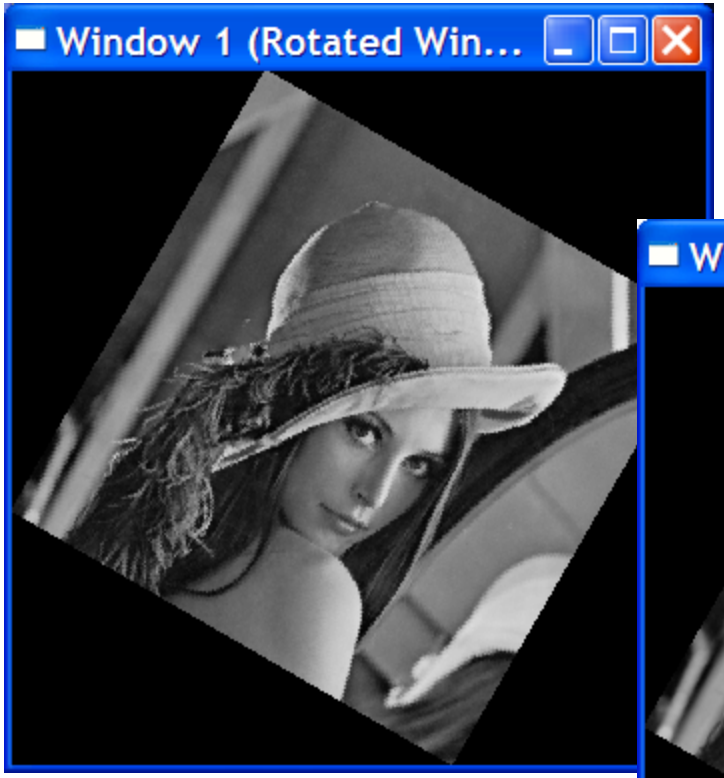
$$DC(i+2, j') = j'\{j'(j'[DC(i+2, j+3) - DC(i+2, j+2) + DC(i+2, j+1) - DC(i+2, j)] + [DC(i+2, j+2) - DC(i+2, j+3) - 2DC(i+2, j+1) + 2DC(i+2, j)]) + [DC(i+2, j+2) - DC(i+2, j)]\} + DC(i+2, j+1)$$

$$DC(i+3, j') = j'\{j'(j'[DC(i+3, j+3) - DC(i+3, j+2) + DC(i+3, j+1) - DC(i+3, j)] + [DC(i+3, j+2) - DC(i+3, j+3) - 2DC(i+3, j+1) + 2DC(i+3, j)]) + [DC(i+3, j+2) - DC(i+3, j)]\} + DC(i+3, j+1)$$

$$DC(i', j') = i'\{i'(i'[DC(i+3, j') - DC(i+2, j') + DC(i+1, j') - DC(i, j')] + [DC(i+2, j') - DC(i+3, j') - 2DC(i+1, j') + 2DC(i, j')]) + [DC(i+2, j') - DC(i, j')]\} + DC(i+1, j')$$

# Geometric Manipulation

## Rotational Mapping Function - Example

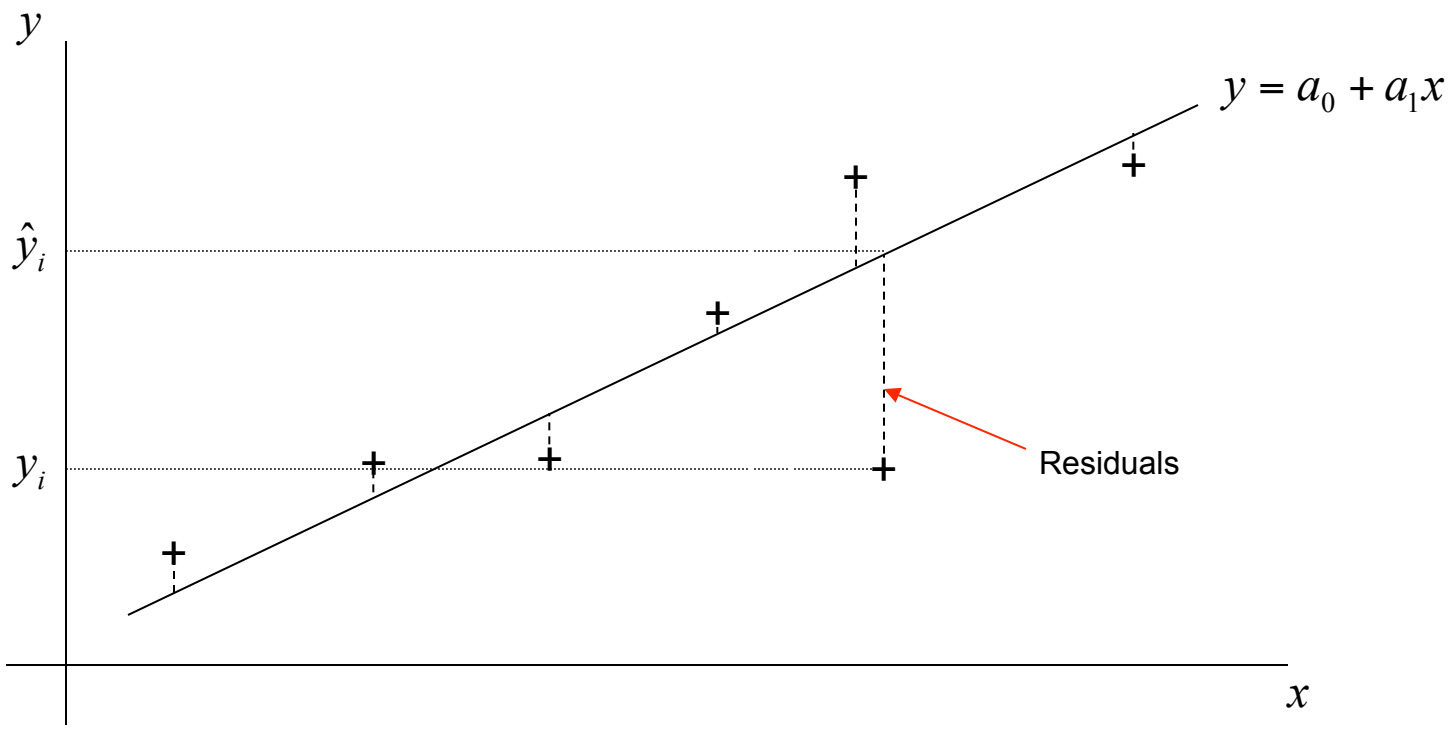




Rotation 30° CW Mapping
Nearest Neighbor Interpolation


Rotation 30° CW Mapping
Bilinear Interpolation

# Geometric Manipulation

## Simple Linear Least Squares Regression




The goal of simple linear least squares regression is to find the line which fits your data that minimizes the sum of the squared residuals

$$\sum_{i->\text{all points}} (y_i - \hat{y}_i)^2$$

# Geometric Manipulation

## Computation of Simple Linear Least Squares Coefficients

$$y = a_0 + a_1 x$$

$$\vec{a} = (\vec{X}^t \vec{X})^{-1} \vec{X}^t \vec{Y}$$

*where*

$$\vec{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \\ 1 & x_n \end{bmatrix} \qquad \vec{Y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

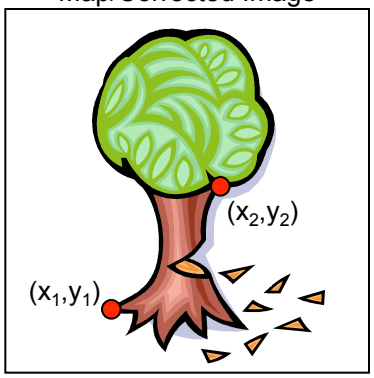
Simple implies a single independent variable

# Geometric Manipulation

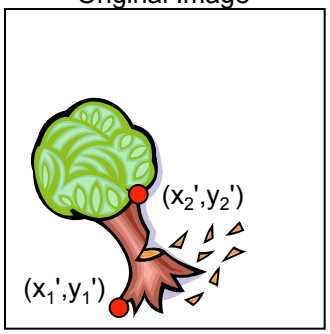
## Ground Control Points

When you can not define a mathematical model for the sensor distortion

Map/Corrected Image




Original Image




Map/Corrected Image Coordinates

| $x$ | $y$ |
|---|---|
| $x_1$ | $y_1$ |
| $x_2$ | $y_2$ |
| $x_3$ | $y_3$ |
| $\vdots$ | $\vdots$ |
| $x_n$ | $y_n$ |

Original Image Coordinates

| $x'$ | $y'$ |
|---|---|
| $x_1'$ | $y_1'$ |
| $x_2'$ | $y_2'$ |
| $x_3'$ | $y_3'$ |
| $\vdots$ | $\vdots$ |
| $x_n'$ | $y_n'$ |

# Geometric Manipulation

## Computation of Mapping Functions

$$x' = f(x,y)$$
$$y' = g(x,y)$$

*Mapping Functions*

$$x' = a_0 + a_1 x + a_2 y + a_3 xy + a_4 x^2 + a_5 y^2$$
$$y' = b_0 + b_1 x + b_2 y + b_3 xy + b_4 x^2 + b_5 y^2$$

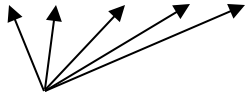
*Example Second Order Polynomial Mapping Function*
*(will account for most scale, rotation, and perspective distortions)*

$$\vec{a} = (\vec{X}^t \vec{X})^{-1} \vec{X}^t \vec{Y}_{x'}$$
$$\vec{b} = (\vec{X}^t \vec{X})^{-1} \vec{X}^t \vec{Y}_{y'}$$

*Least Squares Solution for Mapping Function Coefficients*

*where*

$$\vec{X} = \begin{bmatrix} 1 & x_1 & y_1 & x_1 y_1 & x_1^2 & y_1^2 \\ 1 & x_2 & y_2 & x_2 y_2 & x_2^2 & y_2^2 \\ 1 & x_3 & y_3 & x_3 y_3 & x_3^2 & y_3^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & y_n & x_n y_n & x_n^2 & y_n^2 \end{bmatrix} \quad \vec{Y}_{x'} = \begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ \vdots \\ x_n' \end{bmatrix} \quad \vec{Y}_{y'} = \begin{bmatrix} y_1' \\ y_2' \\ y_3' \\ \vdots \\ y_n' \end{bmatrix}$$



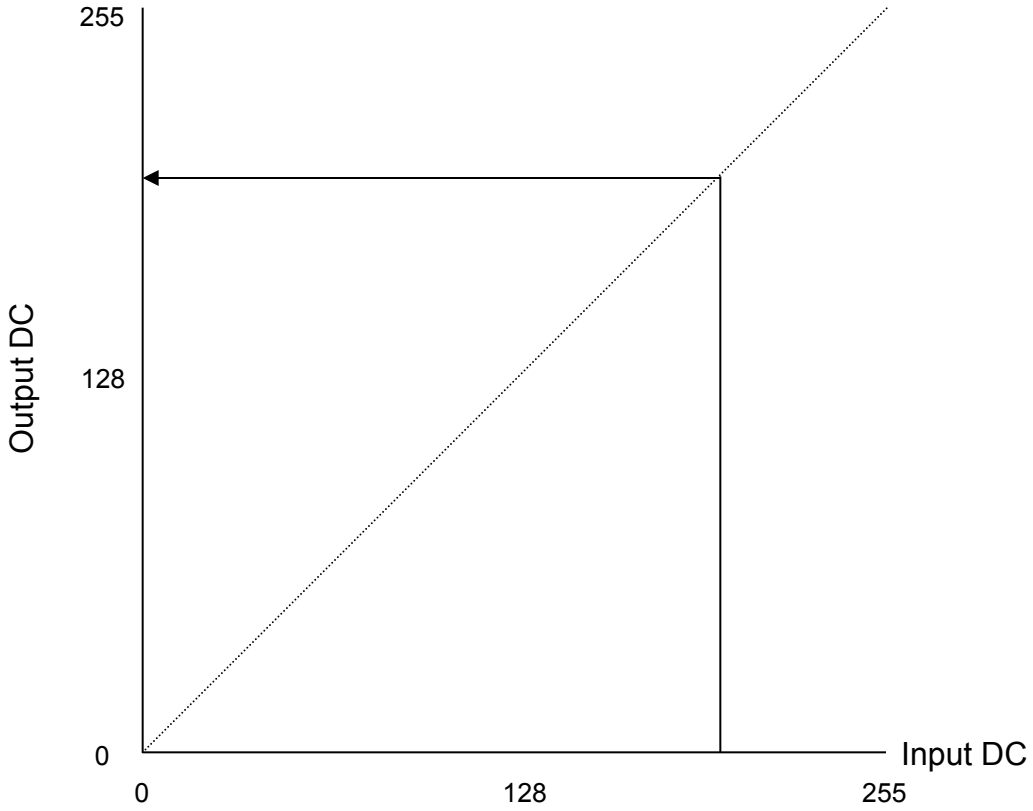
Multiple linear least squares regression involves more than one independent variable

(x,y) is the coordinate you are currently working on in the newly formed, geometrically corrected image

(x',y') is the coordinate where you want to get the pixel's digital count value from in the original image

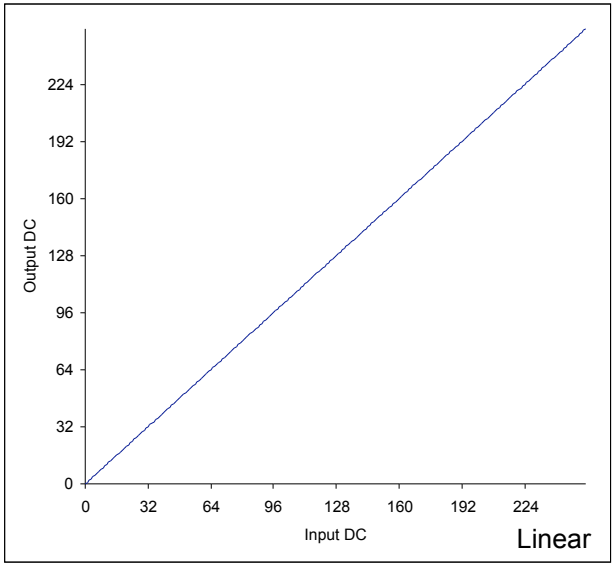
# Image Enhancement

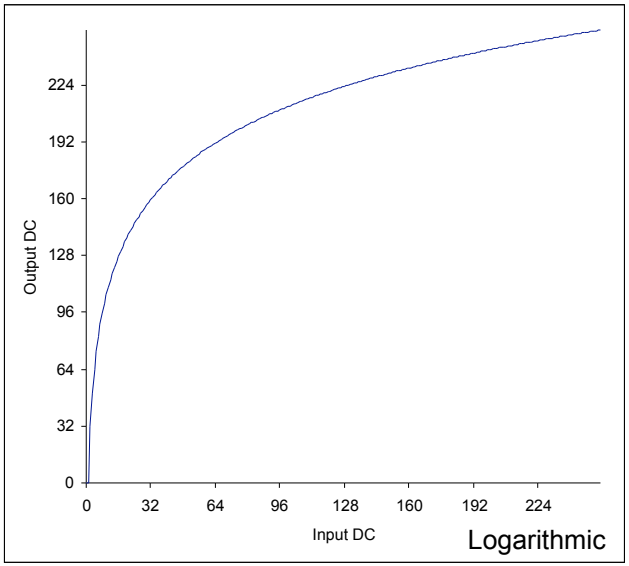
## Lookup Table (LUT) Concept

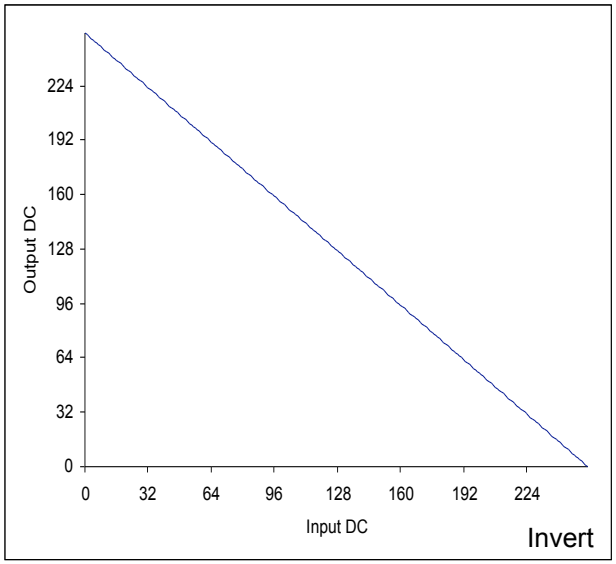


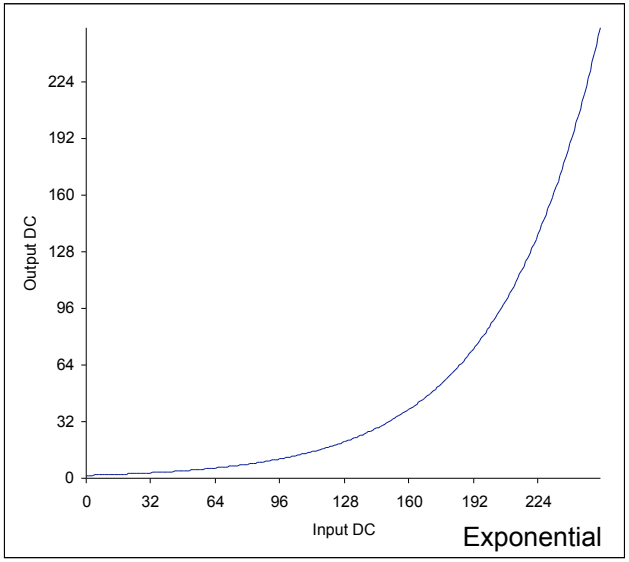
# Image Enhancement

## Lookup Table (LUT) Examples



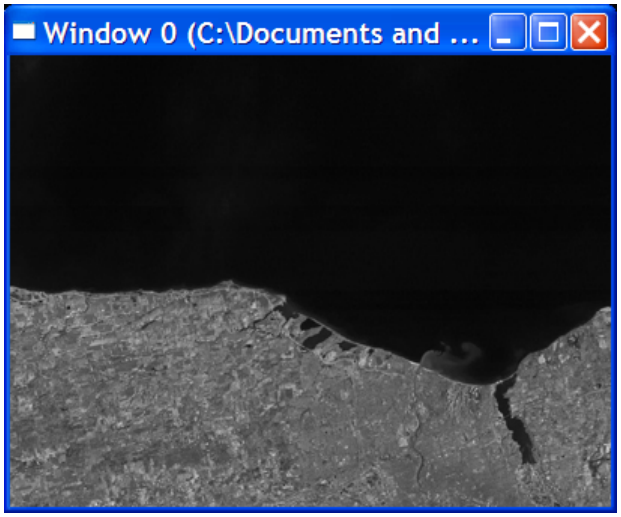



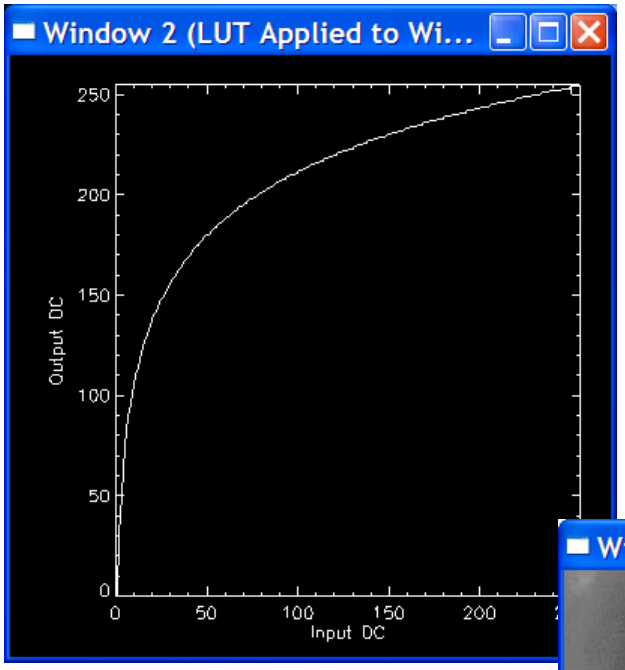



# Image Enhancement

## Lookup Table (LUT) Examples



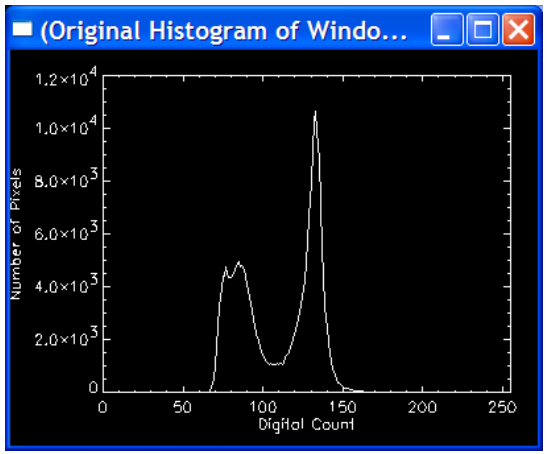

# Image Histogram

The histogram of a digital image with grey levels in the range $[0,L-1]$ is a discrete function

$$h(r_k) = n_k$$

where $r_k$ is the $k^{th}$ grey level and $n_k$ is the number of pixels in the image having that grey level. Commonly, the histogram is normalized by the total number of pixels in the image, $n$, yielding an estimate of the probability of occurrence of each grey level $r_k$.


$$p(r_k) = \frac{n_k}{n}$$



# Image Enhancement

## Histogram Equalization

output digital count ↘ $s = T(r)$ ↙ input digital count $\quad 0 \le r \le 1$

where $r$ represents the gray levels of the image to be enhanced, and
$T(r)$ represents the enhancement function satisfying the assumptions that

a) $T(r)$ is a single-valued and monotonically increasing function in the interval $0 \le r \le 1$ and
b) $0 \le T(r) \le 1$ for $0 \le r \le 1$

guarantees that the inverse transformation exists

preserves the increasing order from black to white

the input and output gray levels will be in the same range

$$r = T^{-1}(s) \qquad 0 \le s \le 1$$

Viewing the gray levels in an image as a random variable in the interval *[0,1]*, the most fundamental descriptor of that random variable is the probability density function (PDF). If we let $p_r(r)$ and $p_s(s)$ denote the PDF of random variables $r$ and $s$, then elementary probability theory states that if $p_r(r)$ and $T(r)$ are known and $T^{-1}(s)$ satisfies a) above, then the PDF $p_s(s)$ of the transformed variable $s$ is

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right|$$

# Image Enhancement

## Histogram Equalization

$$s = T(r) = \int_0^r p_r(w)dw$$

Cumulative Distribution Function

(note that this function satisfies all the conditions set forth previously)

$$\frac{ds}{dr} = \frac{dT(r)}{dr}$$

$$= \frac{d}{dr}\left[\int_0^r p_r(w)dw\right]$$

$$= p_r(r)$$

Leibniz's rule states that the derivative of a definite integral with respect to it upper limit is simply the integrand evaluated at that upper limit

substituting into the equation on the previous page

$$p_s(s) = p_r(r)\left|\frac{dr}{ds}\right|$$

$$= p_r(r)\left|\frac{1}{p_r(r)}\right|$$

$$= 1 \qquad 0 \le s \le 1, \text{ and } 0 \text{ elsewhere}$$

the *uniform* probability density function

# Image Enhancement

## Histogram Equalization

$$p_r(r_k) = \frac{n_k}{n} \qquad k = 0, 1, 2, ..., \mathrm{L} - 1$$

represents the probability of occurrence of the discrete gray level $r_k$ where $n$ is the total number of pixels in the image, $n_k$ is the number of pixels that have the gray level $r_k$, and $L$ is the number of possible gray levels.

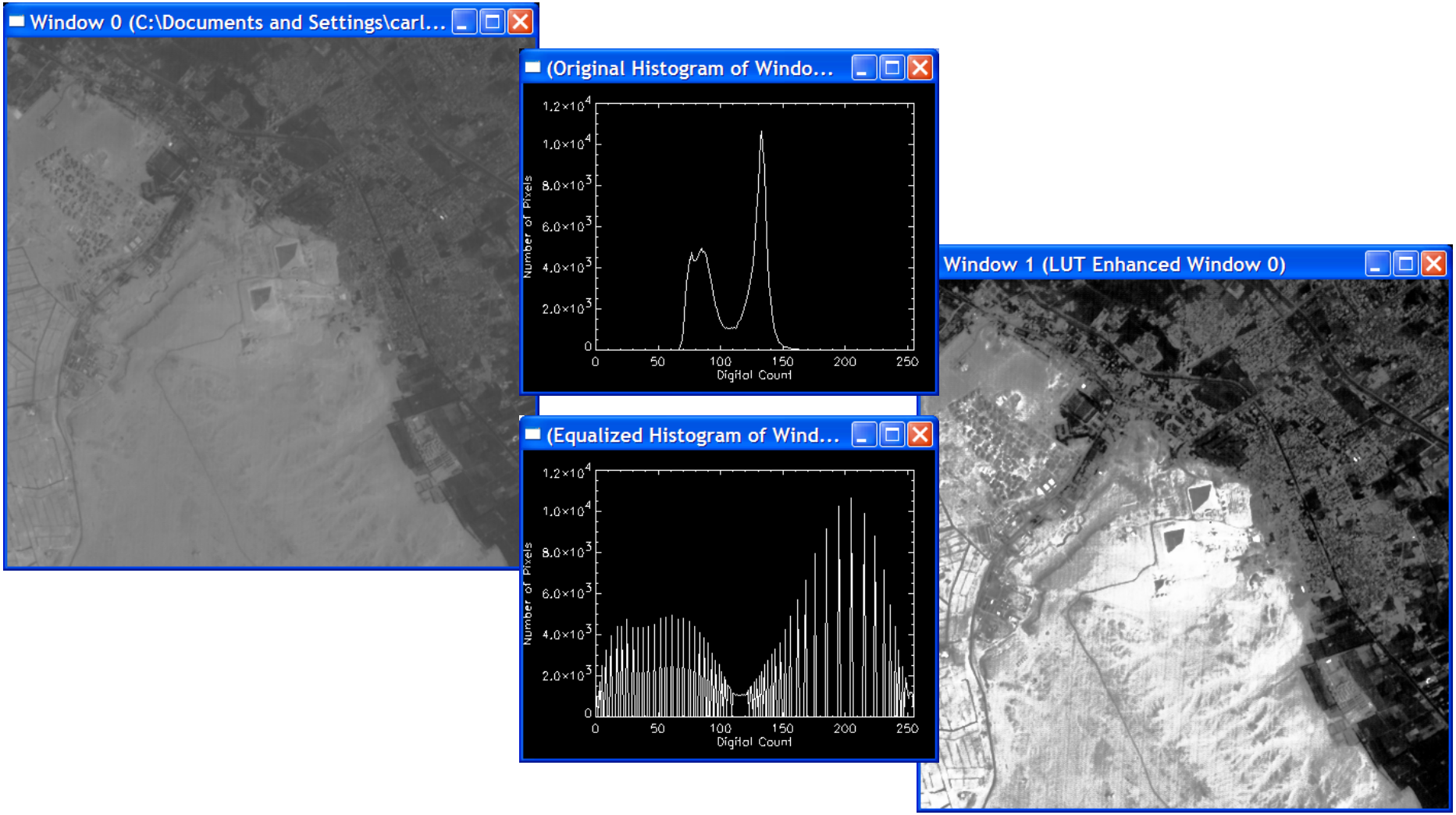
The discrete version of the transformation function is then

$$s_k = T(r_k) = \sum_{j=0}^{k} p_r(r_j)$$

$$= \sum_{j=0}^{k} \frac{n_j}{n} \qquad k = 0, 1, 2, ..., \mathrm{L} - 1$$

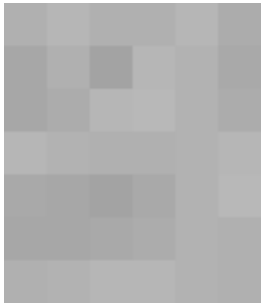
# Image Enhancement

## Histogram Equalization - Example



# Image Enhancement

## Histogram Equalization - Numeric Example

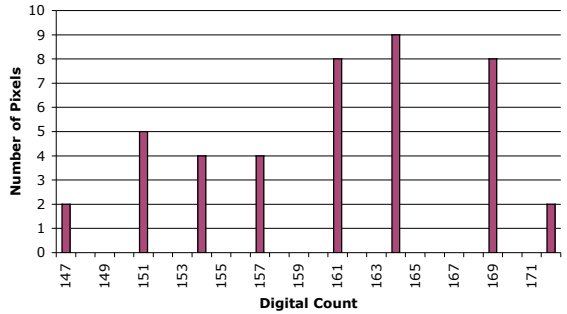


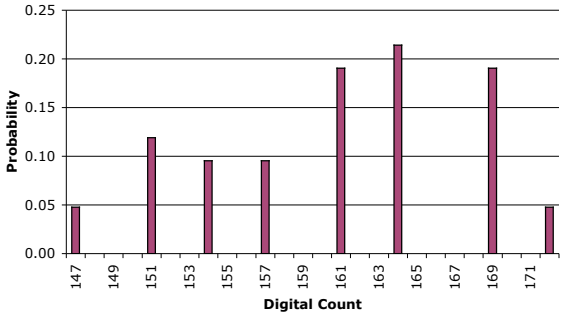
| | | | | | |
|---|---|---|---|---|---|
| 161 | 169 | 161 | 161 | 169 | 157 |
| 151 | 161 | 147 | 169 | 164 | 154 |
| 151 | 157 | 169 | 172 | 164 | 157 |
| 169 | 164 | 161 | 161 | 164 | 169 |
| 154 | 151 | 147 | 154 | 164 | 172 |
| 151 | 151 | 154 | 157 | 164 | 164 |
| 161 | 164 | 169 | 169 | 164 | 161 |

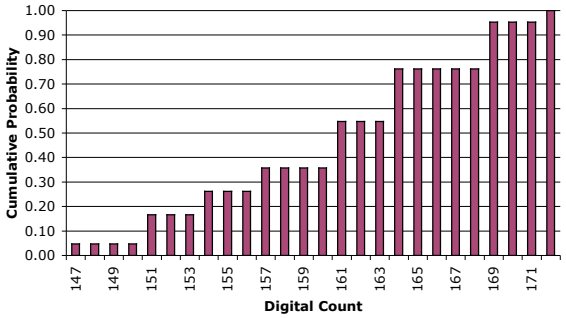
| DC | # of Pixels | PDF | CDF |
|---|---|---|---|
| 147 | 2 | 0.05 | 0.05 |
| 148 | 0 | 0.00 | 0.05 |
| 149 | 0 | 0.00 | 0.05 |
| 150 | 0 | 0.00 | 0.05 |
| 151 | 5 | 0.12 | 0.17 |
| 152 | 0 | 0.00 | 0.17 |
| 153 | 0 | 0.00 | 0.17 |
| 154 | 4 | 0.10 | 0.26 |
| 155 | 0 | 0.00 | 0.26 |
| 156 | 0 | 0.00 | 0.26 |
| 157 | 4 | 0.10 | 0.36 |
| 158 | 0 | 0.00 | 0.36 |
| 159 | 0 | 0.00 | 0.36 |
| 160 | 0 | 0.00 | 0.36 |
| 161 | 8 | 0.19 | 0.55 |
| 162 | 0 | 0.00 | 0.55 |
| 163 | 0 | 0.00 | 0.55 |
| 164 | 9 | 0.21 | 0.76 |
| 165 | 0 | 0.00 | 0.76 |
| 166 | 0 | 0.00 | 0.76 |
| 167 | 0 | 0.00 | 0.76 |
| 168 | 0 | 0.00 | 0.76 |
| 169 | 8 | 0.19 | 0.95 |
| 170 | 0 | 0.00 | 0.95 |
| 171 | 0 | 0.00 | 0.95 |
| 172 | 2 | 0.05 | 1.00 |

# Image Enhancement

## Histogram Equalization - Numeric Example




| DC | CDF | LUT |
|---|---|---|
| 0 | 0.00 | 0 |
| 1 | 0.00 | 0 |
| . | 0.00 | 0 |
| . | 0.00 | 0 |
| . | 0.00 | 0 |
| 147 | 0.05 | 13 |
| 148 | 0.05 | 13 |
| 149 | 0.05 | 13 |
| 150 | 0.05 | 13 |
| 151 | 0.17 | 43 |
| 152 | 0.17 | 43 |
| 153 | 0.17 | 43 |
| 154 | 0.26 | 66 |
| 155 | 0.26 | 66 |
| 156 | 0.26 | 66 |
| 157 | 0.36 | 92 |
| 158 | 0.36 | 92 |
| 159 | 0.36 | 92 |
| 160 | 0.36 | 92 |
| 161 | 0.55 | 140 |
| 162 | 0.55 | 140 |
| 163 | 0.55 | 140 |
| 164 | 0.76 | 194 |
| 165 | 0.76 | 194 |
| 166 | 0.76 | 194 |
| 167 | 0.76 | 194 |
| 168 | 0.76 | 194 |
| 169 | 0.95 | 242 |
| 170 | 0.95 | 242 |
| 171 | 0.95 | 242 |
| 172 | 1.00 | 255 |
| . | 1.00 | 255 |
| . | 1.00 | 255 |
| 255 | 1.00 | 255 |

# Image Enhancement

## Histogram Specification/Matching

Let $r$ and $z$ represent the digital count values of your input and output (processed) images, respectively. Subsequently, $p_r(r)$ and $p_z(z)$ will represent the probability density functions of these two images. The density function $p_r(r)$ is computed from the original input image and $p_z(z)$ is specified to represent the desired density function for the output image.

The transformation desired for histogram specification (matching) would follow the rationale

$$s = T(r) = \int_0^r p_r(w)dw = \int_0^z p_z(t)dt = G(z) = s$$

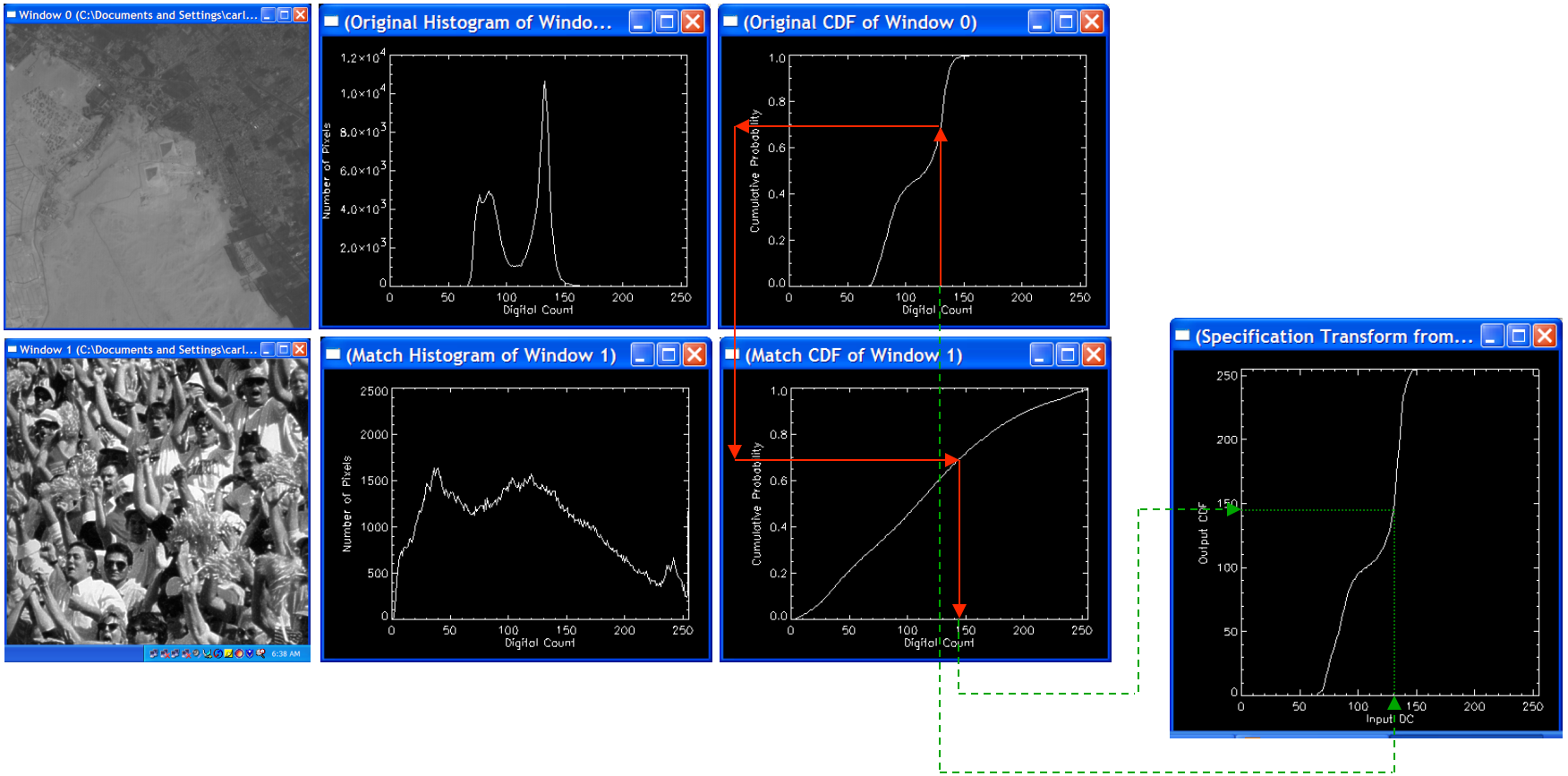
and the output (processed) image digital counts, $z$, would be found as

$$z = G^{-1}(s)$$
$$= G^{-1}\left[T(r)\right]$$

Steps to perform histogram specification
1) determine $T(r)$ as the cumulative distribution function for the original image,
2) determine $G(z)$ as the cumulative distribution function for the specified density function that the output (processed) image is to have,
3) determine the inverse transformation $z=G^{-1}(s)$ of $G(z)$ in the form of a LUT,
4) compute the output (processed) image utilizing the LUT determined in 3).

# Image Enhancement

## Histogram Specification/Matching

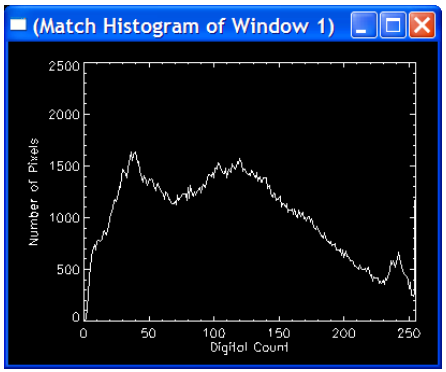


# Image Enhancement

## Histogram Specification/Matching

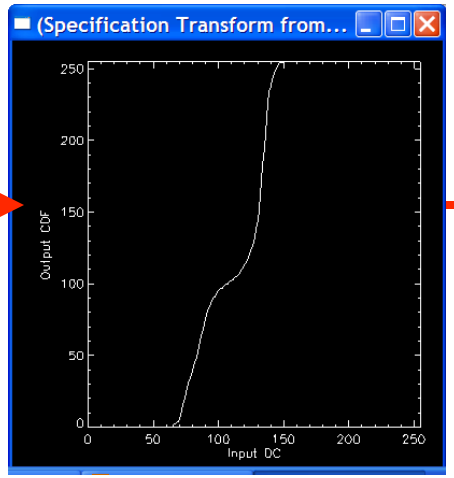
Specified Image


Specified Histogram



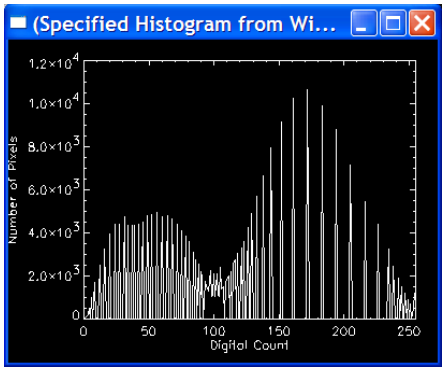

Original Image


Specification Mapping Transform




Processed Image


Histogram of Processed Image



# Image Enhancement

## Histogram Statistics

Let $r$ denote a discrete random variable representing the digital count values of your image in the range $[0,L-1]$ with a normalized histogram (probability density function) given by $p_r(r)$. The $nth$ moment of $r$ about its mean $m$ is

$$\mu_n(r) = \sum_{i=0}^{L-1} (r_i - m)^n \, p(r_i)$$

where $$m = \sum_{i=0}^{L-1} r_i \, p(r_i)$$ MEAN

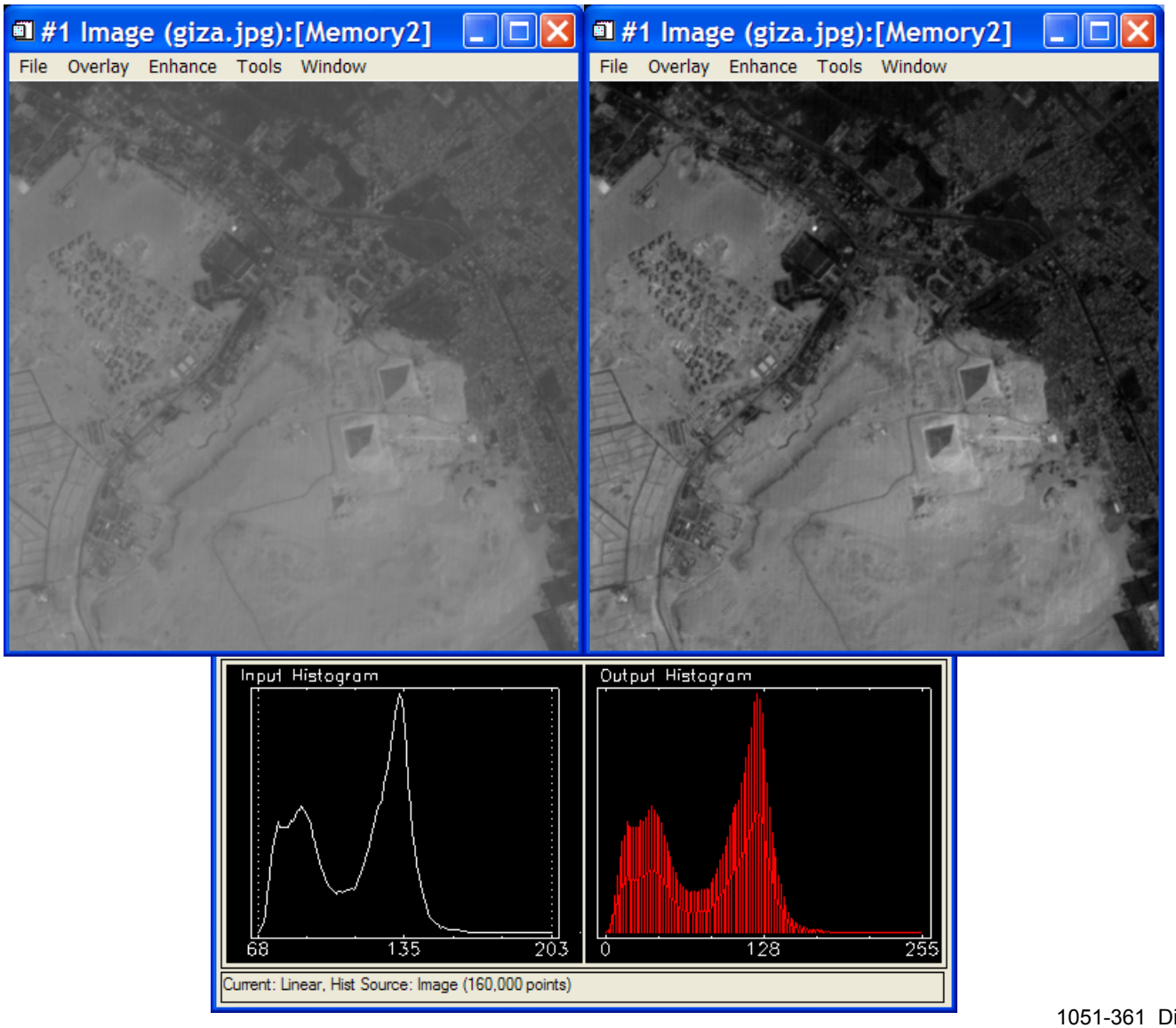
$$\mu_0(r) = \sum_{i=0}^{L-1} (r_i - m)^0 \, p(r_i) = \sum_{i=0}^{L-1} p(r_i) = 1$$

$$\mu_1(r) = \sum_{i=0}^{L-1} (r_i - m)^1 \, p(r_i) = \sum_{i=0}^{L-1} (r_i - m) \, p(r_i) = 0$$

$$\mu_2(r) = \sum_{i=0}^{L-1} (r_i - m)^2 \, p(r_i)$$ VARIANCE

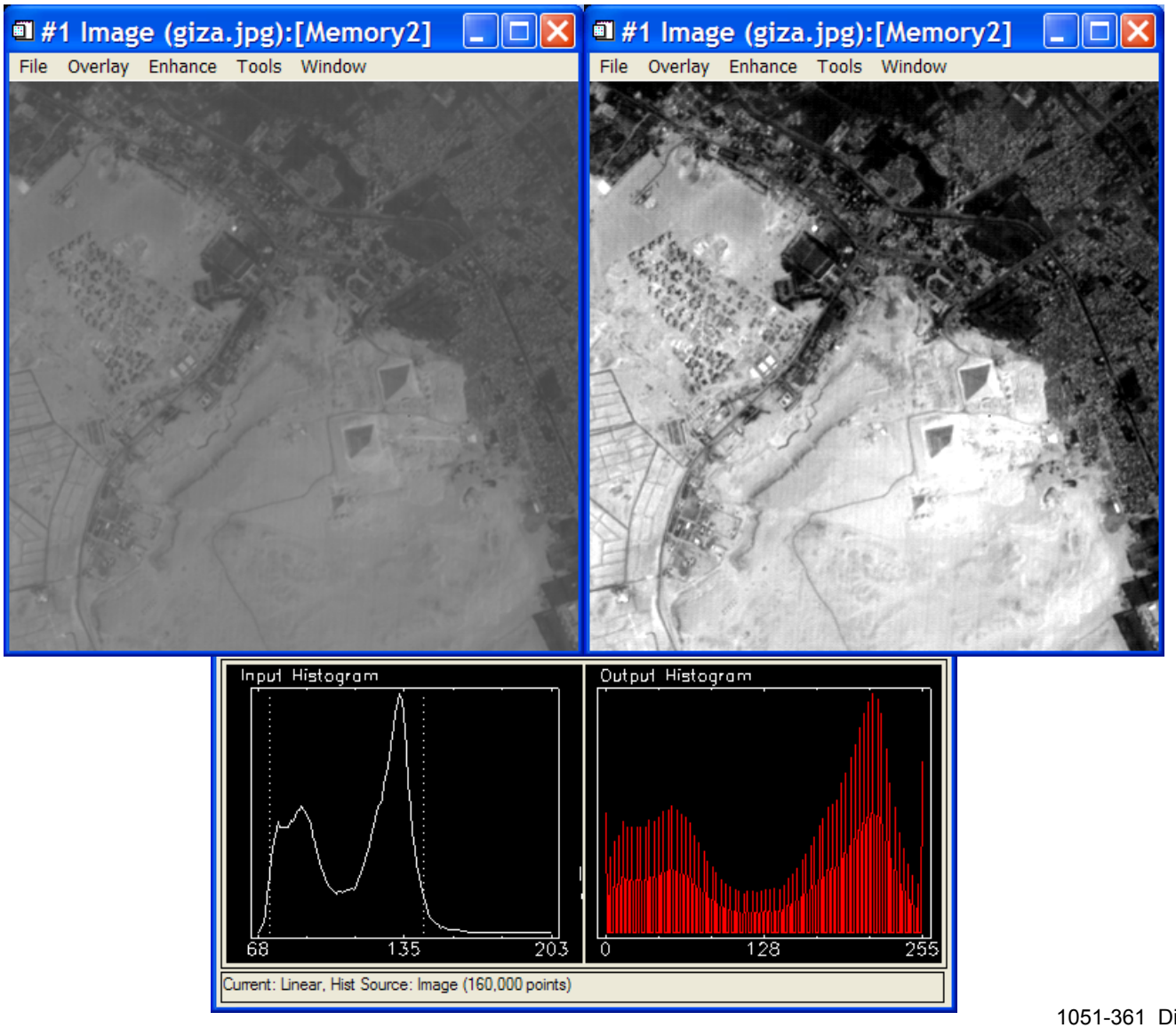
# Image Enhancement

## Linear



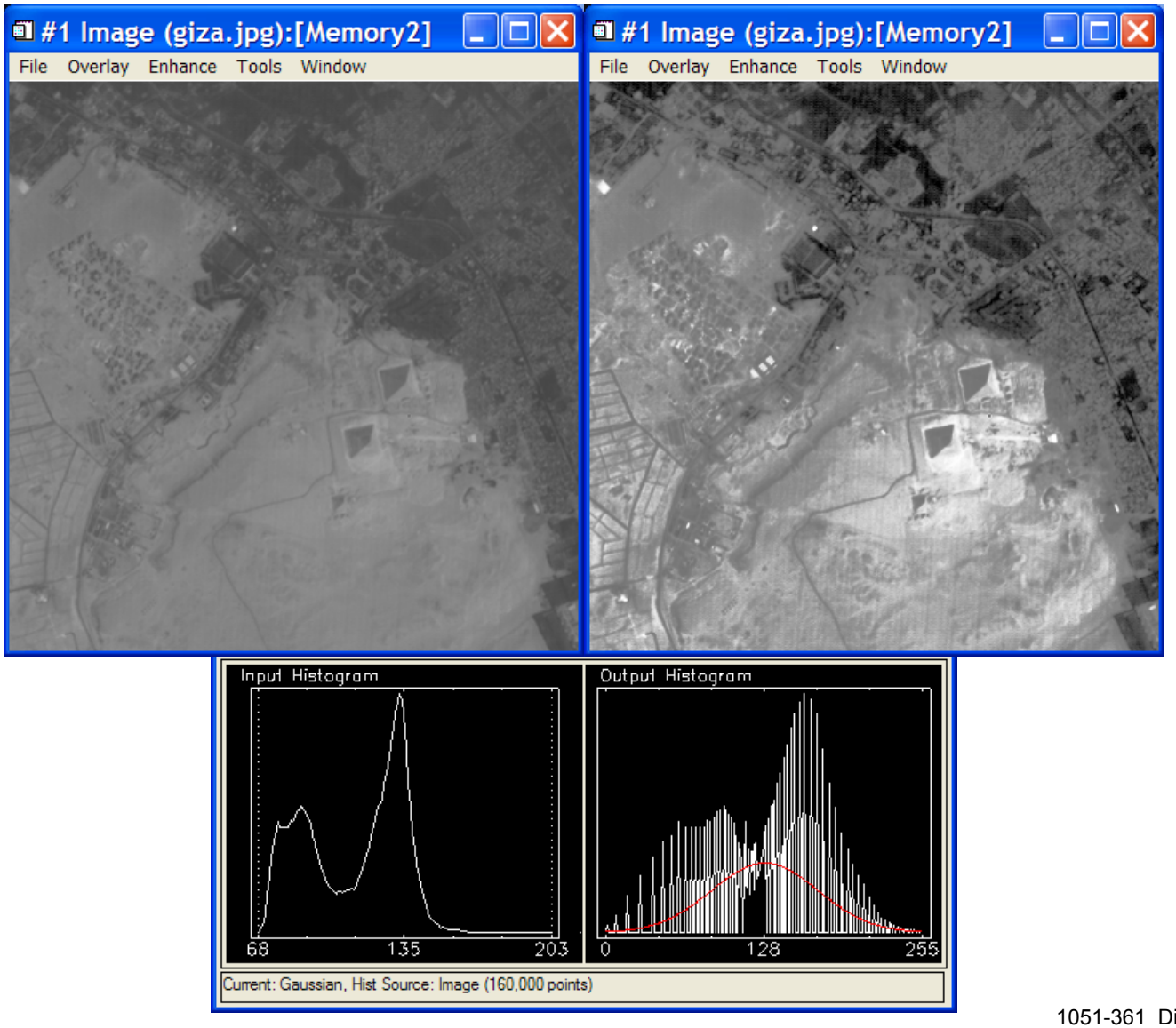
# Image Enhancement

Linear 2%



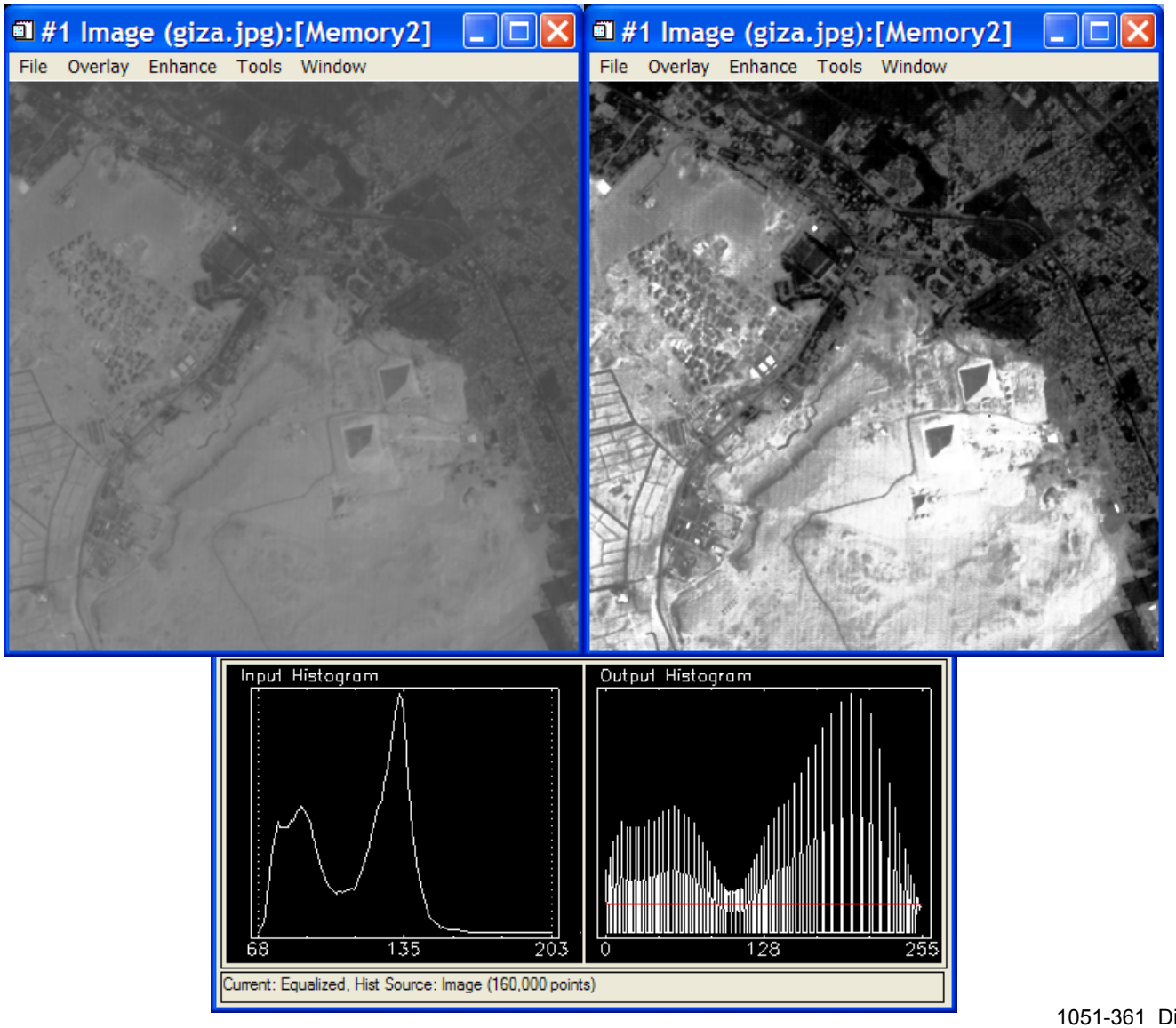
# Image Enhancement

## Gaussian Specification



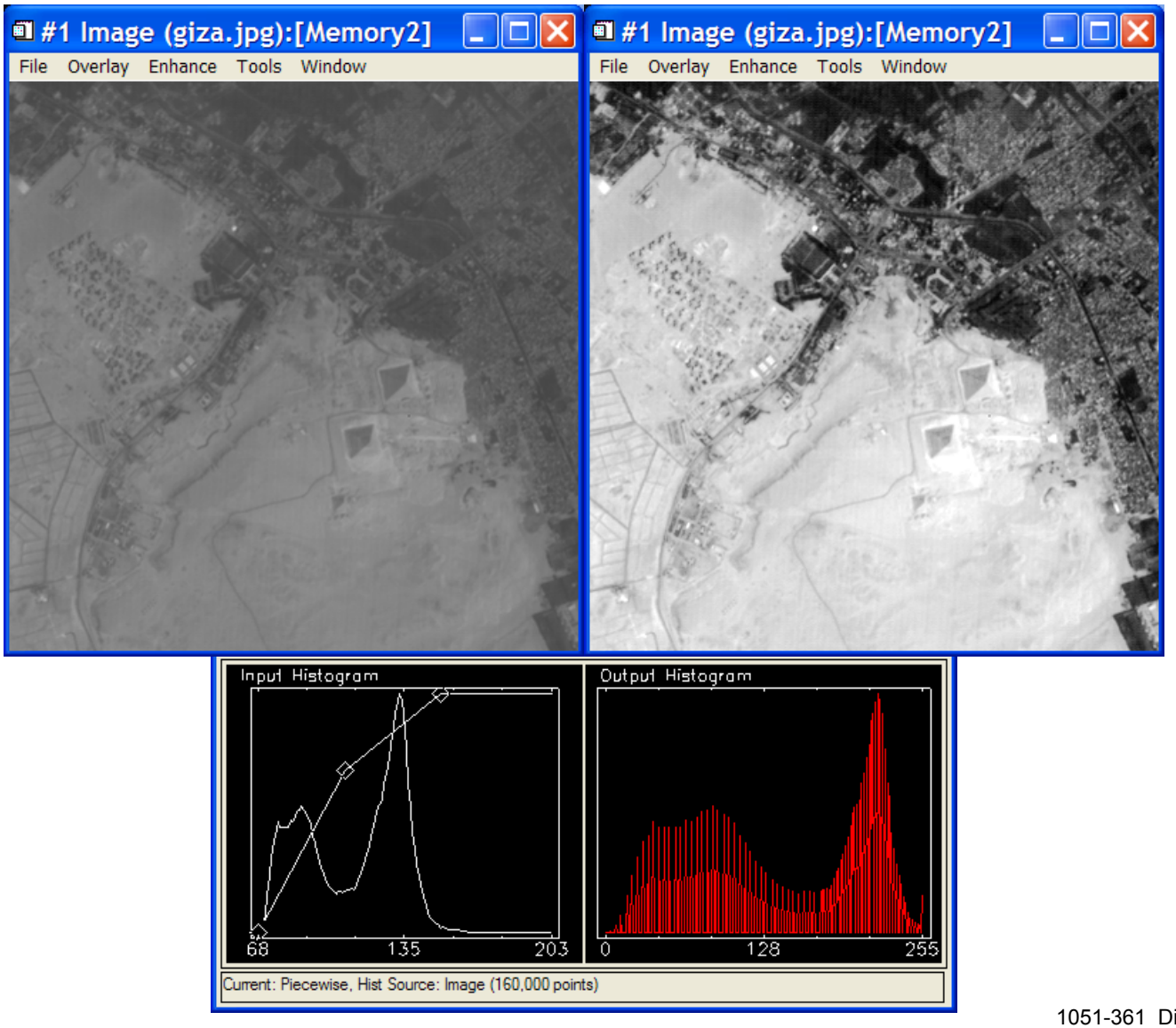
# Image Enhancement

## Uniform Specification (Equalization)



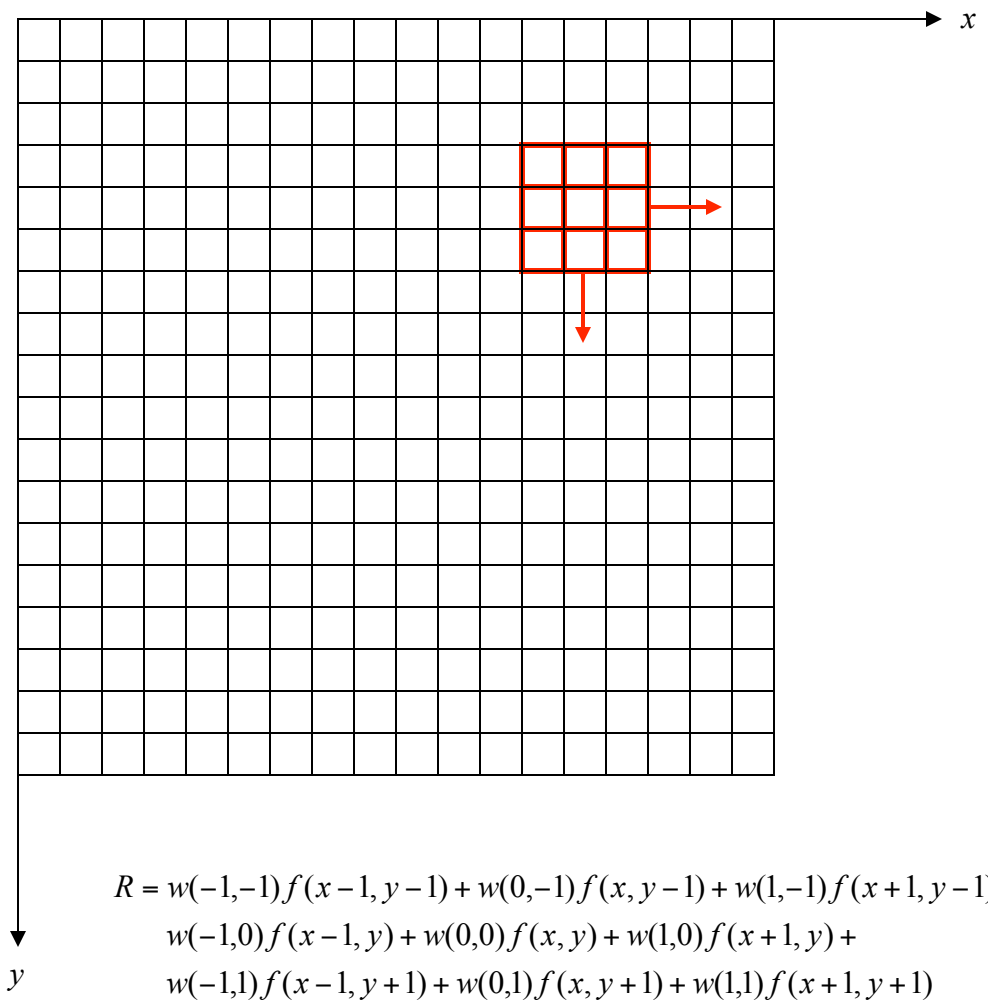
# Image Enhancement

## Piecewise Linear



# Spatial Filtering

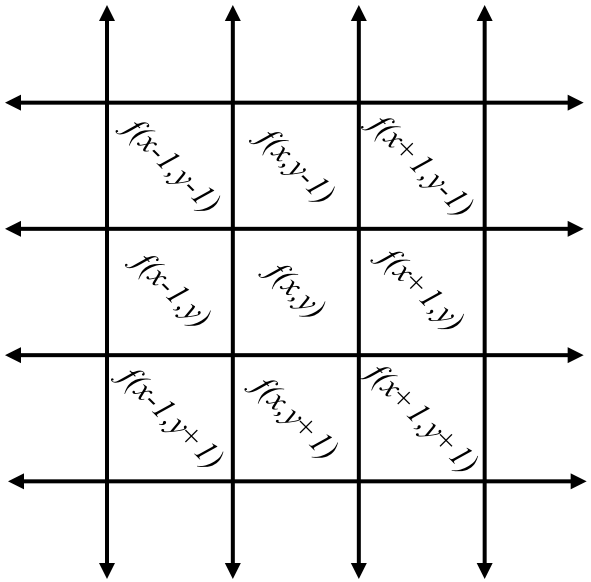
## Basics of Linear Spatial Filtering




Filter

| | | |
|---|---|---|
| w(-1,-1) | w(0,-1) | w(1,-1) |
| w(-1,0) | w(0,0) | w(1,0) |
| w(-1,1) | w(0,1) | w(1,1) |

Image DC beneath Filter




$$R = w(-1,-1)f(x-1,y-1) + w(0,-1)f(x,y-1) + w(1,-1)f(x+1,y-1) +$$
$$w(-1,0)f(x-1,y) + w(0,0)f(x,y) + w(1,0)f(x+1,y) +$$
$$w(-1,1)f(x-1,y+1) + w(0,1)f(x,y+1) + w(1,1)f(x+1,y+1)$$

# Spatial Filtering

## Generalized Form of a Linear Spatial Filter

$$g(x,y) = \sum_{t=-b}^{b} \sum_{s=-a}^{a} w(s,t) f(x+s, y+t)$$

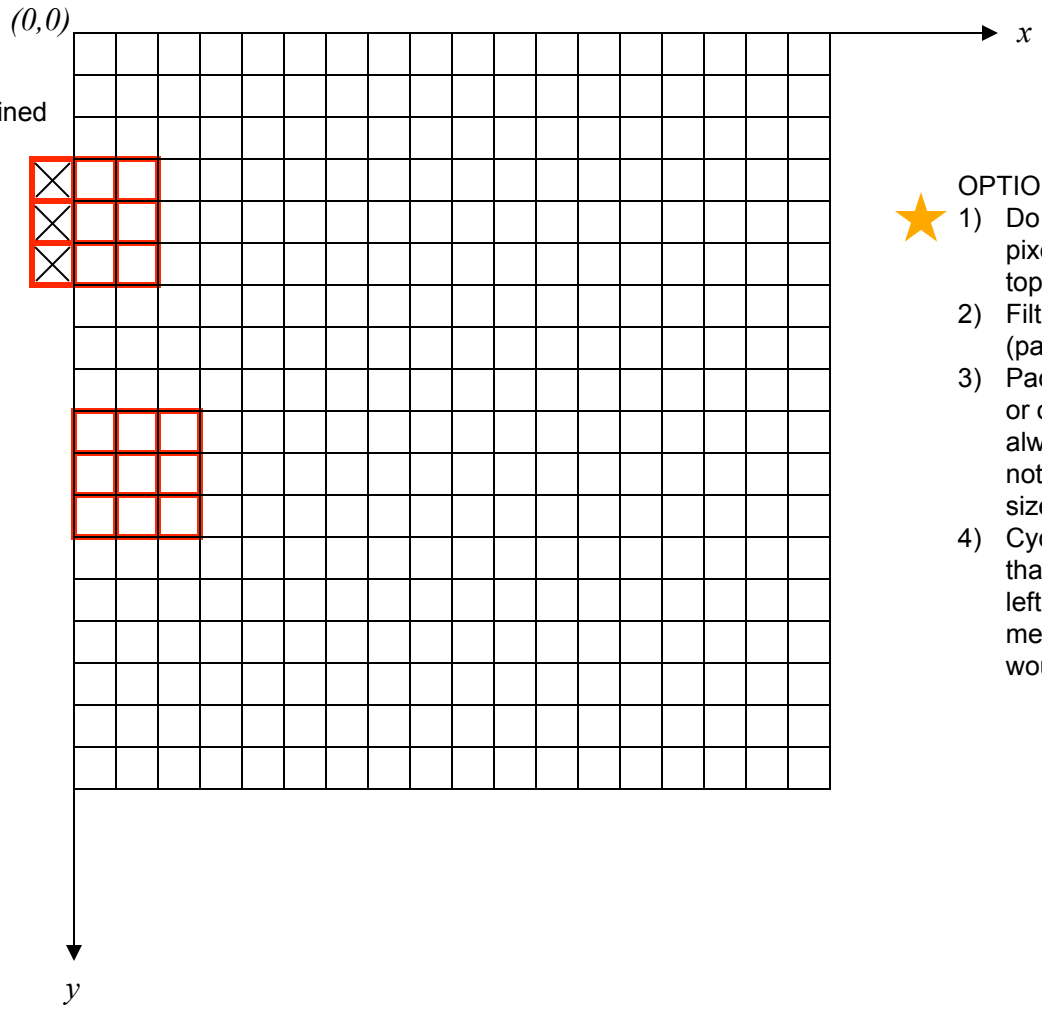
where $a = \frac{m-1}{2}$, $b = \frac{n-1}{2}$ for a filter of size $m$ x $n$
($m$ and $n$ usually chosen to be odd)

Linear spatial filtering is often referred to as "convolving a mask with an image"

A filter is often referred to as a convolution kernel or mask

# Spatial Filtering

## Treatment of Edge Pixels




There are no defined image values at these points beneath the filter

OPTIONS

1) Do not allow the filter to come within *(m-1)/2* pixels of either edge and *(n-1)/2* pixels of the top or bottom
2) Filter only the pixels that fall beneath the filter (partial filtering of edge pixels)
3) Pad the image with rows and columns of *0*'s or other constant value to allow the filter to always cover valid pixels (can lead to very noticeable edge effects as the kernel grows in size)
4) Cyclical treatment wraps the image around so that the rightmost column of pixels meets the leftmost column and the top row of pixels meets the bottom to define pixel values that would fall beneath the spatial filter

# Spatial Filtering

## Smoothing/Averaging/Lowpass Linear Filters

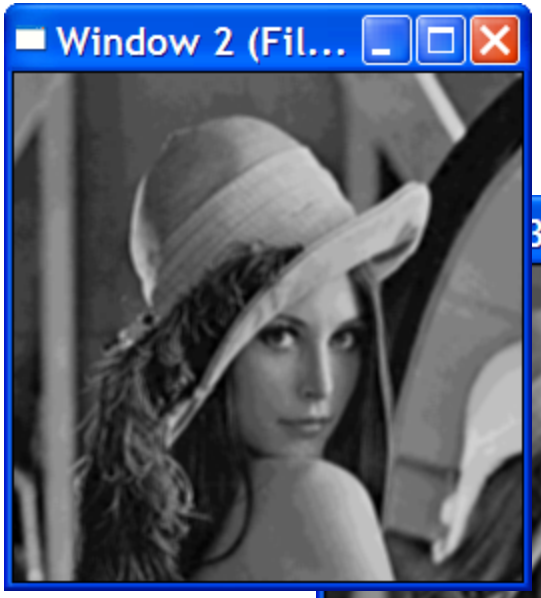
**Purpose**

- Blurring edges
- Noise reduction (smoothing)
- Removal of small details (irrelevant features)
- Bridging small gaps in linear or curvilinear features

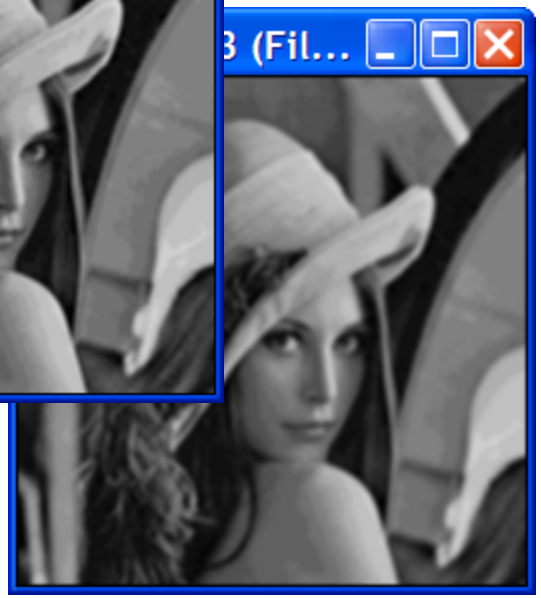
Original Image (256 Gray Levels)


Original Image (16 Gray Levels)


Blurred




More Blurred



# Spatial Filtering

## Generalized Form of a Smoothing Linear Filter

$$g(x,y) = \frac{\sum_{t=-b}^{b}\sum_{s=-a}^{a} w(s,t) f(x+s, y+t)}{\sum_{t=-b}^{b}\sum_{s=-a}^{a} w(s,t)}$$

where $a = \frac{m-1}{2}$, $b = \frac{n-1}{2}$ for a filter of size $m$ x $n$
($m$ and $n$ usually chosen to be odd)

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

*weight=9*

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 9 | 1 |
| 1 | 1 | 1 |

*weight=17*

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 2 | 4 | 2 |
| 1 | 2 | 1 |

*weight=16*

# Spatial Filtering

## Order-Statistics Filters - Non-Linear Spatial Filters

**Median Filter** This filter will replace the center pixel value beneath the filter by the median of the neighborhood of image pixel values covered by the filter

The median, $\xi$, of a set of values is defined such that half of the values in the set are greater than $\xi$ and half of the values in the set are less than $\xi$.

In order to implement this filter

1) Define a list containing all the pixel values in the neighborhood that falls beneath the filter,
2) Sort the list in either ascending or descending order,
3) Find the value in the middle of the list and replace the neighborhood center value with this median.

# Spatial Filtering

## Order-Statistics Filters - Non-Linear Spatial Filters

MIN Filter

This filter will replace the center pixel value beneath the filter by the minimum of the neighborhood of image pixel values covered by the filter, $S_{x,y}$

$$g(x,y) = \min_{(s,t)\in S_{x,y}} \{f(s,t)\}$$

MAX Filter

This filter will replace the center pixel value beneath the filter by the maximum of the neighborhood of image pixel values covered by the filter, $S_{x,y}$

$$g(x,y) = \max_{(s,t)\in S_{x,y}} \{f(s,t)\}$$

# Spatial Filtering

## Order-Statistics Filters - Non-Linear Spatial Filters

Midpoint Filter  This filter will replace the center pixel value beneath the filter by the midpoint between the minimum and maximum of the neighborhood of image pixel values covered by the filter, $S_{x,y}$

$$g(x,y) = \frac{1}{2}\left[\max_{(s,t)\in S_{x,y}}\{f(s,t)\} + \min_{(s,t)\in S_{x,y}}\{f(s,t)\}\right]$$

This filter is best used for reducing randomly distributed Gaussian or uniform noise

# Spatial Filtering

## Order-Statistics Filters - Non-Linear Spatial Filters

Alpha Trimmed Mean Filter

This filter will replace the center pixel value beneath the filter by the mean of the remaining pixels in the neighborhood beneath the filter, $S_{x,y}$, after the d/2 lowest and d/2 highest gray levels are removed

$$g(x,y) = \frac{1}{mn - d} \sum_{(s,t) \in S_{x,y}} f(s,t)$$

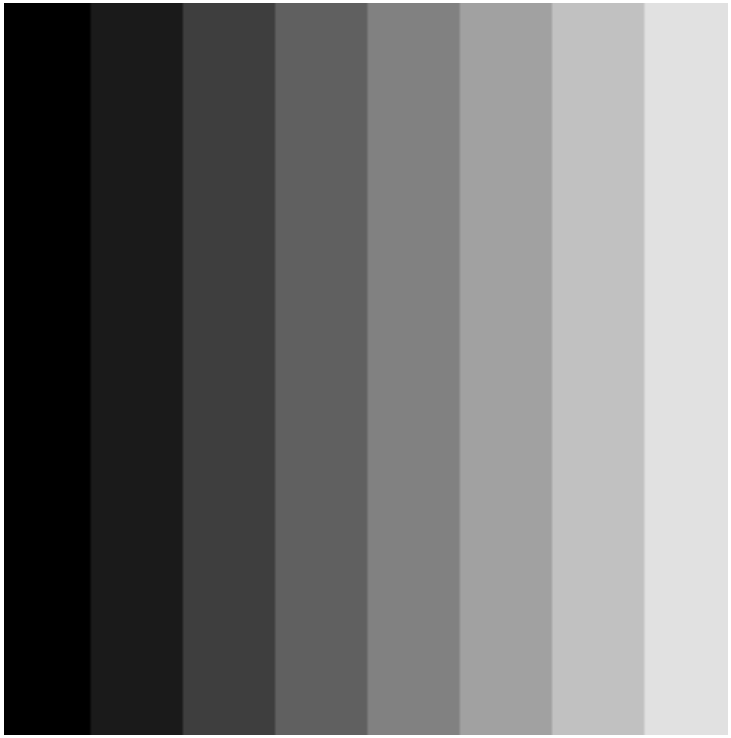
$$g(x,y) = \frac{1}{mn} \sum_{(s,t) \in S_{x,y}} f(s,t)$$

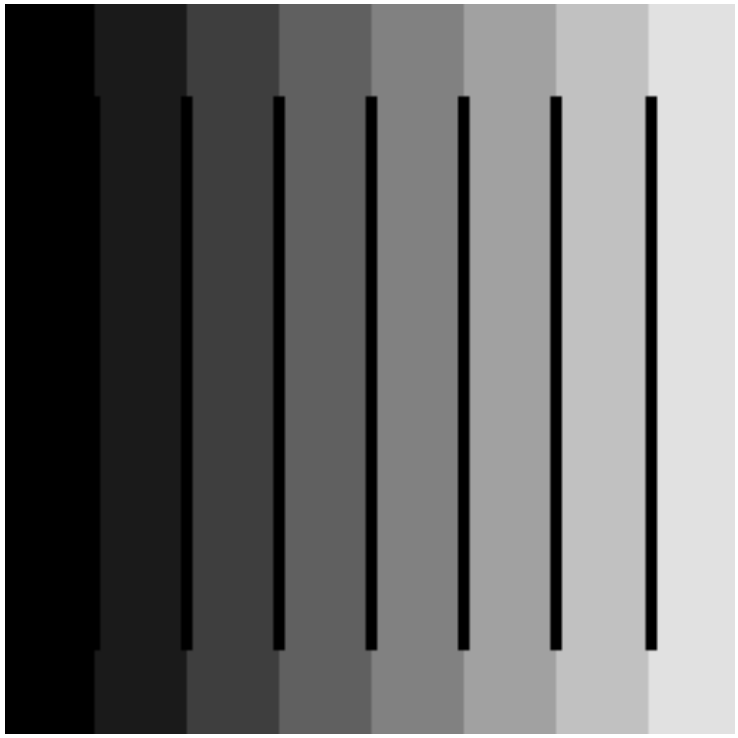
when $d=0$, this reduces to the arithmetic mean

This filter is best used for simultaneously occurring noise terms, e.g. Gaussian and salt-and-pepper noise

# Spatial Filtering

## Sharpening - Introduction





# Spatial Filtering

## Sharpening - Introduction

- Highlight fine detail
- Enhance detail that has been blurred

- As averaging is analogous to integration, sharpening is analogous to differentiation

- The strength of the response of a differentiation operator is proportional to the degree of discontinuity of an image at the point at which the operator is applied
  - Image differentiation will enhance edges
  - It will also enhance noise
  - Image differentiation has little effect on slowly varying gray areas

# Spatial Filtering

## Sharpening - Introduction

*First Derivative*

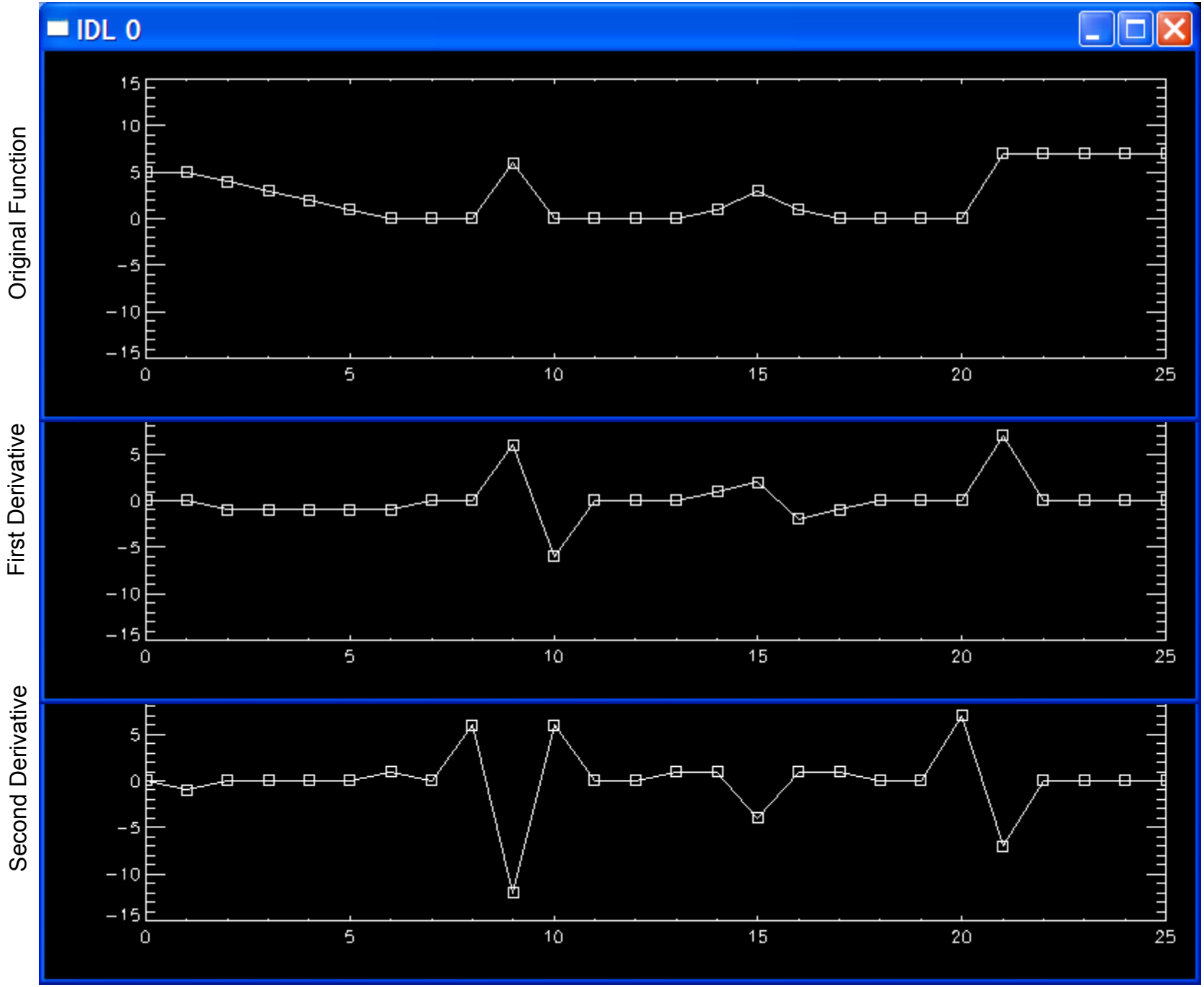
1) must be zero in areas of constant value
2) must be non-zero at the onset of a discontinuity
3) must be non-zero along a ramped discontinuity

$$\frac{\partial f}{\partial x} = f(x+1) - f(x)$$

*Second Derivative*

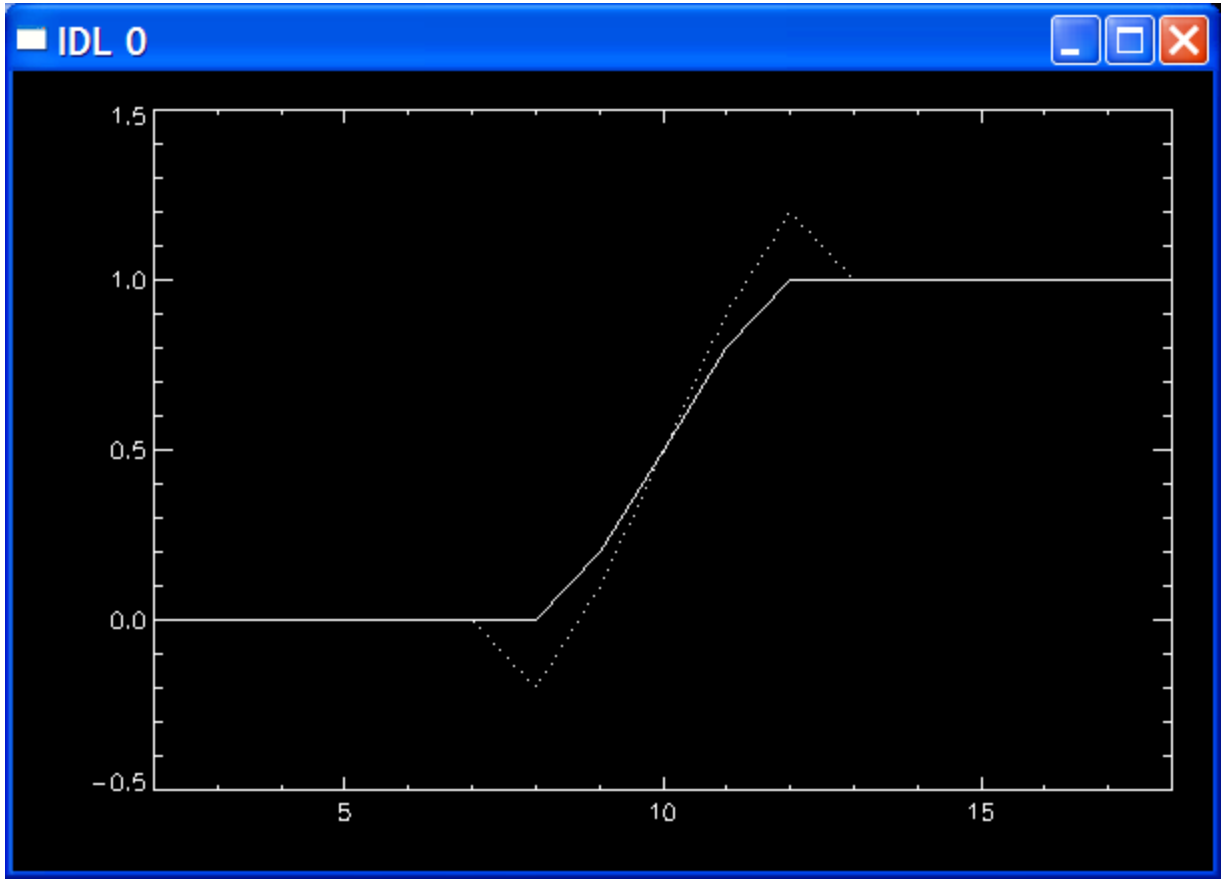
1) must be zero in areas of constant value
2) must be non-zero at the onset and end of a discontinuity
3) must be zero along a ramped discontinuity of constant slope

$$\frac{\partial^2 f}{\partial x^2} = f(x+1) + f(x-1) - 2f(x)$$


IDL 0
Original Function
First Derivative
Second Derivative
15
10
5
0
−5
−10
−15
0
5
10
15
20
25

# Spatial Filtering

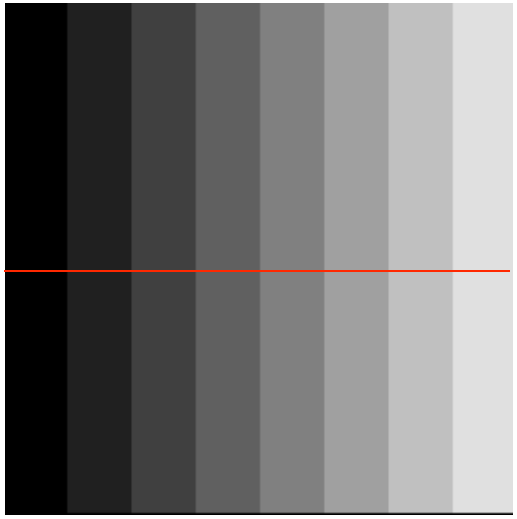
## Sharpening - Introduction

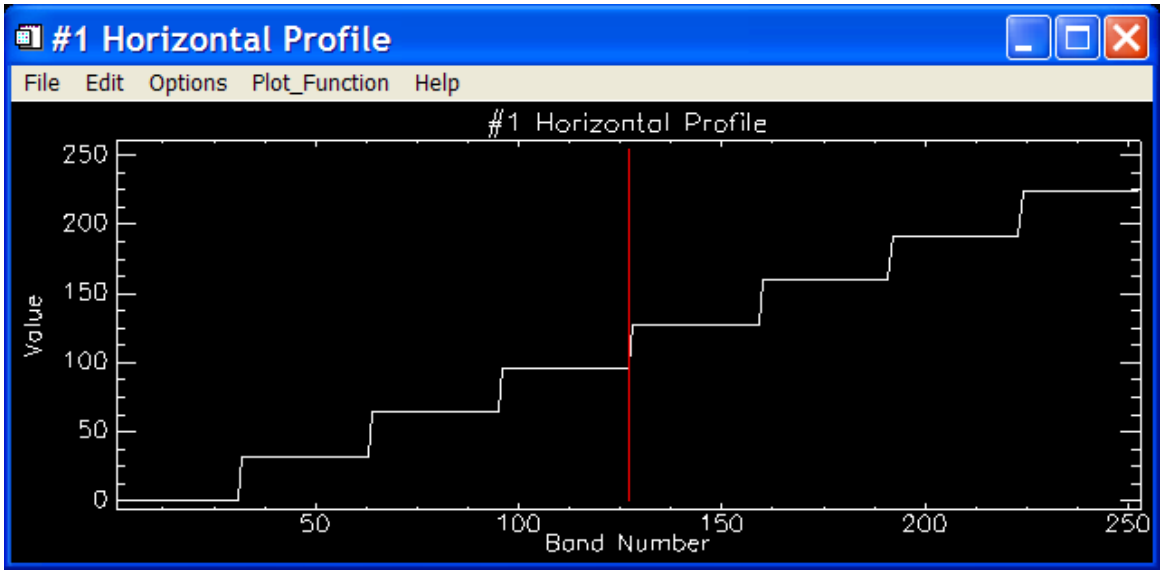


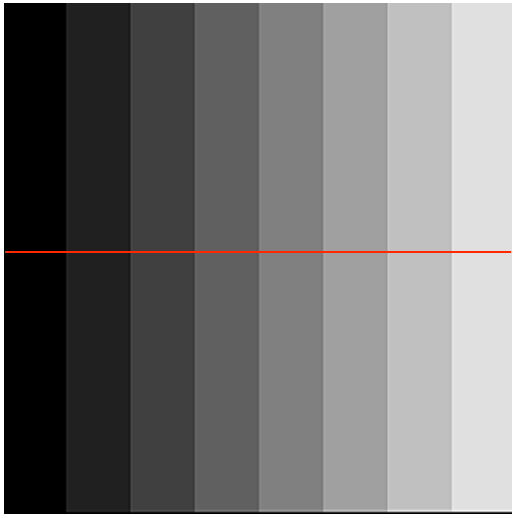

```
IDL> a = [0,0,0,0,0,0,0,0,0,0.2,0.5,0.8,1.0,1,1,1,1,1,1,1,1]
IDL> i = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
IDL> kernel = [-1,3,-1]
IDL> b = CONVOL( a, kernel )
IDL> WINDOW, 0, XSIZE=600, YSIZE=400
IDL> PLOT, i, a, XRANGE=[2,18], XSTYLE=1, YRANGE=[-0.5,1.5], YSTYLE=1
IDL> OPLOT, i, b, LINESTYLE=1
```

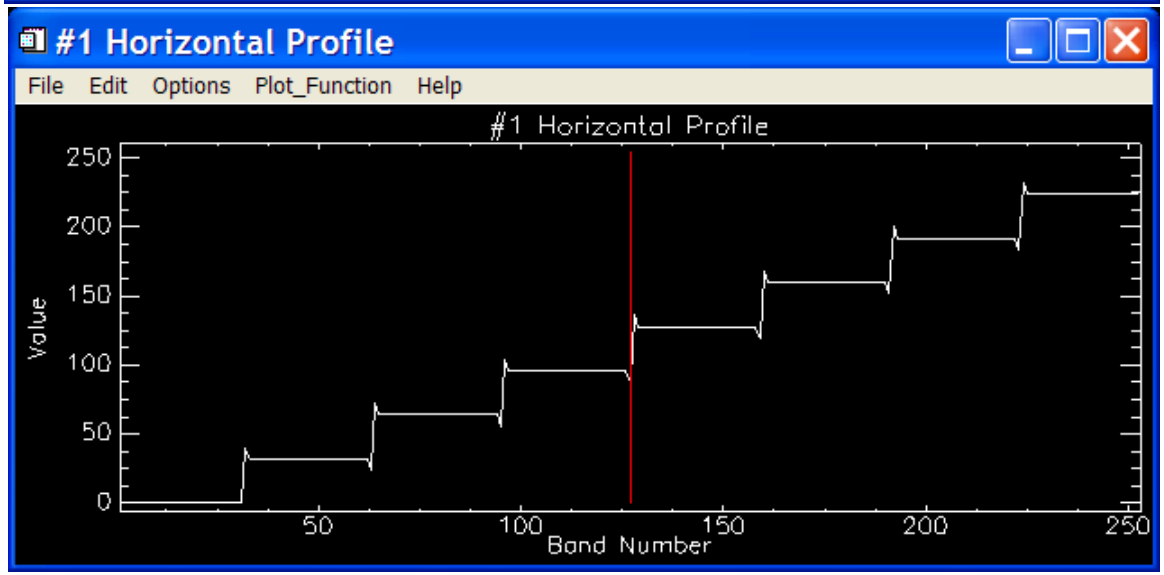
# Spatial Filtering

## Sharpening



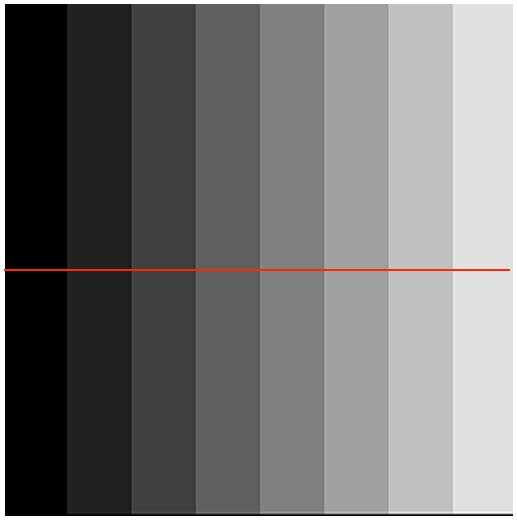


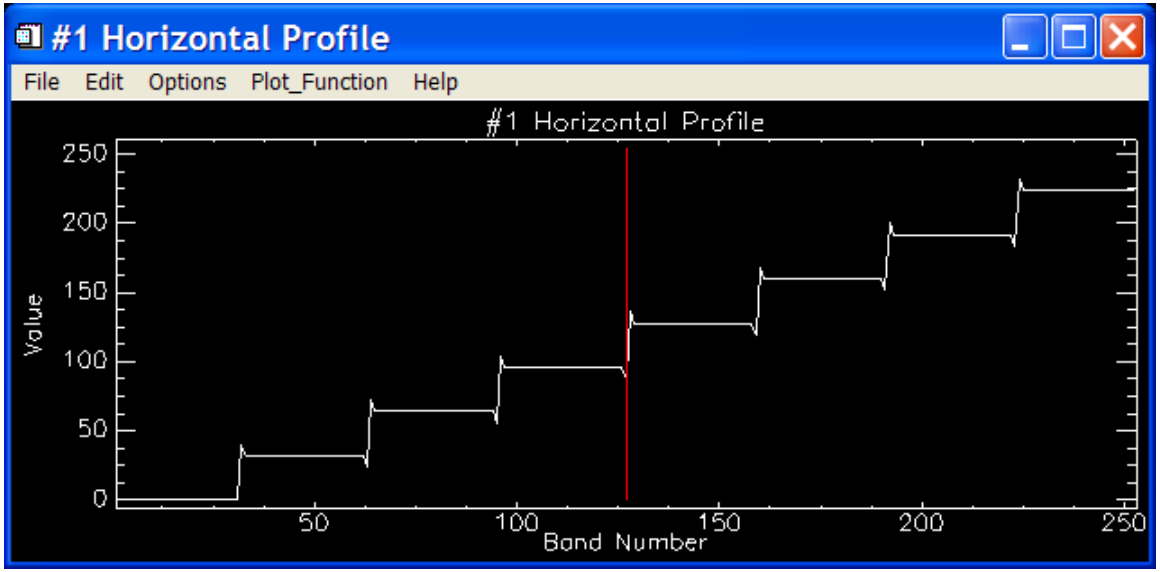



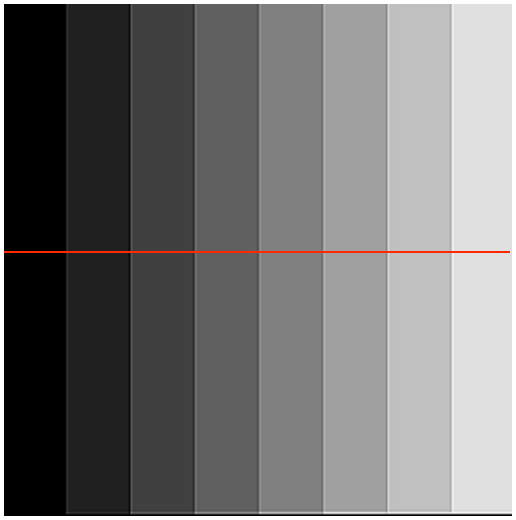


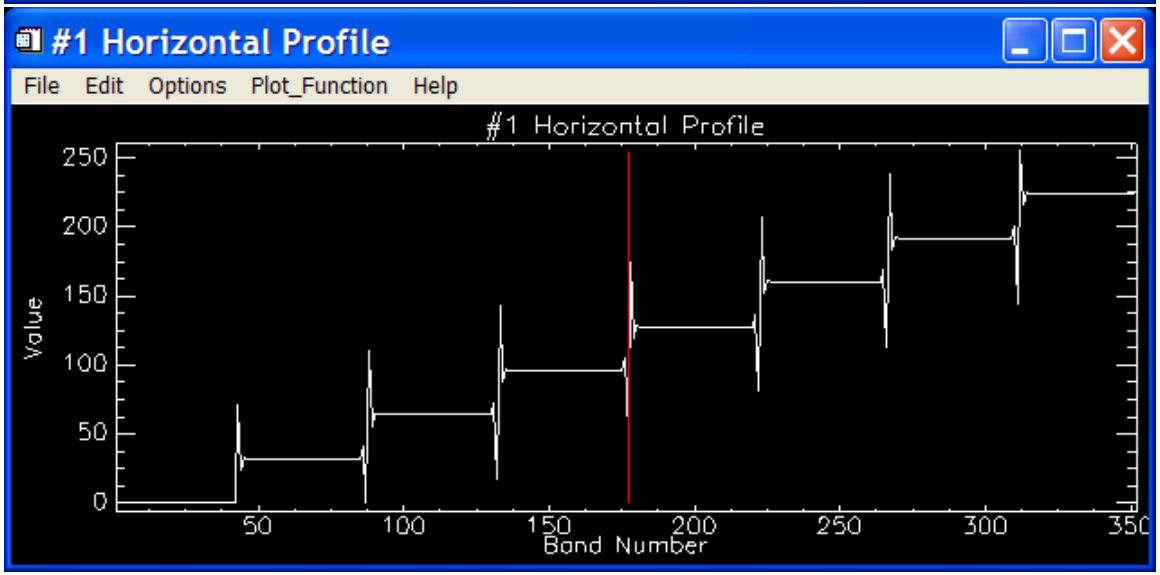
# Spatial Filtering

## Sharpening



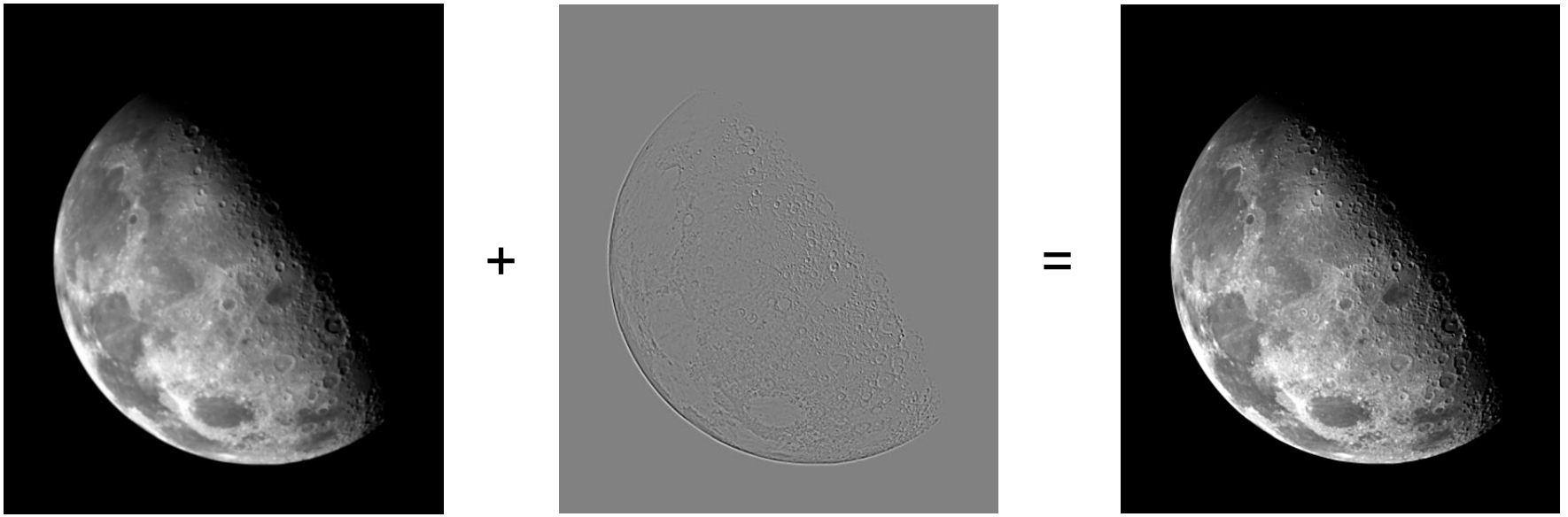




# Spatial Filtering

## Conceptual Approach to Sharpening



# Spatial Filtering

## Sharpening - Laplacian Operator

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y)$$

$$\begin{aligned}\nabla^2 f &= \big(f(x+1, y) + f(x-1, y) - 2f(x, y)\big) + \big(f(x, y+1) + f(x, y-1) - 2f(x, y)\big) \\ &= \big[f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1)\big] - 4f(x, y)\end{aligned}$$

# Spatial Filtering

## Sharpening - Laplacian Operator

$$\nabla^2 f = \big(f(x+1,y) + f(x-1,y) - 2f(x,y)\big) + \big(f(x,y+1) + f(x,y-1) - 2f(x,y)\big)$$
$$= \big[f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)\big] - 4f(x,y)$$

Laplacian Filter

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | -4 | 1 |
| 0 | 1 | 0 |

Isotropic with incremental image rotations of 90 degrees

Other Laplacian Filters

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -8 | 1 |
| 1 | 1 | 1 |

Isotropic with incremental image rotations of 45 degrees

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 4 | -1 |
| 0 | -1 | 0 |

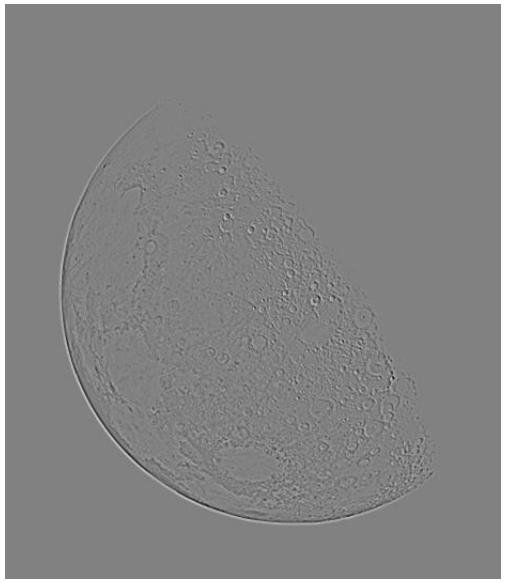
| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 8 | -1 |
| -1 | -1 | -1 |

Negative Laplacian Operators

# Spatial Filtering

## Sharpening Using a Laplacian Linear Filter

Since the second derivative highlights gray-level discontinuities and deemphasizes regions of slowly varying gray levels, the Laplacian produces an image that has features at the edges and is featureless in all other areas.



# Spatial Filtering

## Sharpening Using a Laplacian Linear Filter

So the concept of sharpening can be satisfied by subtracting the enhanced edge information from the original image as

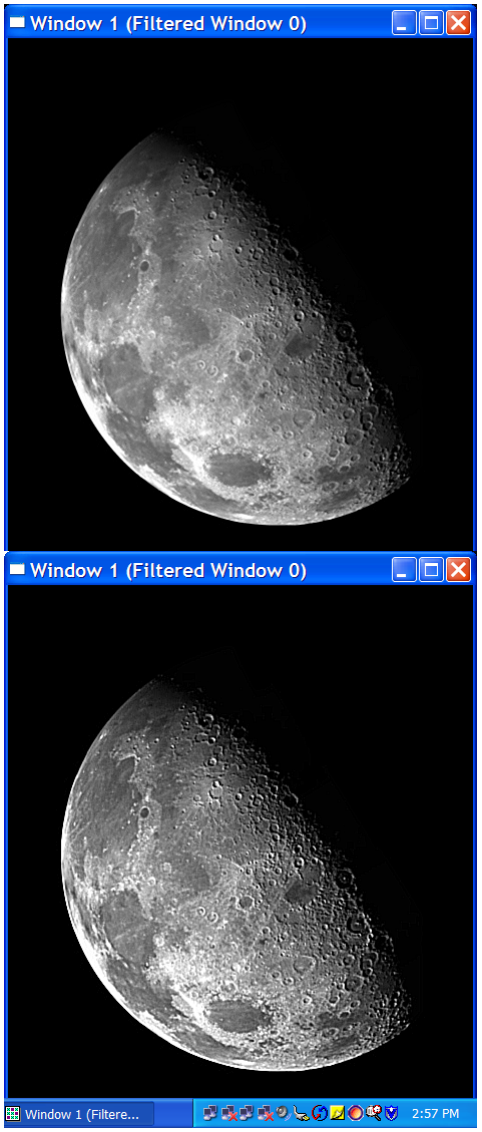
$$f(x,y) - \nabla^2 f = f(x,y) - \left(\left[f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1)\right] - 4f(x,y)\right)$$
$$= 5f(x,y) - f(x+1,y) - f(x-1,y) - f(x,y+1) - f(x,y-1)$$

yielding

$$g(x,y) = \begin{cases} f(x,y) - \nabla^2 f(x,y), & \text{if the center coefficient of the filter is negative} \\ f(x,y) + \nabla^2 f(x,y), & \text{if the center coefficient of the filter is positive} \end{cases}$$

# Spatial Filtering

## Sharpening Using a Laplacian Linear Filter


| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

# Spatial Filtering

## Unsharp Masking

Sharpening Function

$$f_{sharpening}(x,y) = f(x,y) - f_{blurred}(x,y)$$

Sharpened Image

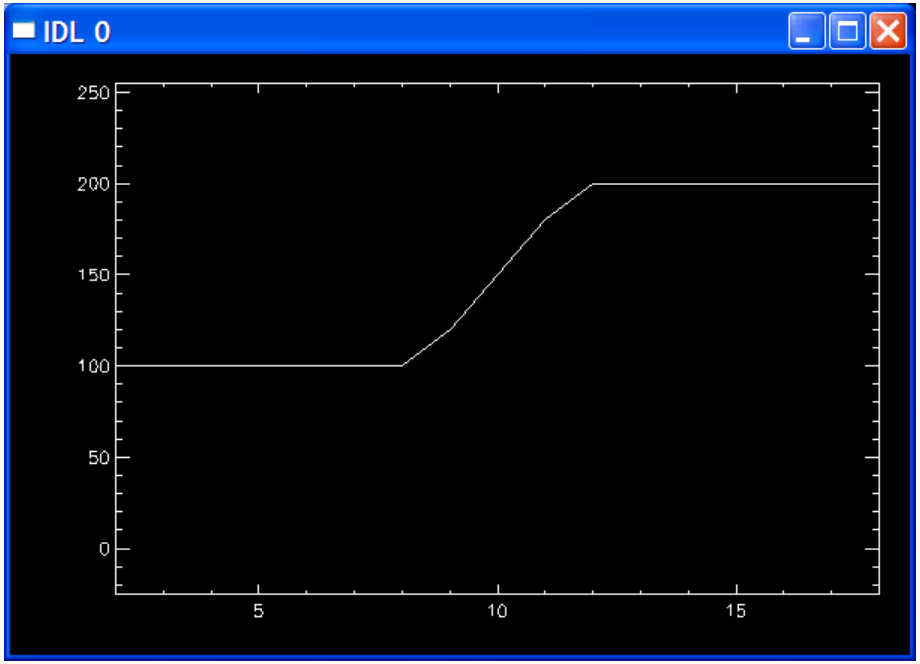
$$f_{sharpened}(x,y) = f(x,y) + f_{sharpening}(x,y)$$

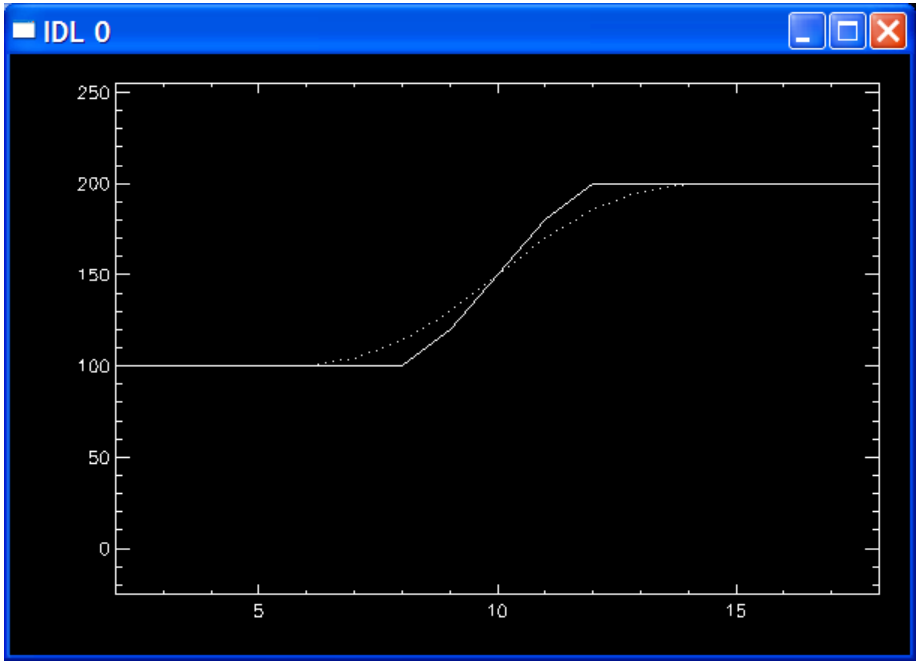
Implementation

$$f_{sharpened}(x,y) = f(x,y) + \left[f(x,y) - f_{blurred}(x,y)\right]$$

# Spatial Filtering

## Unsharp Masking

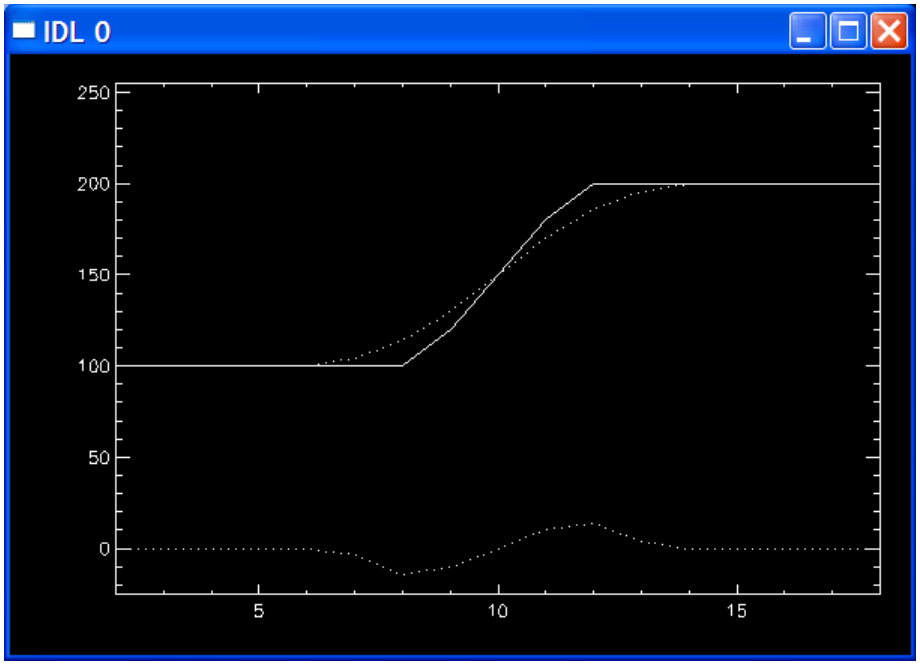


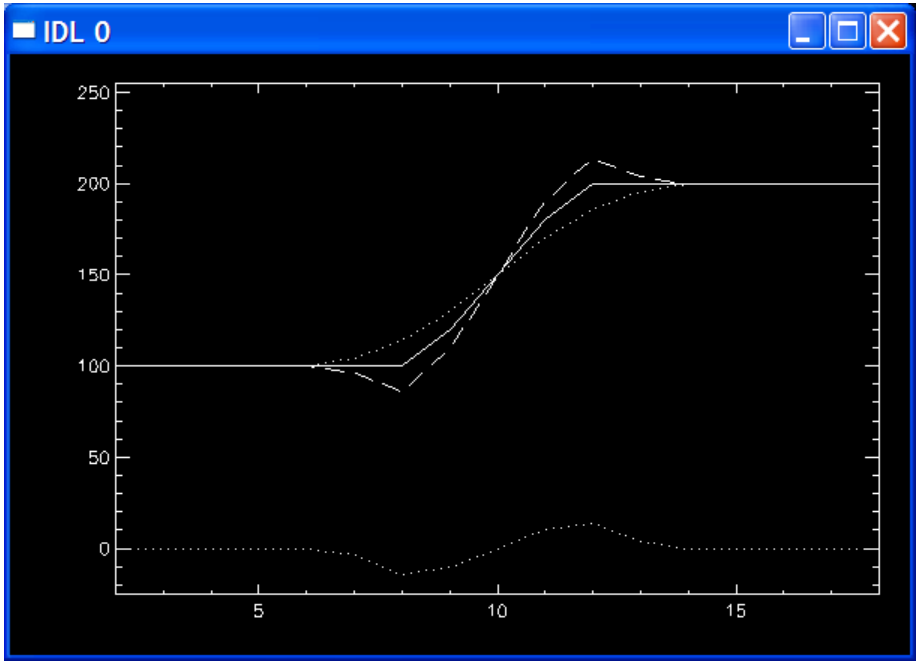




```
IDL> a = [100,100,100,100,100,100,100,100,100,120,150,180,200,200,200,200,200,200,200,200,200]
IDL> i = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
IDL> blur_kernel = [1,1,1,1,1]
IDL> blurred_a = CONVOL( a, blur_kernel,TOTAL(blur_kernel) )
IDL> WINDOW, 0, XSIZE=600, YSIZE=400
IDL> PLOT, i, a, XRANGE=[2,18], XSTYLE=1, YRANGE=[-25,255], YSTYLE=1
IDL> OPLOT, i, blurred_a, LINESTYLE=1
```

# Spatial Filtering

## Unsharp Masking



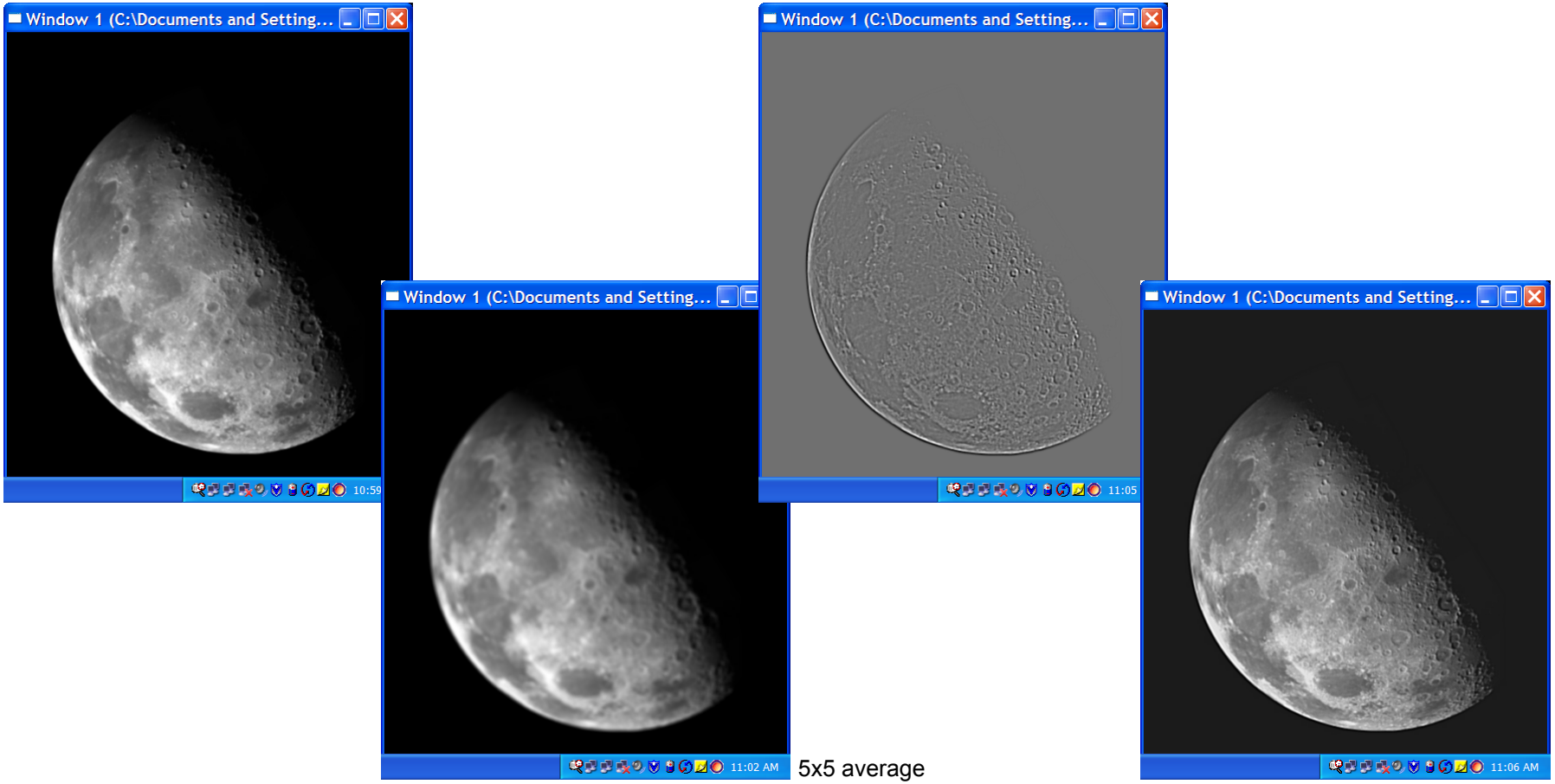




```
IDL> sharpening_function = a - blurred_a
IDL> OPLOT, i, sharpening_function, LINESTYLE=1
IDL> sharpened_a = a + sharpening_function
IDL> OPLOT, i, sharpened_a, LINESTYLE=5
```

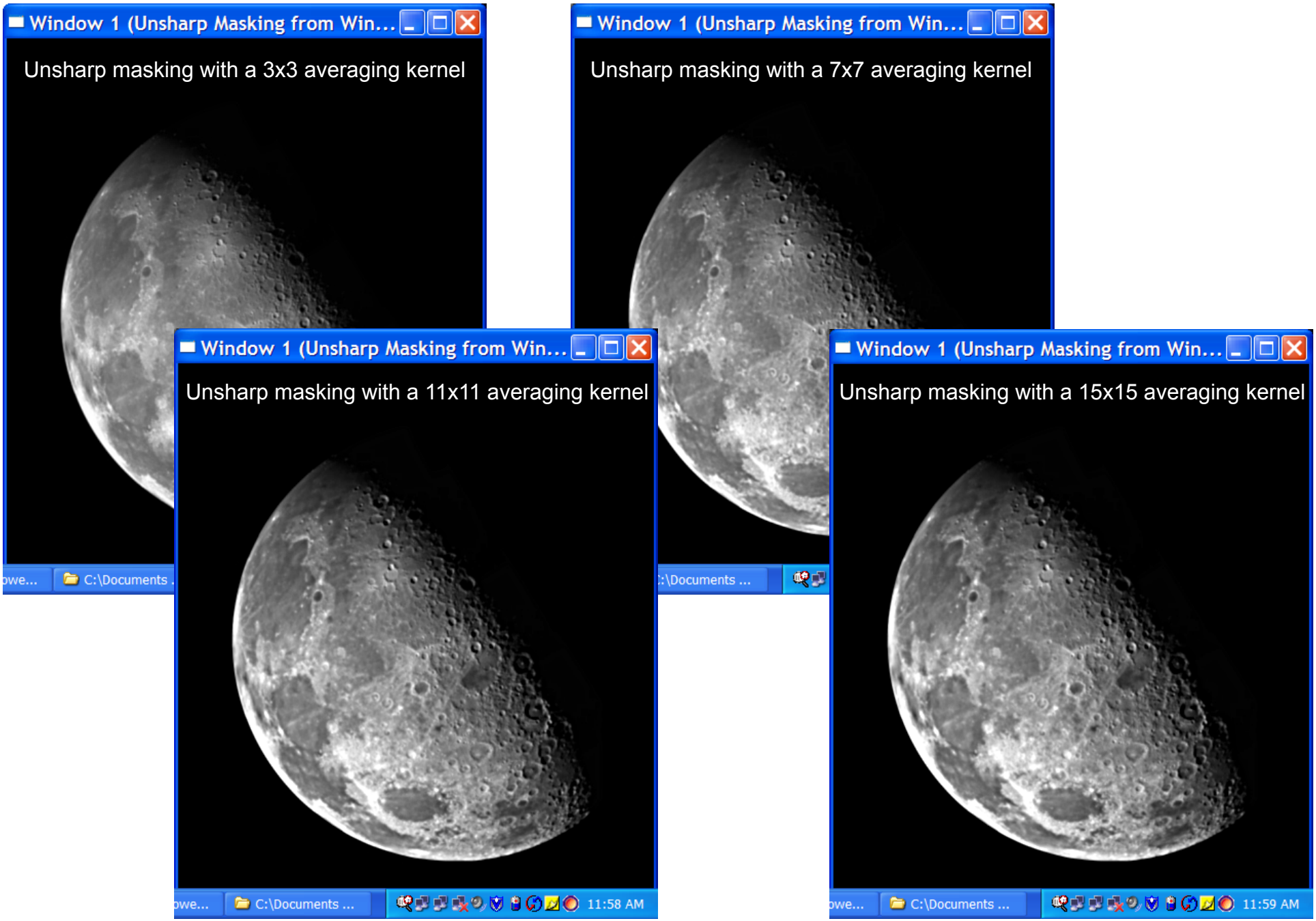
# Spatial Filtering

## Unsharp Masking

$$f_{sharpened}(x,y) = f(x,y) + \left[ f(x,y) - f_{blurred}(x,y) \right]$$




5x5 average


Window 1 (Unsharp Masking from Win...
Unsharp masking with a 3x3 averaging kernel
Window 1 (Unsharp Masking from Win...
Unsharp masking with a 7x7 averaging kernel
Window 1 (Unsharp Masking from Win...
Unsharp masking with a 11x11 averaging kernel
Window 1 (Unsharp Masking from Win...
Unsharp masking with a 15x15 averaging kernel
C:\Documents ...
11:58 AM
11:59 AM

# Gradient

## Definition

First derivatives in image processing are implemented using the magnitude of the gradient.

The gradient of f at the coordinate (x,y) is defined as the two-dimensional vector

$$\nabla\vec{f} = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

$$\begin{aligned} \nabla f &= mag(\nabla\vec{f}) \\ &= \left[G_x^2 + G_y^2\right]^{1/2} \\ &= \left[\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2\right]^{1/2} \end{aligned}$$

# Gradient

## Implementation

For computational efficiency

$$\nabla f = \left[ \left( \frac{\partial f}{\partial x} \right)^2 + \left( \frac{\partial f}{\partial y} \right)^2 \right]^{1/2}$$

is often implemented as

$$\nabla f \approx |G_x| + |G_y|$$

# Gradient

## Roberts Cross Gradient

| | | |
|---|---|---|
| $z_1$ | $z_2$ | $z_3$ |
| $z_4$ | $z_5$ | $z_6$ |
| $z_7$ | $z_8$ | $z_9$ |

$$G_x = (z_9 - z_5)$$
$$G_x = (z_8 - z_6)$$

By definition

$$\nabla f = \left[(z_9 - z_5)^2 + (z_8 - z_6)^2\right]^{1/2}$$

which is often implemented as

$$\nabla f \approx |z_9 - z_5| + |z_8 - z_6|$$

for computational efficiency

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | -1 | 0 |
| 0 | 0 | 1 |

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 0 | -1 |
| 0 | 1 | 0 |

# Gradient

## Sobel Gradient

| | | |
|---|---|---|
| $z_1$ | $z_2$ | $z_3$ |
| $z_4$ | $z_5$ | $z_6$ |
| $z_7$ | $z_8$ | $z_9$ |

$$G_x = (z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)$$
$$G_y = (z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)$$

By definition

$$\nabla f = \left[\left((z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)\right)^2 + \left((z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)\right)^2\right]^{1/2}$$

which is often implemented as

$$\nabla f \approx \left|(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)\right| + \left|(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)\right|$$

for computational efficiency

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

# Gradient

## Examples - Roberts Cross Gradient



Absolute Value


Magnitude

# Gradient

## Examples - Sobel Gradient

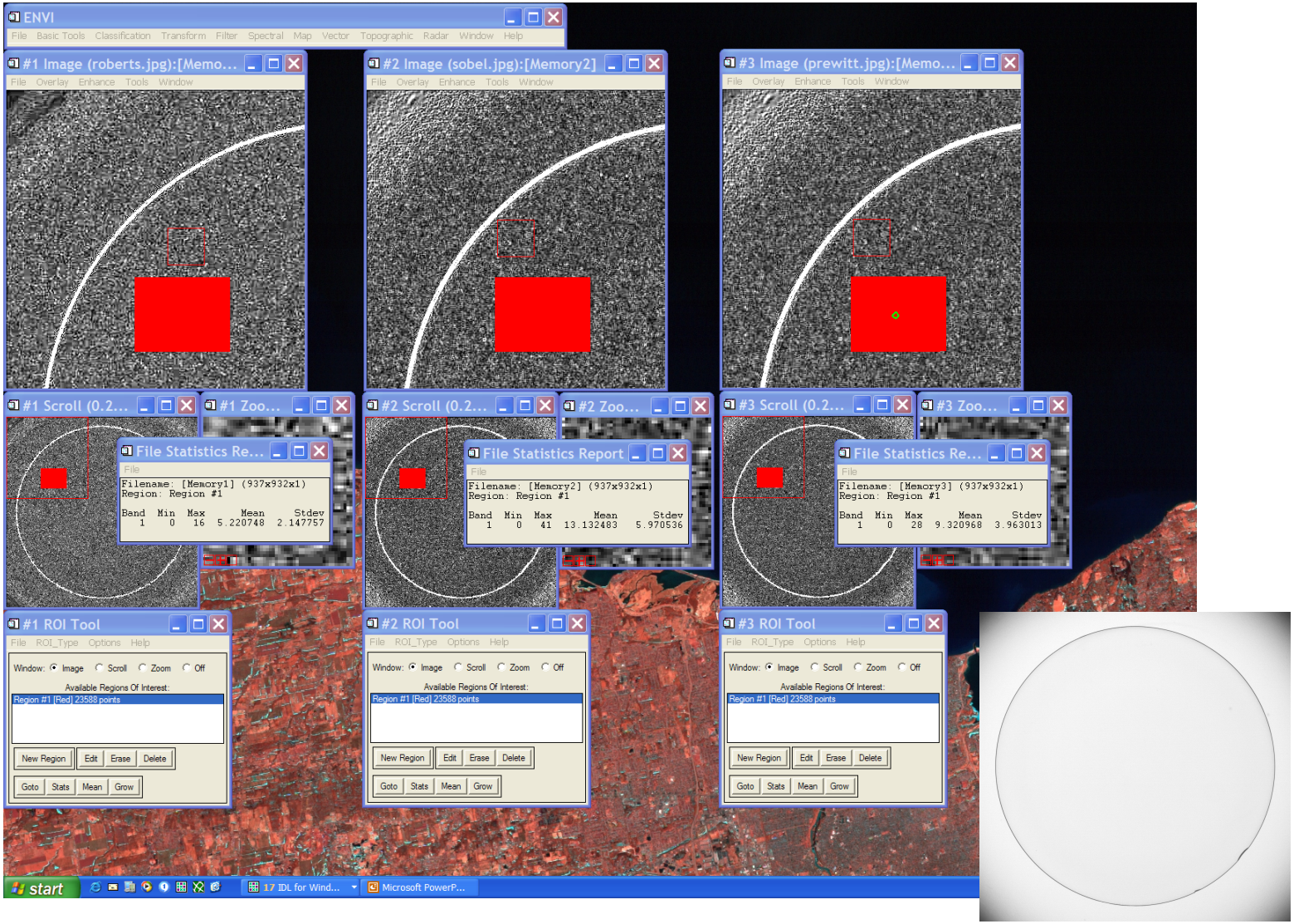


Absolute Value


Magnitude

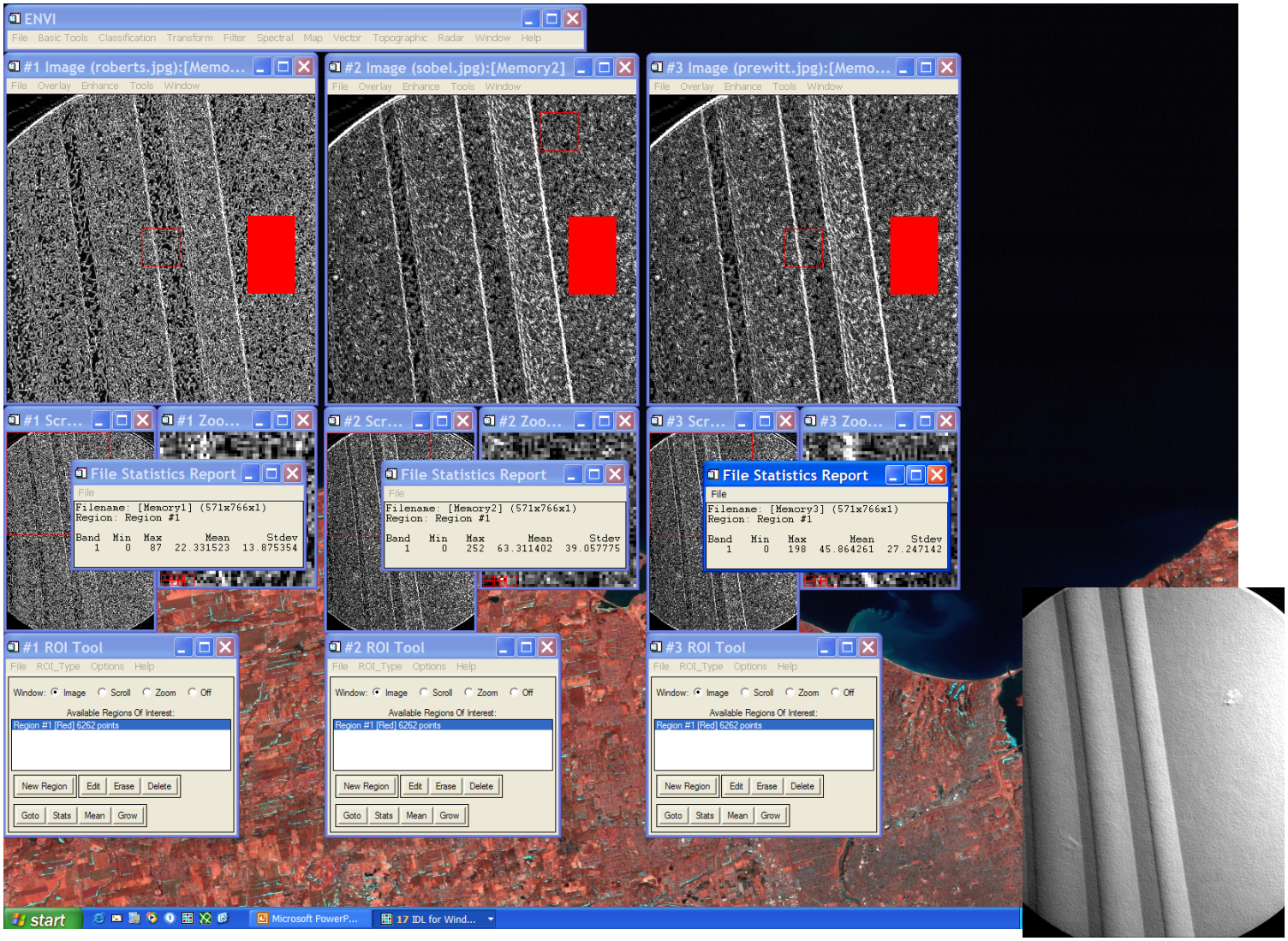
# Gradient

## Noise



# Gradient

## Noise



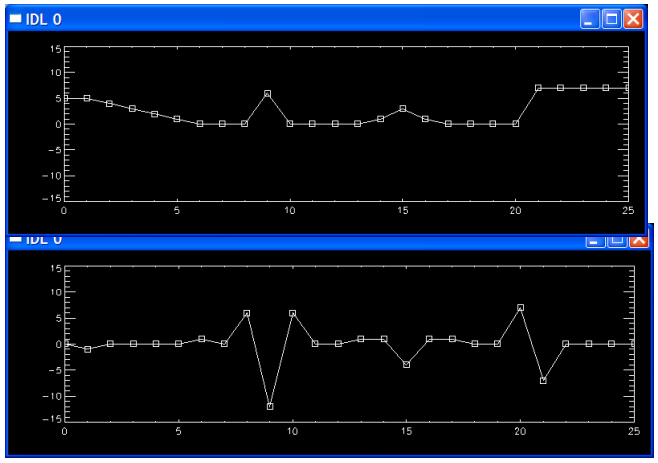
# Point Detection

## Introduction

- Let a *point* be defined as a single pixel whose value is *significantly different* than all of its neighbors
- *Detection* involves the segmentation of this single pixel discontinuity, typically accomplished by thresholding

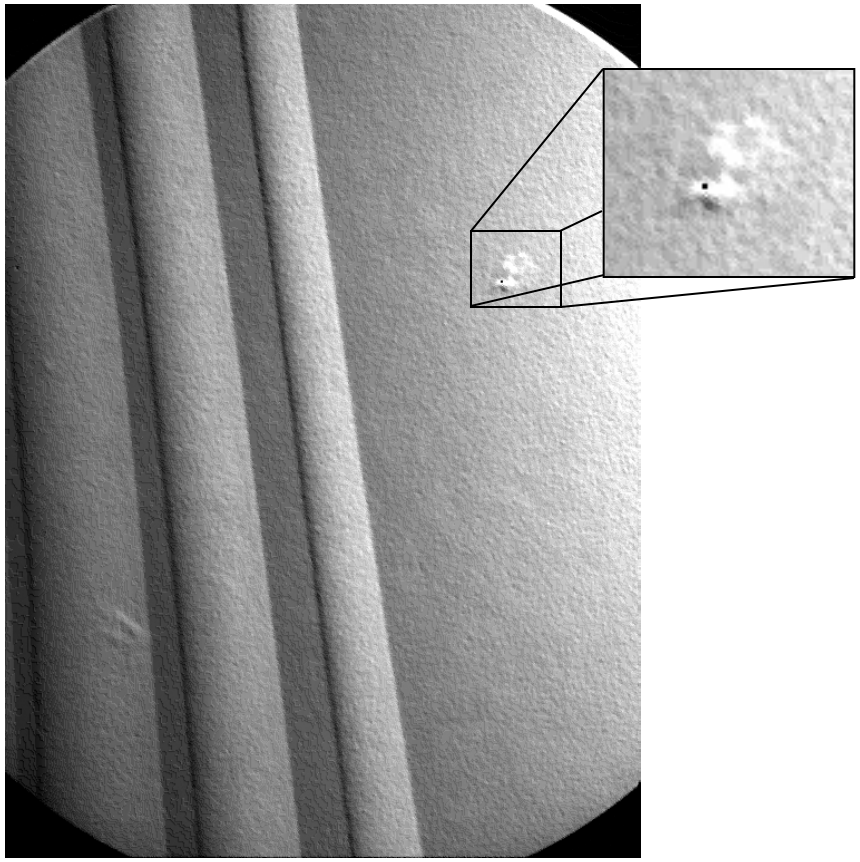
- Applications
  - Defect detection
  - Target location

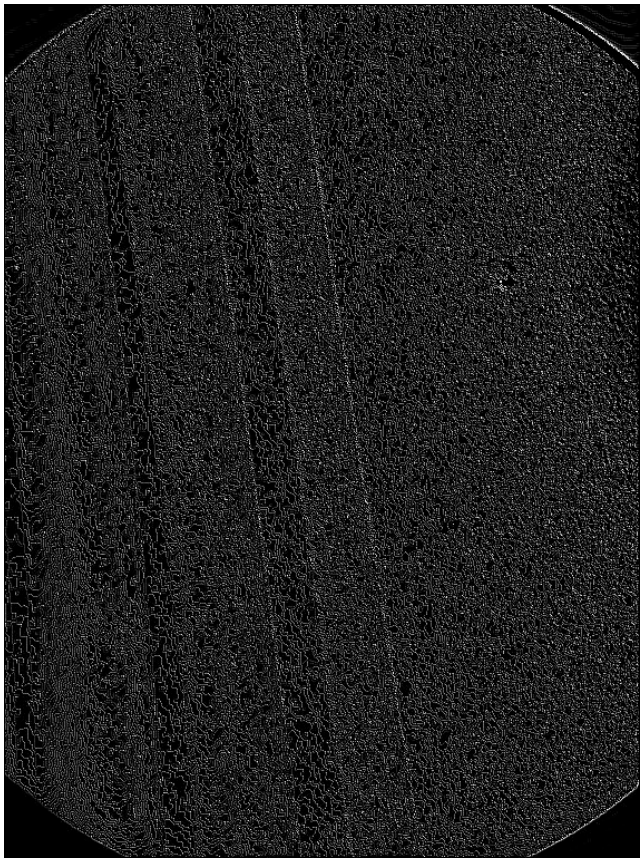
A second derivative linear operator (like the Laplacian) locates regions in the image where the grey level slope changes; a point involves a sudden change in grey level slope at the onset and end of the discontinuity



# Point Detection

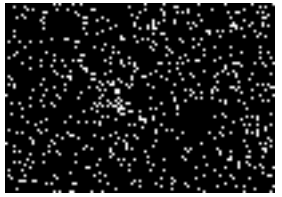
## Laplacian Filtering + Thresholding



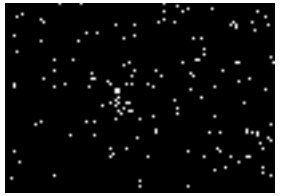


| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -8 | 1 |
| 1 | 1 | 1 |

|R|>0



|R|>128

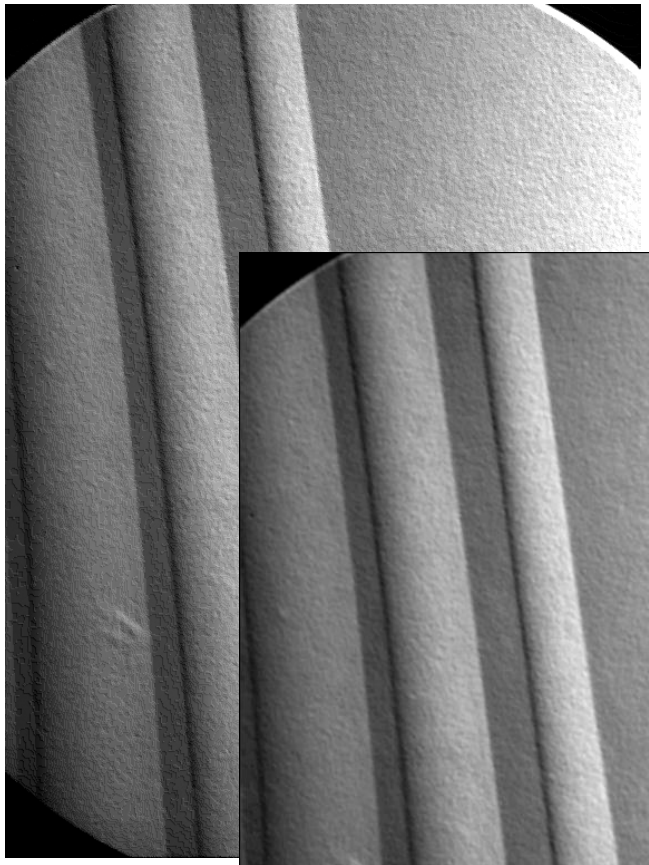


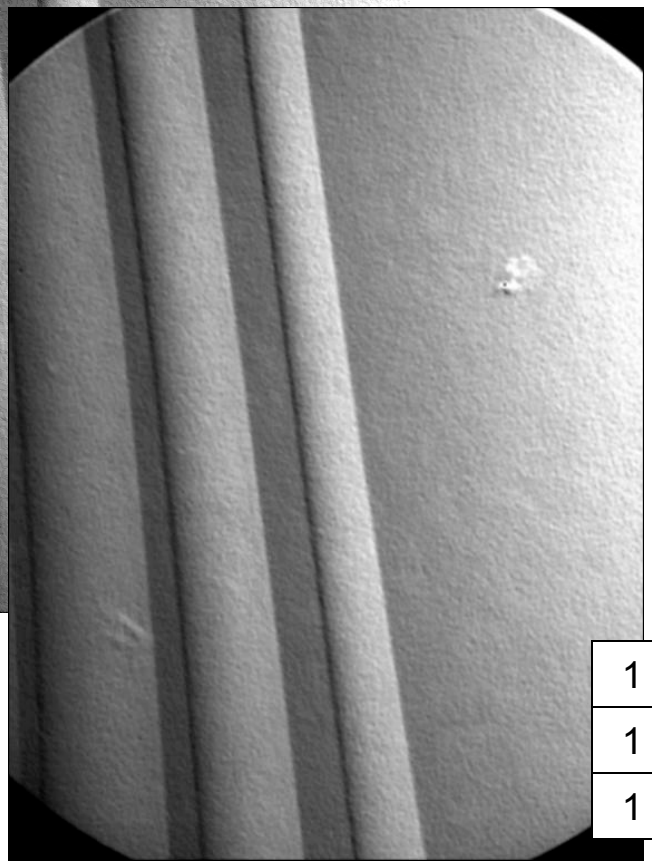
|R|>192

|R|>255

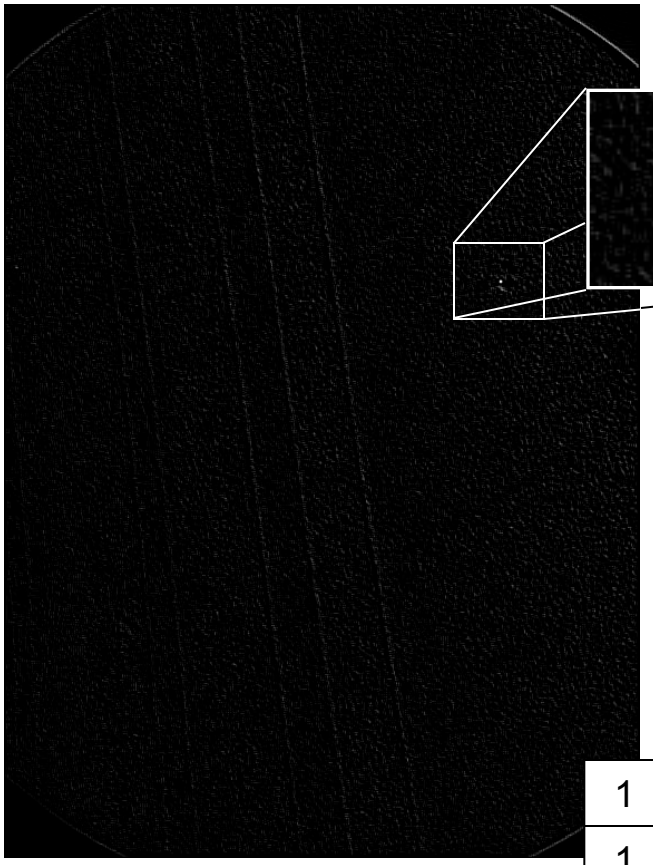
# Point Detection

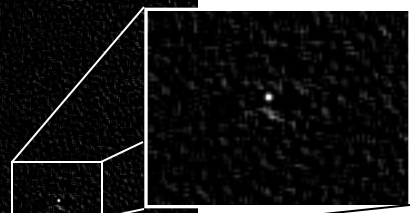
Smoothing + Laplacian Filtering + Thresholding





| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |





|R|>255

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | -8 | 1 |
| 1 | 1 | 1 |

# Line Detection

## Introduction

Using a first derivative operator, like a Sobel Filter, one can find and highlight pixels that comprise the borders of linear feature in an image

Image

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 5 |
| 1 | 1 | 1 | 1 | 1 | 1 | 5 | 1 |
| 1 | 1 | 1 | 1 | 1 | 5 | 1 | 1 |
| 1 | 1 | 1 | 1 | 5 | 1 | 1 | 1 |
| 1 | 1 | 1 | 5 | 1 | 1 | 1 | 1 |
| 1 | 1 | 5 | 1 | 1 | 1 | 1 | 1 |
| 1 | 5 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Sobel Diagonal Filter (45 degrees)

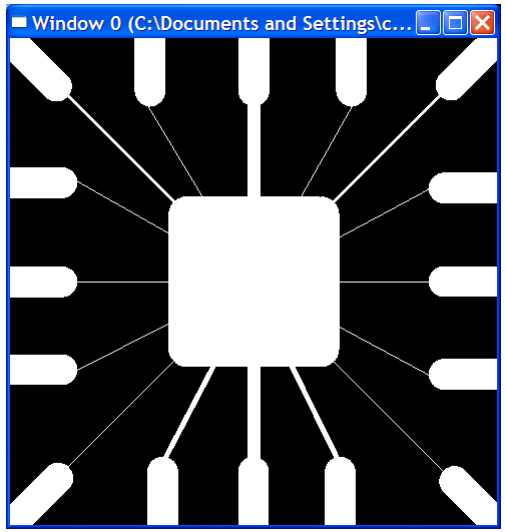
| | | |
|---|---|---|
| -2 | -1 | 0 |
| -1 | 0 | 1 |
| 0 | 1 | 2 |

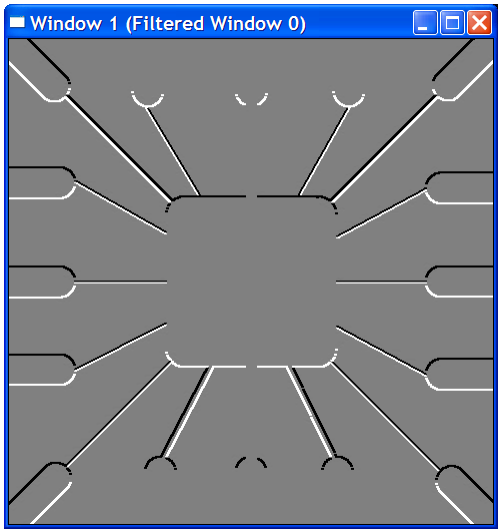
Response

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |
| | 0 | 0 | 0 | 8 | 8 | 0 | |
| | 0 | 0 | 8 | 8 | 0 | -8 | |
| | 0 | 8 | 8 | 0 | -8 | -8 | |
| | 8 | 8 | 0 | -8 | -8 | 0 | |
| | 8 | 0 | -8 | -8 | 0 | 0 | |
| | 0 | -8 | -8 | 0 | 0 | 0 | |
| | | | | | | | |

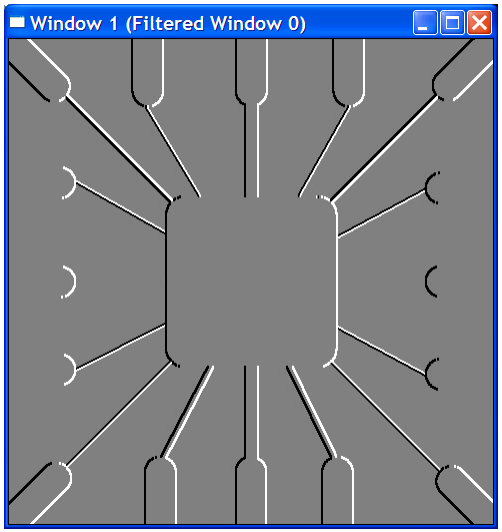
# Line Detection

## Sobel Horizontal/Vertical Filters



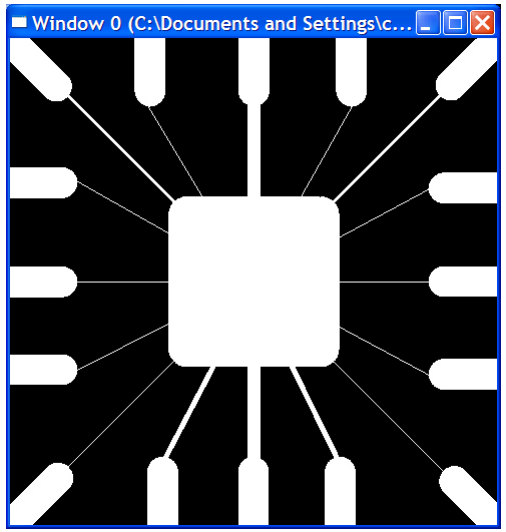

| | | |
|---|---|---|
| -1 | -2 | -1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

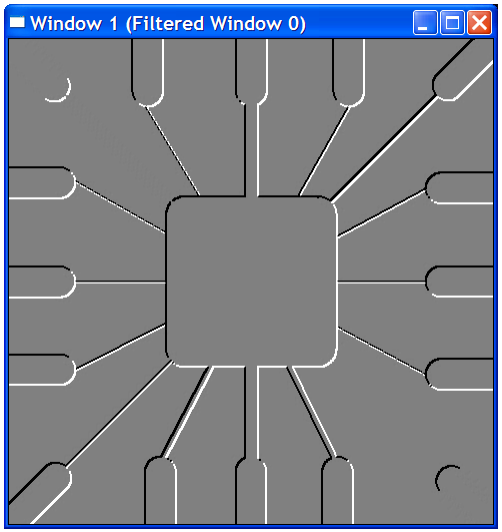



| | | |
|---|---|---|
| -1 | 0 | 1 |
| -2 | 0 | 2 |
| -1 | 0 | 1 |

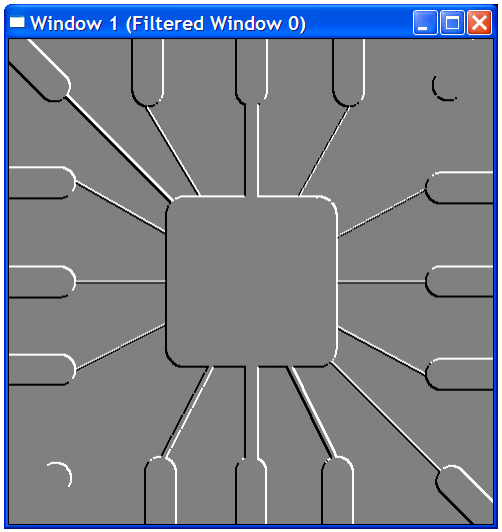
# Line Detection

## Sobel Diagonal Filters




| | | |
|---|---|---|
| -2 | -1 | 0 |
| -1 | 0 | 1 |
| 0 | 1 | 2 |




| | | |
|---|---|---|
| 0 | 1 | 2 |
| -1 | 0 | 1 |
| -2 | -1 | 0 |

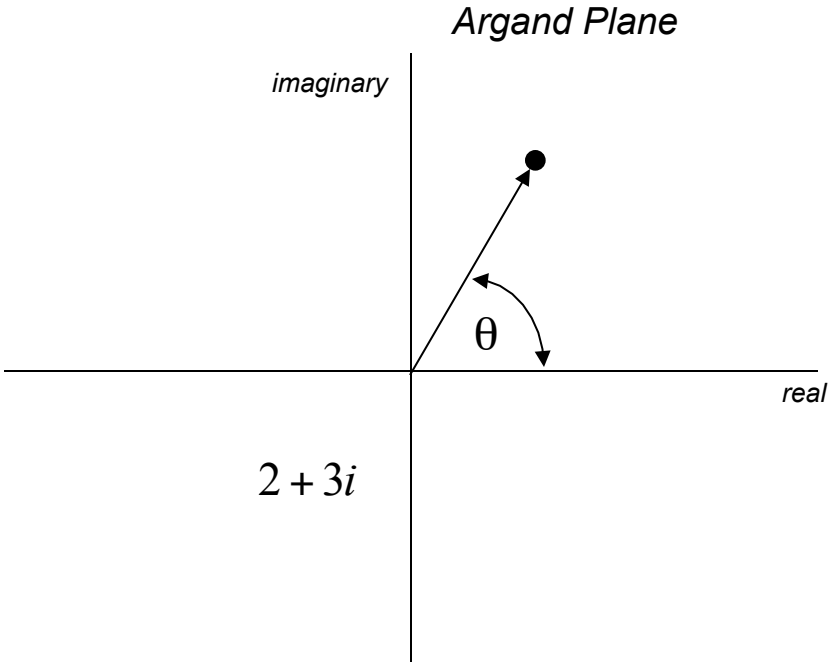
# Frequency Domain

## Complex Numbers

$$i = \sqrt{-1}$$

$$real + imaginary$$

$$0 + 1i$$




$$|c| = \sqrt{c_{real}^2 + c_{imaginary}^2}$$

$$\theta = \tan^{-1}\left(\frac{c_{imaginary}}{c_{real}}\right)$$

$$c_{real} = |c|\cos\theta$$

$$c_{imaginary} = |c|\sin\theta$$

$$a + b = (a_{real} + b_{real}) + i(a_{imaginary} + b_{imaginary})$$

$$a - b = (a_{real} - b_{real}) + i(a_{imaginary} - b_{imaginary})$$

$$a \cdot b = (a_{real} + ia_{imaginary}) \cdot (b_{real} + ib_{imaginary})$$

$$= (a_{real} \cdot b_{real}) + i(a_{real} \cdot b_{imaginary}) + i(a_{imaginary} \cdot b_{real}) + i^2(a_{imaginary} \cdot b_{imaginary})$$

$$= (a_{real} \cdot b_{real} - a_{imaginary} \cdot b_{imaginary}) + i(a_{real} \cdot b_{imaginary} + a_{imaginary} \cdot b_{real})$$

# Frequency Domain

## Introduction

*Fourier Series*

... any function that periodically repeats itself can be expressed as the sum of sines and/or cosines of different frequencies, each multiplied by a different coefficient.

*Jean Baptiste Joseph Fourier, 1822*
*La Theorie Analitique de la Chaleur (The Analytic Theory of Heat)*

*Fourier Transform*

Even functions that are not periodic but whose area under the curve is finite can be expressed as the integral of sines and/or cosines multiplied by a weighting function.

# Frequency Domain

## Fourier Transform

For a continuous function

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i 2\pi u x} dx$$

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{i 2\pi u x} du$$

For a discrete function

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) e^{-i 2\pi u x / M} \qquad \text{for } u = 0, 1, 2, ..., M\text{-}1$$

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{i 2\pi u x / M} \qquad \text{for } x = 0, 1, 2, ..., M\text{-}1$$

# Frequency Domain

## Fourier Transform

$$e^{i\theta} = \cos\theta + i\sin\theta$$ Euler's formula

For physical clarity, the discrete Fourier transform (DFT) can be rewritten using Euler's formula and the fact that *cos(-θ) = cos(θ)*, where the DFT

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)e^{-i2\pi ux/M}$$

becomes

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)\left[\cos\left(2\pi ux/M\right) - i\sin\left(2\pi ux/M\right)\right]$$

for $u = 0, 1, 2, ..., M\text{-}1$

# Frequency Domain

## Fourier Transform

$$F(u) = \frac{1}{M} \sum_{x=0}^{M-1} f(x) \left[ \cos\left(2\pi ux / M\right) - i \sin\left(2\pi ux / M\right) \right]$$

so each term of the Fourier transform, $F(u)$, is composed of the sum of all values of the function $f(x)$, multiplied by sines and cosines of various frequencies, $u$.

The range of frequencies ($u's$) over which the values of $F(u)$ are defined is referred to as the *frequency domain* for the function, and each value of $F(u)$ is referred to as a *frequency component*.

# Frequency Domain

## Fourier Transform - Example Computation

| | | ***u*** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ***x*** | ***f(x)*** | 0 | | 1 | | 2 | | 3 | | 4 | |
| 0 | 0 | 0.0000 | 0.0000i | 0.0000 | 0.0000i | 0.0000 | 0.0000i | 0.0000 | 0.0000i | 0.0000 | 0.0000i |
| 1 | 1 | 1.0000 | 0.0000i | 0.3090 | -0.9511i | -0.8090 | -0.5878i | -0.8090 | 0.5878i | 0.3090 | 0.9511i |
| 2 | 3 | 3.0000 | 0.0000i | -2.4271 | -1.7634i | 0.9271 | 2.8532i | 0.9271 | -2.8532i | -2.4271 | 1.7634i |
| 3 | 1 | 1.0000 | 0.0000i | -0.8090 | 0.5878i | 0.3090 | -0.9511i | 0.3090 | 0.9511i | -0.8090 | -0.5878i |
| 4 | 0 | 0.0000 | 0.0000i | 0.0000 | 0.0000i | 0.0000 | 0.0000i | 0.0000 | 0.0000i | 0.0000 | 0.0000i |
| | ***F(u)*** | 1.0000 | 0.0000i | -0.5854 | -0.4253i | 0.0854 | 0.2629i | 0.0854 | -0.2629i | -0.5854 | 0.4253i |

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)\left[\cos\left(2\pi ux/M\right) - i\sin\left(2\pi ux/M\right)\right]$$

for $u = 0, 1, 2, \ldots, M\text{-}1$

# Frequency Domain

## Fourier Transform

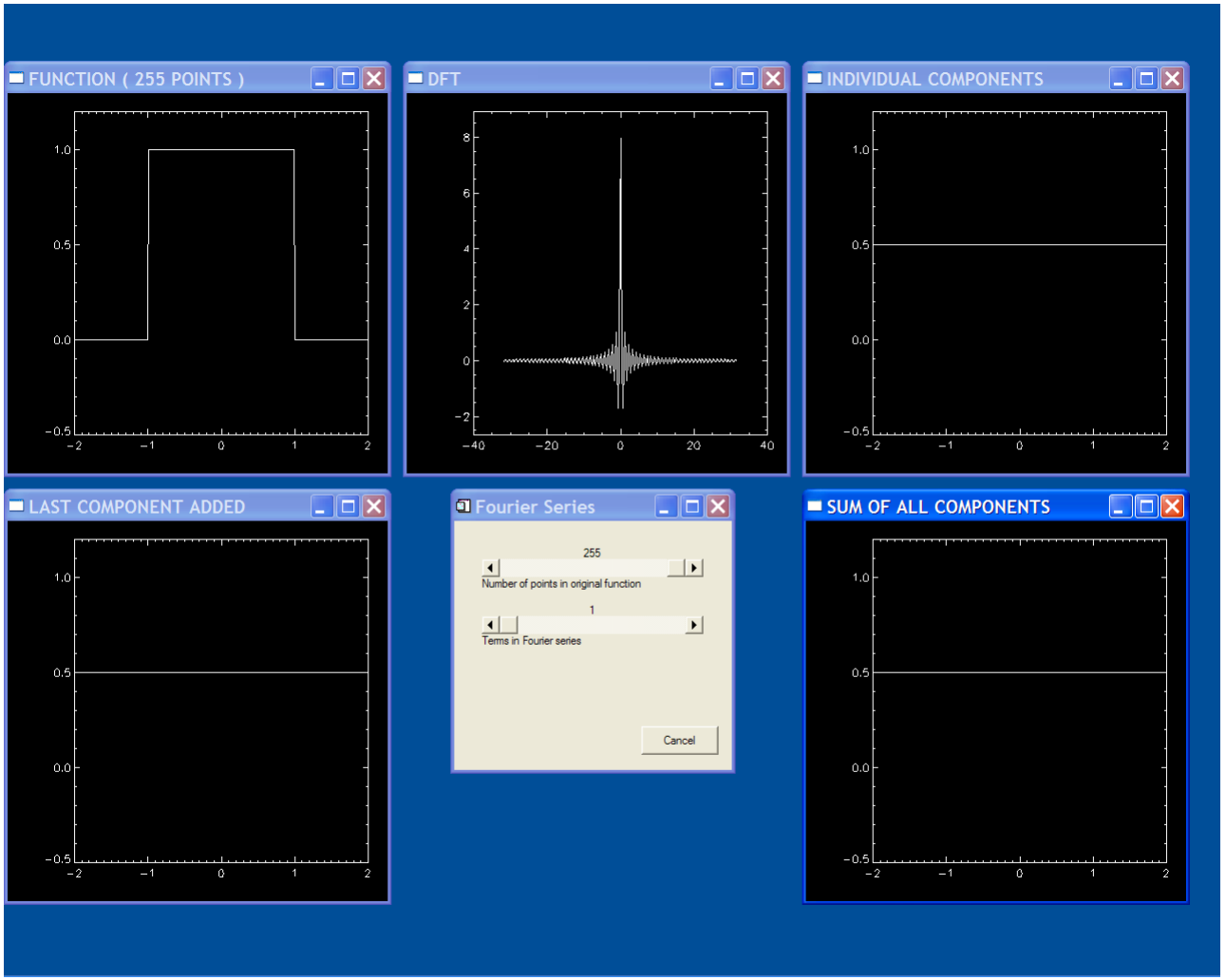
Given the relationship between the forward and inverse Fourier transforms, the dimensional terms for the units on these functions are inversely related as

$$\Delta u = \frac{1}{M\Delta x}$$

which proves relevant when measurements have physical implications with regard to the sample being imaged.

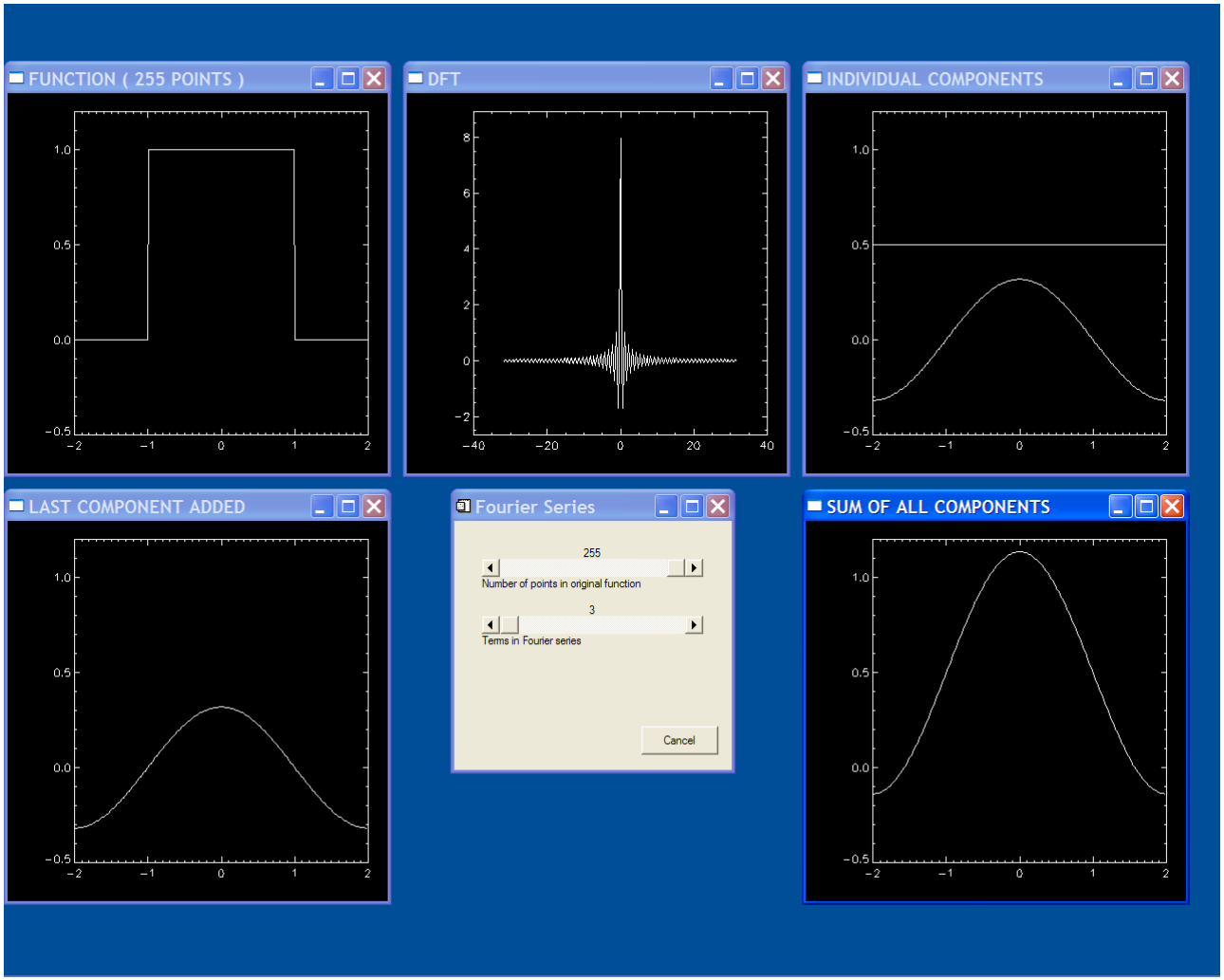
# Frequency Domain

## Discrete Fourier Transform - DC Component



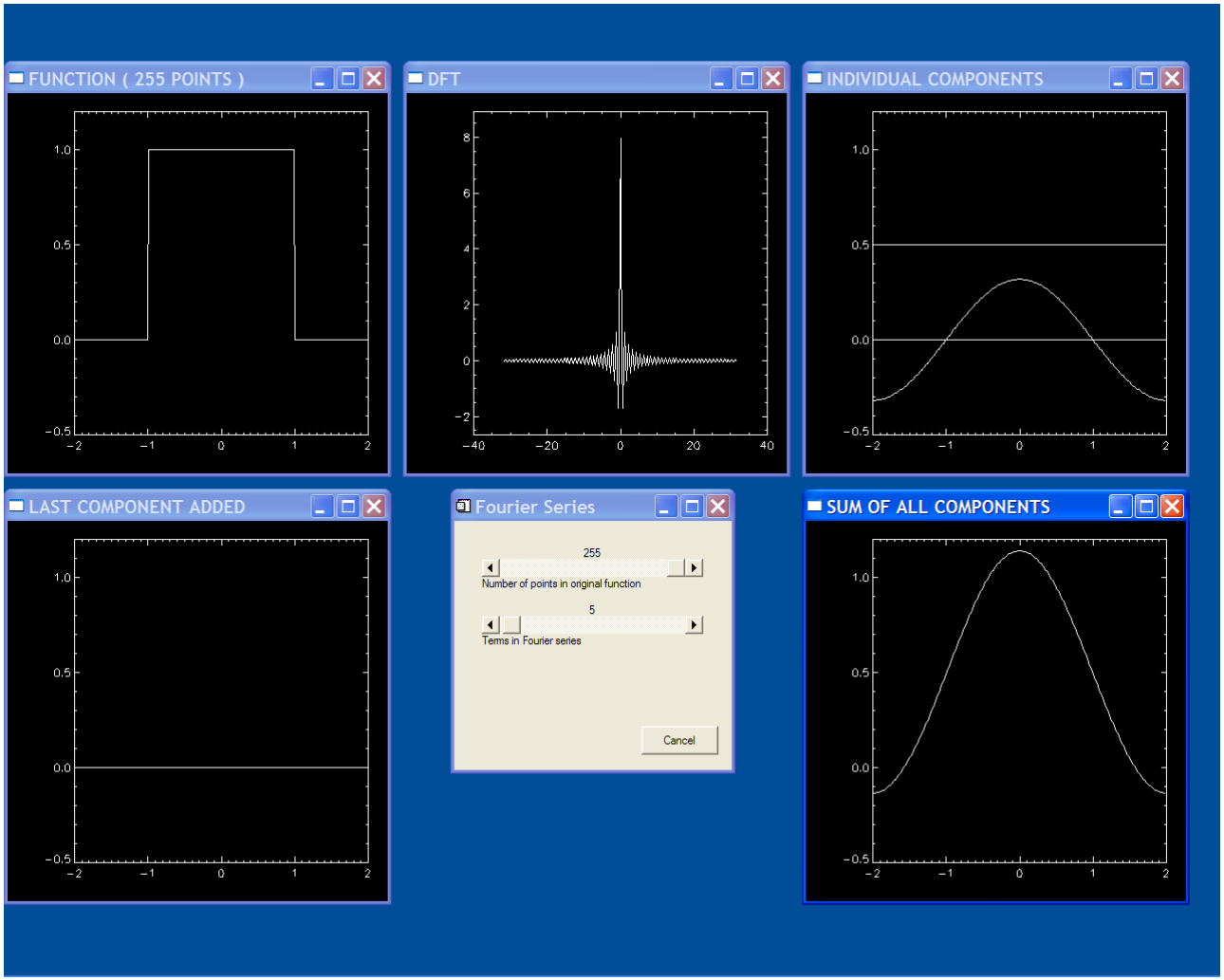
# Frequency Domain

## Discrete Fourier Transform - 3 Components



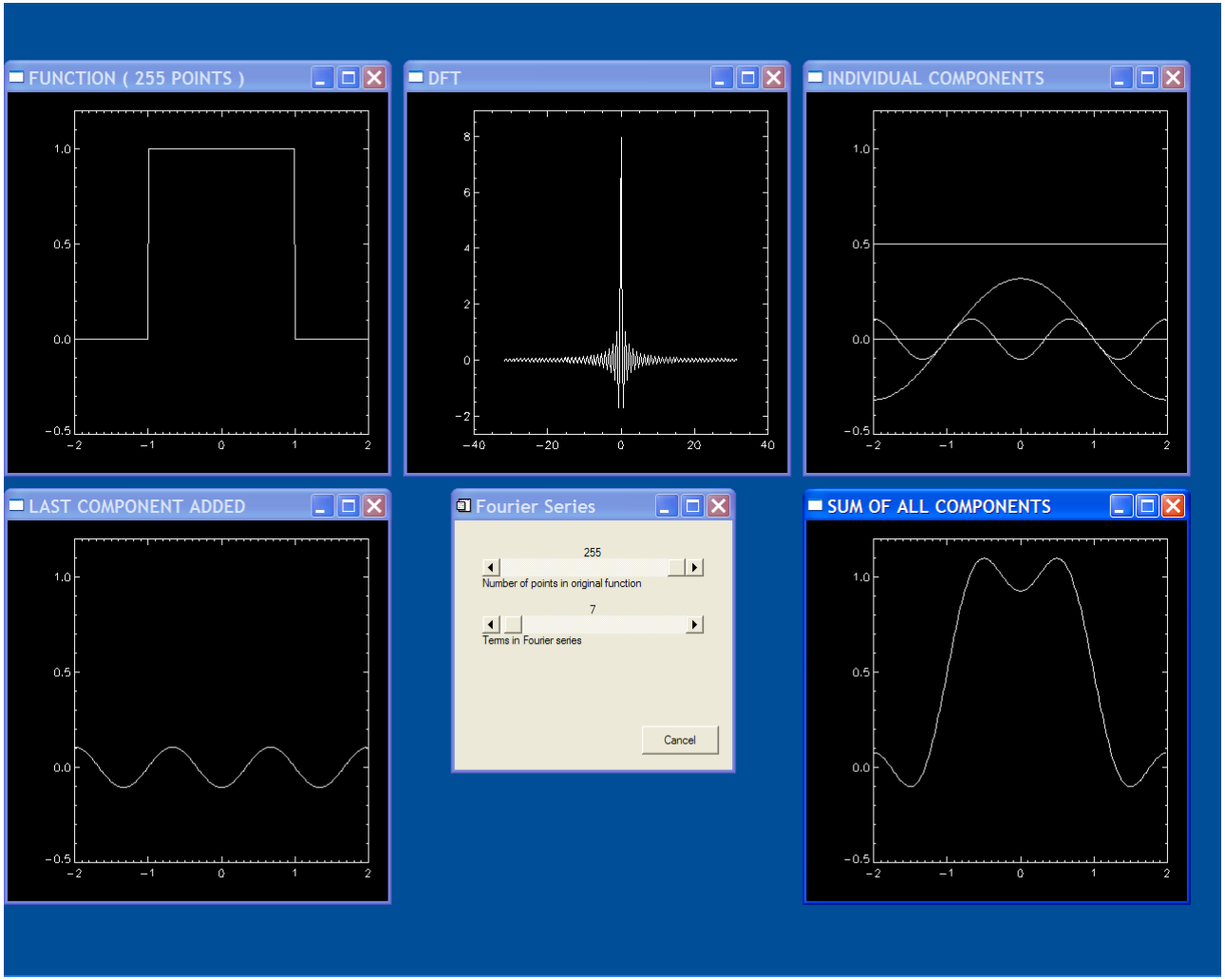
# Frequency Domain

## Discrete Fourier Transform - 5 Components (Zero Crossing)



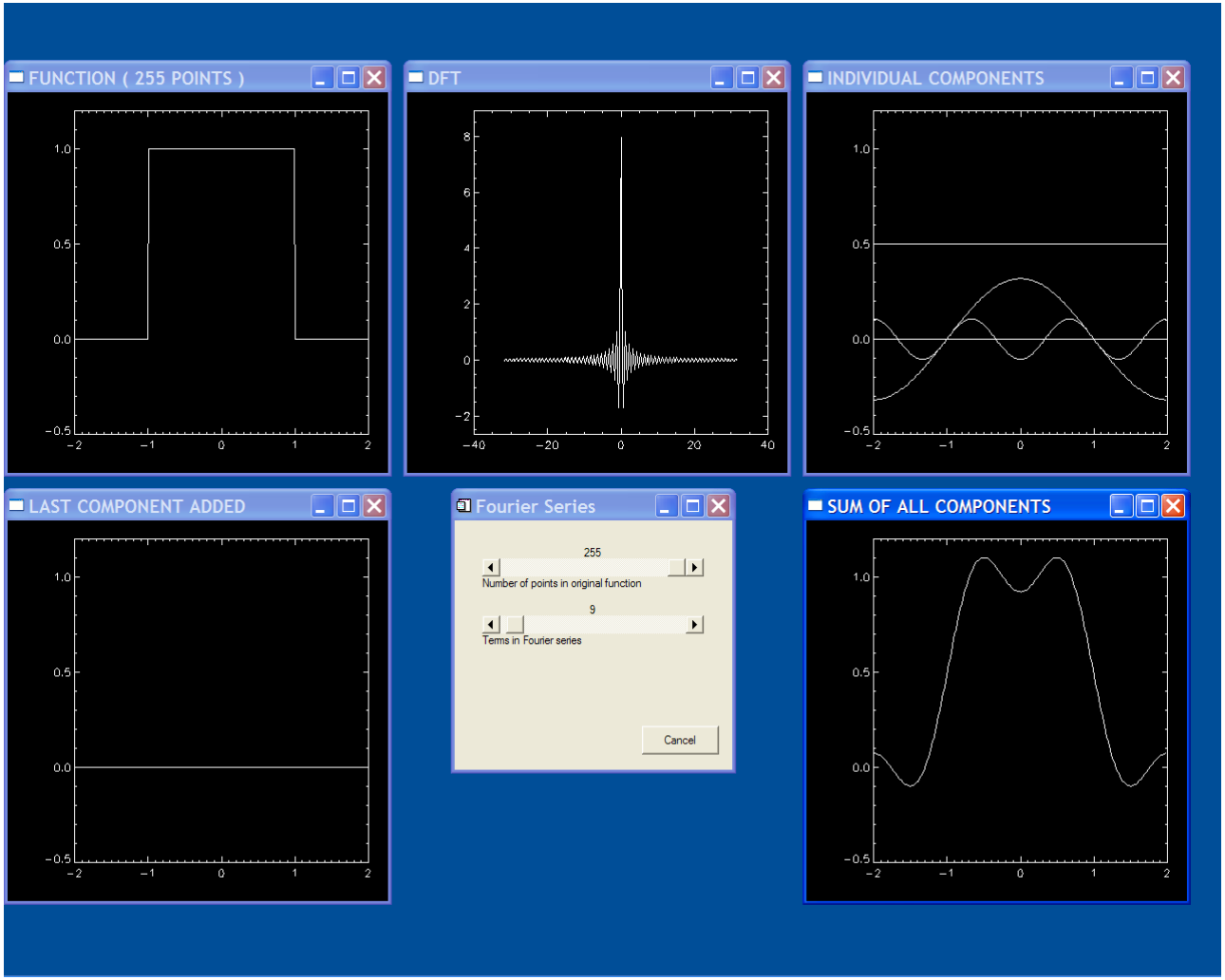
# Frequency Domain

## Discrete Fourier Transform - 7 Components



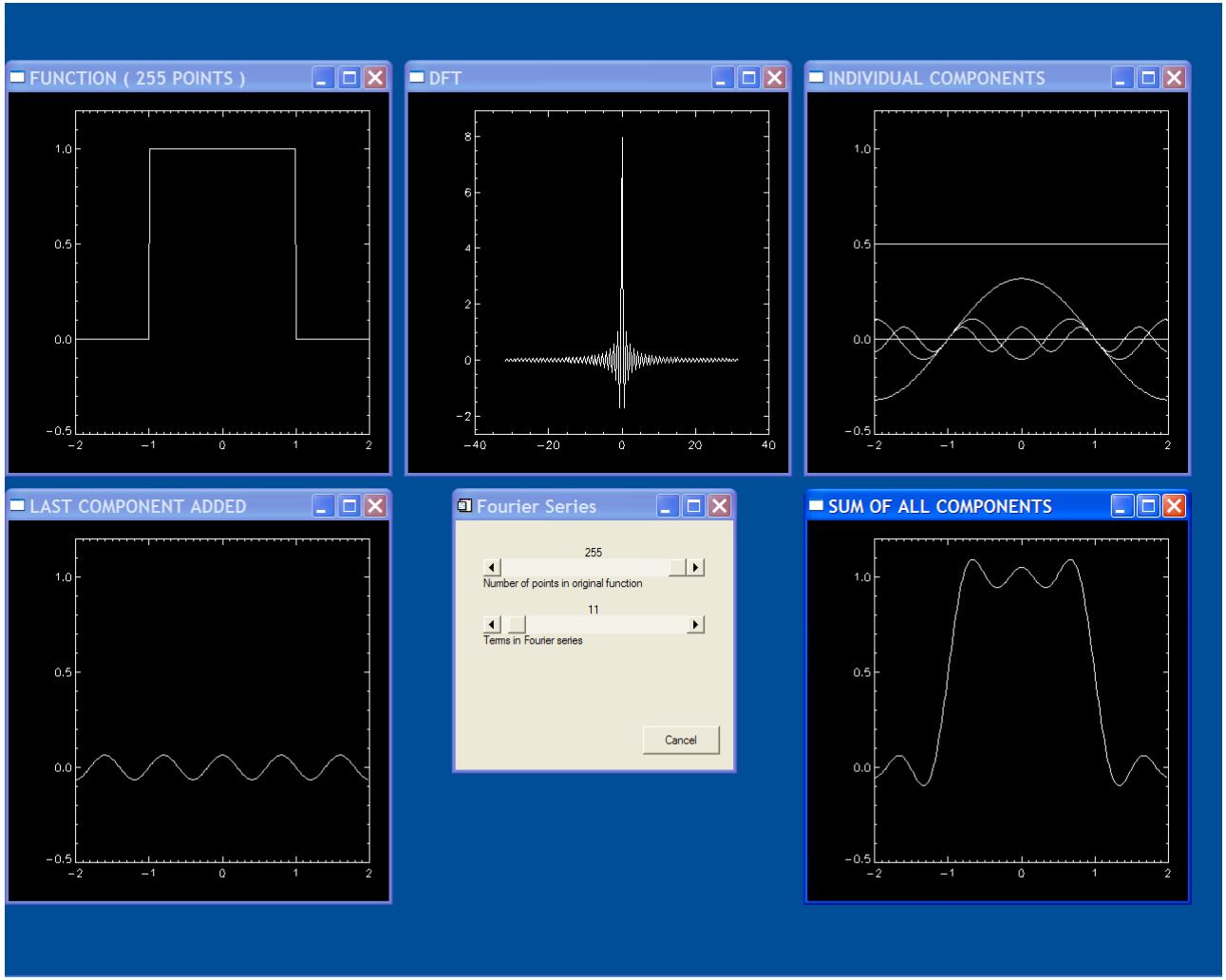
# Frequency Domain

## Discrete Fourier Transform - 9 Components (Zero Crossing)



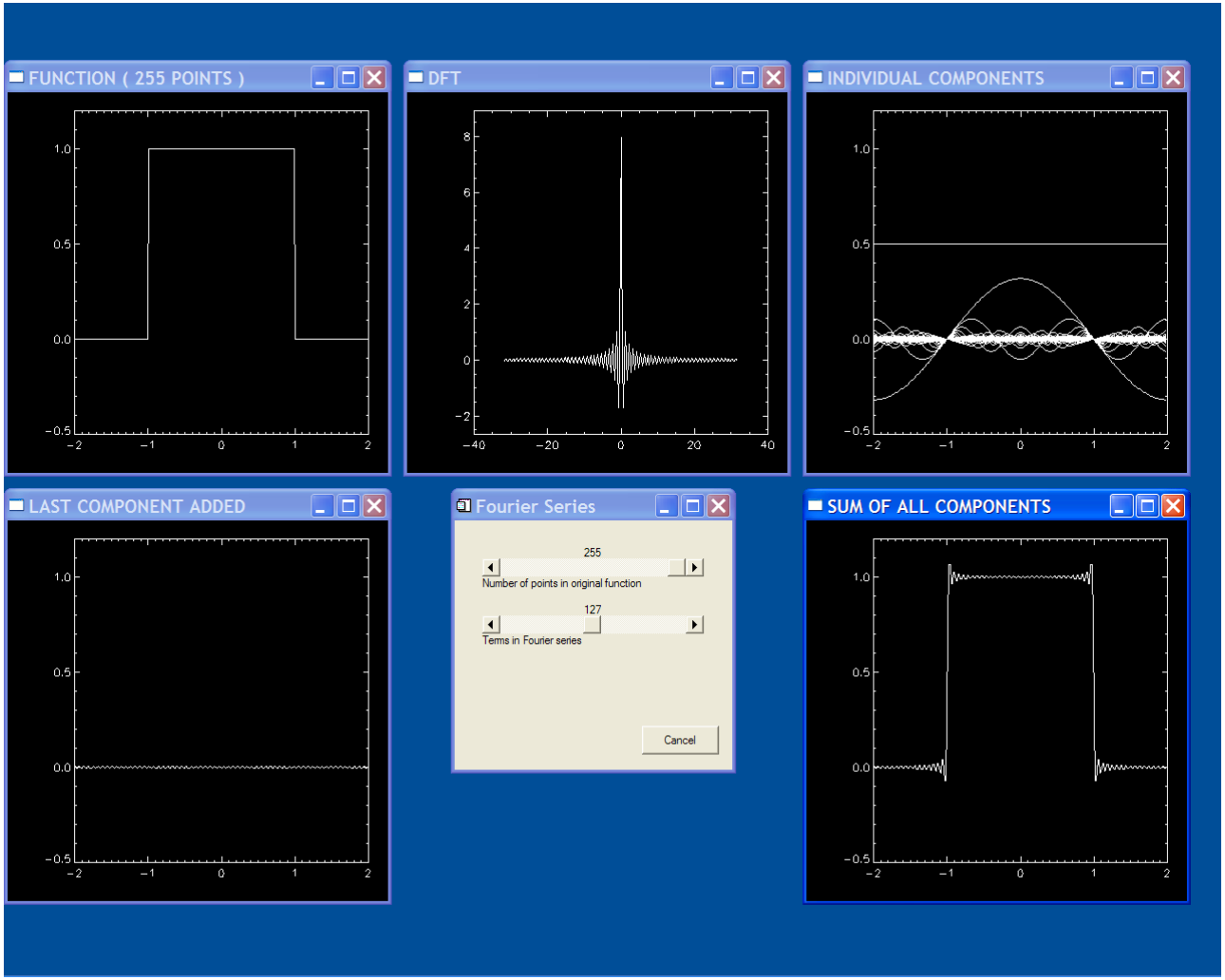
# Frequency Domain

## Discrete Fourier Transform - 11 Components



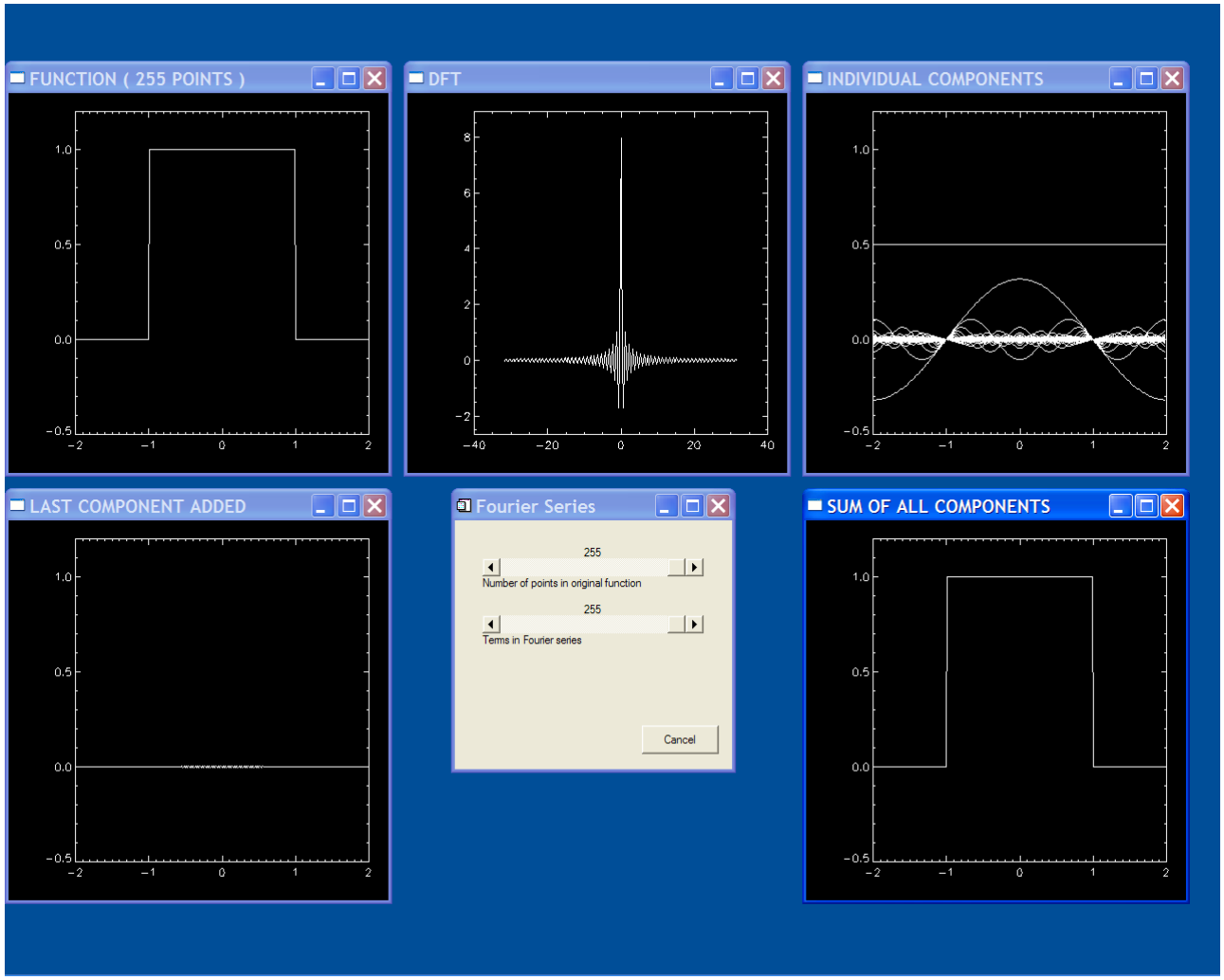
# Frequency Domain

## Discrete Fourier Transform - 127 Components



# Frequency Domain

## Discrete Fourier Transform - 255 Components (Complete Series)



# Frequency Domain

## Two-Dimensional Discrete Fourier Transform Pair

$$F(u,v) = \frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-i2\pi\left(ux/M + vy/N\right)}$$

for $u = 0, 1, 2, ..., M\text{-}1$ and $v = 0, 1, 2, ..., N\text{-}1$

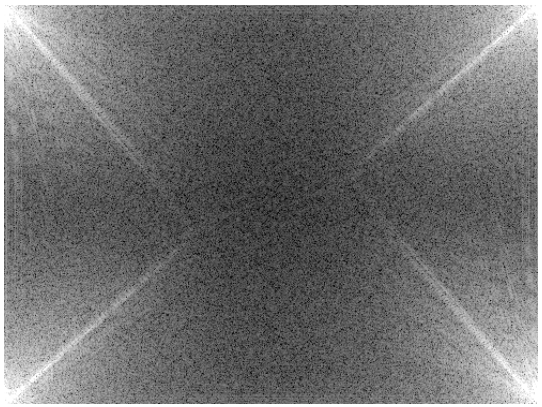
$$f(x,y) = \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{i2\pi\left(ux/M + vy/N\right)}$$

for $u = 0, 1, 2, ..., M\text{-}1$ and $v = 0, 1, 2, ..., N\text{-}1$

# Frequency Domain

## Two Dimensional Discrete Fourier Transform - Computational Tricks

A forward discrete Fourier transform computed on an image with the coordinate $(0,0)$ in the upper left hand corner, will produce a frequency domain spectrum that appears as



One can cause the DC component to appear in the center of the transform by shifting the coordinates of the function, $f(x,y)$, by an amount equal to one-half the image size in each dimension prior to computing the component at each frequency, but this adds more time to an already lengthy computation. We could take advantage of the *translation property* of the Fourier transform pair to be more efficient.

# Frequency Domain

## Two Dimensional Discrete Fourier Transform - Computational Tricks

Translation property of the Fourier transform pair

$$f(x,y)e^{i2\pi\left(u_0x/M + v_0y/N\right)} \Leftrightarrow F(u-u_0,v-v_0)$$

$$f(x-x_0,y-y_0) \Leftrightarrow F(u,v)e^{-i2\pi\left(ux_0/M + vy_0/N\right)}$$

If you let $u_0=M/2$ and $v_0=N/2$ then

$$\begin{aligned} e^{i2\pi\left(u_0x/M + v_0y/N\right)} &= e^{i2\pi\left(\frac{M}{2}x/M + \frac{N}{2}y/N\right)} \\ &= e^{i2\pi\left(\frac{1}{2}\left[Mx/M + Ny/N\right]\right)} \\ &= e^{i\pi(x+y)} \\ &= \left[\cos\pi + i\sin\pi\right]^{(x+y)} \\ &= (-1)^{(x+y)} \end{aligned}$$

and the Fourier transform pairs become

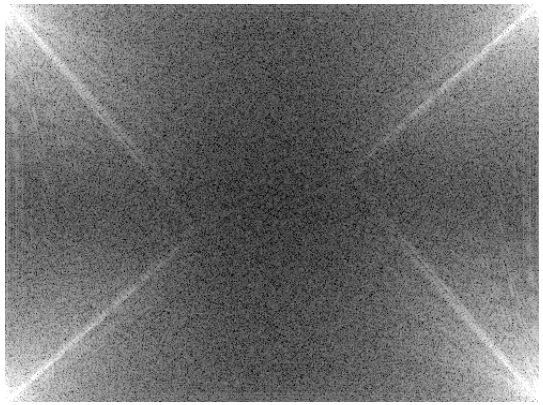
$$f(x,y)(-1)^{(x+y)} \Leftrightarrow F\left(u-\frac{M}{2},v-\frac{N}{2}\right)$$

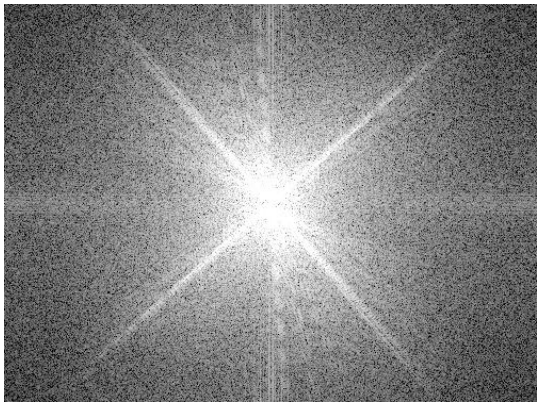
$$f\left(x-\frac{M}{2},y-\frac{N}{2}\right) \Leftrightarrow F(u,v)(-1)^{(u+v)}$$

# Frequency Domain

## Two Dimensional Discrete Fourier Transform - Computational Tricks

This results in a rearrangement of the un-translated transform to have the DC component appear in the center





As indicated on the previous slide, this is also true for the inverse Fourier transform pair

# Frequency Domain

## Two Dimensional Discrete Fourier Transform - Computational Tricks

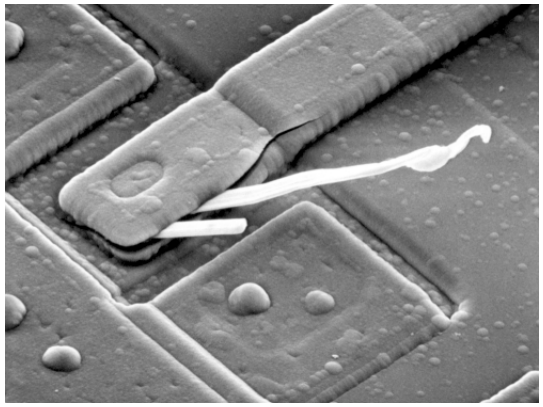
If *f(x,y)* is real, then the Fourier transform is conjugate symmetric

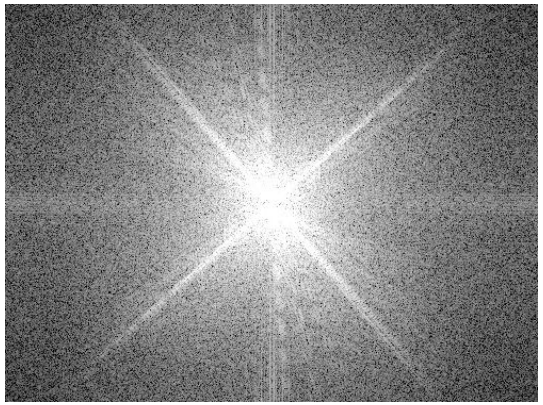
$$F[u,v] = F^*[-u,-v]$$

from which it follows that

$$|F(u,v)| = |F(-u,-v)|$$

which says that the Fourier spectrum is symmetric. This conjugate symmetry and the centering property can reduce the computational requirements by half.





# Frequency Domain

## Two Dimensional Discrete Fourier Transform - Computational Tricks

The Fourier transform of *f(x,y)*

$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi\left(ux/M + vy/N\right)}$$

for $u = 0, 1, 2, ..., M\text{-}1$ and $v = 0, 1, 2, ..., N\text{-}1$

can be rewritten in the separable form

$$F(u,v) = \frac{1}{M}\sum_{x=0}^{M-1} e^{-i2\pi ux/M} \cdot \frac{1}{N}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi vy/N}$$

$$= \frac{1}{M}\sum_{x=0}^{M-1} e^{-i2\pi ux/M} \cdot F(x,v)$$

$$= \frac{1}{M}\sum_{x=0}^{M-1} F(x,v)e^{-i2\pi ux/M}$$

# Frequency Domain

## Two Dimensional Discrete Fourier Transform - Computational Tricks

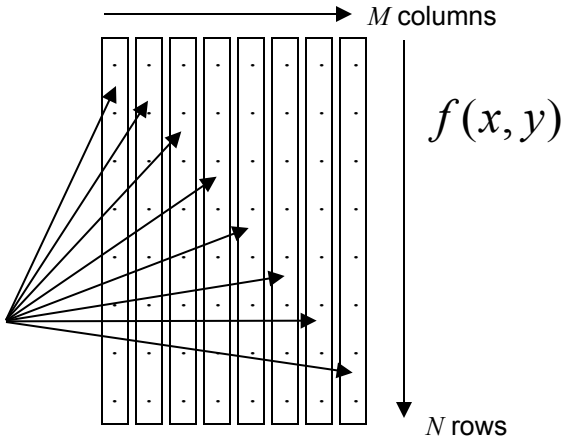
In the previous equation, $F(x,v)$, is written as

$$F(x,v) = \frac{1}{N}\sum_{y=0}^{N-1} f(x,y)e^{-i2\pi vy/N}$$

for each value of $x$ and $v = 0, 1, 2, ..., N-1$

and is a complete one-dimensional discrete Fourier transform along one column of $f(x,y)$. That is $F(x_0,v)$ is the one-dimensional discrete Fourier transform of the first column of the image for $v = 0, 1, 2, ..., N-1$. In all of these one-dimensional transforms on columns, the frequency value $u$ remains constant.

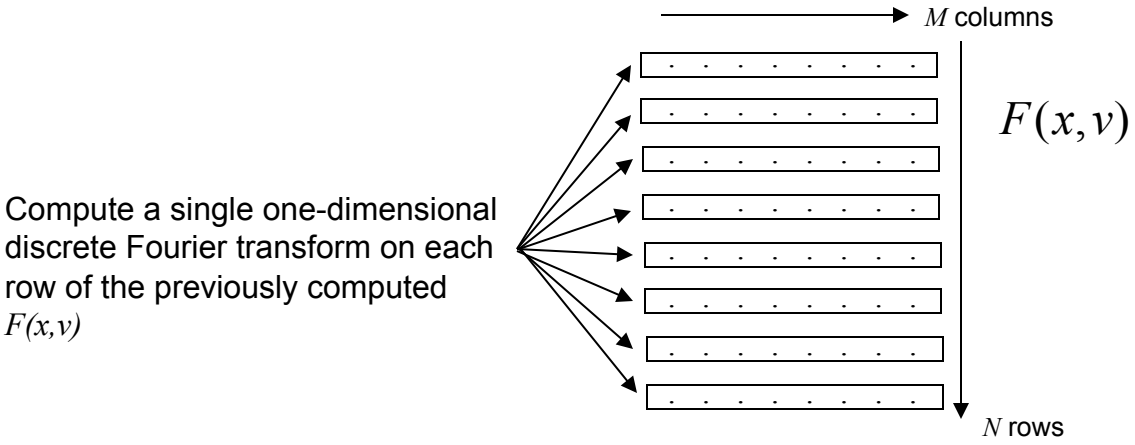
Compute a single one-dimensional discrete Fourier transform on each column



# Frequency Domain

## Two Dimensional Discrete Fourier Transform - Computational Tricks

Now, to complete the two-dimensional discrete Fourier transform, the frequency variable $u$ needs to vary over its range from $0$ to $M-1$. This is done in a similar fashion by computing the one-dimensional discrete Fourier transforms along each row of $F(x,v)$.




Compute a single one-dimensional discrete Fourier transform on each row of the previously computed $F(x,v)$

With this approach we reduce the computational load from

$$n_{multiplies} = (M \cdot N) \cdot (M \cdot N) = (100 \cdot 100) \cdot (100 \cdot 100) = 10{,}000 \cdot 10{,}000 = 100{,}000{,}000$$

to

$$n_{multiplies} = (N^2 \cdot M) + (M^2 \cdot N) = (100^2 \cdot 100) + (100^2 \cdot 100) = 1{,}000{,}000 + 1{,}000{,}000 = 2{,}000{,}000$$

# Frequency Domain

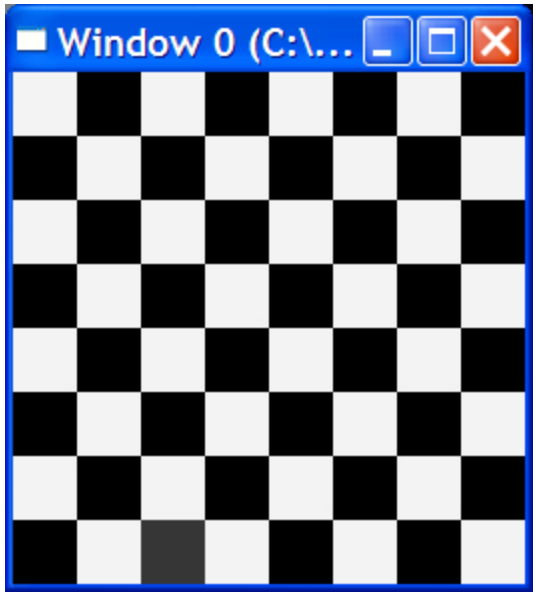
## Two Dimensional Discrete Fourier Transform - Computational Tricks

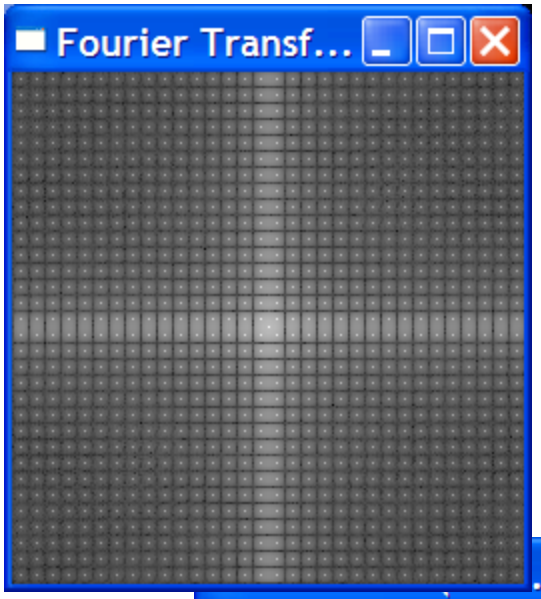
| | | DFT | DFT Executed as 1-D Transforms on Columns and Rows | | DFT | DFT Executed as 1-D Transforms on Columns and Rows |
|---|---|---|---|---|---|---|
| Number of Rows | Number of Columns | Number of Complex Multiplications | Number of Complex Multiplications | Multiple | Time (sec) | Time (sec) |
| 64 | 64 | 16,777,216 | 524,288 | 32 | 66.8 | 1.5 |
| 128 | 128 | 268,435,456 | 4,194,304 | 64 | 546.2 | 12.4 |
| 256 | 256 | 4,294,967,296 | 33,554,432 | 128 | 4,367.1 | 98.1 |
| 512 | 512 | 68,719,476,736 | 268,435,456 | 256 | 34,940.7 | 785.2 |
| 1,024 | 1,024 | 1,099,511,627,776 | 2,147,483,648 | 512 | *yeh right!* | 6,280.1 |

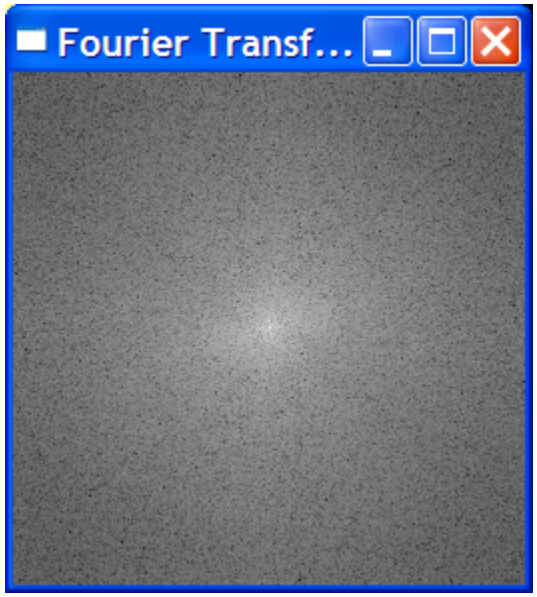
- Computations carried out on a 2 GHz Dell Inspiron running IDL 5.5

# Frequency Domain

## Example Fourier Transform Pairs

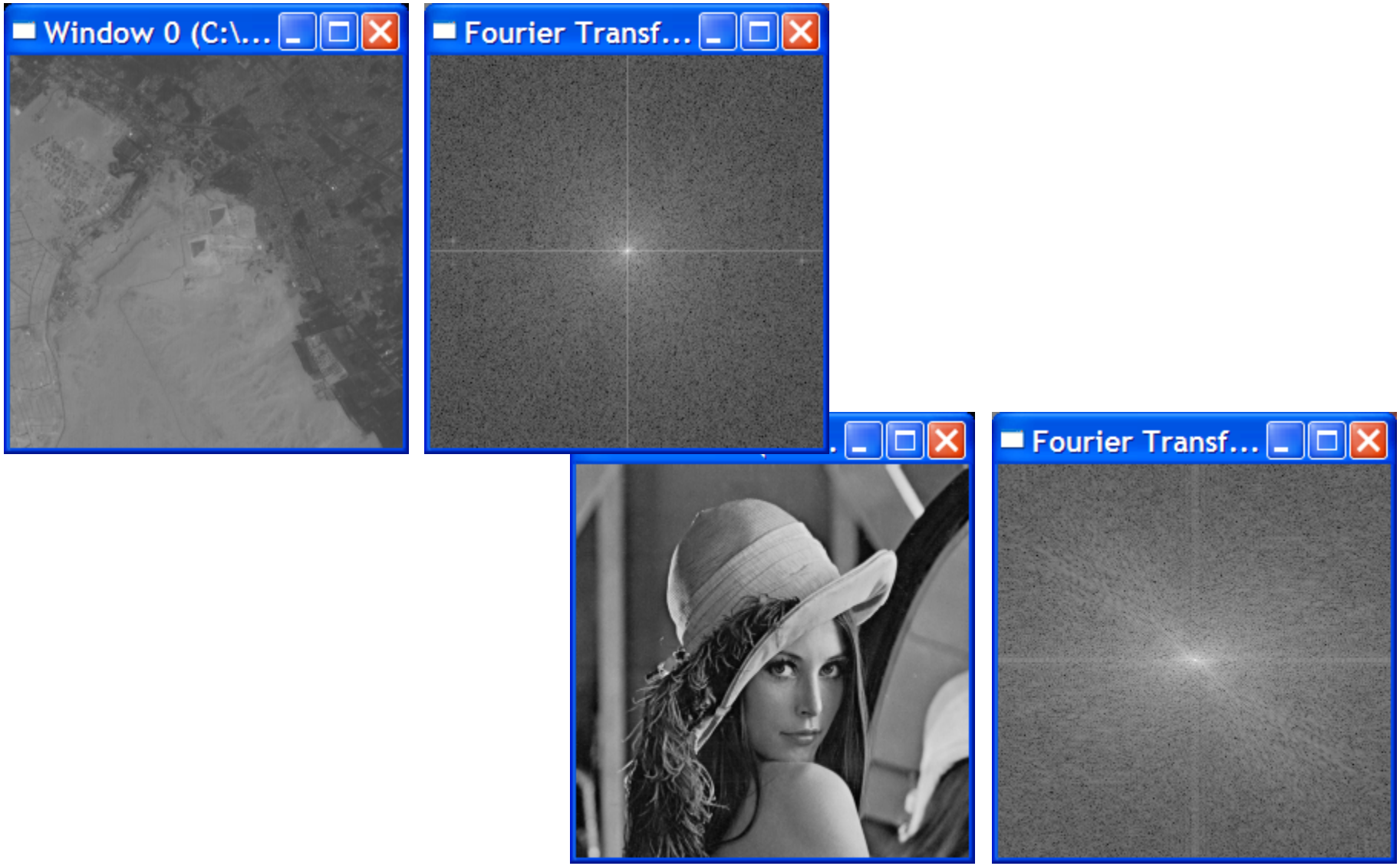


# Frequency Domain

## Example Fourier Transform Pairs

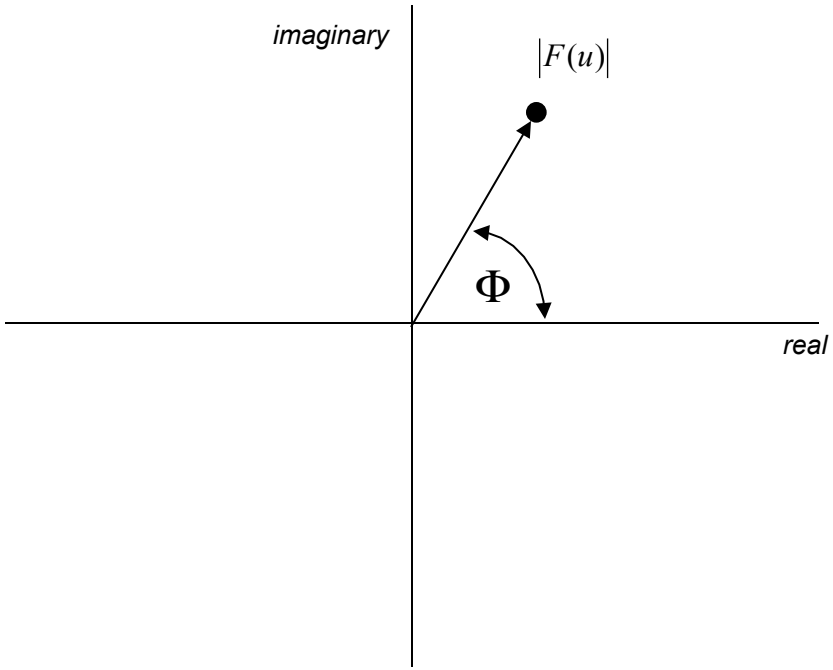


# Frequency Domain

## Amplitude and Phase Representation of Fourier Transform

$$|F(u)| = \sqrt{\mathrm{Re}\{F(u)\}^2 + \mathrm{Im}\{F(u)\}^2}$$

$$\Phi(u) = \tan^{-1}\left(\frac{\mathrm{Im}\{F(u)\}}{\mathrm{Re}\{F(u)\}}\right)$$



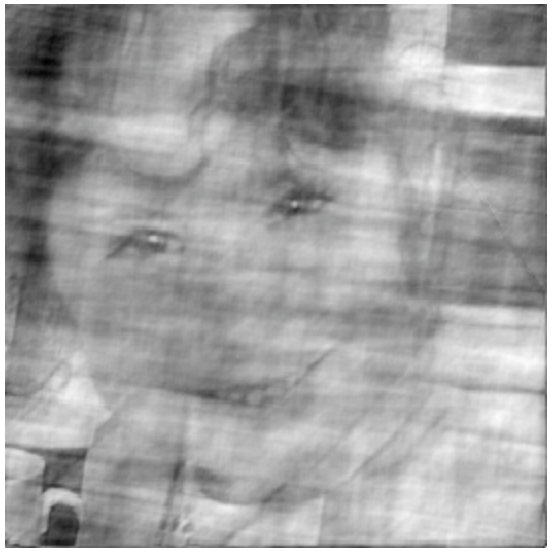

$$F(u) = |F(u)|e^{i\Phi(u)}$$
$$= |F(u)|[\cos(\Phi(u)) + i\sin(\Phi(u))]$$

Amplitude

Phase Angle

# Frequency Domain

## Phase Dominance in Perception

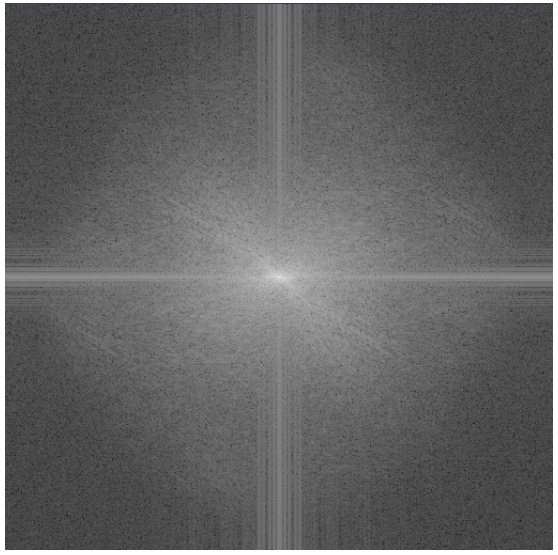


# Frequency Domain

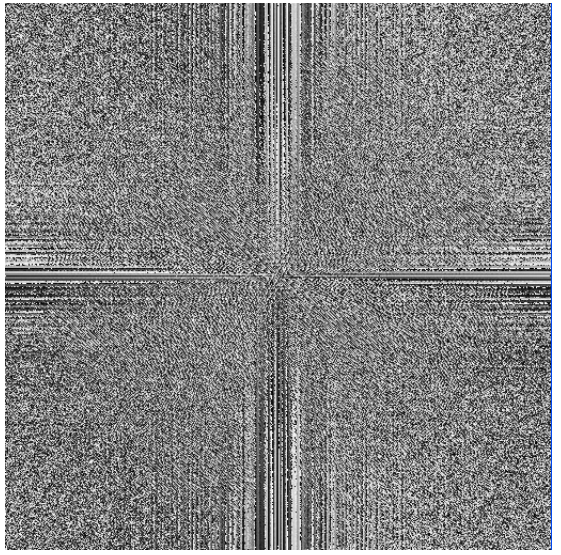
## Phase Angle Substitution

f(x,y)

$\log_{10}( |F(u,v)| )$

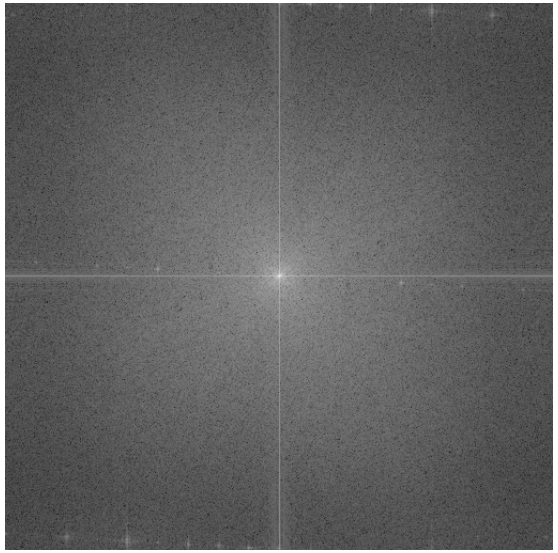


$\Phi(u,v)_F$

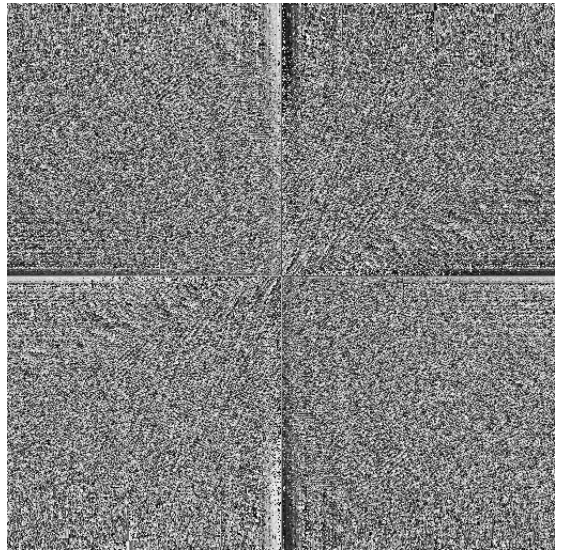


g(x,y)

$\log_{10}( |G(u,v)| )$



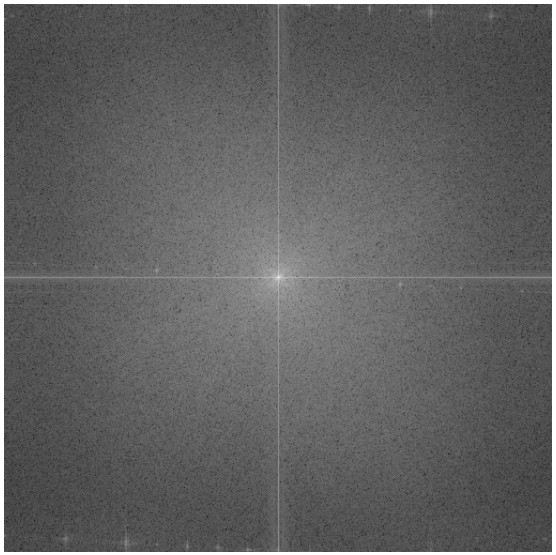
$\Phi(u,v)_G$



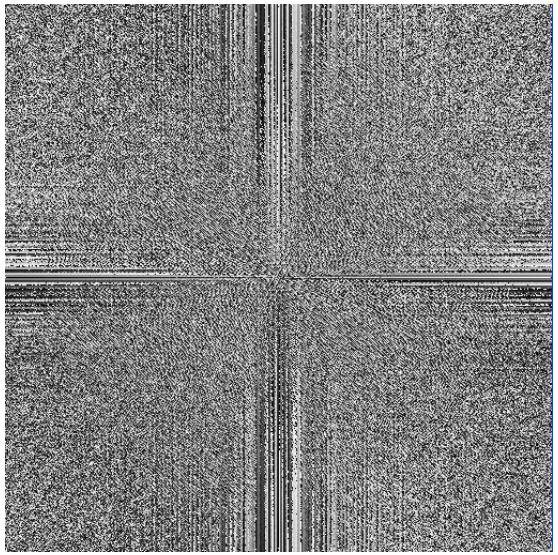
# Frequency Domain

## Phase Angle Substitution

$\log_{10}(|G(u,v)|)$



$\Phi(u,v)_F$

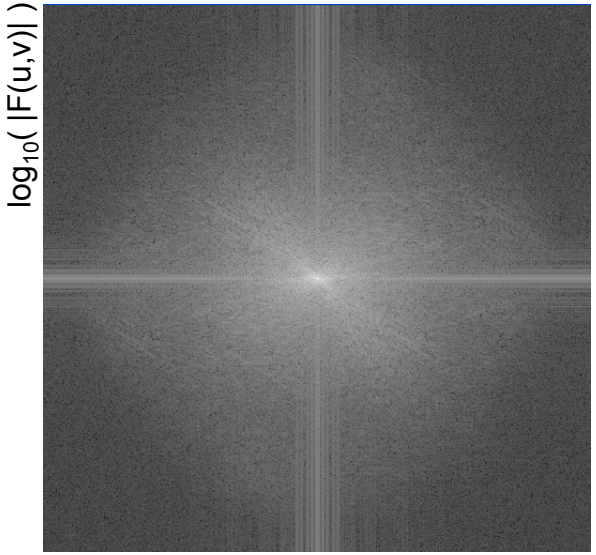


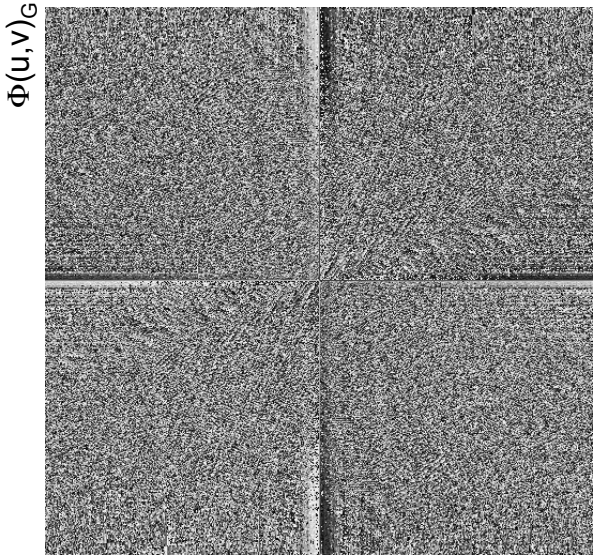
$$H(u,v) = |G(u,v)|e^{i\Phi(u,v)_F}$$
$$= |G(u,v)|\left[\cos\left(\Phi(u,v)_F\right) + i\sin\left(\Phi(u,v)_F\right)\right]$$

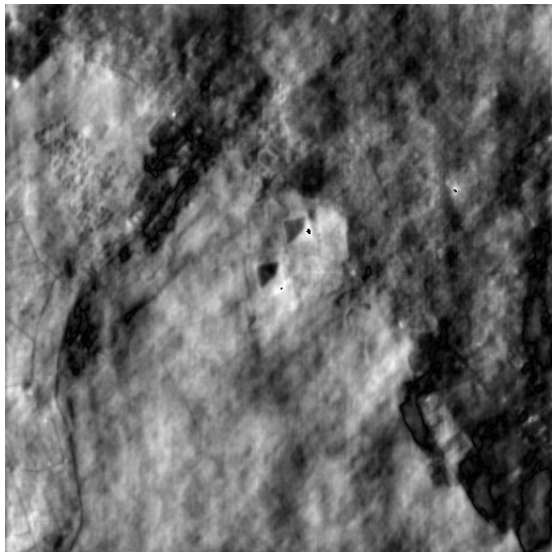
$$h(x,y) = \Im^{-1}\left\{G(u,v)\left[\cos\left(\Phi(u,v)_F\right) + i\sin\left(\Phi(u,v)_F\right)\right]\right\}$$

# Frequency Domain

## Phase Angle Substitution



$$H(u,v) = \left|F(u,v)\right| e^{i\Phi(u,v)_G}$$
$$= \left|F(u,v)\right| \left[\cos\left(\Phi(u,v)_G\right) + i\sin\left(\Phi(u,v)_G\right)\right]$$

$$h(x,y) = \Im^{-1}\left\{F(u,v)\left|\left[\cos\left(\Phi(u,v)_G\right) + i\sin\left(\Phi(u,v)_G\right)\right]\right.\right\}$$

# Fast Fourier Transform

Given the Discrete Fourier Transform (DFT)

$$F(u) = \frac{1}{M}\sum_{x=0}^{M-1} f(x)e^{-i2\pi ux/M} \qquad \text{for } u = 0, 1, 2, ..., M\text{-}1$$

if it is assumed that $M$ is an even number, two sequences can be formed from the even and odd numbered samples of the original function $f(x)$

$$g(x) : f(0), f(2), \ldots, f(M-2)$$
$$h(x) : f(1), f(3), \ldots, f(M-1)$$

so that

$$g(x) : f(2x)$$
$$h(x) : f(2x+1) \qquad \text{for } x = 0, 1, 2, ..., (M/2\text{-}1)$$

# Fast Fourier Transform

The DFT can then be re-written (*ignoring the scaling factor for now*) as

$$
\begin{aligned}
F(u) &= \sum_{x=0}^{M/2-1} g(x) e^{-i2\pi u(2x)} + \sum_{x=0}^{M/2-1} h(x) e^{-i2\pi u(2x+1)} \\
&= \sum_{x=0}^{M/2-1} g(x) e^{-i4\pi ux} + e^{-i2\pi u} \sum_{x=0}^{M/2-1} h(x) e^{-i4\pi ux} \\
&= \sum_{x=0}^{M/2-1} g(x) e^{-i2\pi ux} + e^{-i2\pi u} \sum_{x=0}^{M/2-1} h(x) e^{-i2\pi ux} \\
&= G(u) + e^{-i2\pi u} H(u)
\end{aligned}
$$

for $u = 0, 1, 2, ..., M\text{-}1$

where $G(u)$ and $H(u)$ are the discrete Fourier transforms of $g(x)$ and $h(x)$, respectively. This states that the discrete Fourier transform of the original function, $F(u)$, can be obtained as the sum of these two individual transformed functions, $G(u)$ and $H(u)$.

# Fast Fourier Transform

You can continue to subdivide the separated DFTs in this same manner until you arrive at DFTs comprised of only two (2) points each. Each two-point DFT is computed simply as

$$F(u) = f(0)\left[\cos\left(2\pi u(0)/M\right) - i\sin\left(2\pi u(0)/M\right)\right] + f(1)\left[\cos\left(2\pi u(1)/M\right) - i\sin\left(2\pi u(1)/M\right)\right]$$

$$= f(0)\left[\cos(0) - i\sin(0)\right] + f(1)\left[\cos\left(2\pi u/M\right) - i\sin\left(2\pi u/M\right)\right]$$

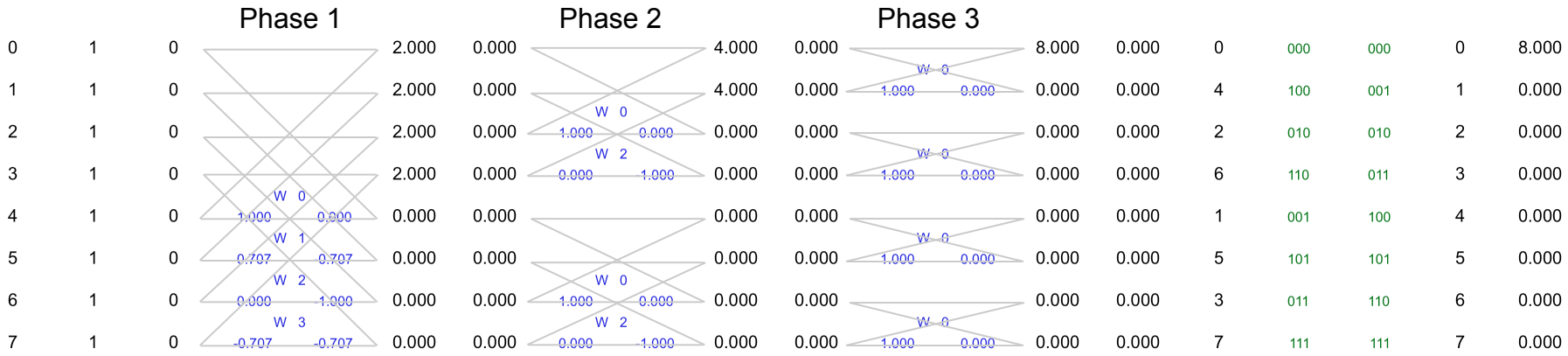
$$= f(0) + f(1)\left[\cos\left(2\pi u/M\right) - i\sin\left(2\pi u/M\right)\right]$$

The complete computation of the Fast Fourier Transform (FFT) by the reduction of the DFT to a series of two-point DFTs is often represented by a "butterfly" diagram.

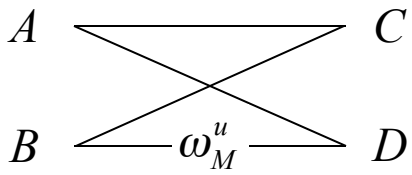
# Fast Fourier Transform

## 8-Point Butterfly Diagram

## rect(x)




For any butterfly in the diagram above




where

$$C = A + B$$

$$D = (A - B)\omega_M^u$$

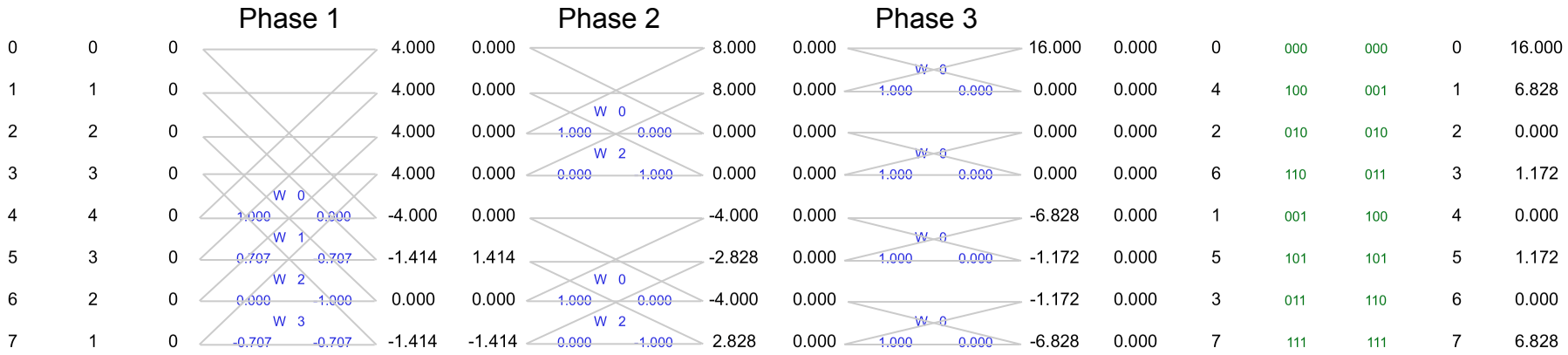
$$\omega_M^u = \left( \cos\left[\frac{2\pi u}{M}\right] - i \sin\left[\frac{2\pi u}{M}\right] \right)$$

NOTE: A $2^n$-point function, $f(x)$, will have $n$ phases in its butterfly diagram

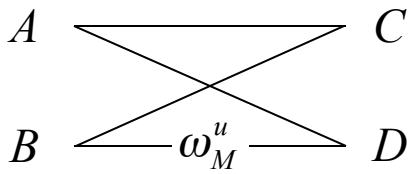
# Fast Fourier Transform

## 8-Point Butterfly Diagram

## tri(x)




For any butterfly in the diagram above




where

$$C = A + B$$

$$D = (A - B)\omega_M^u$$

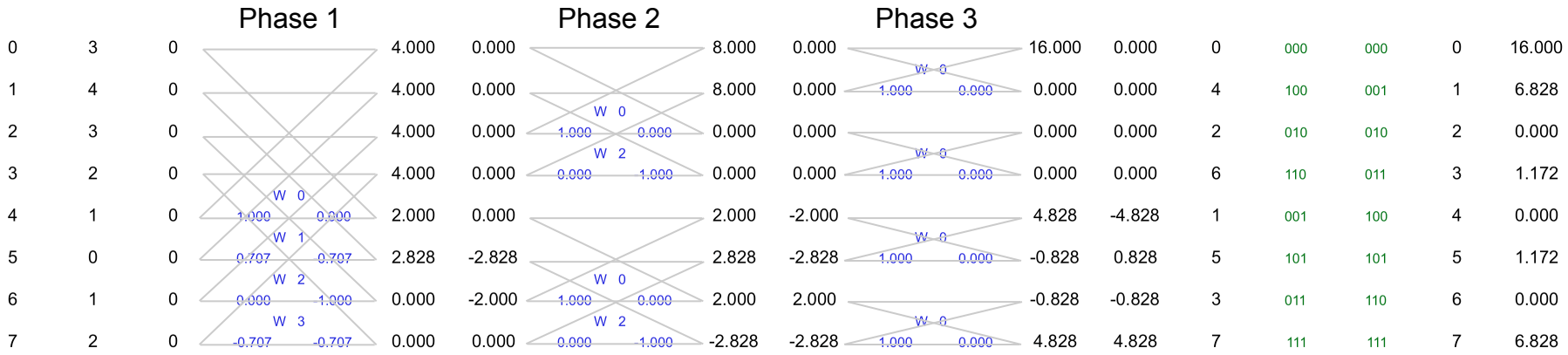
$$\omega_M^u = \left( \cos\left[\frac{2\pi u}{M}\right] - i \sin\left[\frac{2\pi u}{M}\right] \right)$$

NOTE: A $2^n$-point function, $f(x)$, will have $n$ phases in its butterfly diagram

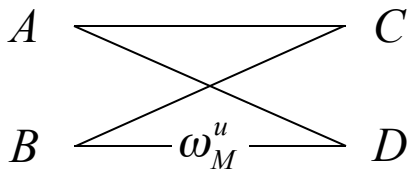
# Fast Fourier Transform

## 8-Point Butterfly Diagram

## tri(x+3)




For any butterfly in the diagram above




where

$$C = A + B$$

$$D = (A - B)\omega_M^u$$

$$\omega_M^u = \left( \cos\left[\frac{2\pi u}{M}\right] - i \sin\left[\frac{2\pi u}{M}\right] \right)$$

NOTE: A $2^n$-point function, $f(x)$, will have $n$ phases in its butterfly diagram

# Frequency/Spatial Domain Filtering

Relationship

## Convolution Theorem

$$f(x,y) * h(x,y) \Leftrightarrow F(u,v) \cdot H(u,v)$$
$$f(x,y) \cdot h(x,y) \Leftrightarrow F(u,v) * H(u,v)$$

## Convolution*

$$f(x,y) * h(x,y) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) \cdot h(x-m, y-n)$$

* Do not confuse this with the linear spatial filtering. Implied in the definition of convolution is a flip of the filter and a shifting of its position.

# Frequency Domain Filtering

## Basic Steps

- Multiply the input image by $(-1)^{x+y}$ to center the transform
- Compute $F(u,v)$ using the Fourier transform on the image from the previous step
- Multiply $F(u,v)$ by the filter function $H(u,v)$ (pixel-by-pixel)
- Compute the inverse Fourier transform of the result from the previous step
- Obtain the magnitude of the inverse transformed image
- Multiply the result of the previous step by $(-1)^{x+y}$

# Frequency Domain Filtering

## Basic Steps Using IDL

- Create a filter image in the frequency domain, $H(u,v)$
- Shift the filter image so that it will align with the Fourier transform, $F(u,v)$

  $shifted\ H(u,v)$ = SHIFT( $H(u,v)$, $n_{columns}/2$, $n_{rows}/2$ )
- Compute the Fourier transform using the FFT function

  $F(u,v)$ = FFT( $f(x,y)$ )
- Multiply the resulting Fourier transform image by the shifted filter image

  $G(u,v) = shifted\ H(u,v) * F(u,v)$
- Compute the inverse Fourier transform of the filtered Fourier transform

  $g(x,y)$ = FFT( $G(u,v)$, /INVERSE )
- Display the magnitude of $g(x,y)$ to see your enhanced image

# Frequency Domain Filtering

## Basic Filter Types

- Notch Filter
  - a constant valued function with a hole or a series of holes in it designed to either pass or block specific frequency values
- Lowpass Filter
  - a function that attenuated high frequency information while allowing low frequency information to pass
- Highpass Filter
  - a function that attenuated low frequency information while allowing high frequency information to pass

# Frequency Domain Filtering

## Ideal Lowpass Filter

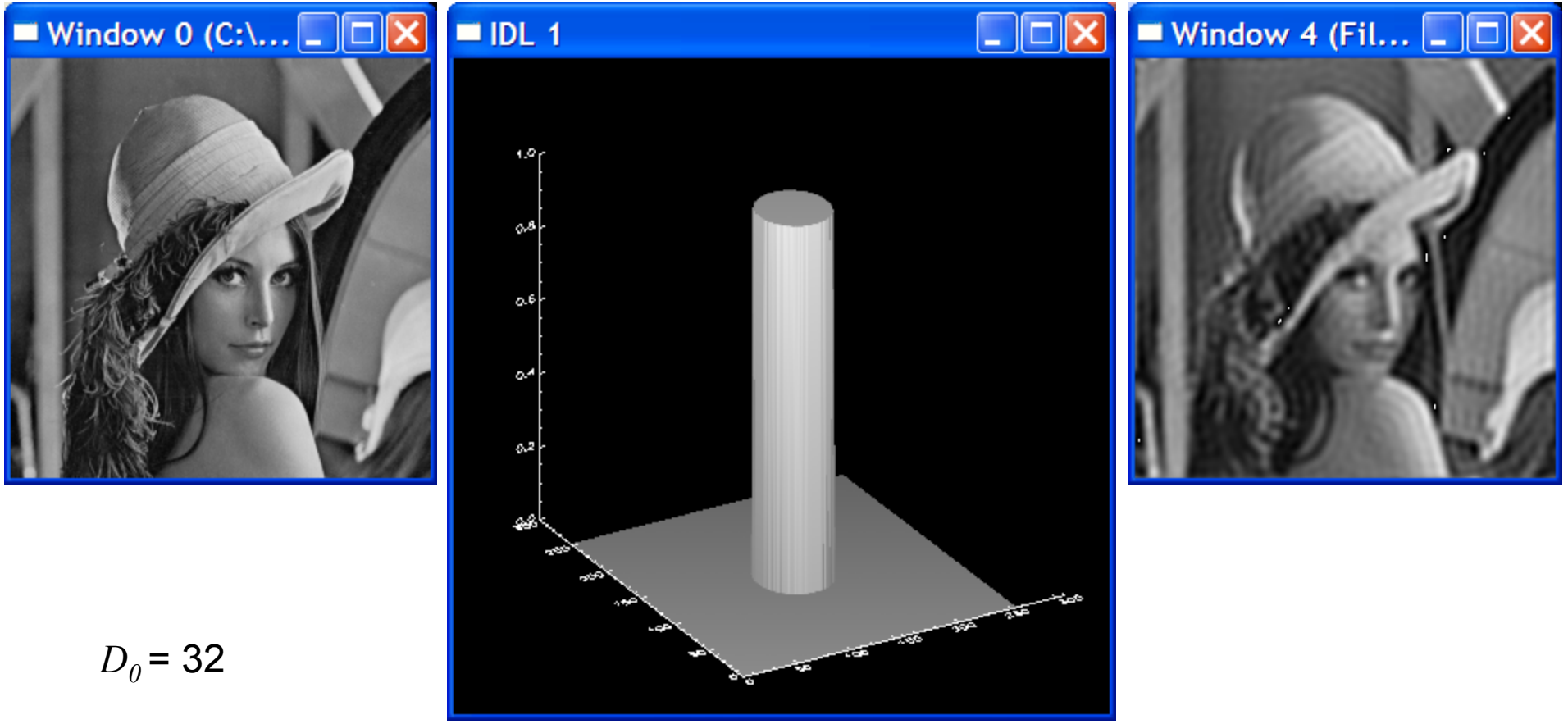
$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

where $D(u,v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$

and $D_0$ is defined as the cutoff frequency (a non-negative quantity)

# Frequency Domain Filtering

## Ideal Lowpass Filter




$D_0 = 32$

# Frequency Domain Filtering

## Butterworth Lowpass Filter

$$H(u,v) = \frac{1}{1 + \left[\frac{D(u,v)}{D_0}\right]^{2n}}$$

where $D(u,v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$

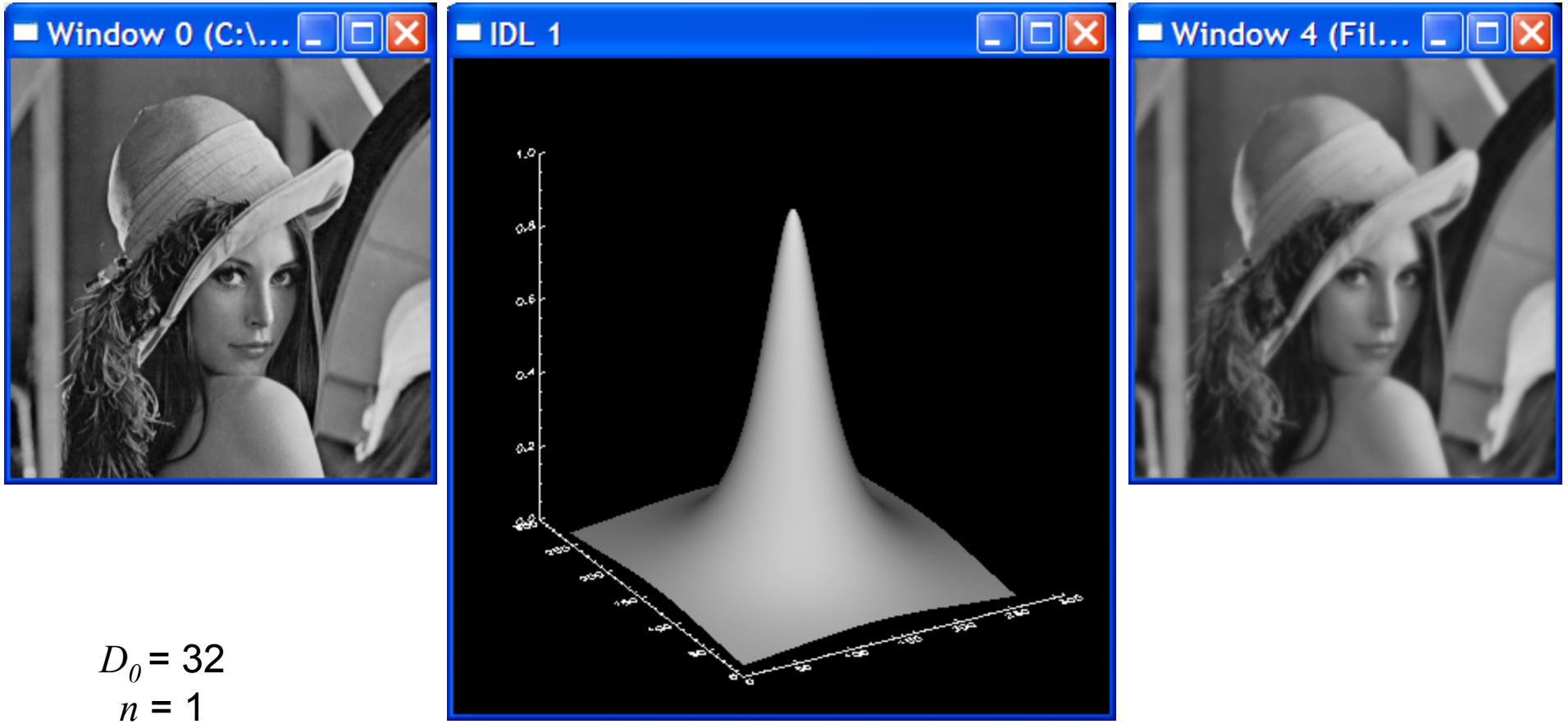
$D_0$ is defined as the cutoff frequency (a non-negative quantity),

and $n$ is the *order* of the filter

* Note that when $D(u,v) = D_0$, the filter is down to 0.5 of its maximum value

# Frequency Domain Filtering

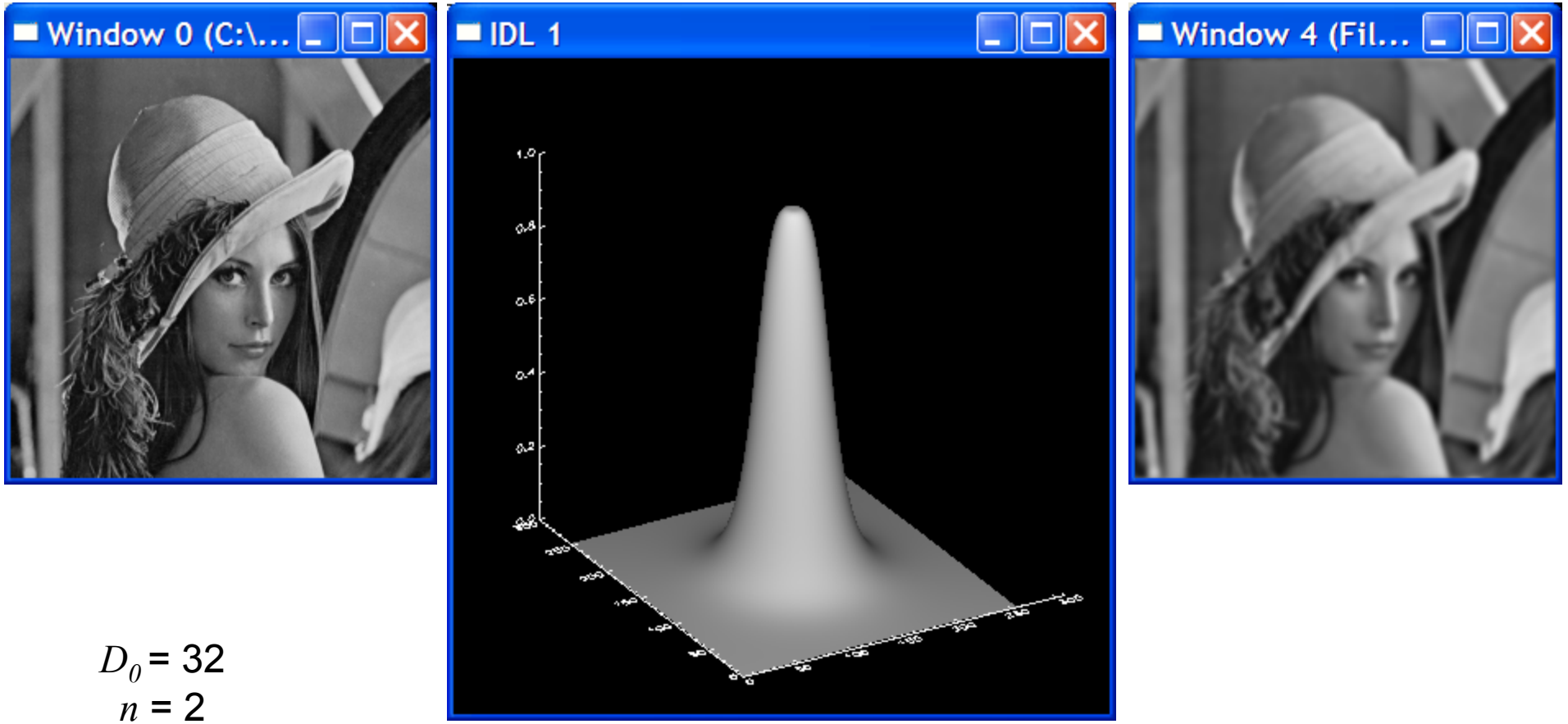
## Butterworth Lowpass Filter




$D_0 = 32$
$n = 1$

# Frequency Domain Filtering

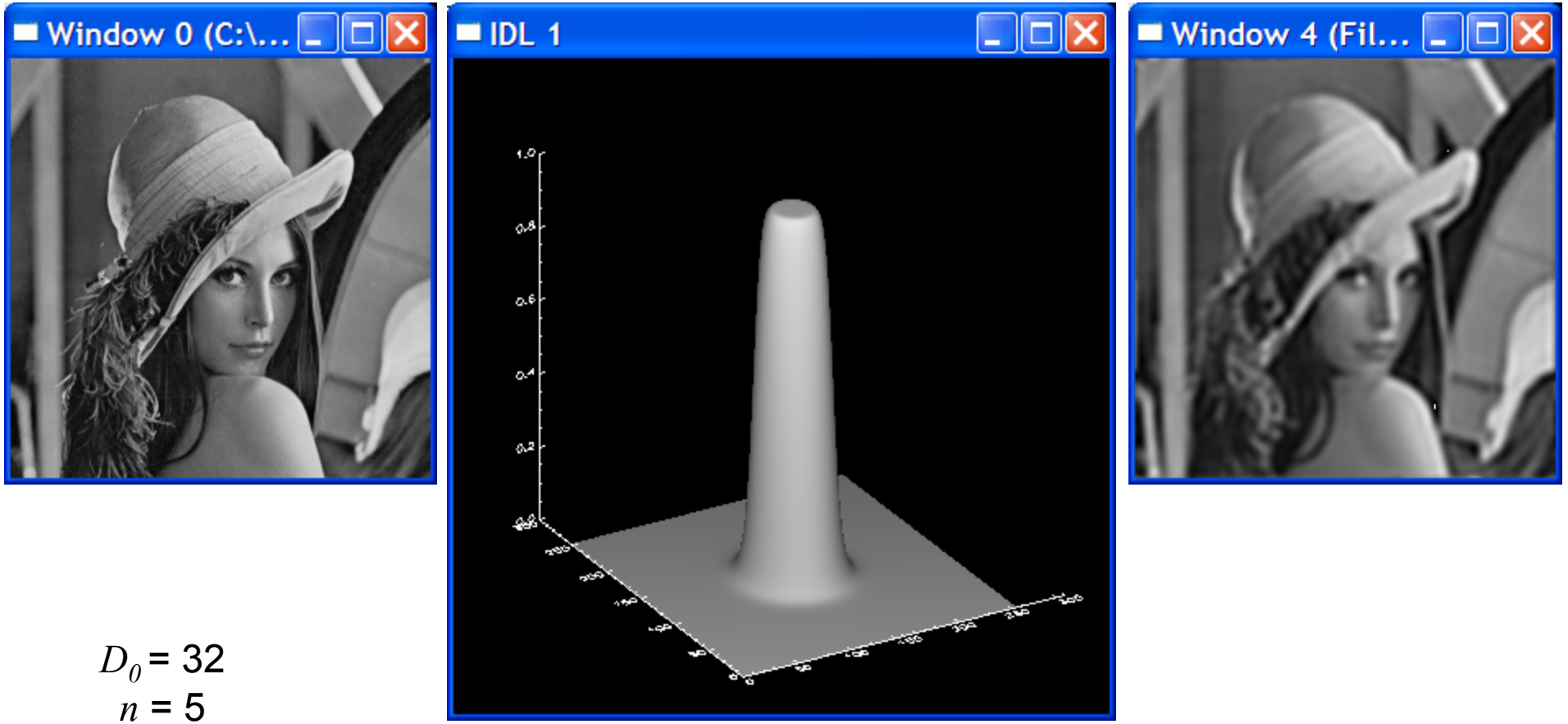
## Butterworth Lowpass Filter




$D_0 = 32$
$n = 2$

# Frequency Domain Filtering

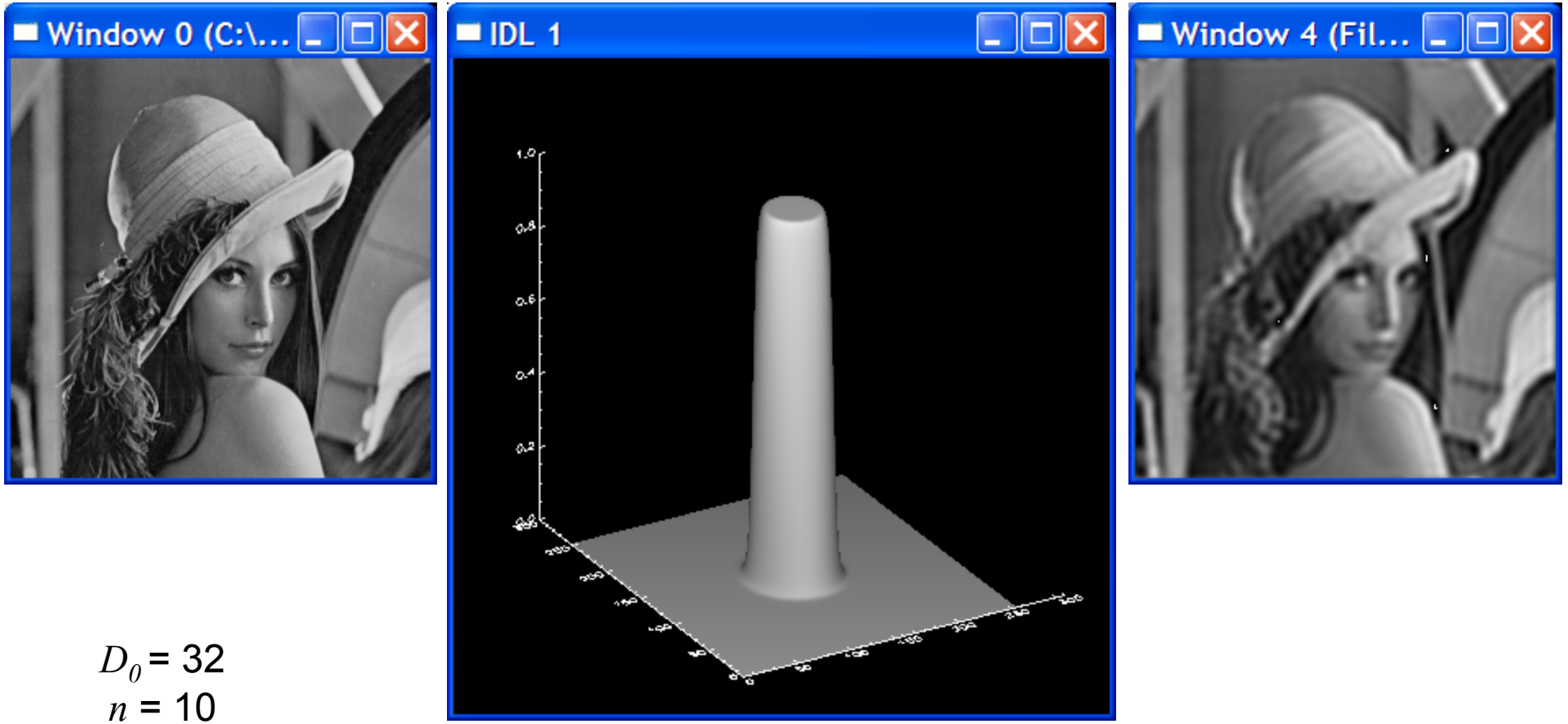
## Butterworth Lowpass Filter




$D_0 = 32$
$n = 5$

# Frequency Domain Filtering

## Butterworth Lowpass Filter




$D_0 = 32$
$n = 10$

# Frequency Domain Filtering

## Gaussian Lowpass Filter

$$H(u,v) = e^{-\frac{D(u,v)^2}{2D_0^2}}$$

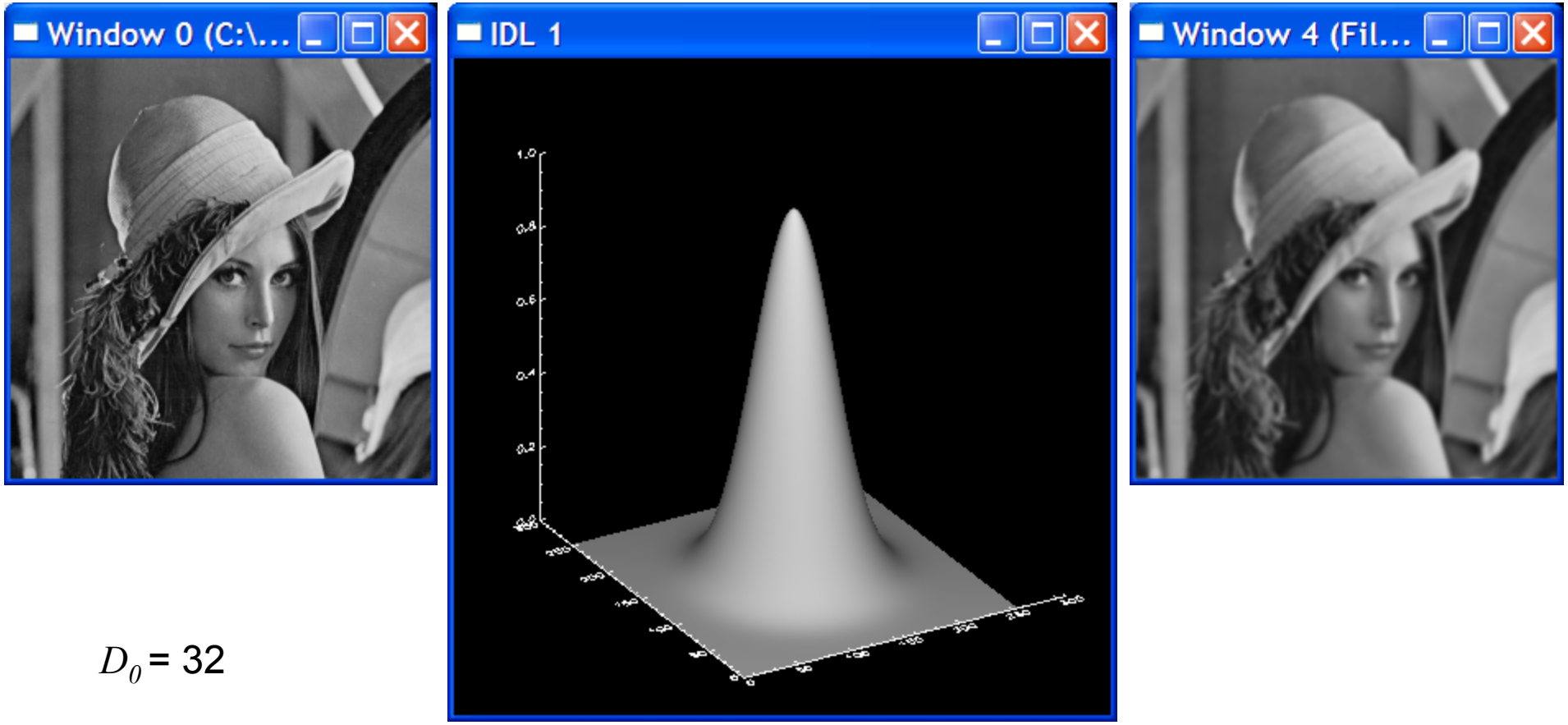
where $D(u,v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$

and $D_0$ is defined as the cutoff frequency (a non-negative quantity) (in this case, it is analogous to the standard deviation of the Gaussian distribution)

* Note that when $D(u,v) = D_0$, the filter is down to 0.667 of its maximum value

# Frequency Domain Filtering

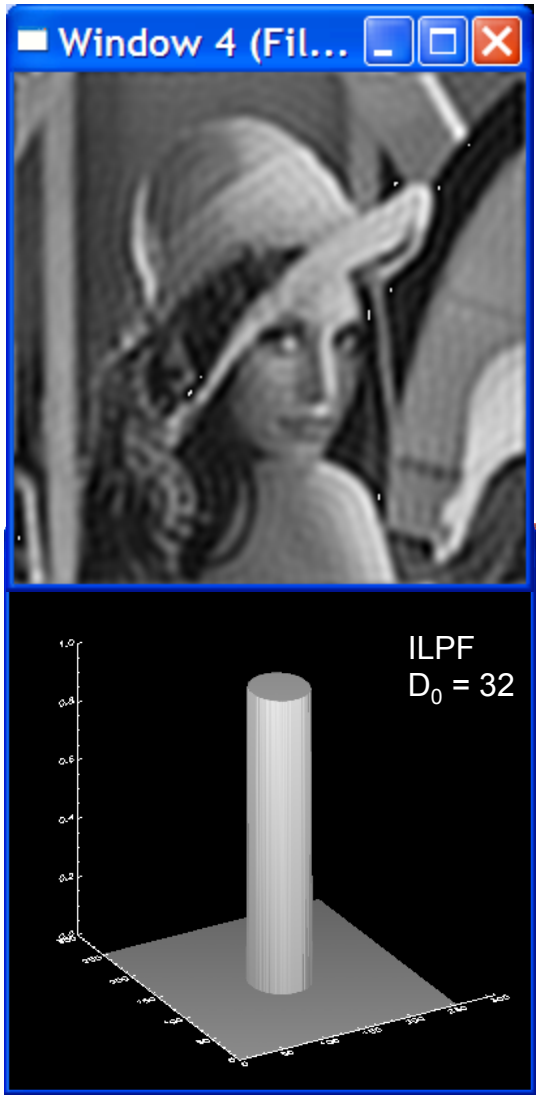
## Gaussian Lowpass Filter

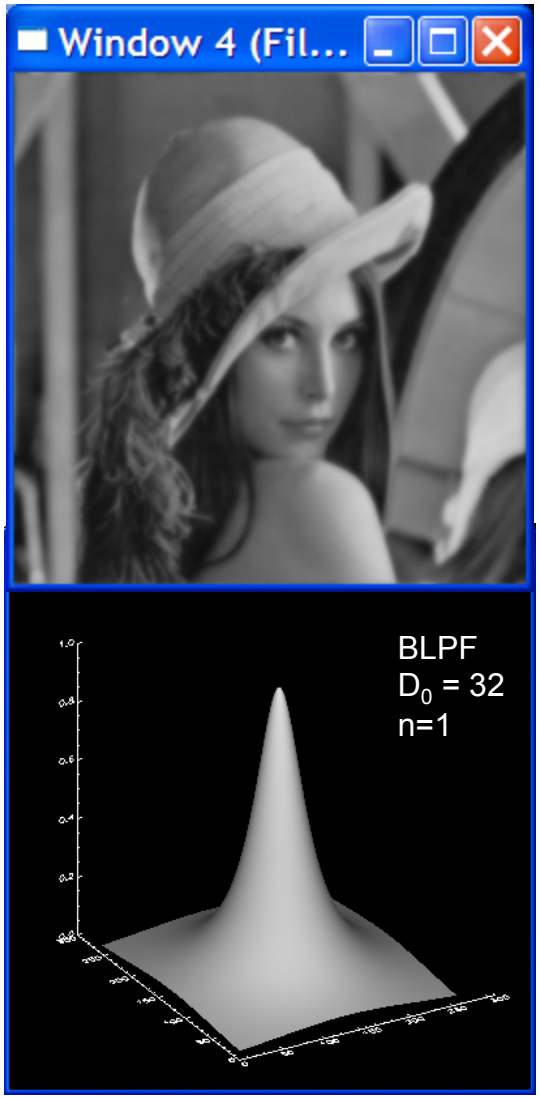



$D_0 = 32$

# Frequency Domain Filtering

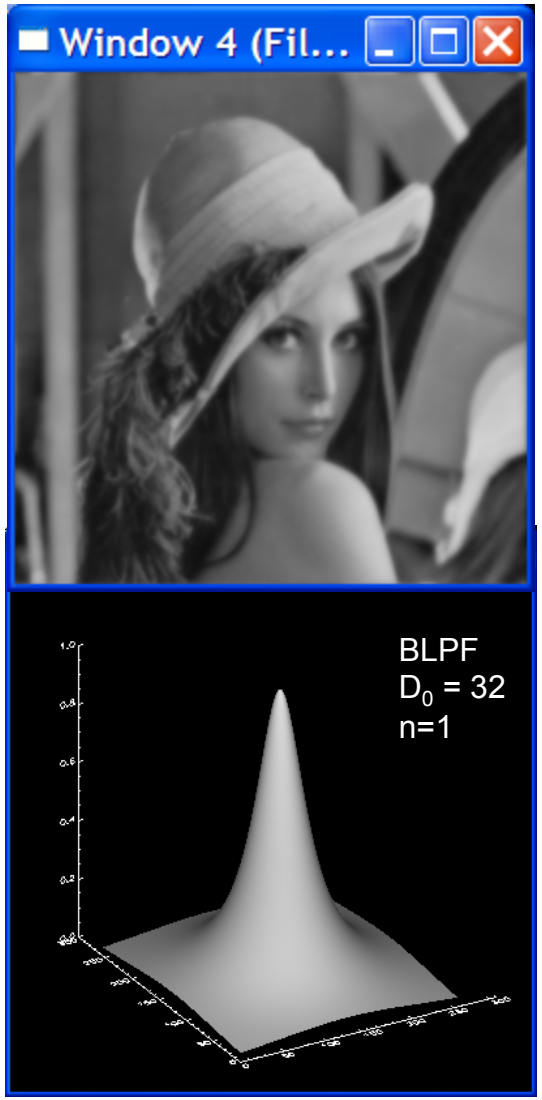
## Comparing Lowpass Filters

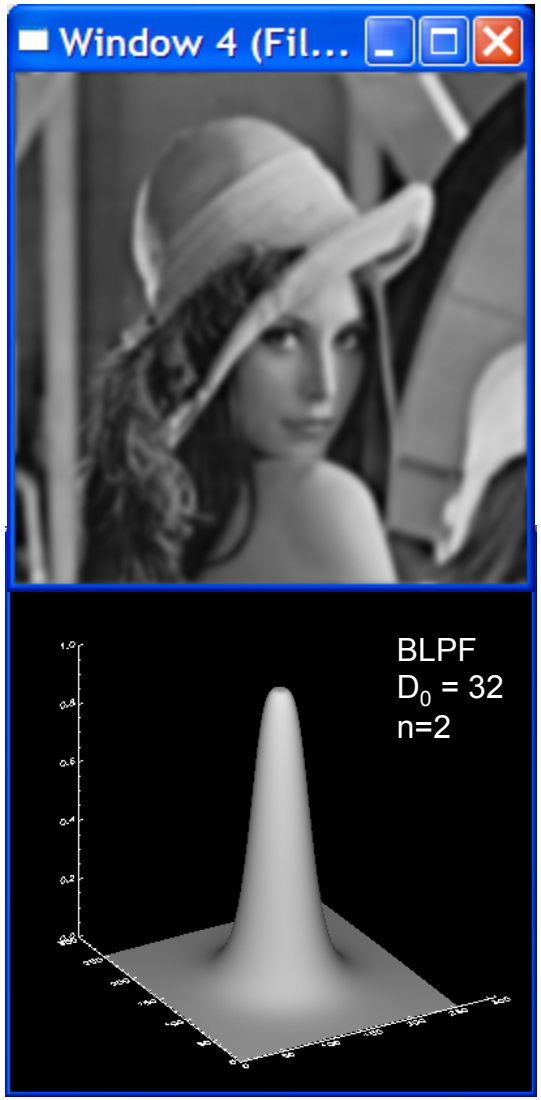


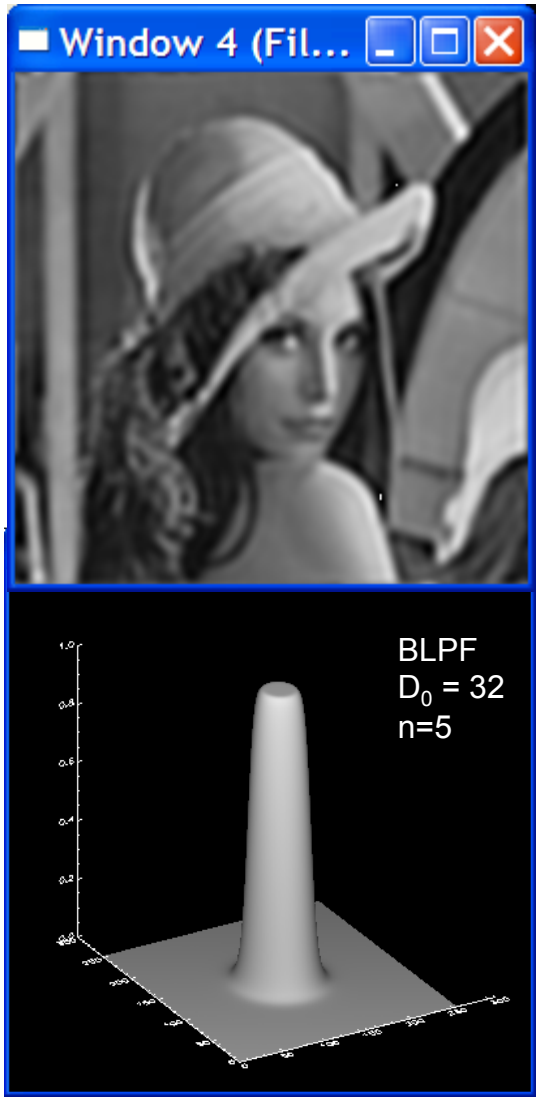




# Frequency Domain Filtering

## Comparing Butterworth Lowpass Filters



# Frequency Domain Filtering

## Ideal Highpass Filter

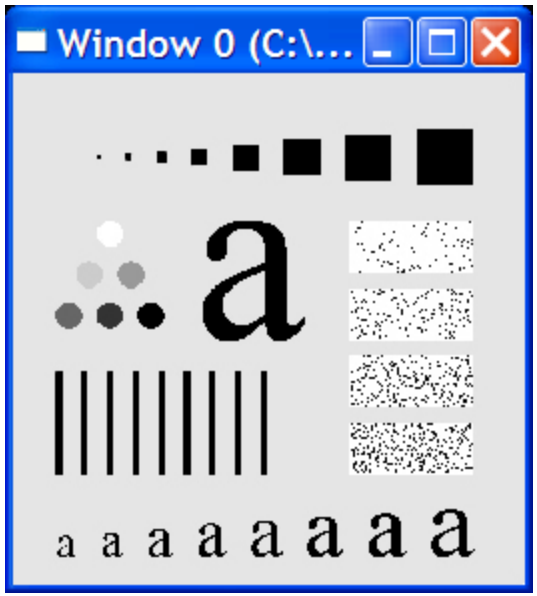
$$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \le D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases}$$

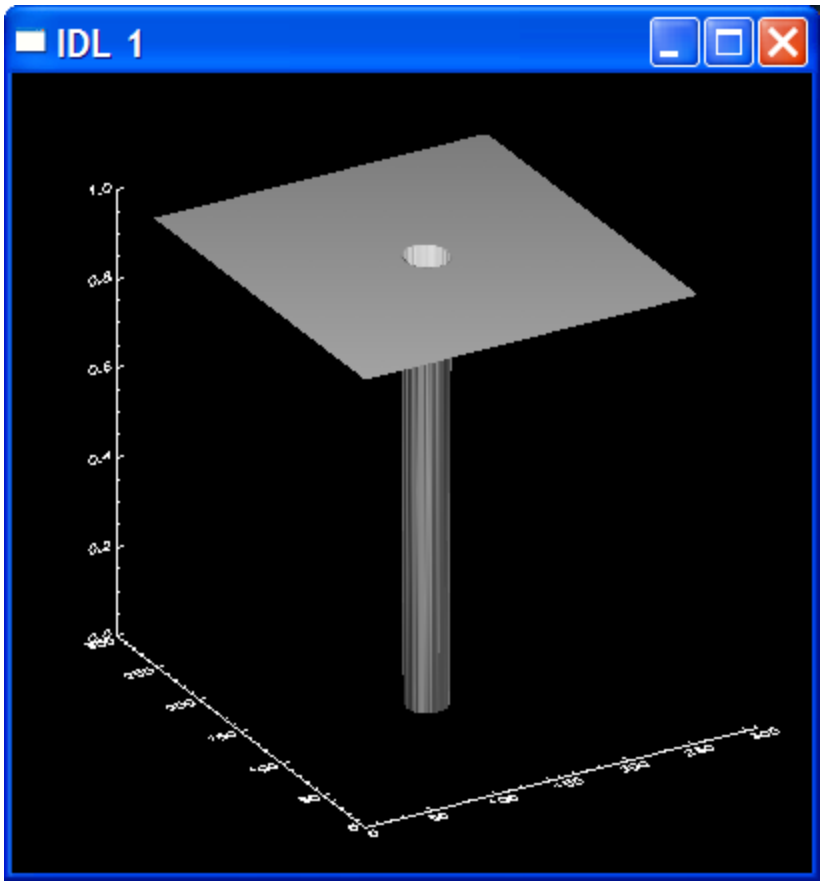
where $D(u,v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$

and $D_0$ is defined as the cutoff frequency (a non-negative quantity)

# Frequency Domain Filtering

## Ideal Highpass Filter





$D_0 = 15$

# Frequency Domain Filtering

## Butterworth Highpass Filter

$$H(u,v) = \frac{1}{1 + \left[\frac{D_0}{D(u,v)}\right]^{2n}}$$

where $D(u,v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$

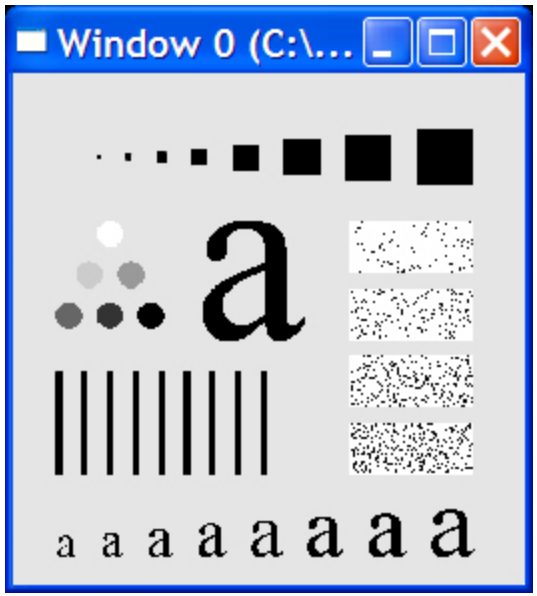
$D_0$ is defined as the cutoff frequency (a non-negative quantity),

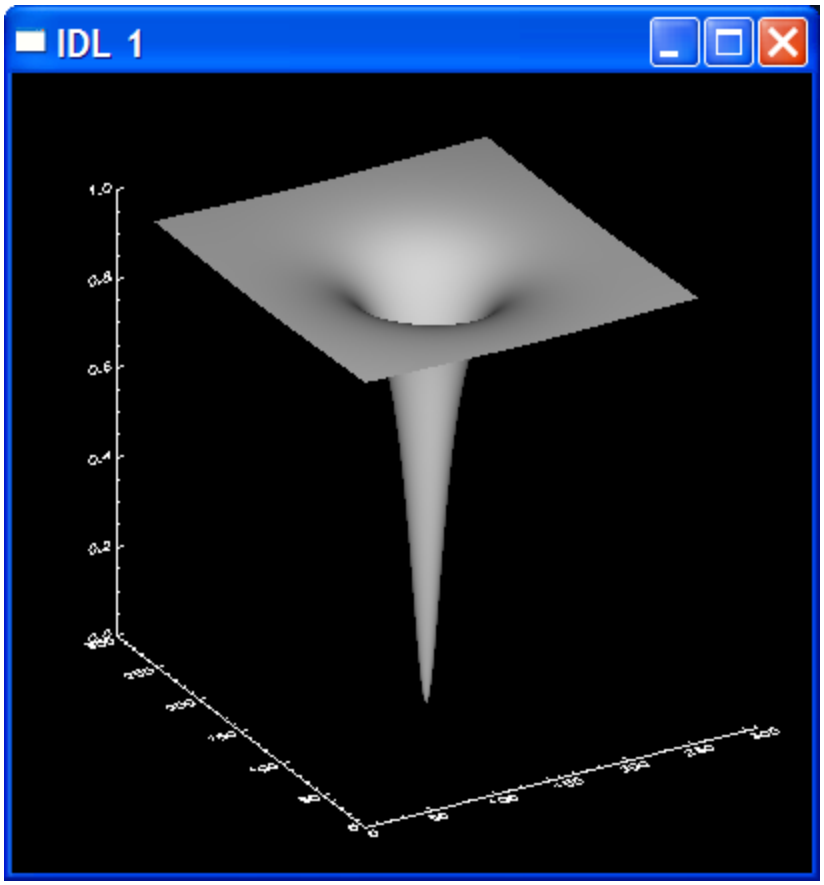
and $n$ is the *order* of the filter

* Note that when $D(u,v) = D_0$, the filter is up 0.5 from its minimum value

# Frequency Domain Filtering

## Butterworth Highpass Filter



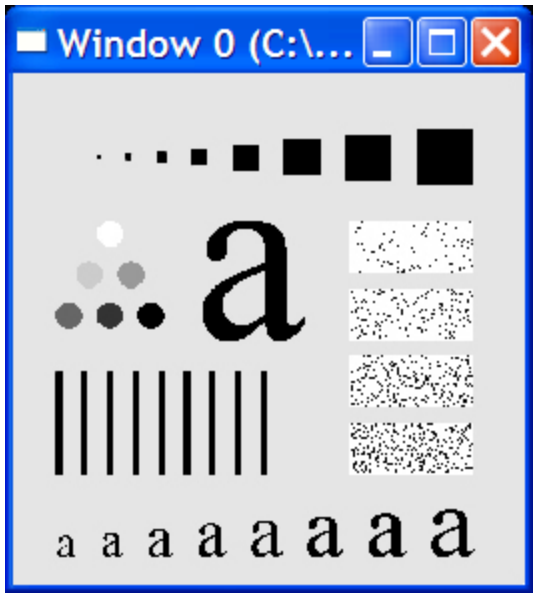



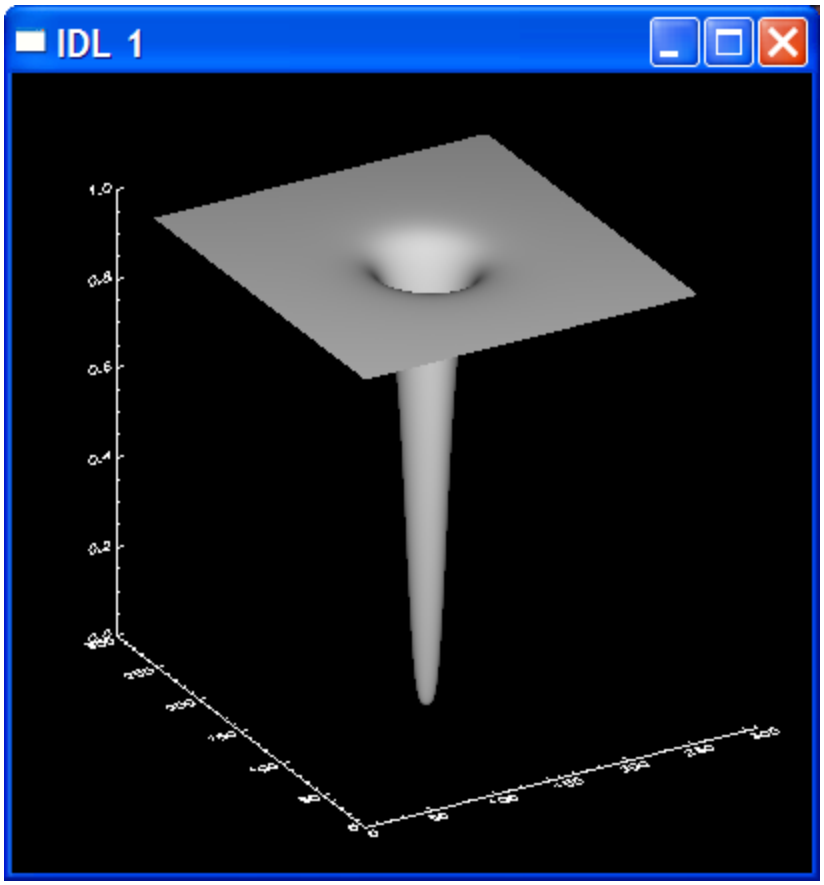


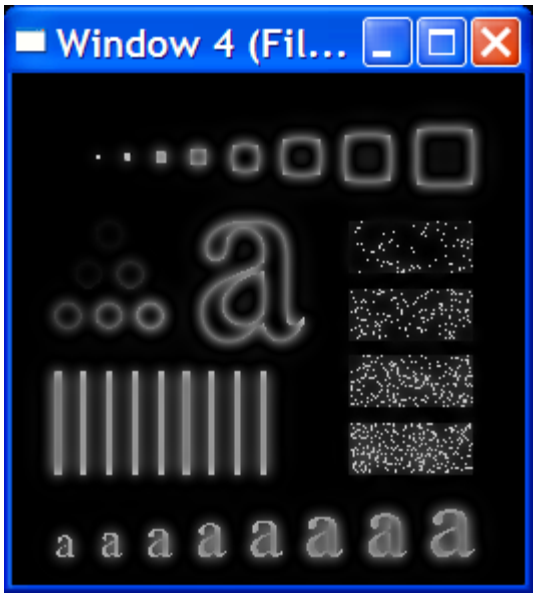
$D_0 = 15$
$n = 1$

# Frequency Domain Filtering

## Butterworth Highpass Filter



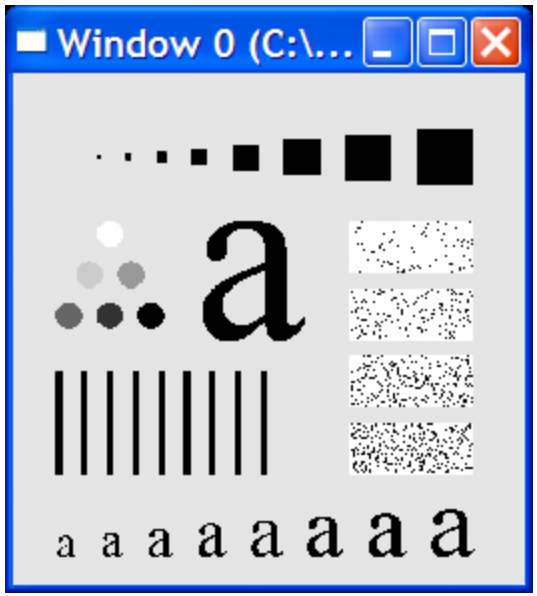



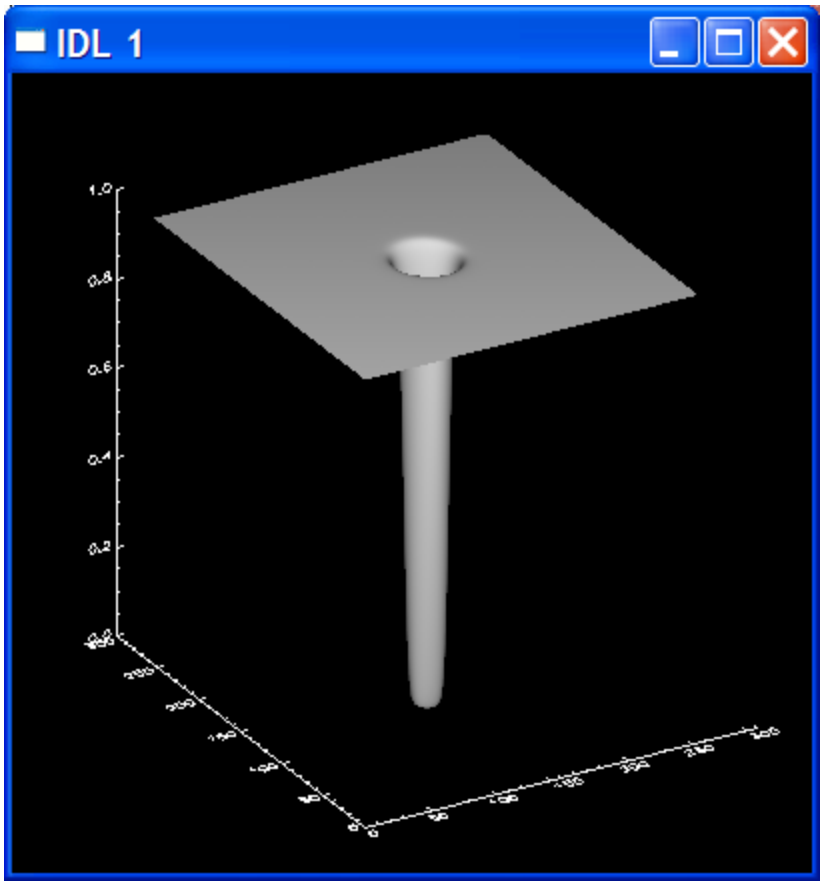



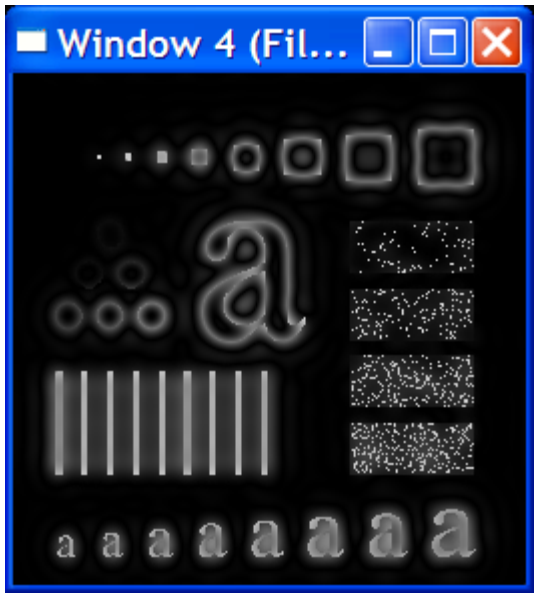

$D_0 = 15$
$n = 2$

# Frequency Domain Filtering

## Butterworth Highpass Filter



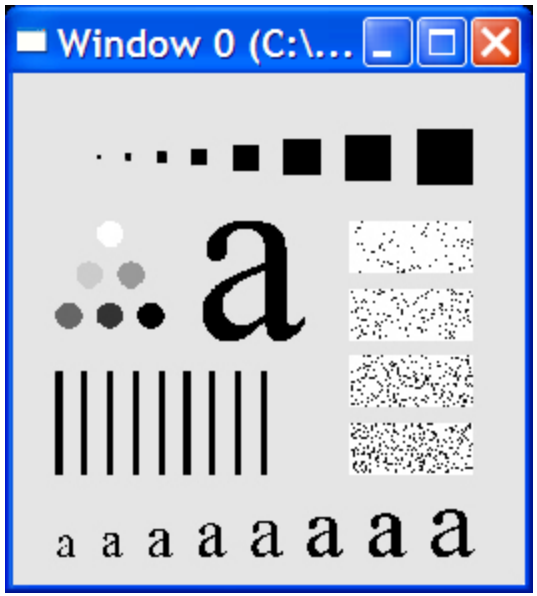



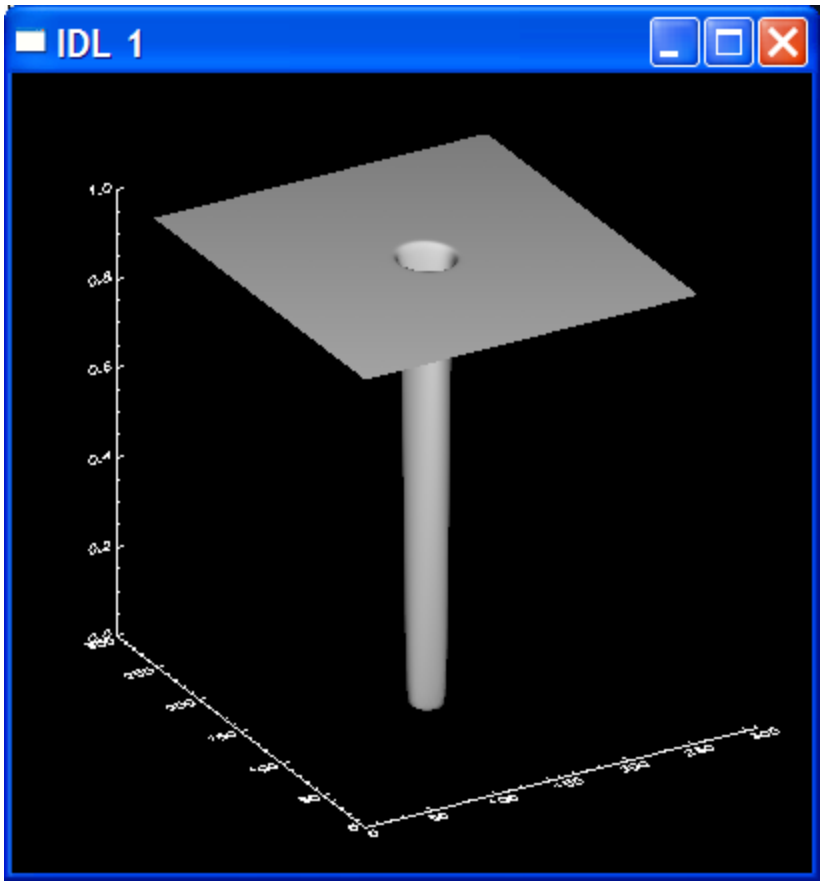



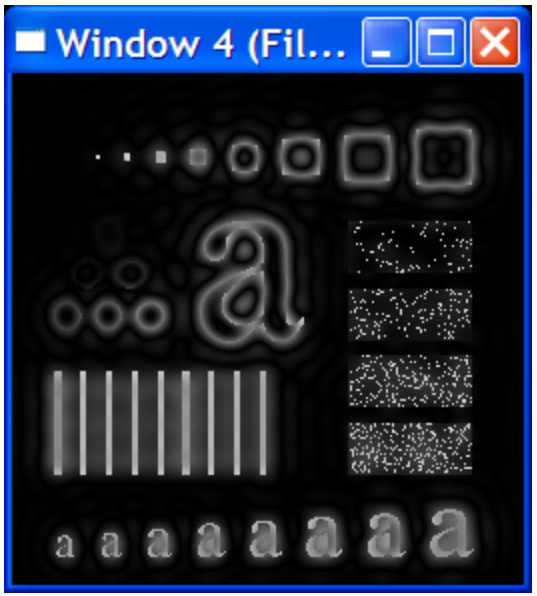

$D_0 = 15$
$n = 5$

# Frequency Domain Filtering

## Butterworth Highpass Filter




$D_0 = 15$
$n = 10$

# Frequency Domain Filtering

## Gaussian Highpass Filter

$$H(u,v) = 1 - e^{-\frac{D(u,v)^2}{2D_0^2}}$$

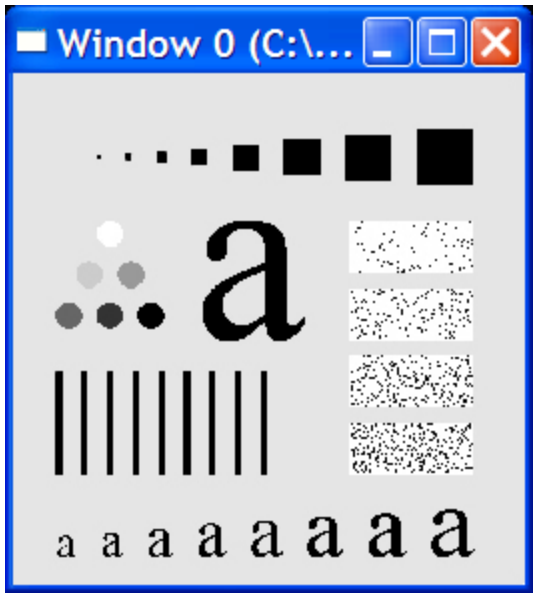
where $D(u,v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$

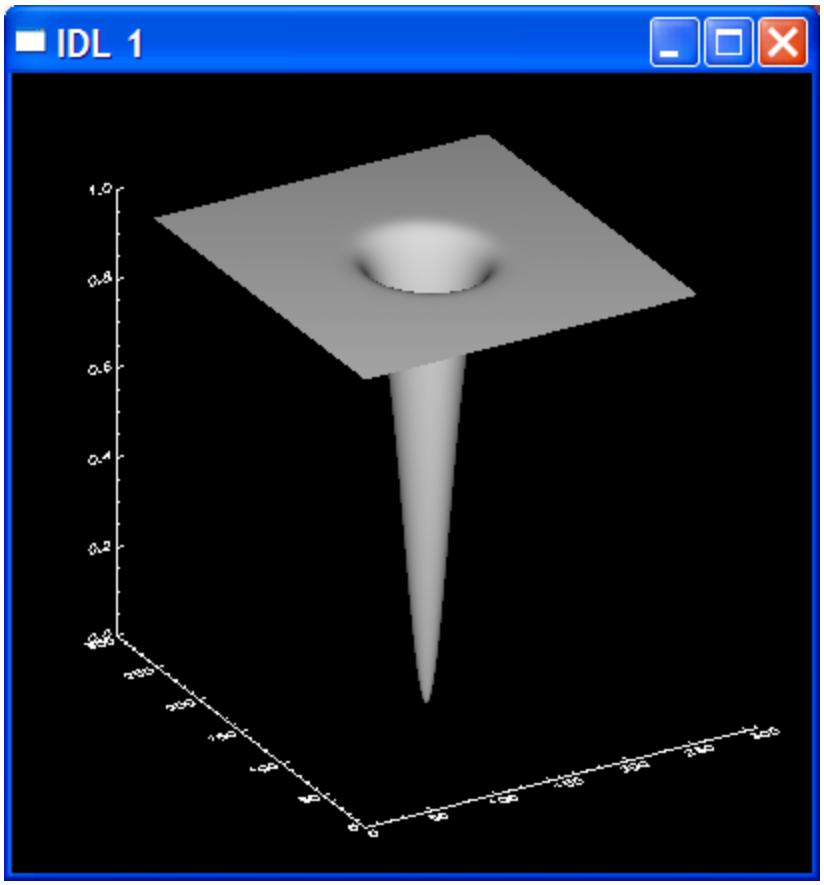
and $D_0$ is defined as the cutoff frequency (a non-negative quantity) (in this case, it is analogous to the standard deviation of the Gaussian distribution)

* Note that when $D(u,v) = D_0$, the filter is up 0.667 over its minimum value

# Frequency Domain Filtering

## Gaussian Highpass Filter



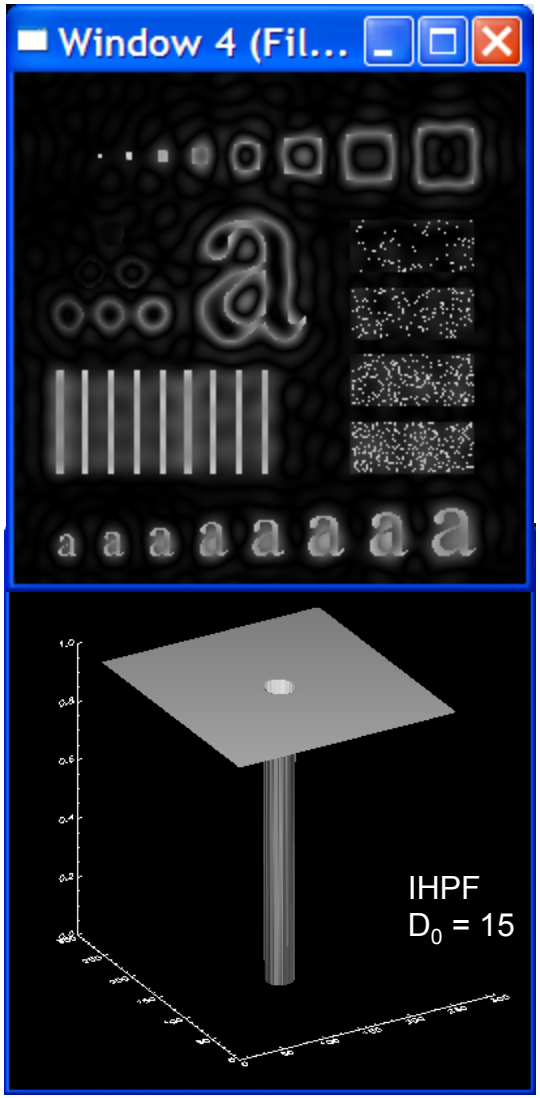



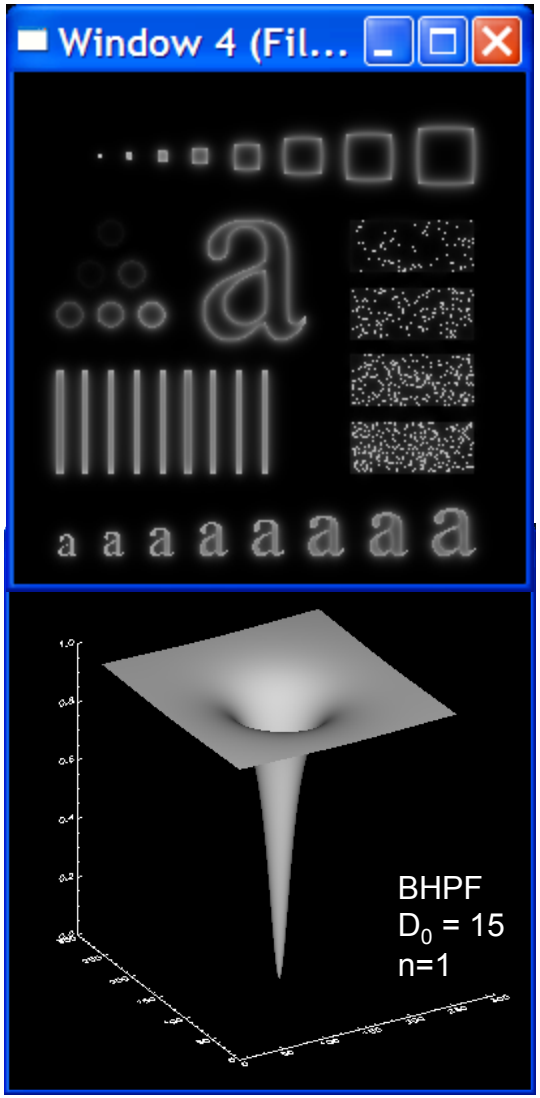


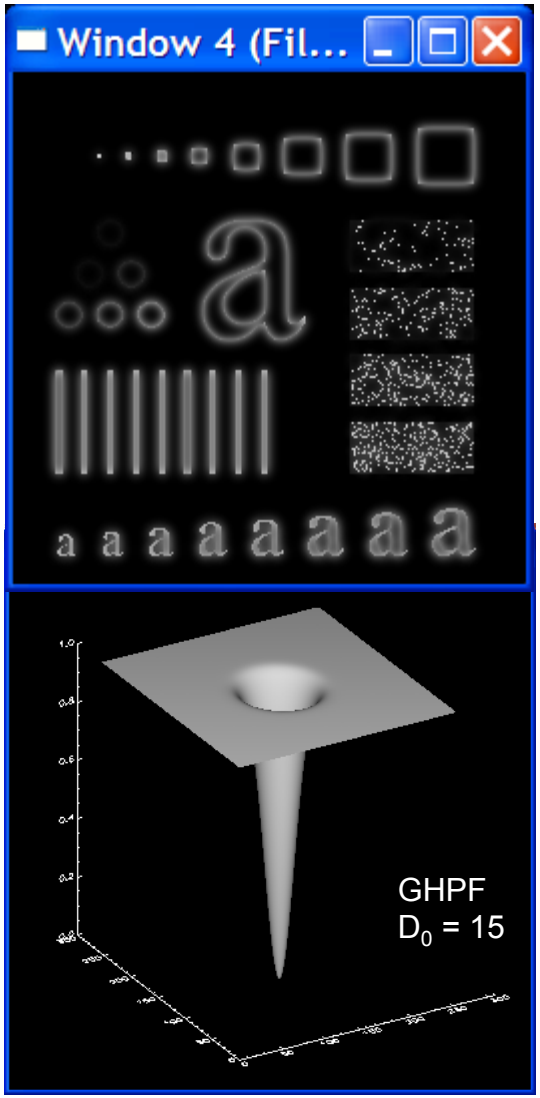
$D_0 = 15$

# Frequency Domain Filtering

## Comparing Highpass Filters

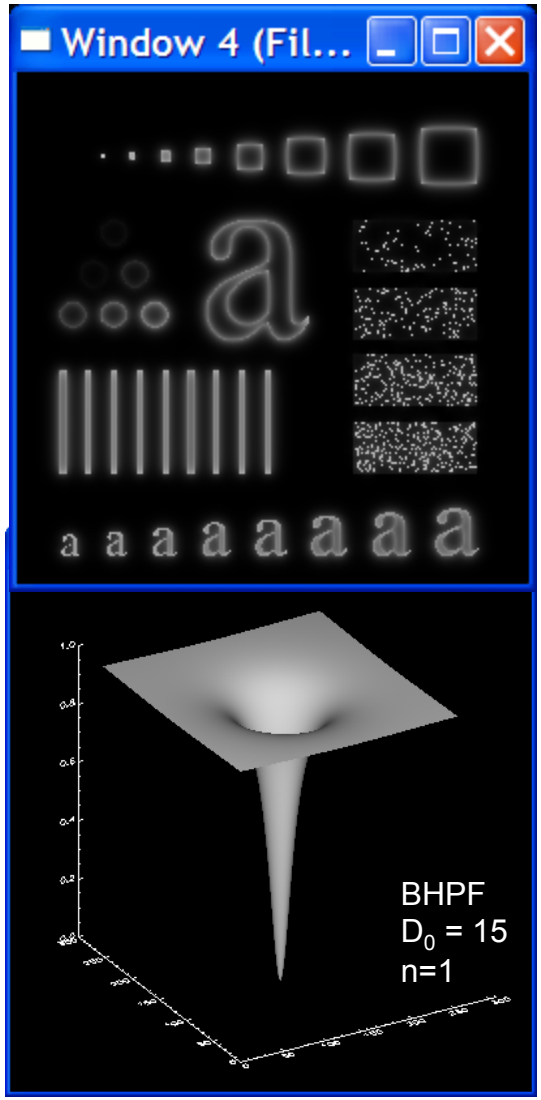


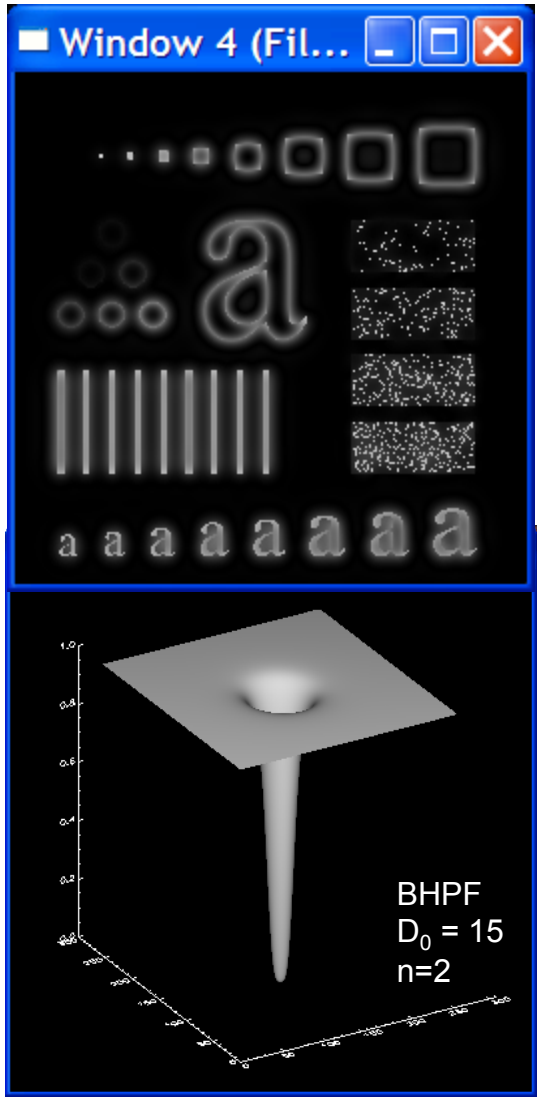



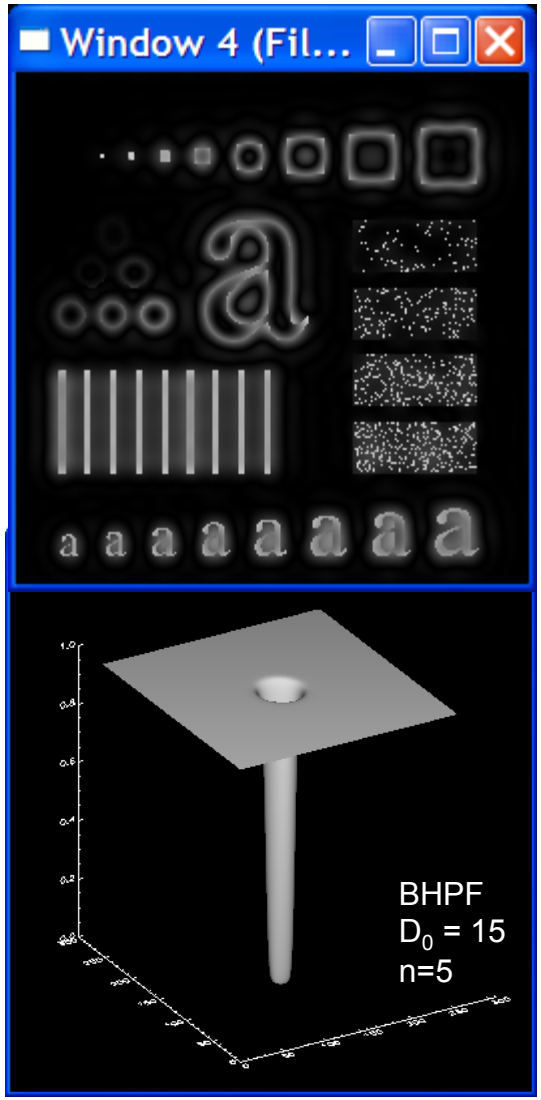



# Frequency Domain Filtering

## Comparing Butterworth Highpass Filters



# Frequency Domain Filtering

## Laplacian Filter

Given the property

$$\Im\left[\frac{d^n f(x)}{dx^n}\right] = (iu)^n F(u)$$

The Laplacian of $f(x,y)$ is arrived at as

$$\Im\left[\frac{d^2 f(x,y)}{dx^2} + \frac{d^2 f(x,y)}{dy^2}\right] = (iu)^2 F(u,v) + (iv)^2 F(u,v)$$

$$= i^2 u^2 F(u,v) + i^2 v^2 F(u,v)$$

$$= -(u^2 + v^2) F(u,v)$$

with the filter in the frequency domain being

$$H(u,v) = -(u^2 + v^2)$$

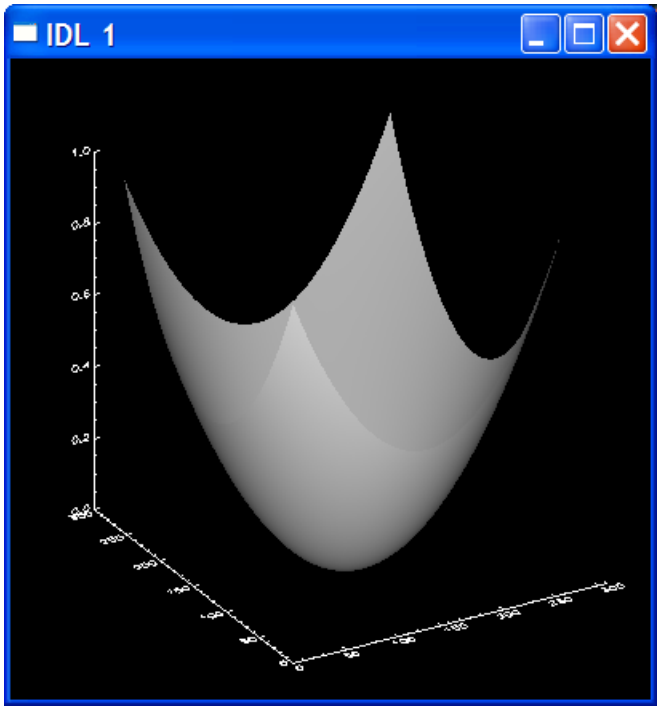
# Frequency Domain Filtering

## Laplacian Filter

The filter

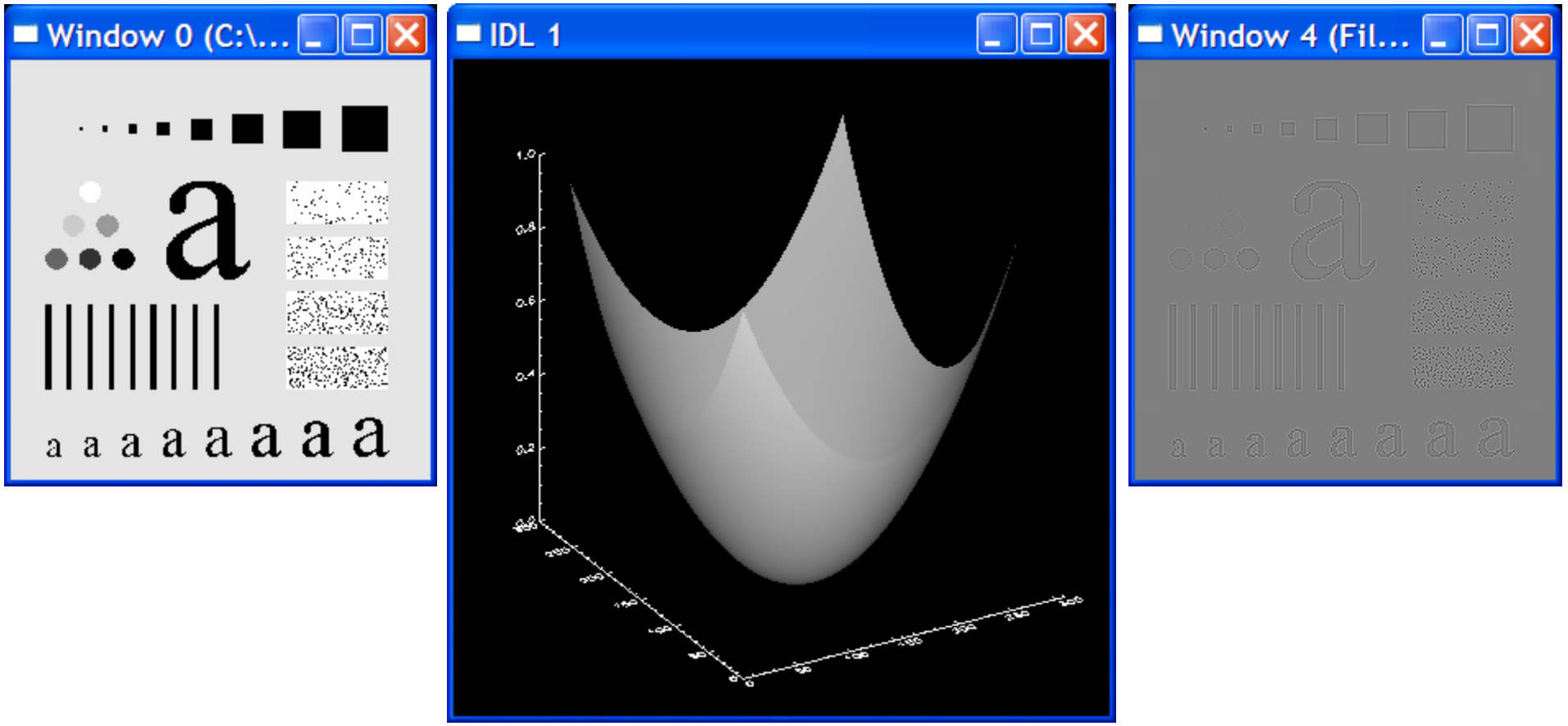
$$H(u,v) = -(u^2 + v^2)$$

can have very large negative values and must be scaled to fall within the range [0,1] by dividing by min( $H(u,v)$ ) to give you a filter that looks like



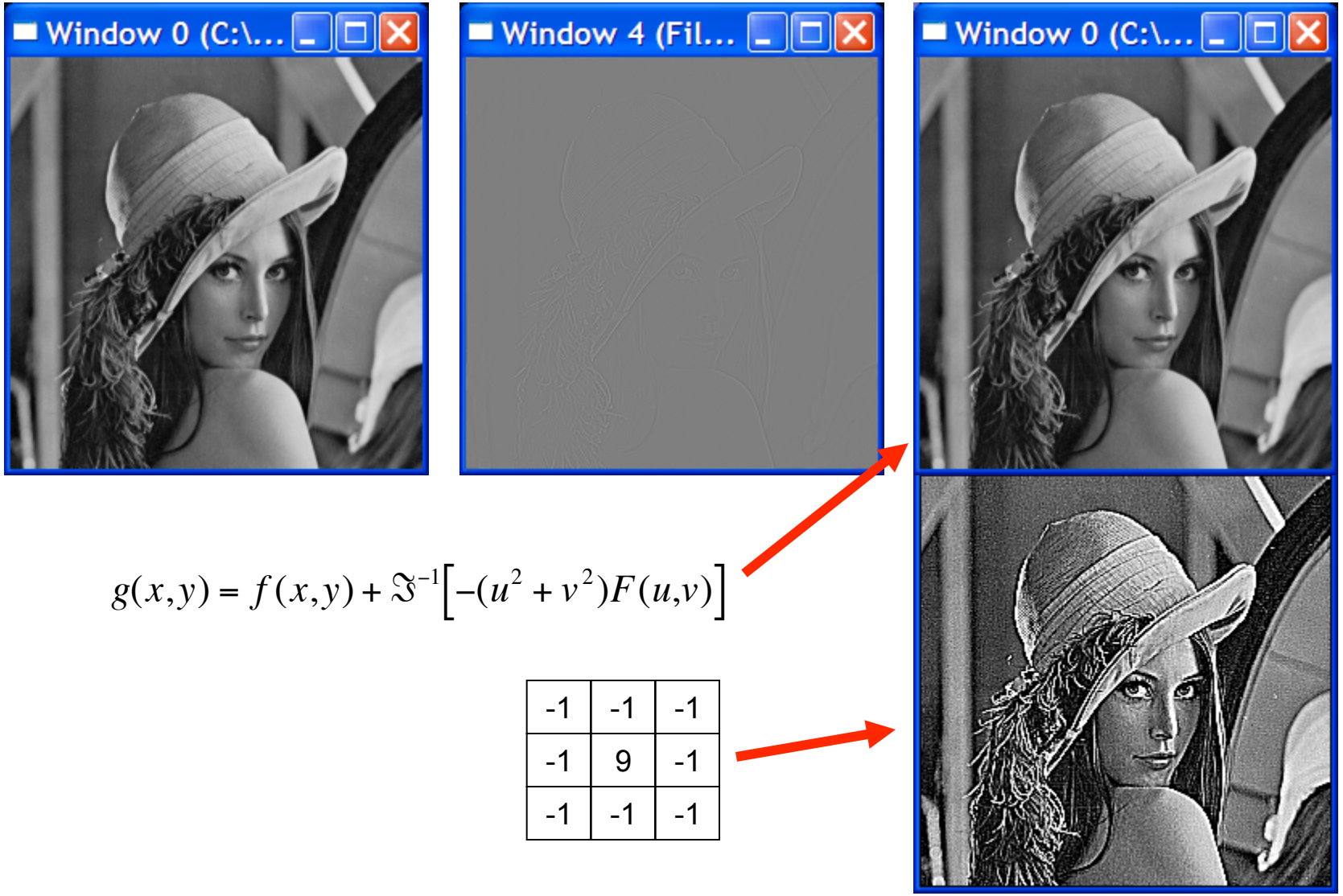
# Frequency Domain Filtering

## Laplacian Filter



# Frequency Domain Filtering

## Composite Laplacian Filter




$$g(x,y) = f(x,y) + \Im^{-1}\left[-(u^2+v^2)F(u,v)\right]$$

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

# Frequency Domain Filtering

## High Boost Filter

We know that a highpass filtered image is given by

$$f_{hp}(x,y) = f(x,y) - f_{lp}(x,y)$$

and that the high boost filtered imaged can be computed as

$$\begin{aligned} f_{hb}(x,y) &= Af(x,y) - f_{lp}(x,y) \\ &= (A-1)f(x,y) + f(x,y) - f_{lp}(x,y) \\ &= (A-1)f(x,y) + f_{hp}(x,y) \end{aligned}$$

Taking the Fourier transform of this equality we get

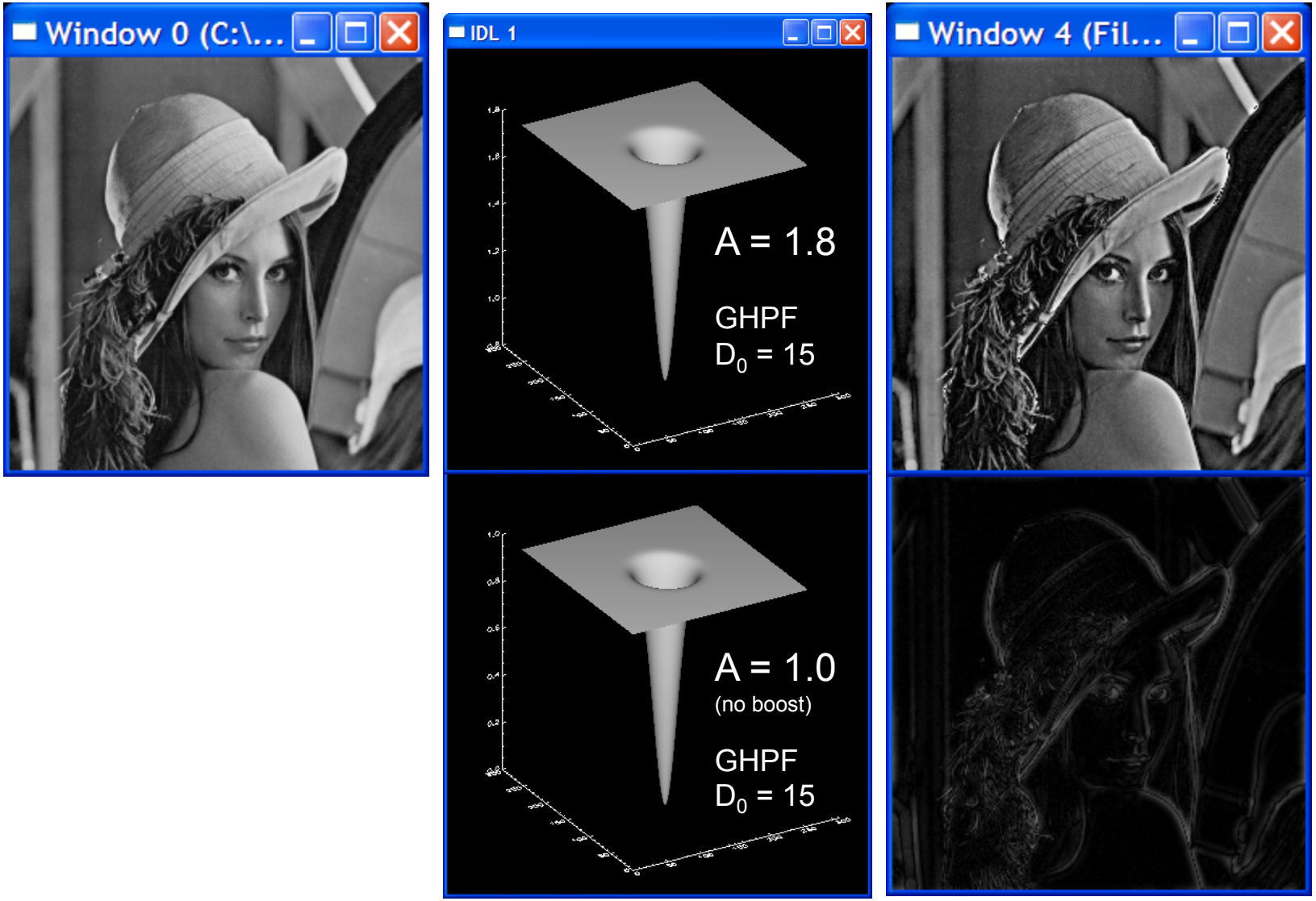
$$\begin{aligned} \Im[f_{hb}(x,y)] &= \Im[(A-1)f(x,y)] + \Im[f_{hp}(x,y)] \\ &= (A-1)F(u,v) + H_{hp}(u,v)F(u,v) \\ &= [(A-1) + H_{hp}(u,v)]F(u,v) \end{aligned}$$

giving us the frequency domain filter for high boost filtering

$$H_{hb}(u,v) = (A-1) + H_{hp}(u,v)$$

# Frequency Domain Filtering

## High Boost Filter



# Image Restoration

## Periodic Noise Reduction

Periodic noise, by its nature, allows for it easy removal in the frequency domain. As we have seen, any function can be broken down into a summation of sinusoids. If the function is periodic, these sinusoids will likely be of discrete, constant frequency and show up as bright, unique power spikes in the Fourier transform.

Just as we have designed ideal, Butterworth and Gaussian lowpass and highpass filters, we can use these same designs to remove power at specific frequencies. Filters of this sort fall into the general classes

Bandreject Filters
Bandpass Filters
Notch Filters

# Image Restoration

## Bandreject Filters

As the name implies, a bandreject filter will attenuate specific frequency values in an annulus about the origin of the frequency domain, or the DC component

Ideal Bandreject Filter

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) < D_0 - \frac{W}{2} \\ 0 & \text{if } D_0 - \frac{W}{2} \le D(u,v) \le D_0 + \frac{W}{2} \\ 1 & \text{if } D(u,v) > D_0 + \frac{W}{2} \end{cases}$$

Butterworth Bandreject Filter

$$H(u,v) = \frac{1}{1 + \left[ \frac{D(u,v)W}{D^2(u,v) - D_0^{\,2}} \right]^{2n}}$$

Gaussian Bandreject Filter

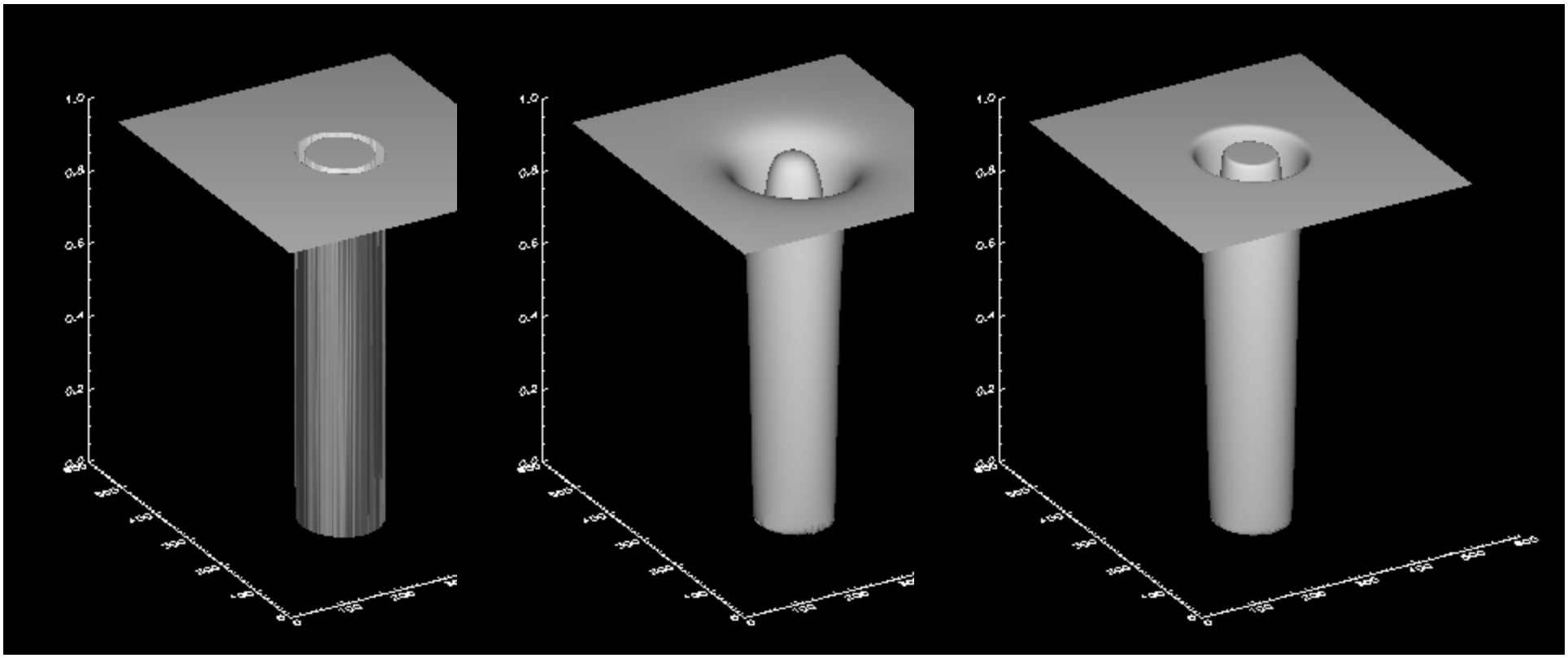
$$H(u,v) = 1 - e^{-\frac{1}{2}\left[ \frac{D^2(u,v) - D_0^{\,2}}{D(u,v)W} \right]^2}$$

where $D(u,v) = \sqrt{(u - M/2)^2 + (v - N/2)^2}$

$D_0$ is the radial center of the band rejection annulus,
$W$ is the width of the band, and
$n$ is the *order* of the filter (Butterworth only)

# Image Restoration

## Bandreject Filters



Ideal Bandreject Filter

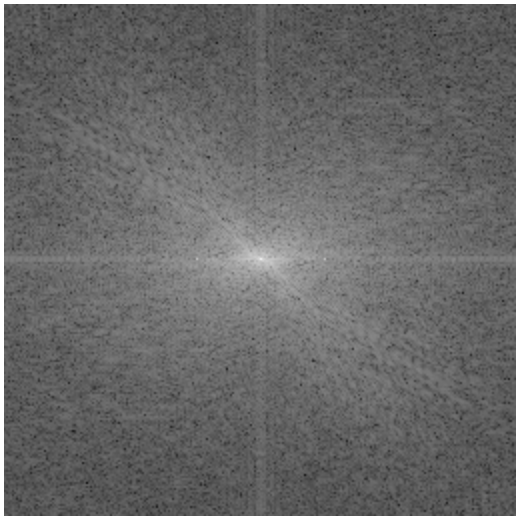
Butterworth Bandreject Filter
Order = 1

Gaussian Bandreject Filter

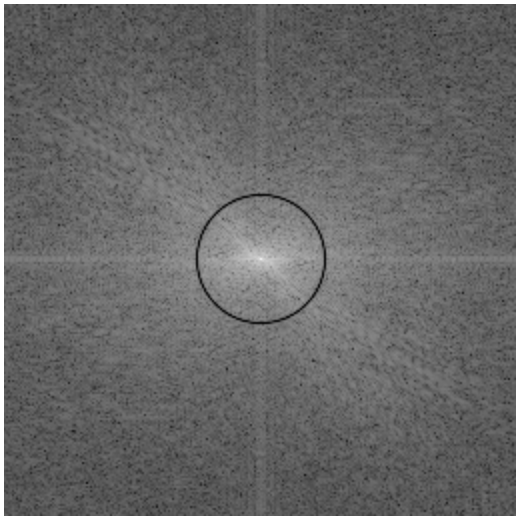
Radial Center = 32
Width of Band = 15

# Image Restoration

## Periodic Noise Reduction



```
image = tvrd()
noise = fltarr(256,256)
u = (findgen(256)-128)*(!PI/4)
for v = 0, 255 do $
        noise(*,v) = 16*sin(u)
noisy_image = image + noise
tv, noisy_image
```



Butterworth Bandreject Parameters

Radial Center = 32
Width of Band = 1
Order = 2

# Image Restoration

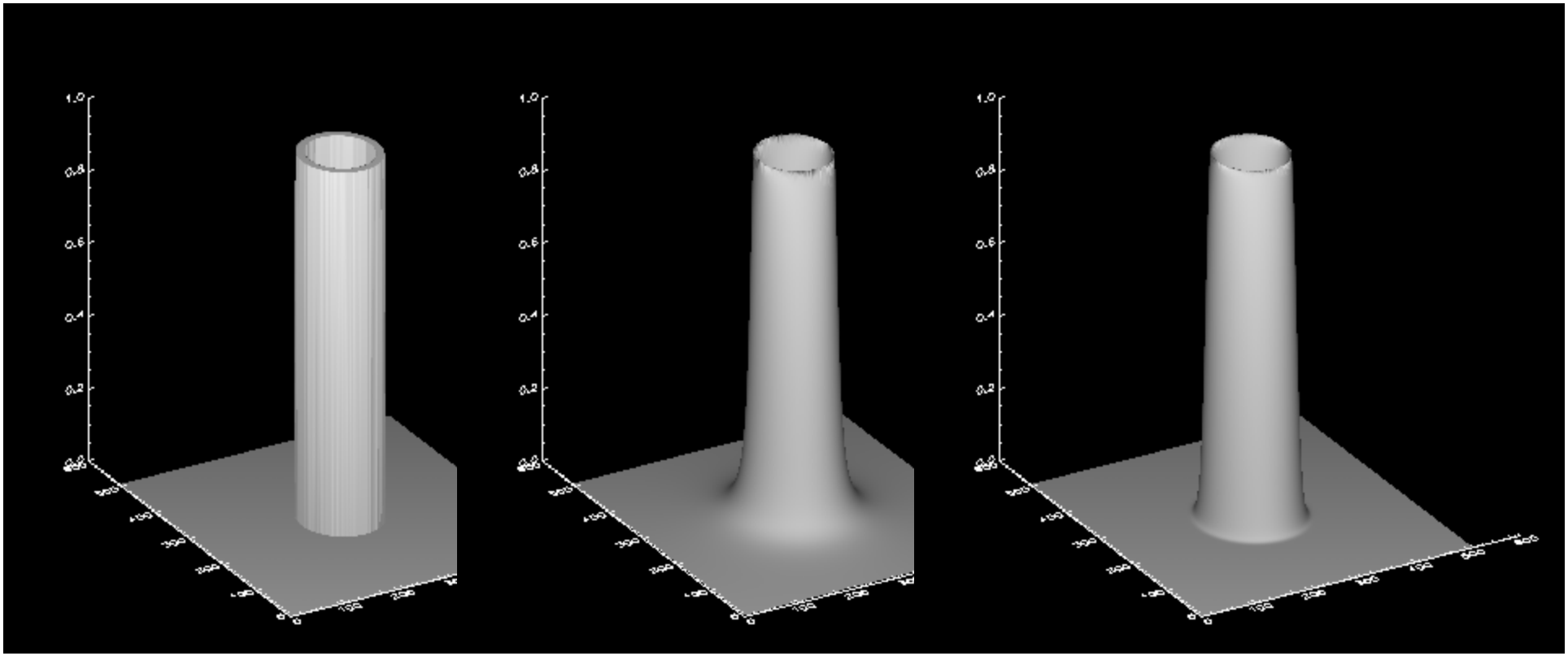
## Bandpass Filters

As the name implies, a bandpass filter will pass only specific frequency values in an annulus about the origin of the frequency domain, or the DC component. These filters are simply represented as the complement of the corresponding bandreject filter

$$H_{bandpass}(u,v) = 1 - H_{bandreject}(u,v)$$

# Image Restoration

## Bandpass Filters



Ideal Bandpass Filter

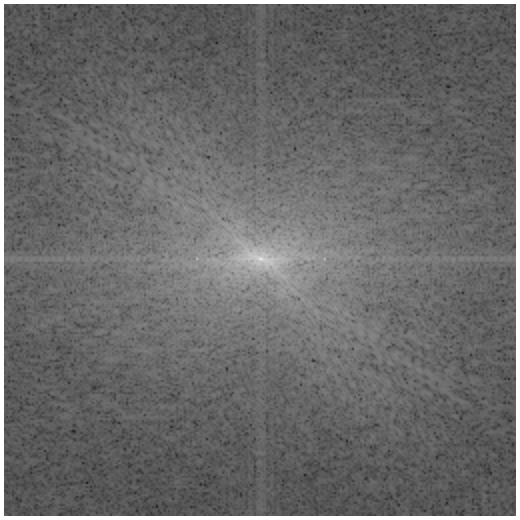
Butterworth Bandpass Filter
Order = 1

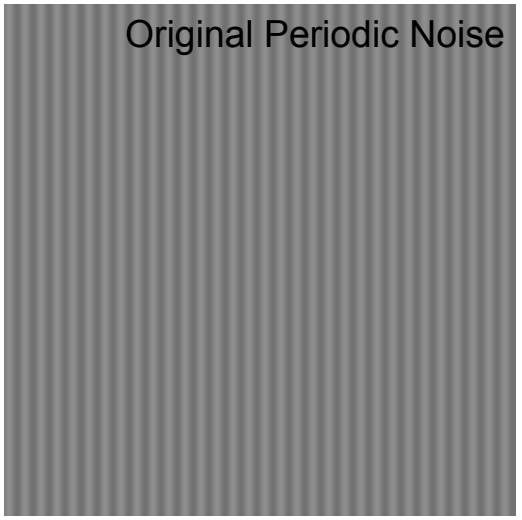
Gaussian Bandpass Filter

Radial Center = 32
Width of Band = 15

# Image Restoration

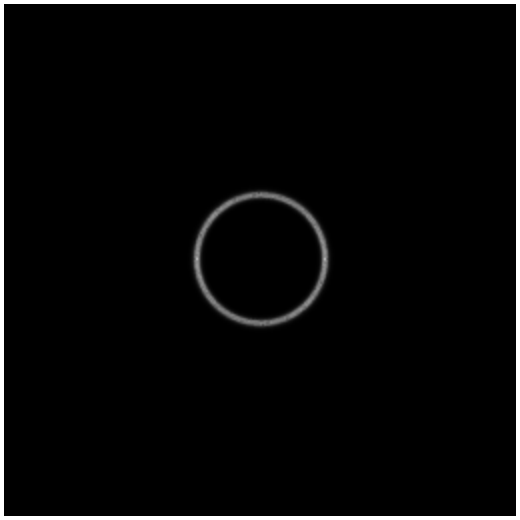
## Periodic Noise Isolation

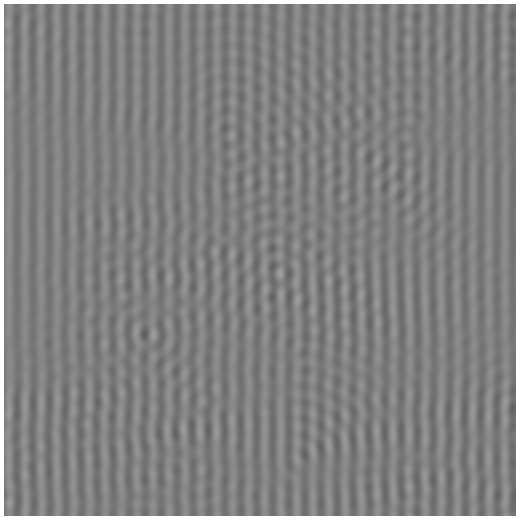





Butterworth Bandpass Parameters

Radial Center = 32
Width of Band = 1
Order = 2





# Image Restoration

## Notch Reject Filters

These filters reject only frequencies in a neighborhood around a specified central frequency. Due to the symmetry of the Fourier transform these filters must appear as symmetric pairs around the central frequency

**Ideal Notch Filter**

$$H(u,v) = \begin{cases} 0 & \text{if } D_1(u,v) \le D_0 \text{ or } D_2(u,v) \le D_0 \\ 1 & \text{otherwise} \end{cases}$$

**Butterworth Notch Reject Filter**

$$H(u,v) = \frac{1}{1 + \left[ \frac{D_0^{\,2}}{D_1(u,v) D_2(u,v)} \right]^n}$$

**Gaussian Notch Reject Filter**

$$H(u,v) = 1 - e^{-\frac{1}{2}\left[ \frac{D_1(u,v) D_2(u,v)}{D_0^{\,2}} \right]}$$

where $D_1(u,v) = \sqrt{(u - M/2 - u_0)^2 + (v - N/2 - v_0)^2}$

$D_2(u,v) = \sqrt{(u - M/2 + u_0)^2 + (v - N/2 + v_0)^2}$

$D_0$ is the radius of the notch filter with centers at $(u_0, v_0)$ and $(-u_0, -v_0)$

$n$ is the *order* of the filter (Butterworth only)

# Image Restoration

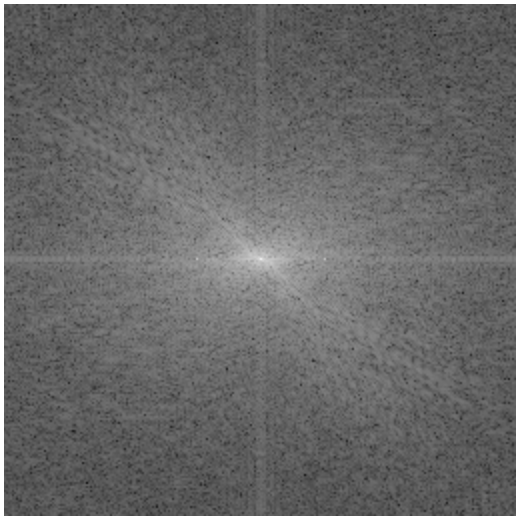
## Notch Pass Filters

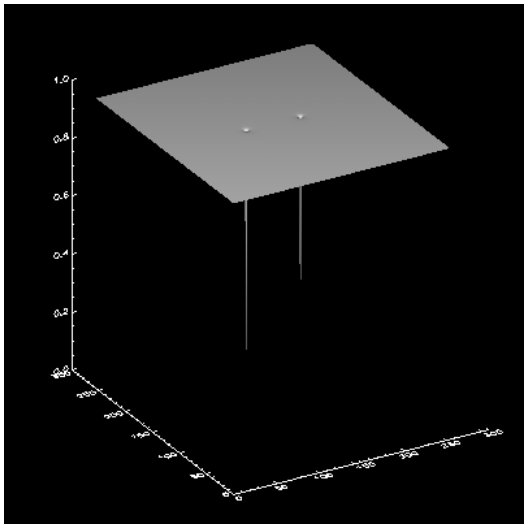
These filters pass only frequencies in a neighborhood around a specified central frequency. Due to the symmetry of the Fourier transform these filters must appear as symmetric pairs around the central frequency

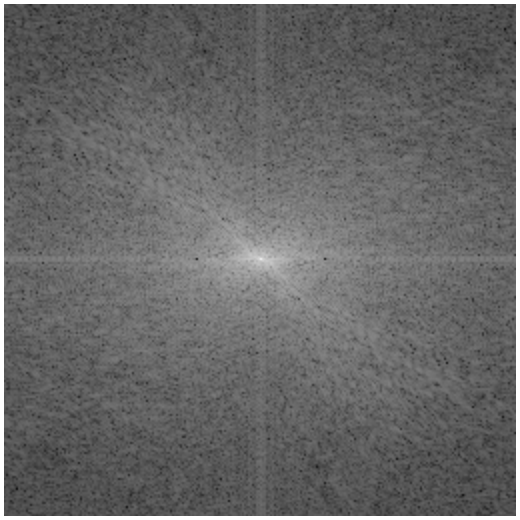
$$H_{notchpass}(u,v) = 1 - H_{notchreject}(u,v)$$

# Image Restoration

## Periodic Noise Reduction





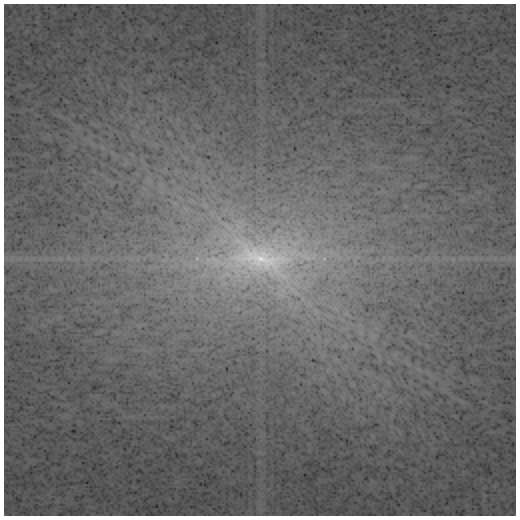


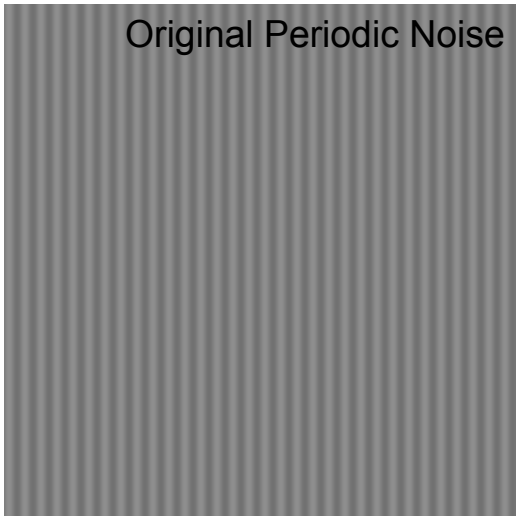
Butterworth Notch Pass Parameters

Radial Center = 32
Width of Notch = 3
Order = 2

# Image Restoration

## Periodic Noise Isolation






Butterworth Notch Pass Parameters

Radial Center = 32
Width of Notch = 3
Order = 2