Frequencies

Agenda

- Texture
- Pyramid matching
- Frequencies

Cost function

$$\min_{Z,D} C(Z, D, X) \quad \text{where} \quad C(Z, D, X) = \sum_i ||x_i - d_{z_i}||^2$$

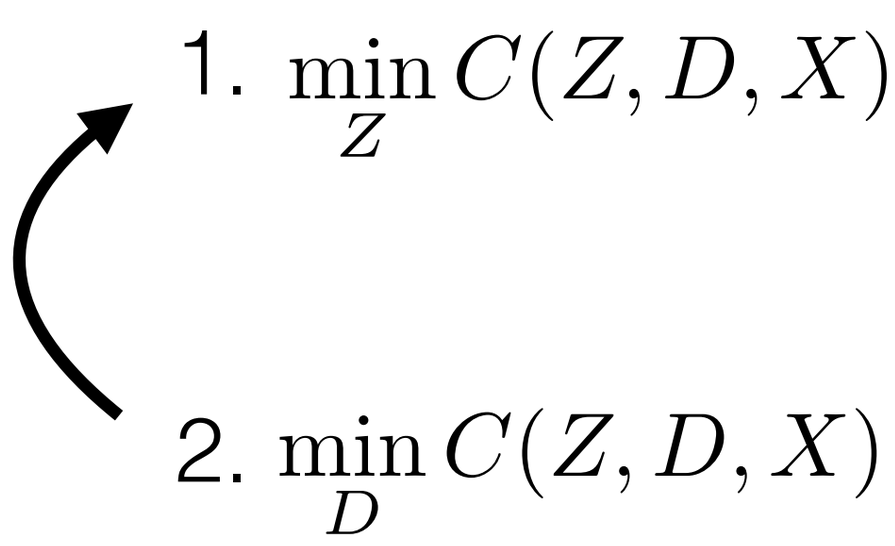
x_i: i^{th} input vector (to be clustered)

$z_i \in \{1 \dots K\}$: i^{th} label

d_k: k^{th} dictionairy element (or mean)

Coordinate descent optimization

$$\min_{Z,D} C(Z, D, X) \quad \text{where} \quad C(Z, D, X) = \sum_i ||x_i - d_{z_i}||^2$$



Aside: what happens if we are just given a *matrix* of pairwise distances?

Sparse reconstructions

$$\min_{D,Z} ||X - DZ||_F^2$$ subject to sparse constraints on Z

$$X = [x_1, \ldots x_n]$$

$$D = [d_1, \ldots, d_K]$$

$$Z = [z_1, \ldots z_n]$$

K-means: $z_i = [\ldots, 0, 1, 0, \ldots]$

L0 sparse-coding: $||z_i||_0 \leq M$ (greedy algorithms known as “matching pursuit”)

L1 sparse-coding: $||z_i||_1 \leq M$ (convex program)

Dictionary learning

$$\min_{D,Z} ||X - DZ||_F^2$$

Some folks claim that k-means should always be replaced by sparse coding (never hurts, sometimes better) even for bag-of-words

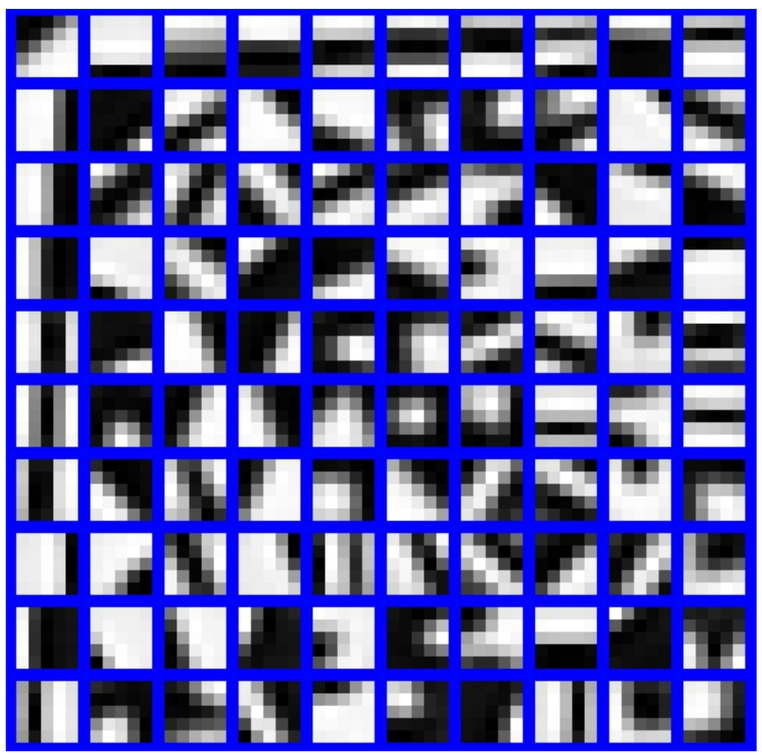
… k-means is far simpler, right?

“In between” k-means and sparse coding

$$\min_{D,Z} ||X - DZ||_F^2$$ subject to sparse constraints on Z

K-means: $z_i = [\ldots, 0, 1, 0, \ldots]$

L0 sparse-coding: $||z_i||_0 \leq M$ (For M = 1, we can solve for Z in closed form)



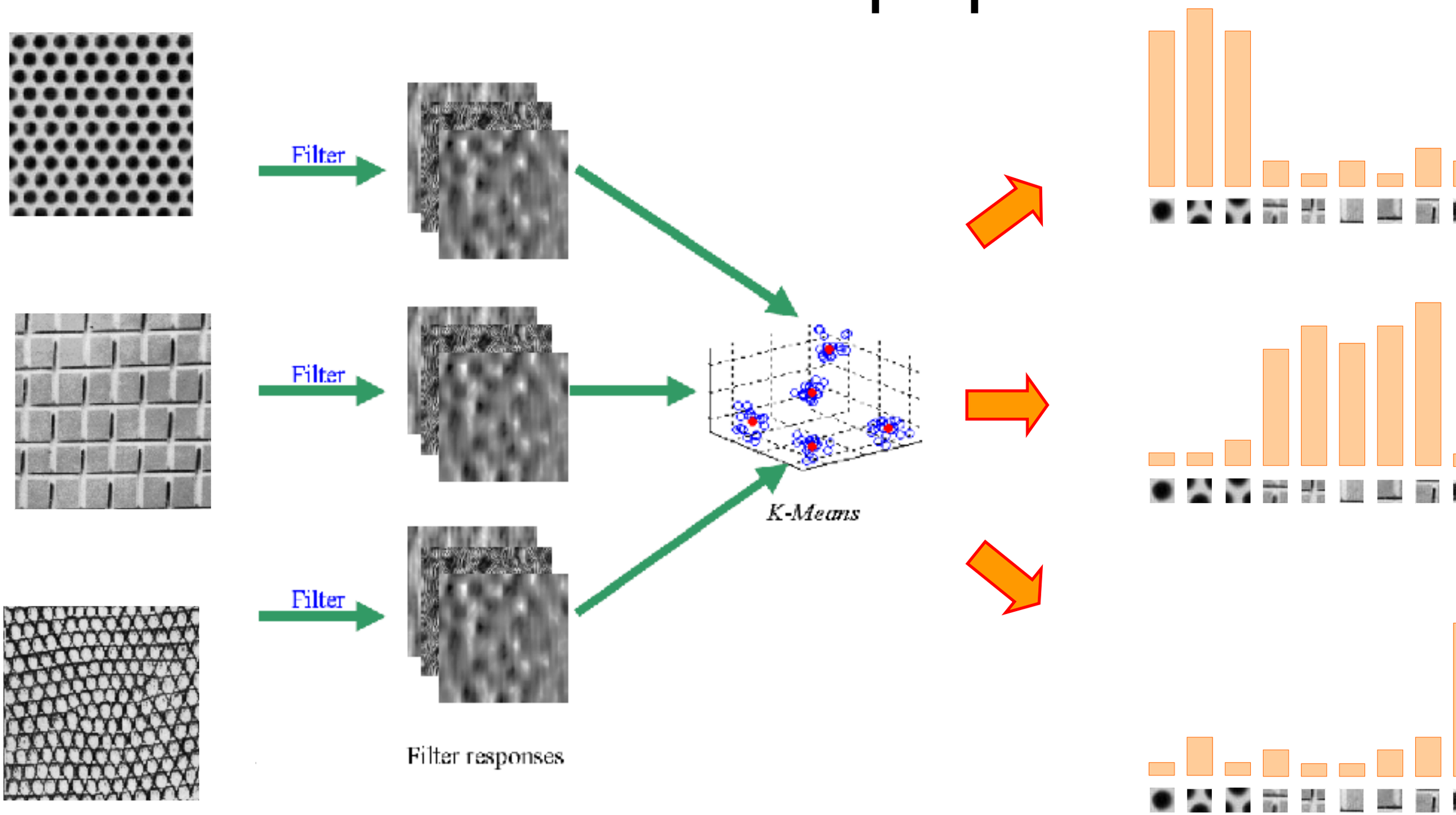
Similar to k-means using cosine distance, but allows for negative matches

Histograms of Sparse Codes for Object Recognition, CVPR13

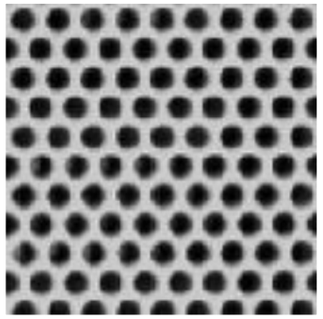
Agenda

- Texture
- Pyramid matching
- Frequencies

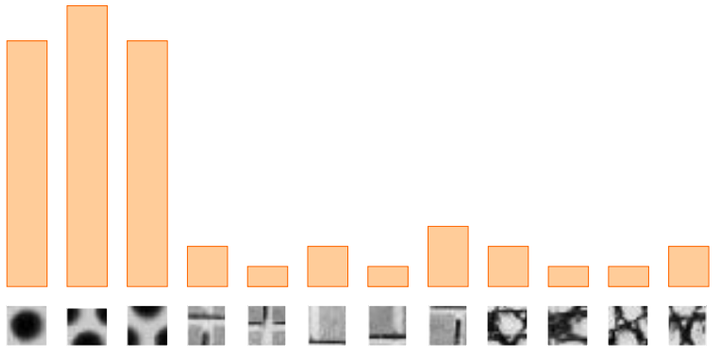
Recall texture pipeline

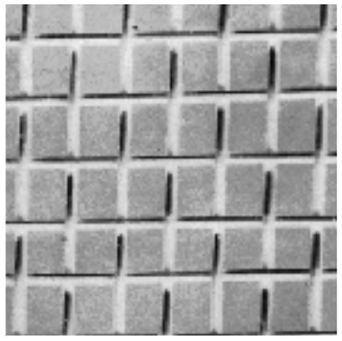


Recall texture pipeline

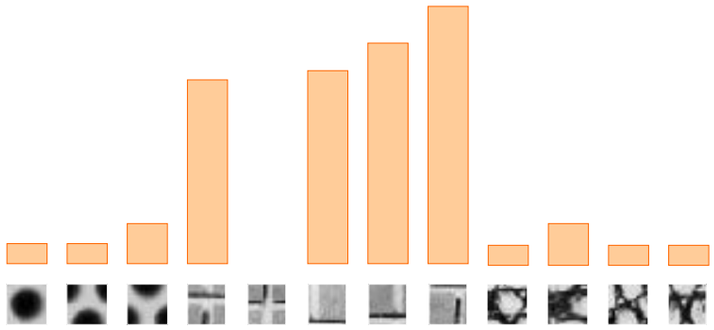


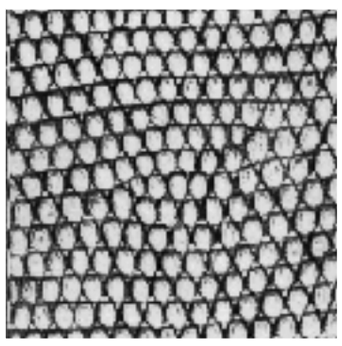
polka-dot



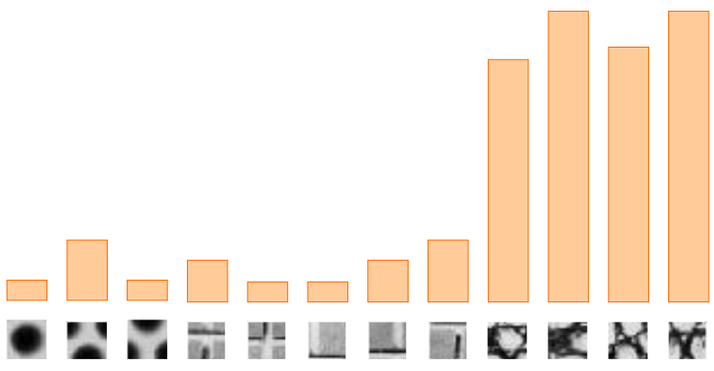


brick

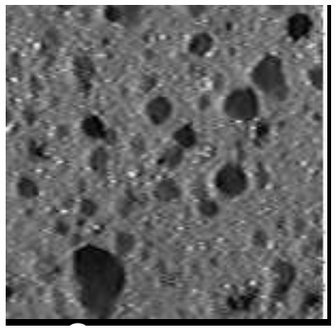


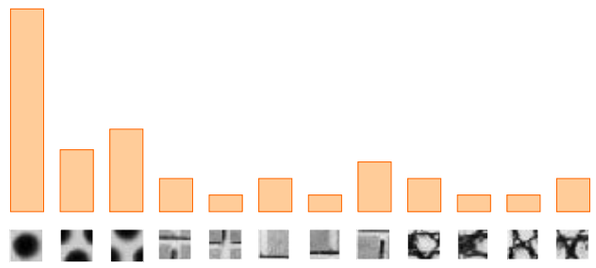


mesh



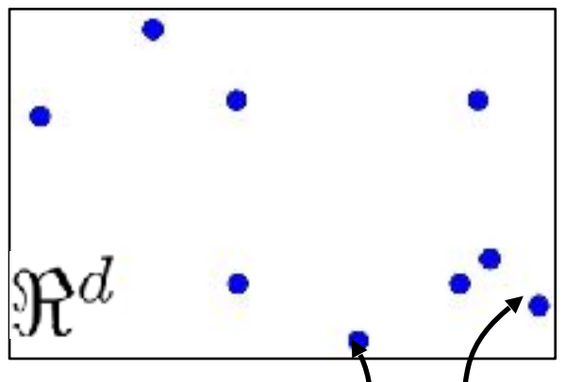
match

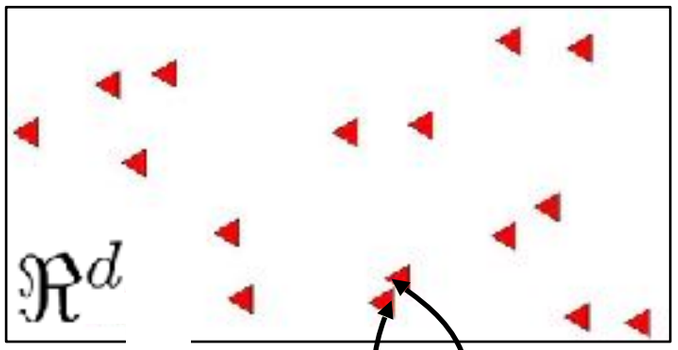


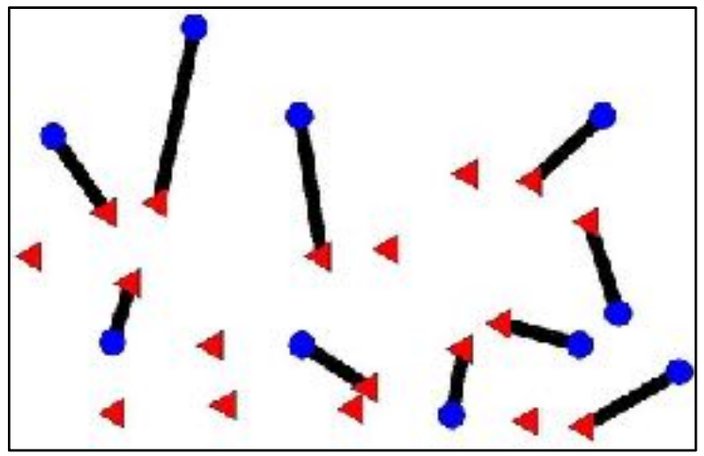


Alternative to quantization: explicitly compute matchings between image pairs

Grauman & Darrell







optimal partial matching

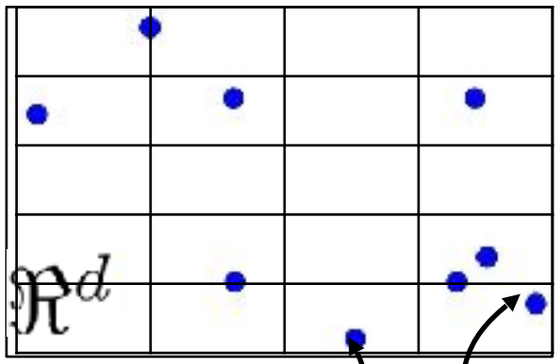
$$\max_{\pi:\mathbf{X}\to\mathbf{Y}} \sum_{\mathbf{x}_i\in\mathbf{X}} \mathcal{S}(\mathbf{x}_i, \pi(\mathbf{x}_i))$$

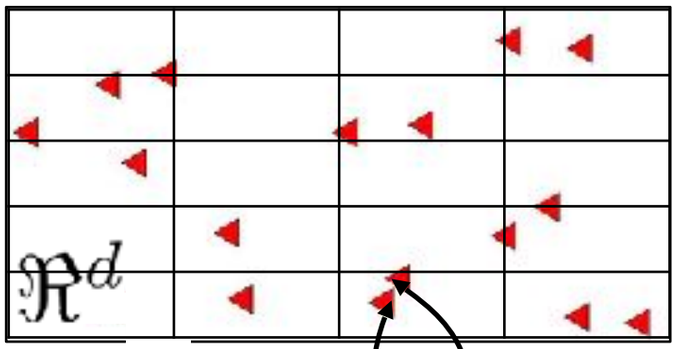
$$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\}$$
$$\vec{\mathbf{x}}_i \in \Re^d$$

$$\mathbf{Y} = \{\vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{y}}_n\}$$
$$\vec{\mathbf{y}}_i \in \Re^d$$

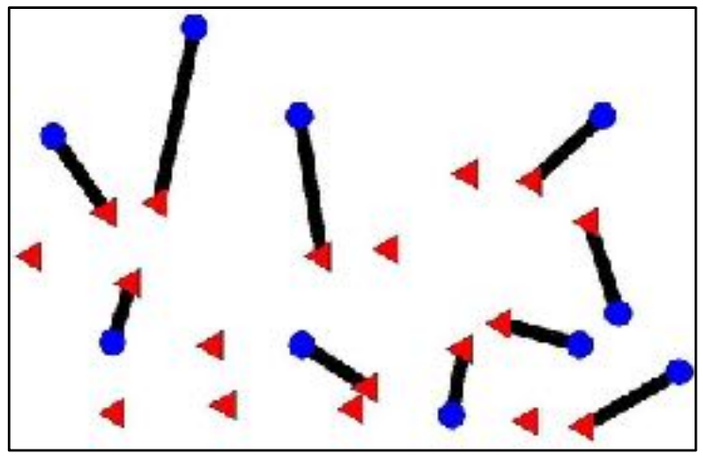
Digression: alternative to quantization

Approximate matching with histogram similarity





$\approx$

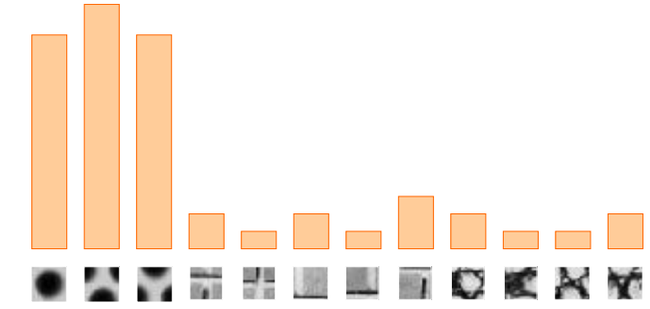


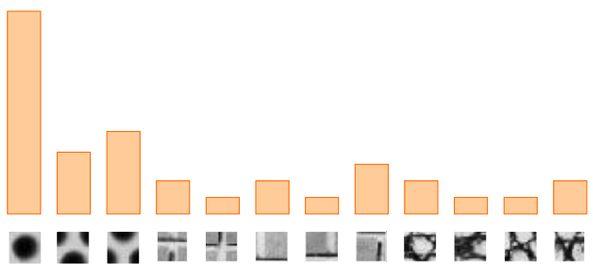
$$\mathbf{X} = \{\vec{\mathbf{x}}_1, \ldots, \vec{\mathbf{x}}_m\}$$
$$\vec{\mathbf{x}}_i \in \Re^d$$

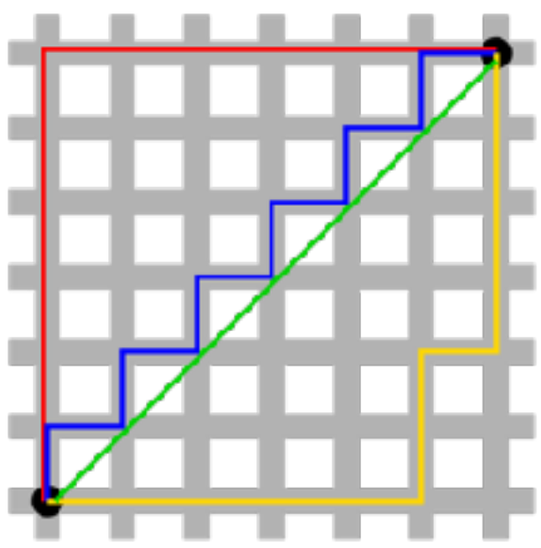
$$\mathbf{Y} = \{\vec{\mathbf{y}}_1, \ldots, \vec{\mathbf{y}}_n\}$$
$$\vec{\mathbf{y}}_i \in \Re^d$$

Aside: what's the "right" way to compare histograms?

$$D_r(x,y) = \Big(\sum_i (x_i - y_i)^r\Big)^{\frac{1}{r}}$$







euclidean (r=2) or manhattan (r=1)

Aside: what's the "right" way to compare histograms?

euclidean (r=2)
or manhattan (r=1)

$$D_r(x,y) = \left(\sum_i (x_i - y_i)^r\right)^{\frac{1}{r}}$$

chi-squared distance
[derived from chi-squared text in statistics]

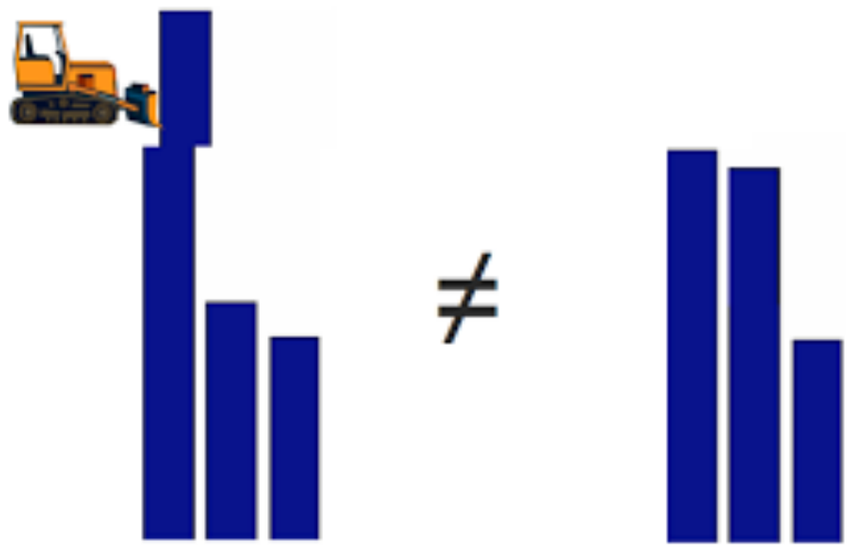
$$Chi(x,y) = \sum_i \frac{(x_i - y_i)^2}{x_i + y_i}$$

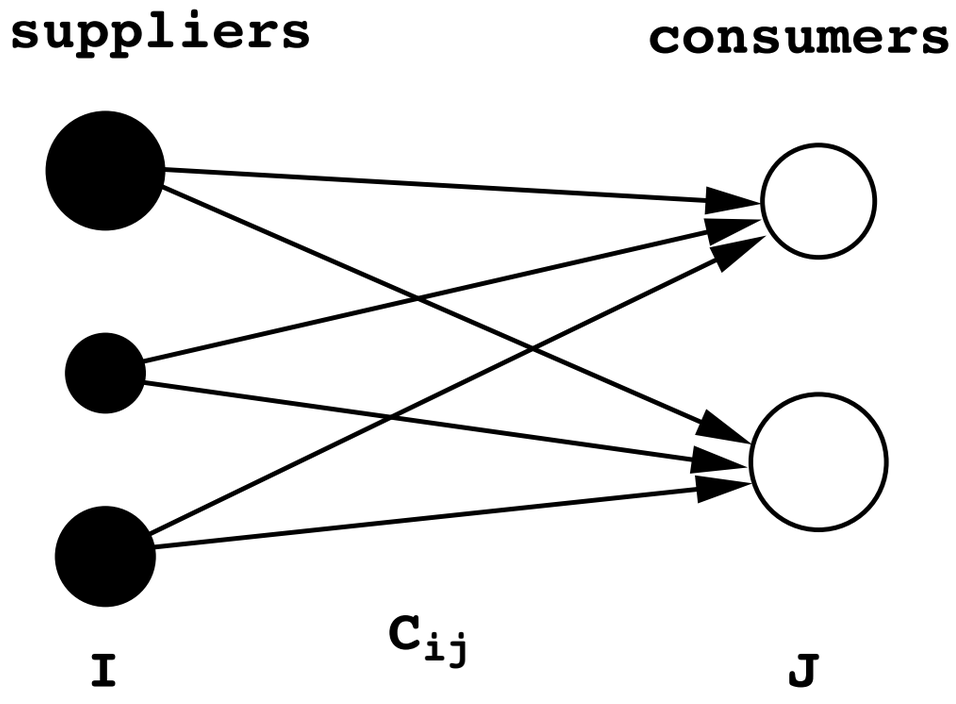
K-L divergence
[log probability of seeing x under model y]

$$D_{KL}(x,y) = \sum_i x_i \log \frac{x_i}{y_i}$$

Earth mover's distance

Cast as "transportation problem"

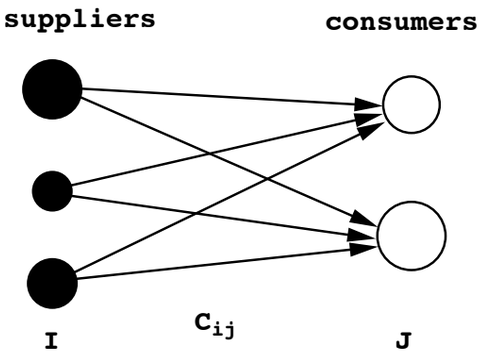




Earth mover's distance

Bipartite network flow

$$\min_{f_{ij}} \sum_{ij} c_{ij} f_{ij} \quad s.t.$$



$$f_{ij} \geq 0$$

$$\sum_i f_{ij} = y_j$$

$$\sum_j f_{ij} = x_i$$

$$EMD(\mathbf{x}, \mathbf{y}) = \sum_{ij} c_{ij} f_{ij}$$

https://www.cs.duke.edu/~tomasi/papers/rubner/rubnerTr98.pdf

Similarity Kernals

[sometimes more intuitive to define than distance functions]

$$D(x,y) = K(x,x) + K(y,y) - 2K(x,y)$$

$$K_{lin}(x,y) = \sum_i x_i y_i$$

[what's corresponding distance function?]

Similarity Kernals

[sometimes more intuitive to define than distance functions]

$$D(x,y) = K(x,x) + K(y,y) - 2K(x,y)$$

$$K_{lin}(x,y) = \sum_i x_i y_i$$

$$K_{int}(x,y) = \sum_i \min(x_i, y_i)$$

What happens if x,y are binary vectors?

Similarity Kernals

[sometimes more intuitive to define than distance functions]

$$D(x,y) = K(x,x) + K(y,y) - 2K(x,y)$$

$$K_{lin}(x,y) = \sum_i x_i y_i$$

$$K_{int}(x,y) = \sum_i \min(x_i, y_i)$$

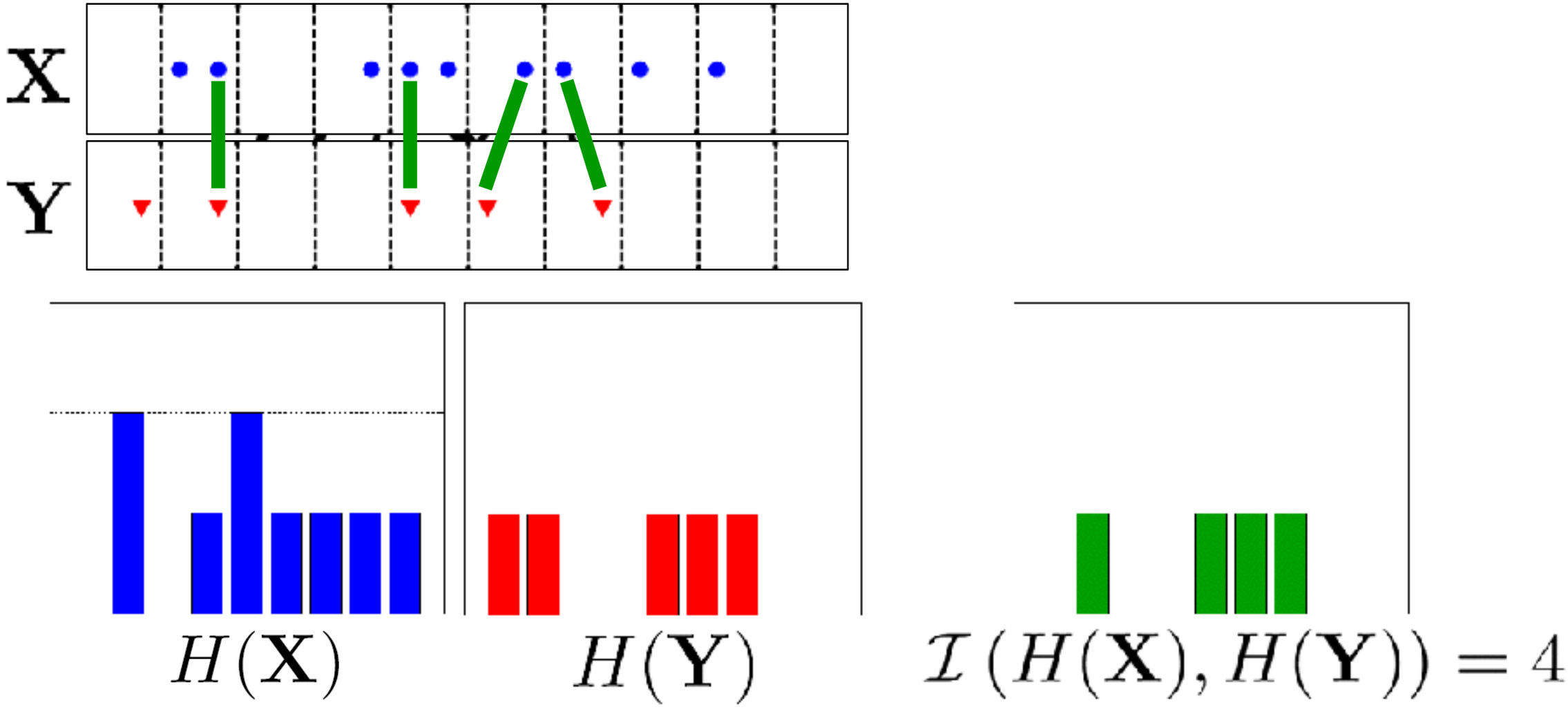
$$K_{bat}(x,y) = \sum_i \sqrt{x_i y_i}$$

It turns out, we can compute transformations f(x) and f(y) such that L2 distance in transformed space corresponds to these kernals (allows use of linear predictors)

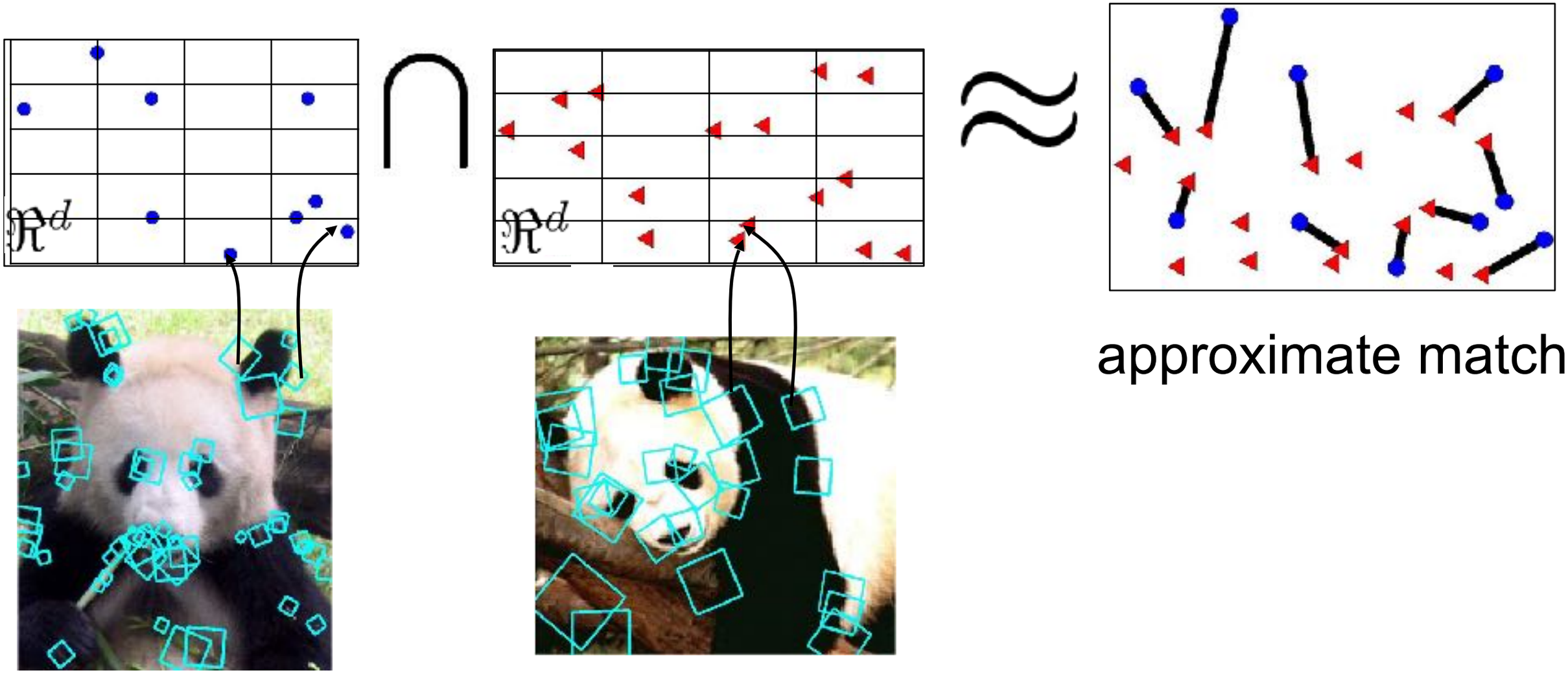
http://www.robots.ox.ac.uk/~vgg/software/homkermap/

Histogram intersection kernal

$$\mathcal{I}(H(\mathbf{X}), X(\mathbf{Y})) = \sum_k \min(H(\mathbf{X}_k), X(\mathbf{Y}_k))$$

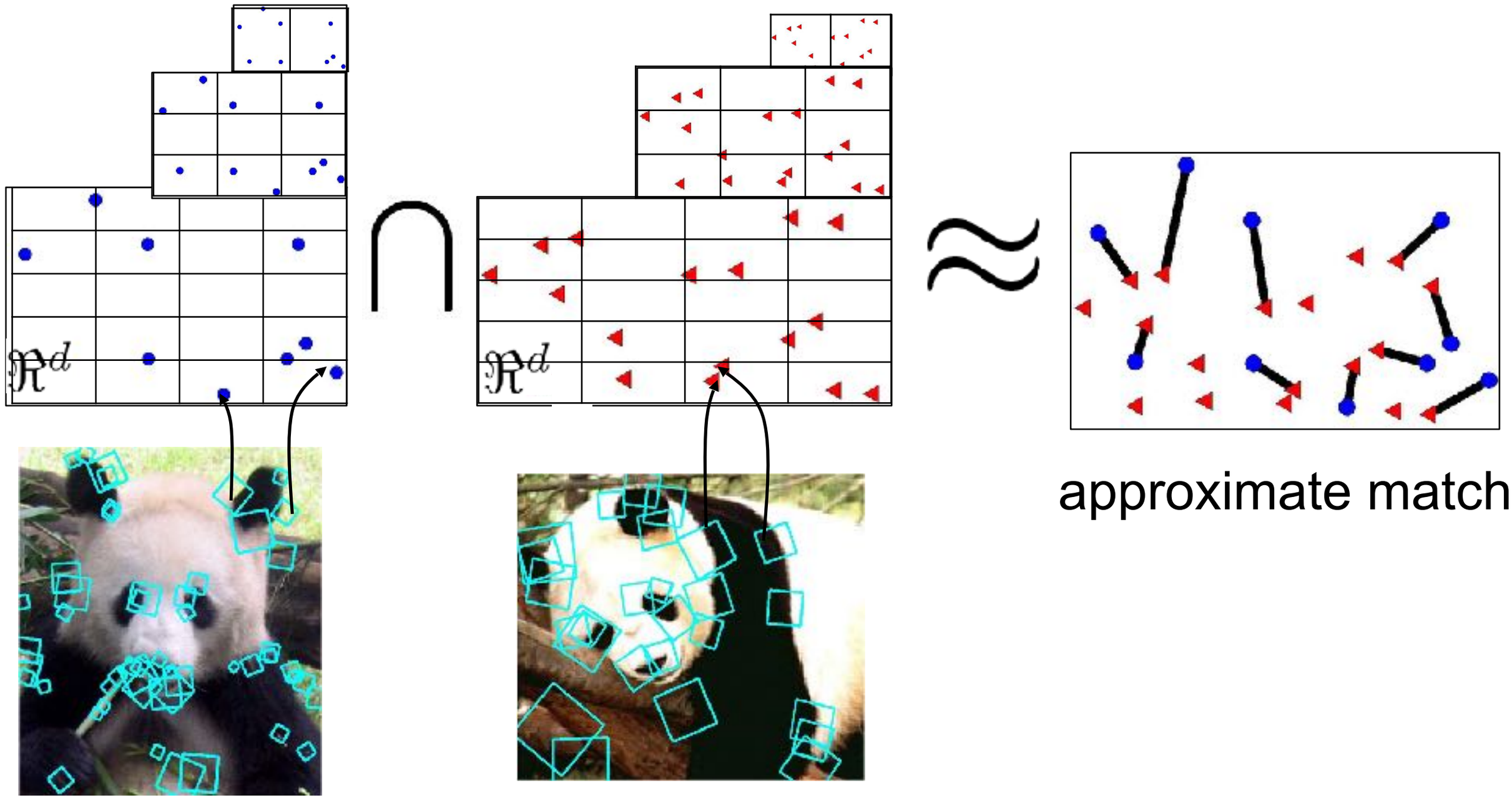


Back to correspondence matching



But what about bin effects (partial credit for near matches)?

Back to correspondence matching



Count matches obtained from larger bins

Counting new matches

Histogram intersection

$$\mathcal{I}\left(H(\mathbf{X}), H(\mathbf{Y})\right) = \sum_{j=1}^{r} \min\left(H(\mathbf{X})_j, H(\mathbf{Y})_j\right)$$

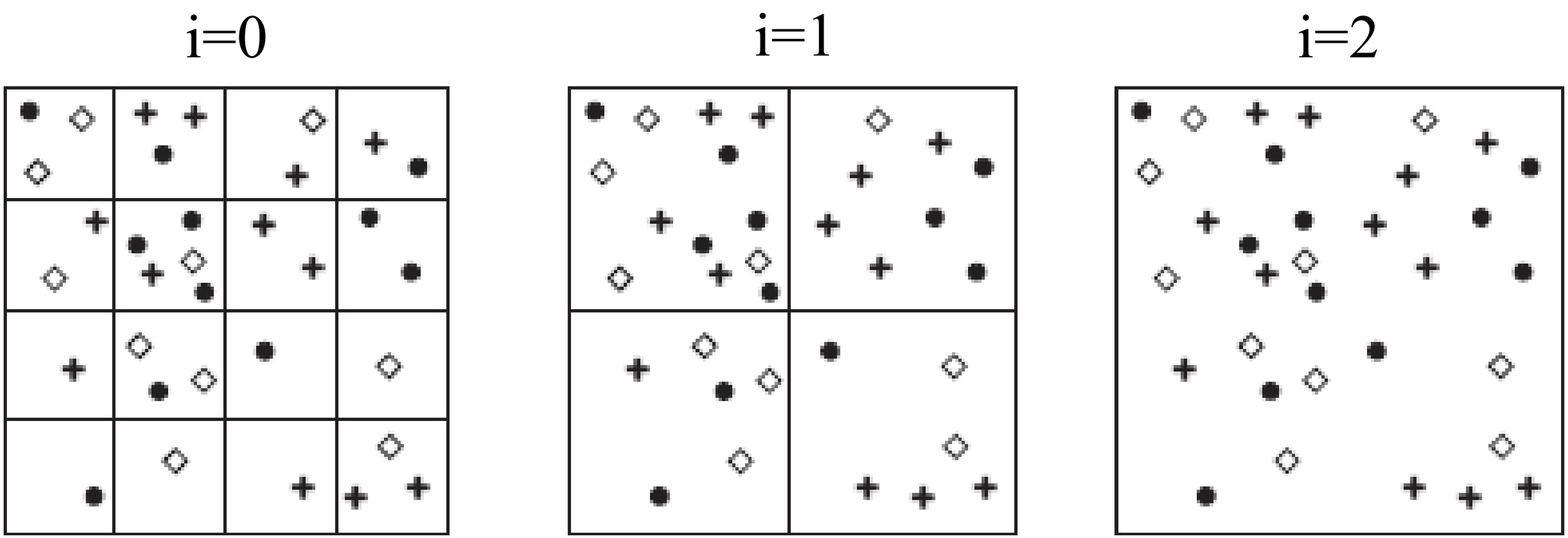
matches at this level

matches at previous level

$$N_i = \mathcal{I}\left(H_i(\mathbf{X}), H_i(\mathbf{Y})\right) - \mathcal{I}\left(H_{i-1}(\mathbf{X}), H_{i-1}(\mathbf{Y})\right)$$

Difference in histogram intersections across levels counts *number of new pairs* matched

Giving partial credit for new matches



Weight new matches inversely porportional to bin size

$$\frac{1}{2^i}$$

Pyramid match kernel

histogram pyramids

$$K_\Delta\left(\overbrace{\Psi(\mathbf{X}), \Psi(\mathbf{Y})}\right) =$$

$$\sum_{i=0}^{L} \frac{1}{2^i} \Big(\underbrace{\mathcal{I}\left(H_i(\mathbf{X}), H_i(\mathbf{Y})\right) - \mathcal{I}\left(H_{i-1}(\mathbf{X}), H_{i-1}(\mathbf{Y})\right)}_{\text{number of newly matched pairs at level } i} \Big)$$

measure of difficulty of a match at level *i*

- Weights inversely proportional to bin size
- Normalize kernel values to avoid favoring large sets

Spatial Pyramid Matching

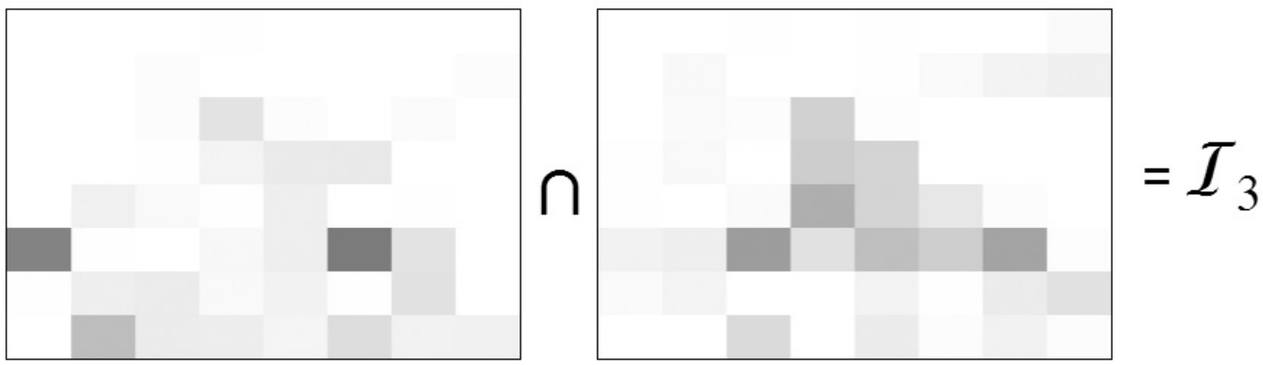
Quantize features into words, but build pyramid in space

Nifty way to encode constraints like “eye” words lie near top of image

Original images

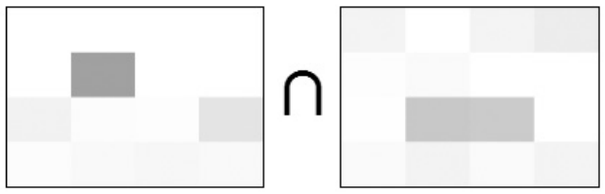
Feature histograms:

Level 3



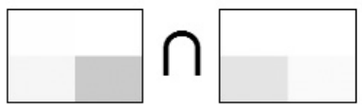
$= \mathcal{I}_3$

Level 2



$= \mathcal{I}_2$

Level 1



$= \mathcal{I}_1$

Level 0

$= \mathcal{I}_0$

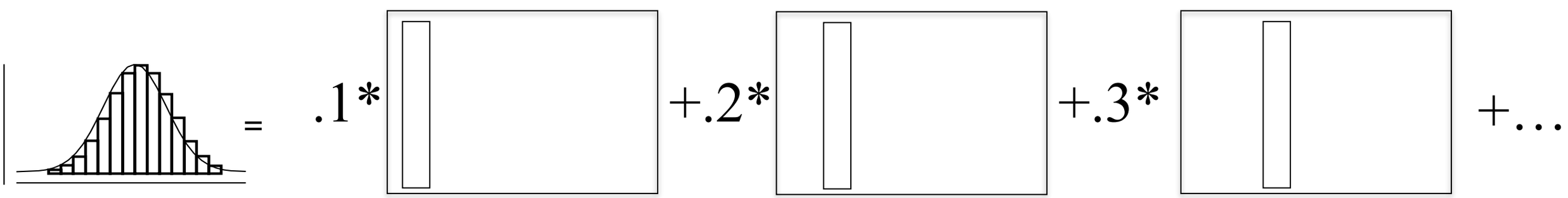
Total weight (value of *pyramid match kernel*): $\mathcal{I}_3 + \frac{1}{2}(\mathcal{I}_2 - \mathcal{I}_3) + \frac{1}{4}(\mathcal{I}_1 - \mathcal{I}_2) + \frac{1}{8}(\mathcal{I}_0 - \mathcal{I}_1)$

Agenda

- Texture
- Pyramid matching
- Frequencies

Motivation

Representing a signal as a linear combination of impulse functions

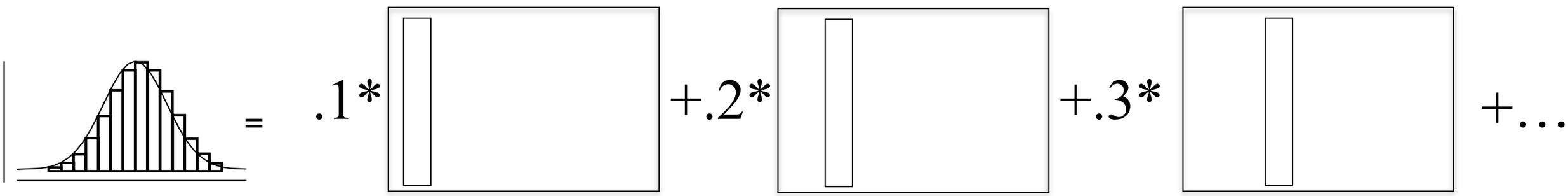


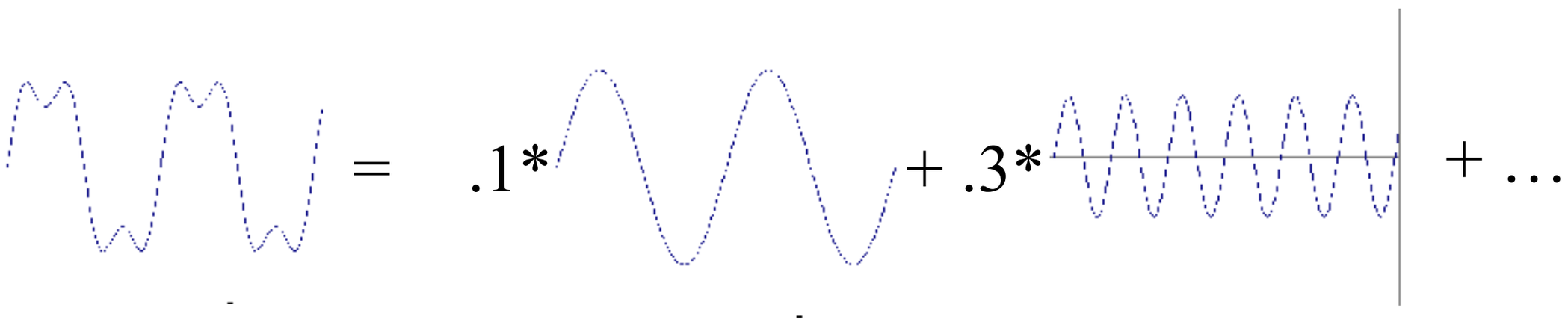
..allows us to characterize linear-shift invariant (LSI) systems by their impulse response

What about other basis functions besides impulses?

Lets pick a smooth basis

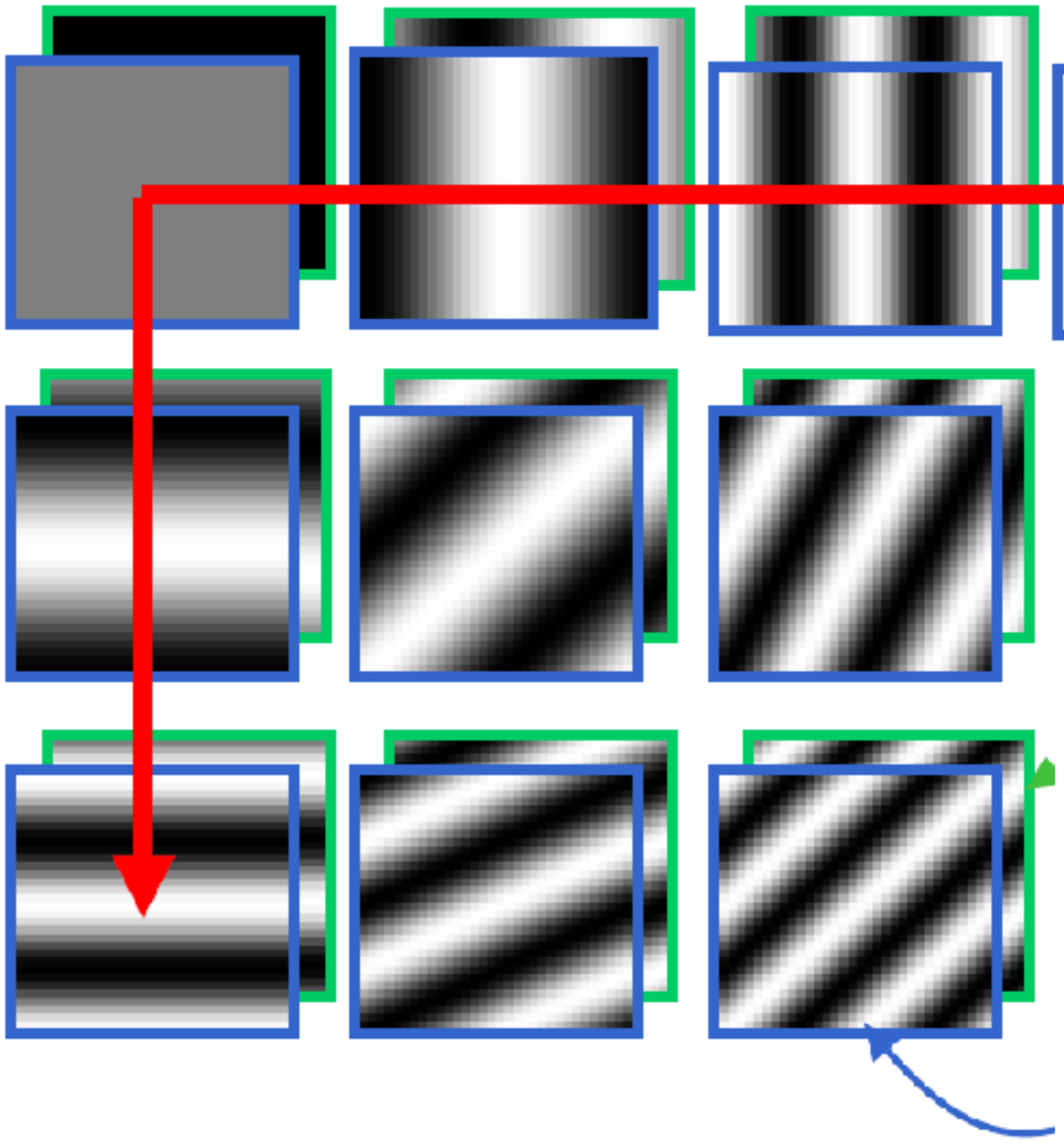
to reflect fact that real-world signals tend to be slow-changing



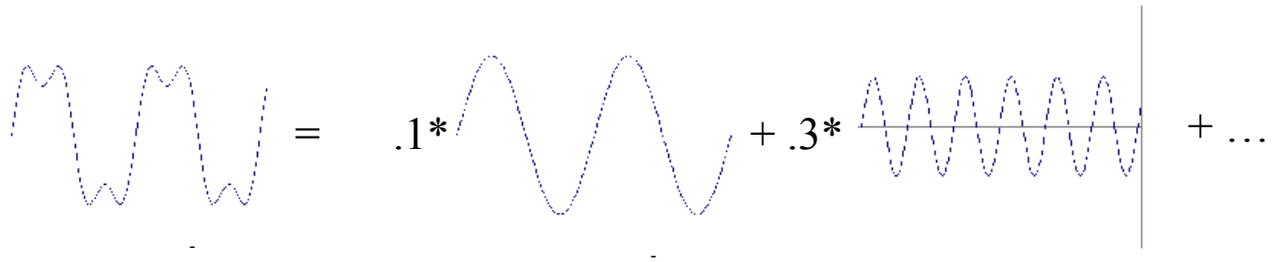


sine functions (sinusoids)

Where we are headed…



Why sinusoidal basis functions?



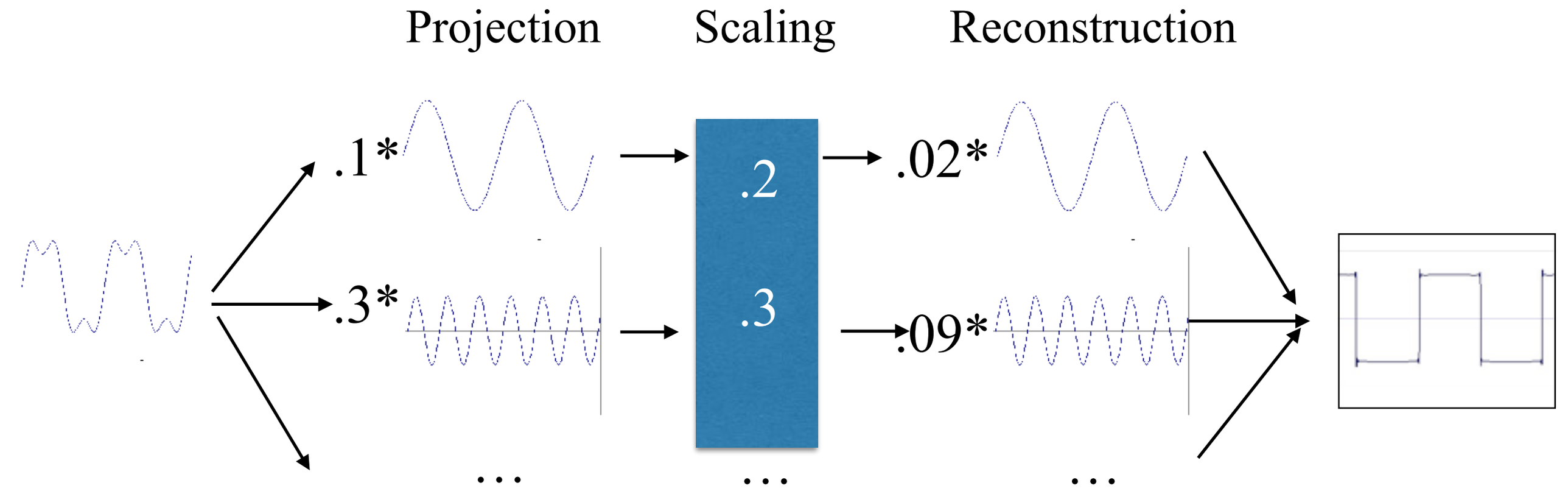
1. Efficient representation

Punchline: better compression (only need to encode low-frequency sinusoids - basis of jpeg)

2. Sinusoids are *eigenfunctions* of LSI systems

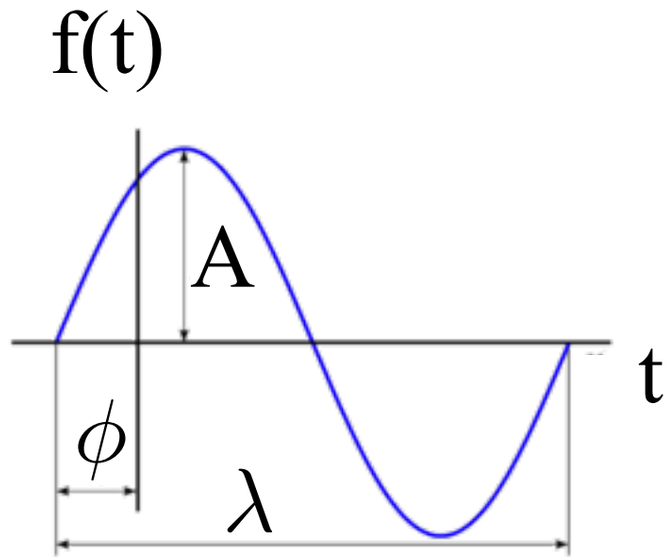
Punchline: LSI systems (or convolutions) can be implemented by …

Implement LSI operations by multiplication



"Replace convolutions with multiplications"

Background: sinusoids



$$f(t) = A\sin(2\pi ft + \phi) = A\sin(\omega x + \phi)$$

A: amplitude

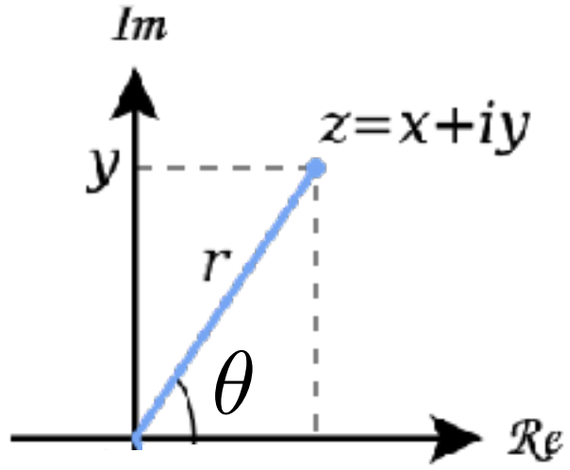
ϕ : phase

f: frequency (cycles in t=1 sec)

ω :

λ :

Background: complex numbers



(often “j” instead of “i”)

Alternative parameterizations: $z = x + iy = re^{i\theta}$

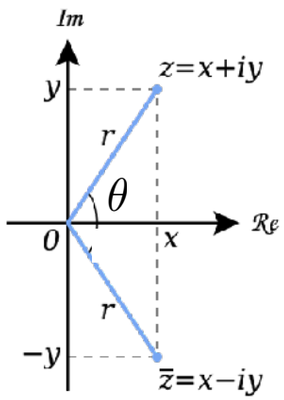
$$e^{i\theta} = \cos\theta + i\sin\theta \quad \text{(Euler's formula)}$$

$$Re(z) =$$

$$Im(z) =$$

$$z^* =$$

Background: complex numbers



$$z = x + jy = re^{j\theta}$$

$$z* = x - jy = re^{-j\theta}$$

$$Re(z) = x = r\cos(\theta)$$

$$Im(z) = y = r\sin(\theta)$$

What's the product of two complex numbers?

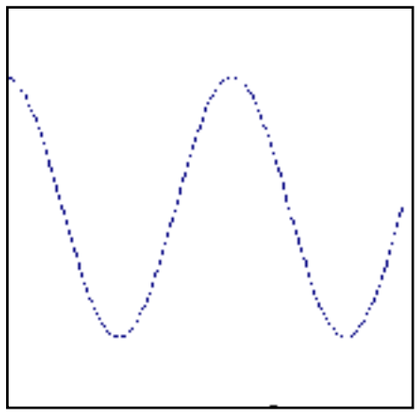
$$r_1 e^{j\theta_1} r_2 e^{j\theta_2} = r_1 r_2 e^{j(\theta_1+\theta_2)}$$

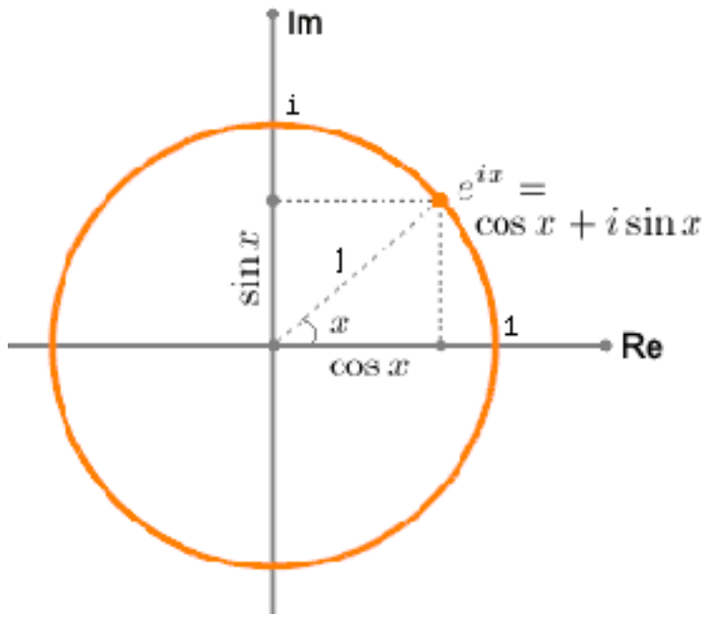
What's the inverse of a complex number?

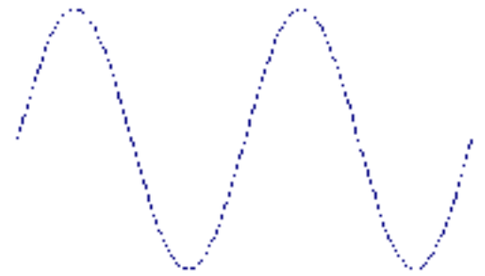
$$Inv(z) = \frac{1}{r} e^{-j\theta} = z^* \quad \text{for} \quad r = 1$$

Background: complex sinusoid

$$f(t) = e^{j\omega t}$$







$$Re\{f(t)\} = \cos(\omega t)$$

$$Im\{f(t)\} = \sin(\omega t)$$

Think of point travelling around unit circle on imaginary plane at $\frac{\omega}{2\pi}$ cycles per sec

Proof: complex sinusoids are eigenfunctions of LSI systems

$$g[u] = \sum_v h[v] f[u-v], \quad f[u] = e^{j\omega u}$$

$$=$$

$$=$$

Proof: complex sinusoids are eigenfunctions of LSI systems

$$g[u] = \sum_v h[v] f[u-v], \quad f[u] = e^{j\omega u}$$

$$= e^{j\omega u} \sum_v h[v] e^{-j\omega v}$$

$$= e^{j\omega u} H(w) \quad \text{where} \quad H(w) = \sum_v h[v] e^{-j\omega v}$$

$$= f[u] H(\omega)$$

If we hit a LSI system with a sinuoid, output is that sinusoid scaled by H(w) (a function of system's impulse response)

Proof: complex sinusoids are eigenfunctions of LSI systems

$$g[u] = \sum_v h[v] f[u-v], \quad f[u] = e^{j\omega u}$$

$$= e^{j\omega u} \sum_v h[v] e^{-j\omega v}$$

$$= e^{j\omega u} H(w) \quad \text{where} \quad H(w) = \sum_v h[v] e^{-j\omega v}$$

$$= f[u] H(\omega)$$

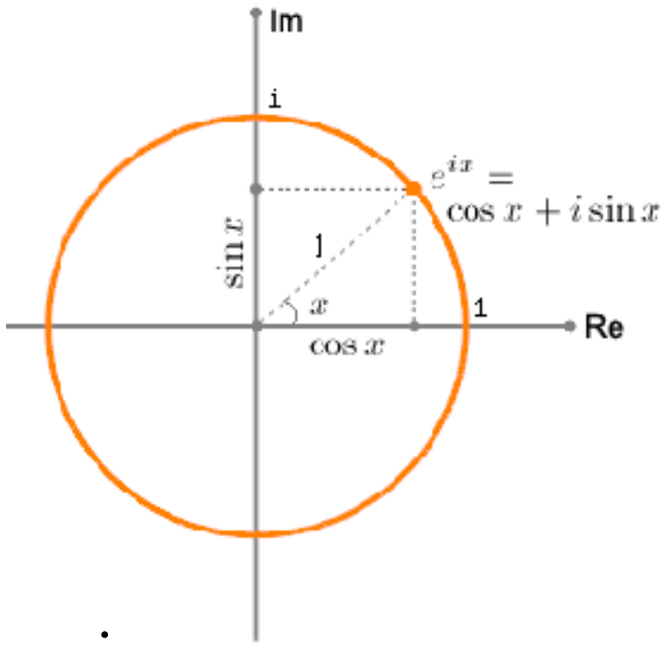
Aside: Could this have worked for $f[u] = z^u$?

The resulting H[z] is known as a z-transform: https://en.wikipedia.org/wiki/Z-transform

Discrete Fourier Transform (DFT)

Evaluate below transformation at N different frequencies

$$F(\omega_k) = \sum_u f[u] e^{-j\omega_k u} \quad \omega_k = \frac{2\pi k}{N}, \quad k = \{0, \ldots, N-1\}$$



k=0: point not moving
k=1: travels around unit circle once every N steps
k=2: travels around unit circle twice every N steps
….

Discrete Fourier Transform (DFT)

$$F[k] = \frac{1}{N} \sum_{u=0}^{N-1} f[u] e^{\frac{-j2\pi ku}{N}}$$

$$Re(F[k]) = \frac{1}{N} \sum_{u=0}^{N-1} f[u] \cos(\frac{-2\pi ku}{N})$$

f[u] ⟶ F[k]

N-long sequence N-long sequence

Discrete Fourier Transform (DFT)

$$F[k=0] = \frac{1}{N}\sum_{u=0}^{N} f[u]$$

$$F[k=1] = \frac{1}{N}\sum_{u=0}^{N} f[u]e^{\frac{-j2\pi u}{N}}$$

$$\dots$$

$$F[k=N/2] = \frac{1}{N}\sum_{u=0}^{N} f[u]e^{-j\pi u}$$

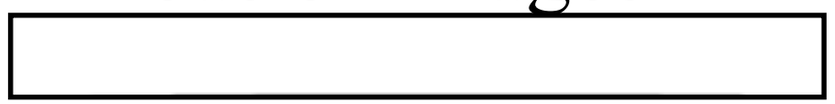
Comput F[k] by taking inner product of f[u] with a complex sinusoid $g_k[u]$

What does the (real part of) this sinuoidal functions look like?

Discrete Fourier Transform (DFT)

$$F[k=0] = \frac{1}{N}\sum_{u=0}^{N} f[u]$$

basis “images”



$$F[k=1] = \frac{1}{N}\sum_{u=0}^{N} f[u]e^{\frac{-j2\pi u}{N}}$$

$$\ldots$$

$$F[k=N/2] = \frac{1}{N}\sum_{u=0}^{N} f[u]e^{-j\pi u}$$

Discrete Fourier Transform (DFT)

$$F[k=0] = \frac{1}{N}\sum_{u=0}^{N} f[u]$$

$$F[k=1] = \frac{1}{N}\sum_{u=0}^{N} f[u]e^{\frac{-j2\pi u}{N}}$$

$$\dots$$

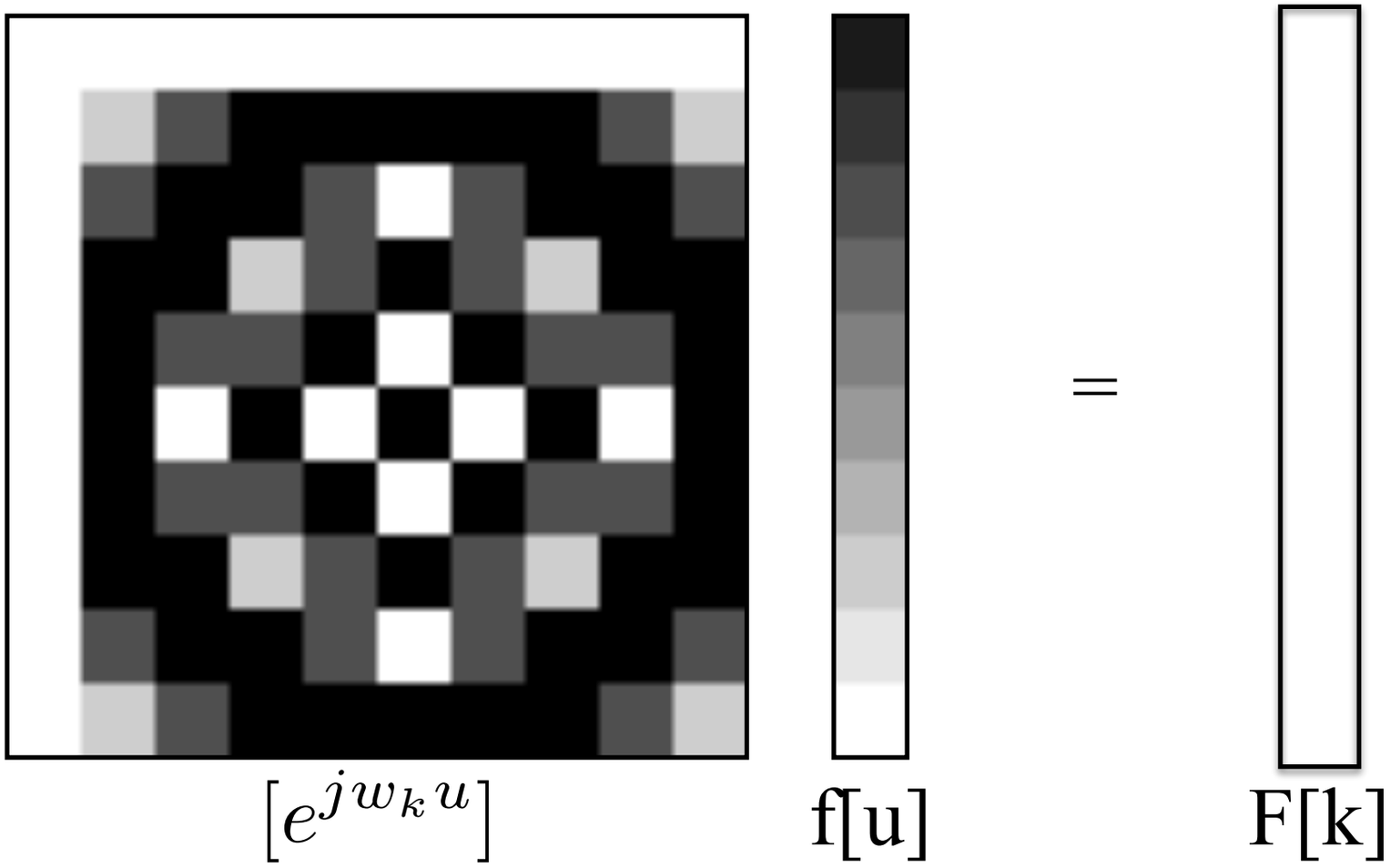
$$F[k=N/2] = \frac{1}{N}\sum_{u=0}^{N} f[u]e^{-j\pi u}$$

$$\dots$$

$$F[k=N] = ?$$

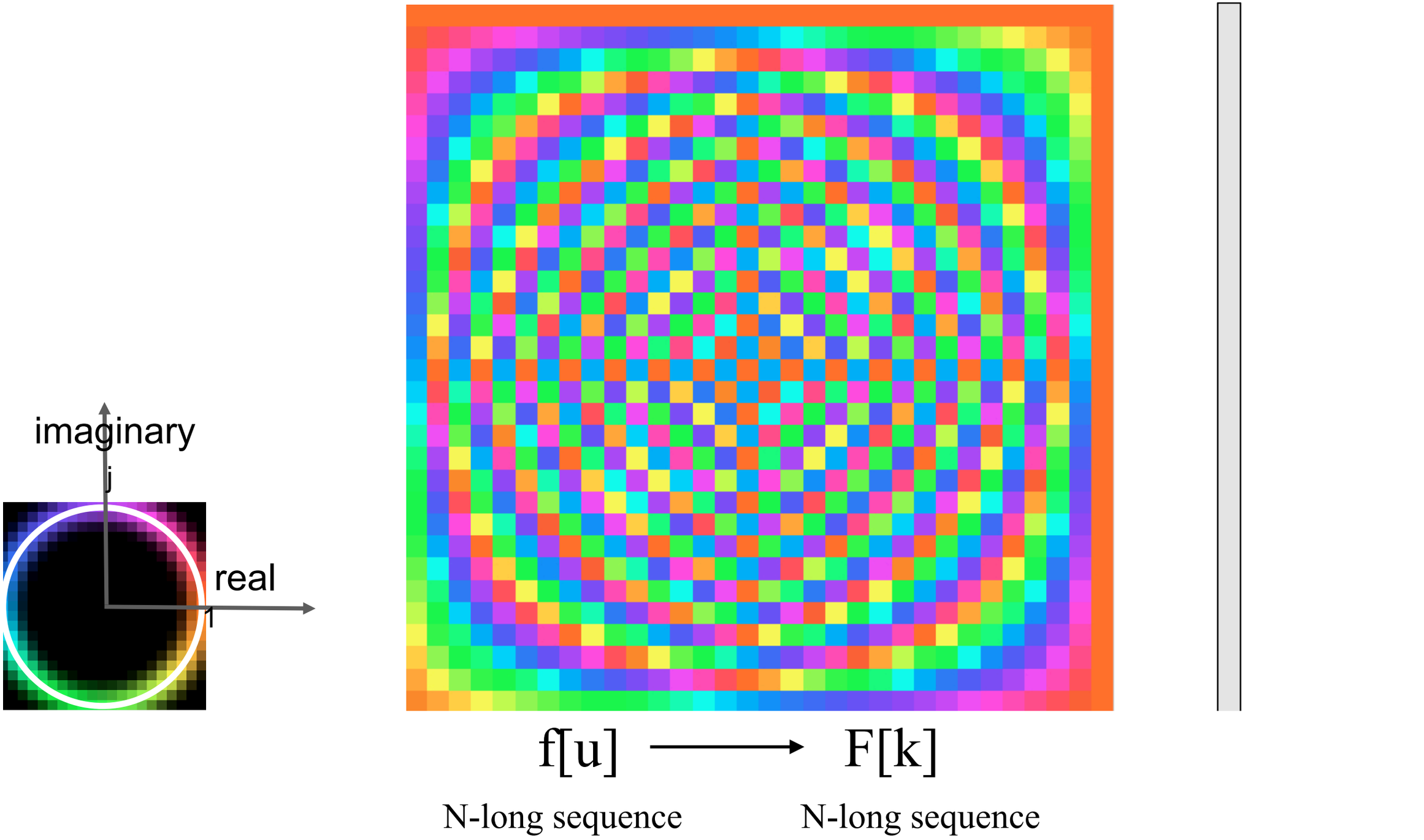
$$F[k] = \frac{1}{N}\sum_{u=0}^{N-1} f[u]e^{\frac{-j2\pi ku}{N}}$$

Linear transformation



How expensive is the conversion?

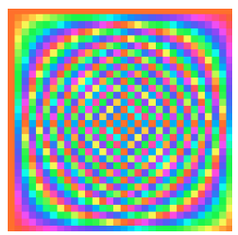
Complex matrix multiplication



Complex orthonormality

$$U^* = [U_{ij}^*]^T$$ "complex conjugate or adjoint"

$$(U^*)U = I$$

()  = I

Implies the inverse transform is obtained by transpose + flipping imaginary part

$$F = U f$$

$$(U^*)F = f$$

Inverse DFT

$$f[u] = \sum_{k=0}^{N-1} F[k] e^{\frac{j2\pi ku}{N}} \qquad \text{(IDFT)}$$

$$F[k] = \frac{1}{N} \sum_{u=0}^{N-1} f[u] e^{\frac{-j2\pi ku}{N}} \qquad \text{(DFT)}$$

F[k] ⟶ f[u]

N-long sequence — N-long sequence

Sometimes the 1/N is moved or factored
(really need sqrt(N) to make transform self-inverting)

General fourier terminology

https://en.wikipedia.org/wiki/Fourier_transform

$$a_n \cos(nx) + b_n \sin(nx)$$

Decompose any continuous periodic signal into a summation of sinusoids of increasing frequency

General fourier terminology

- Continuous fourier transfrom

https://en.wikipedia.org/wiki/Fourier_transform

- Discrete-time fourier transform

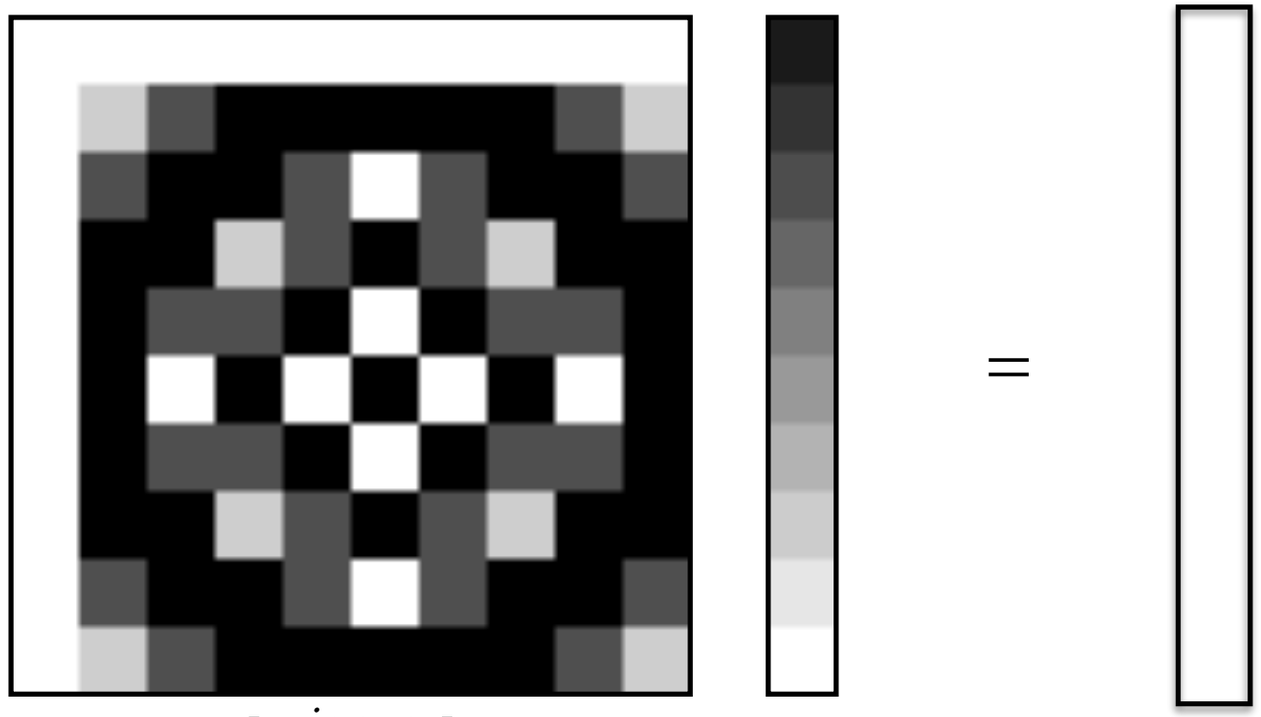
https://en.wikipedia.org/wiki/Discrete-time_Fourier_transform

(useful for analyzing samples of a continuous fn)

- Discrete fourier transform (DFT)

https://en.wikipedia.org/wiki/Discrete_Fourier_transform

DFTs as a change of basis



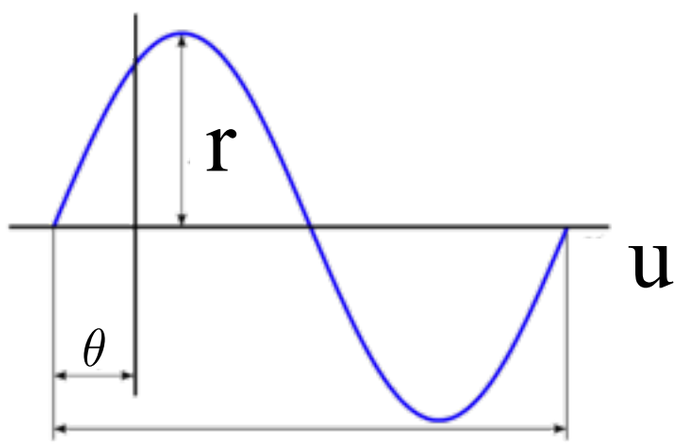
$[e^{jw_k u}]$ f[u] F[k]

How do we interpret complex weighting coefficients?

$$F[k] = a + bj = re^{j\theta}$$

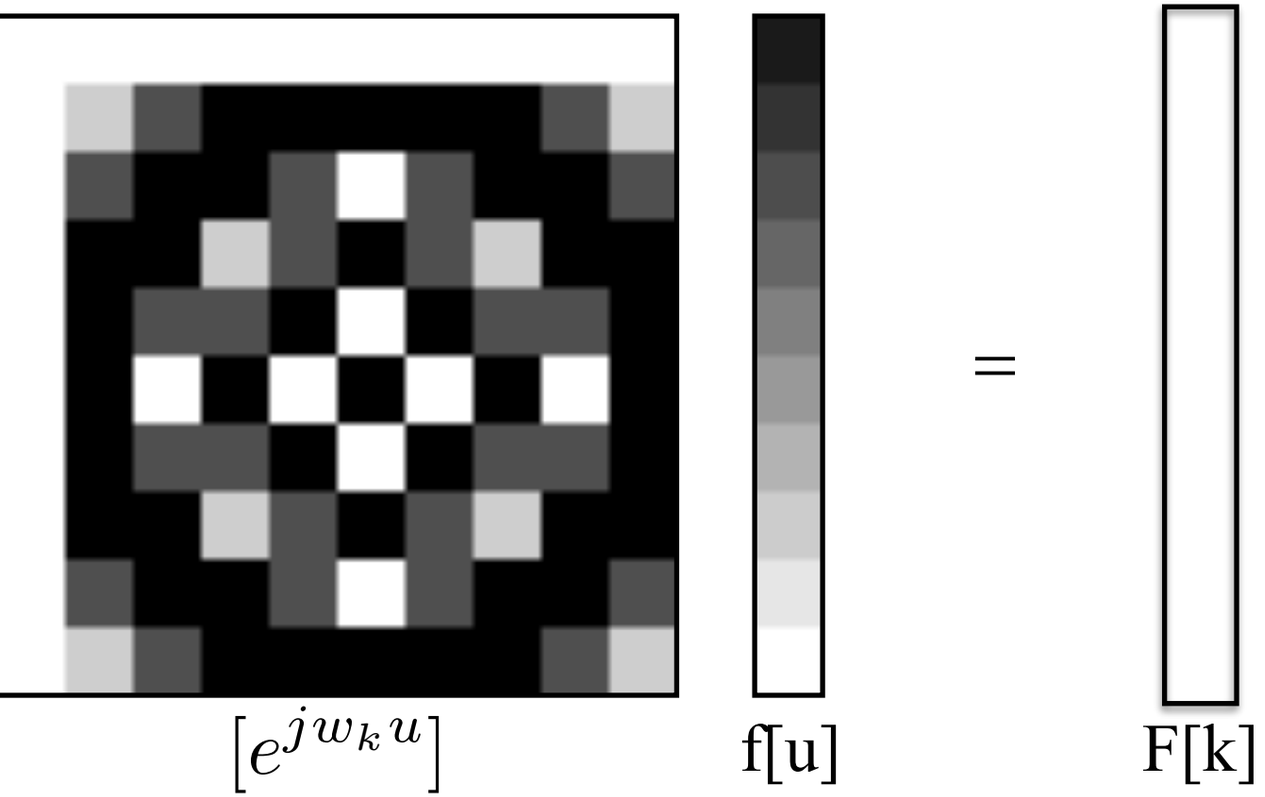
$$re^{j\theta}e^{\frac{j2\pi ku}{N}} = re^{j(\frac{2\pi ku}{N}+\theta)}$$

$$Re\{\cdot\} = r\cos(\frac{2\pi ku}{N} + \theta)$$



Scaling coefficients allow us to shift sinusoids!

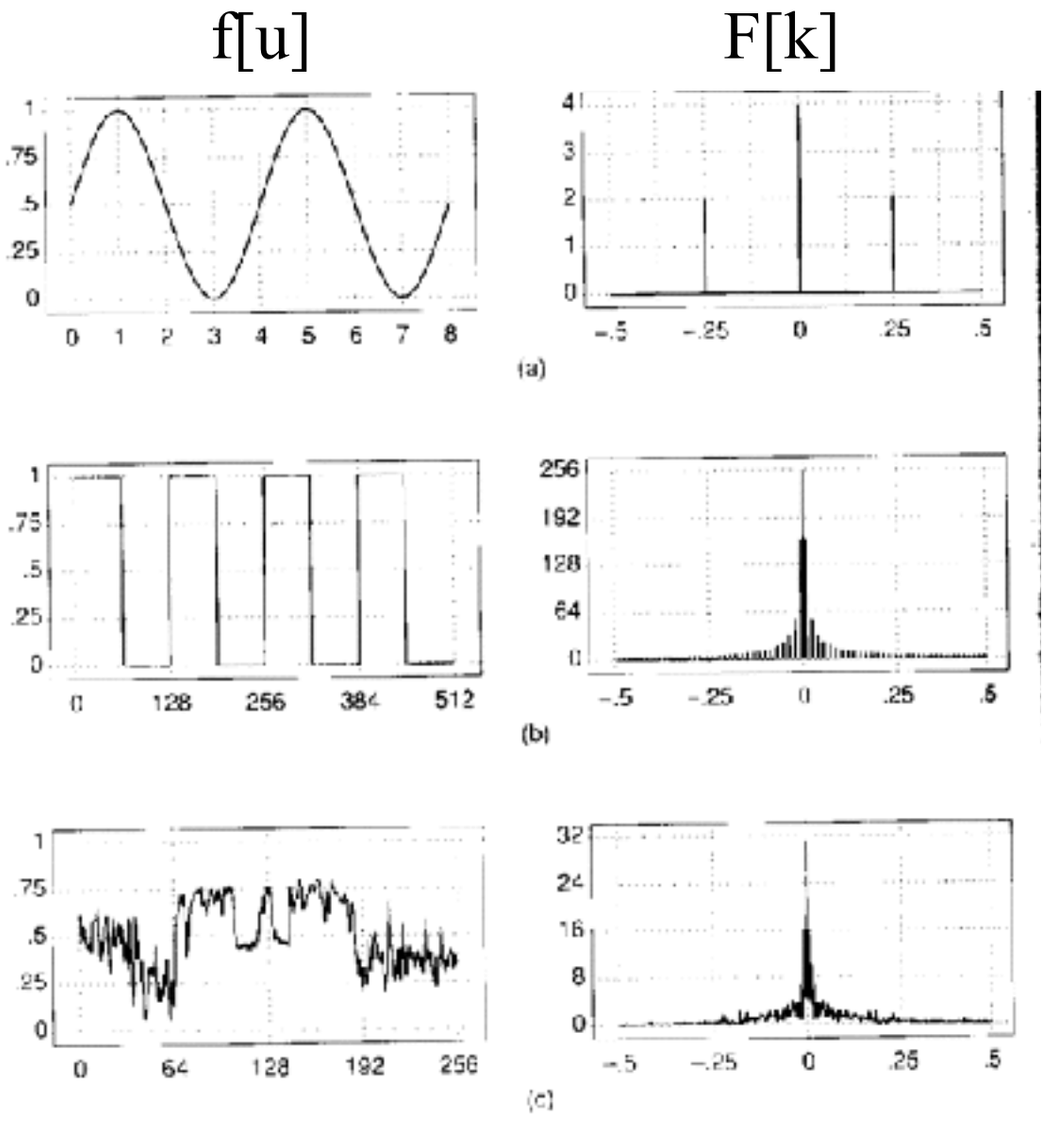
Visualizing DFTs



$[e^{jw_k u}]$

f[u]

F[k]



Visualize frequencies from [-N/2..N/2] instead of [0..N]

(Equivalent info because FTs are periodic)

Proof: DFT of real signal

$$F[k] = \frac{1}{N} \sum_{u=0}^{N-1} f[u] e^{\frac{-j2\pi ku}{N}}$$

$$F^*[-k] =$$

$$=$$

Properties of DFT

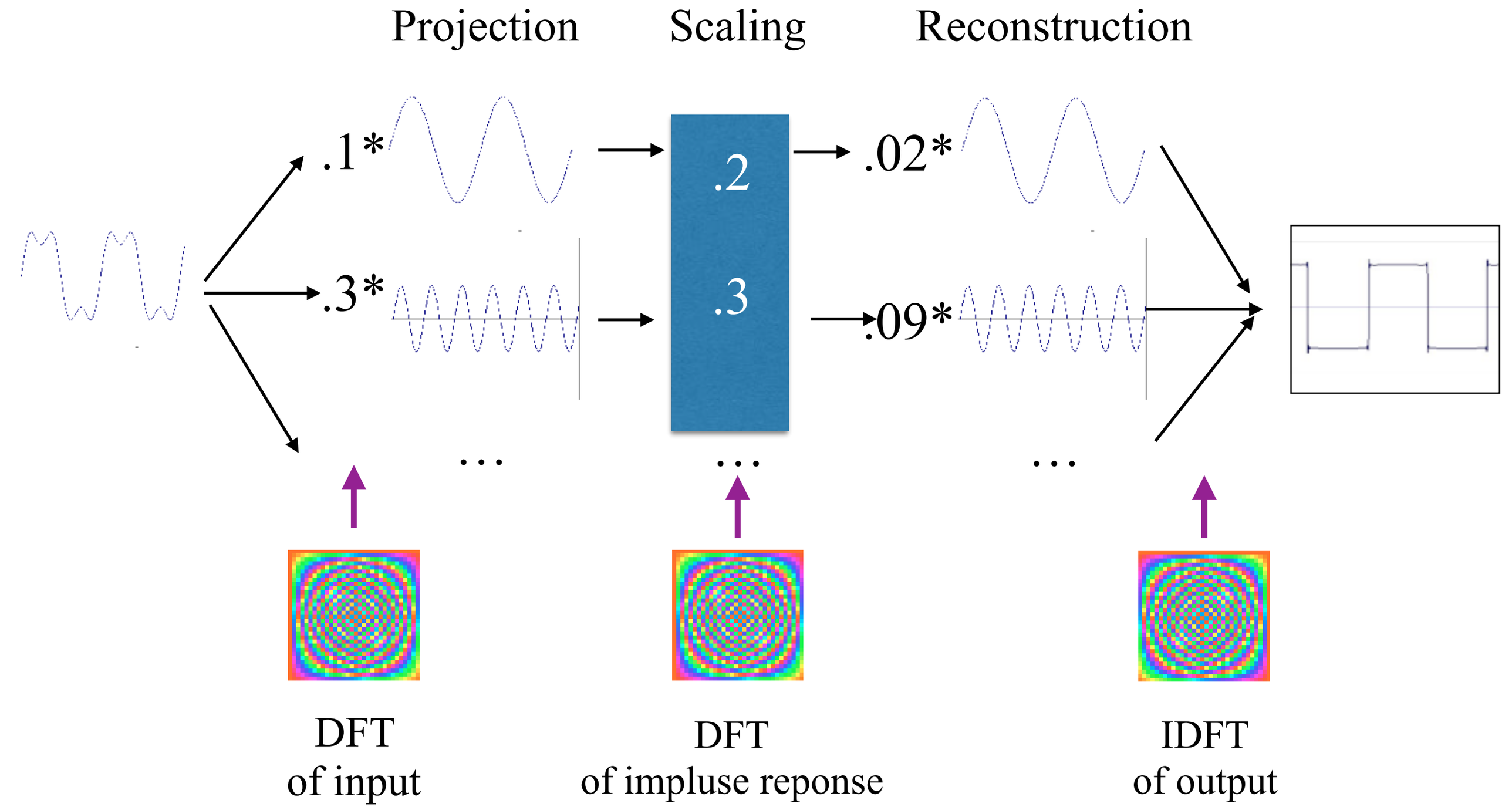
- Symmetric for real-valued f[u]: F[k] = F[-k]* (we only really need N/2 values)

- Periodic: F[k+N] = F[k]. (we only need to encode N values; often use k = [-N/2…N/2])

- Eigenfunction property: the DFT of the convolution of two functions is the product of their DFTs (convolution in time domain is multiplication in frequency domain)

- DFT of a Gaussian with variance sigma is a another Gaussian with variance 1/sigma

Crucial property: convolution is multiplication in frequency space

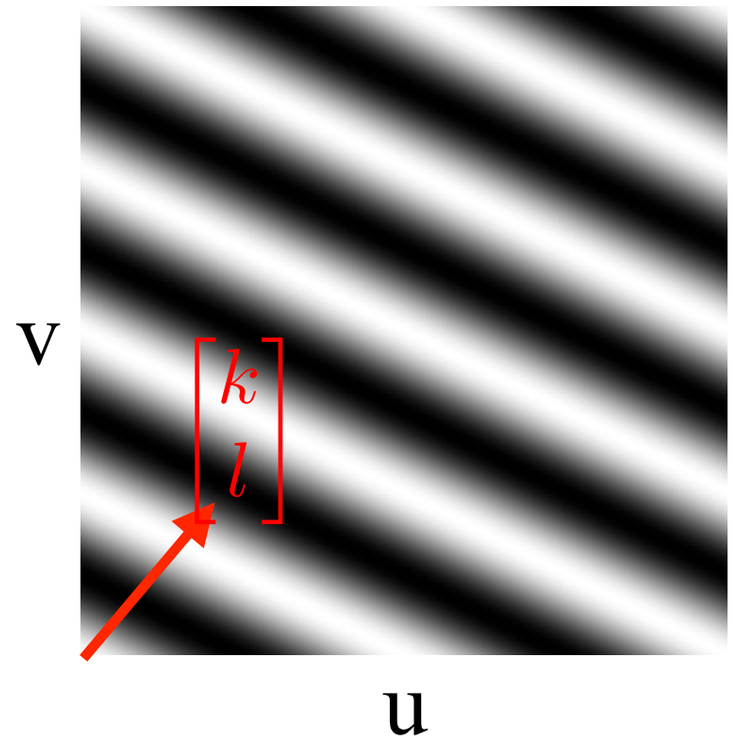


Extensions to 2D

$$F[k,l] = \frac{1}{N^2}\sum_{u}^{N}\sum_{v}^{N} f[u,v]e^{-2j\pi(\frac{ku+lv}{N})}$$

$$g[u,v] = \cos(-2\pi\frac{ku+lv}{N})$$

$$\begin{bmatrix} k \\ l \end{bmatrix}^T \begin{bmatrix} u \\ v \end{bmatrix} = ||a||||b||\cos\theta$$



magnitude of [k,l] gives the frequency

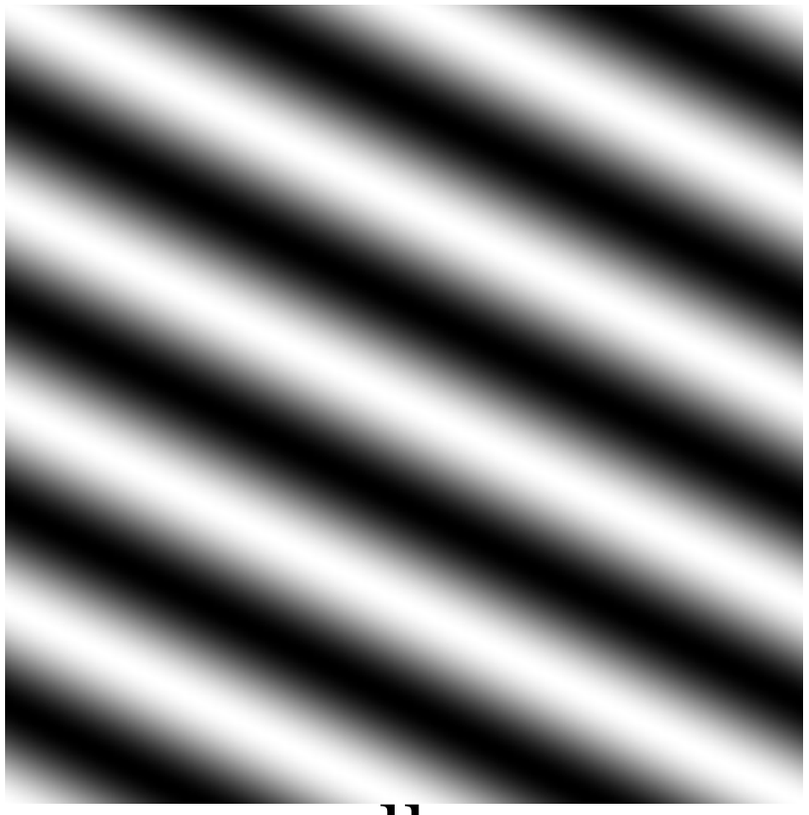
Visualizing DFTs in 2D

Arrange coefficients into an *image*

F[k,l]

$$e^{-2j\pi(\frac{ku+lv}{N})}$$

v



u

Real part of basis image

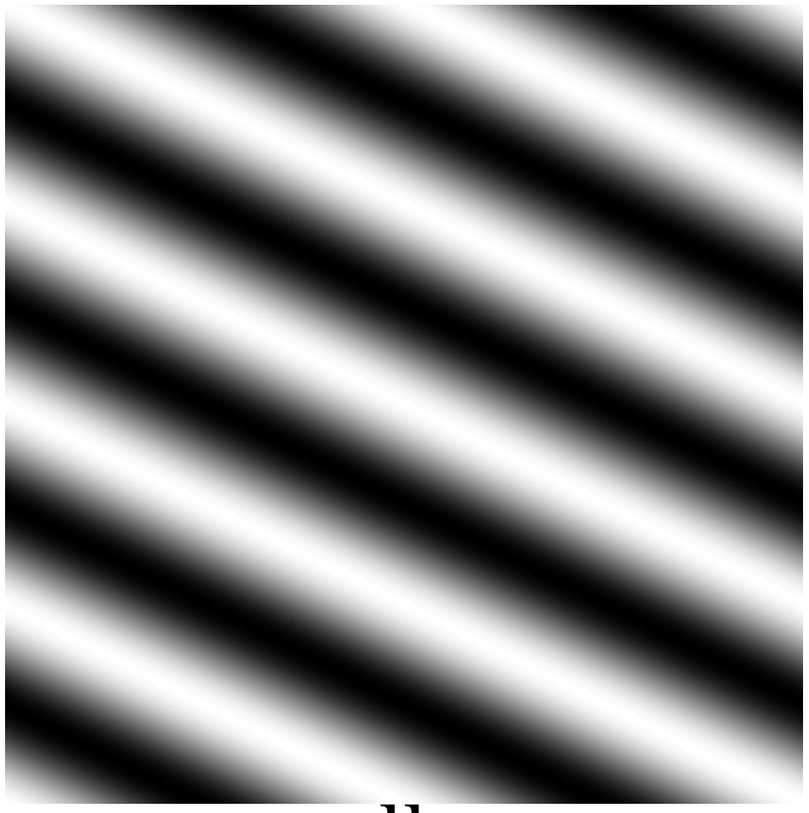
Visualizing DFTs in 2D

What does basis image look like for [-k,-l]?

F[k,l]

$$\begin{matrix} & e^{-2j\pi(\frac{ku+lv}{N})} \\ & \vdots \\ e^{2j\pi(\frac{ku+lv}{N})} & \end{matrix}$$

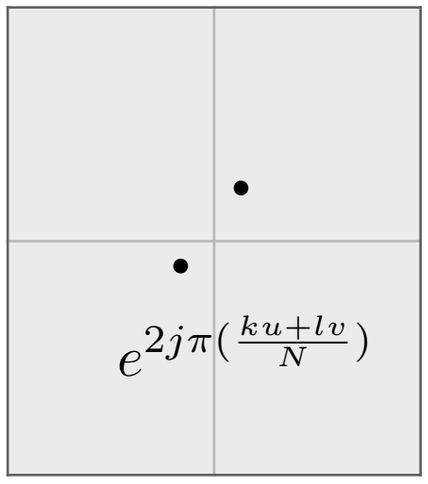
v



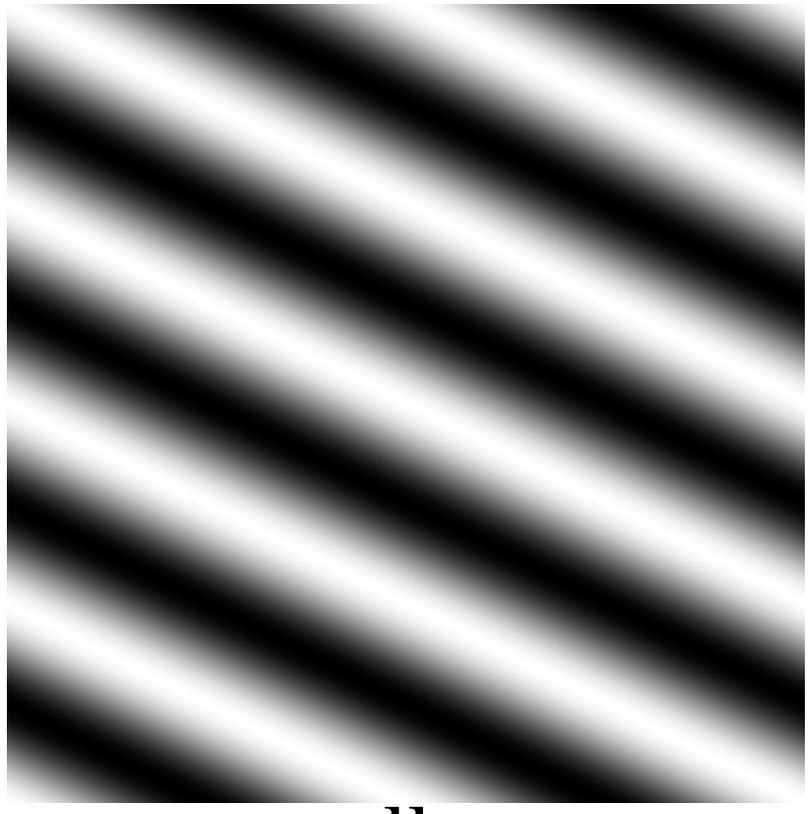
u

Real part of basis image

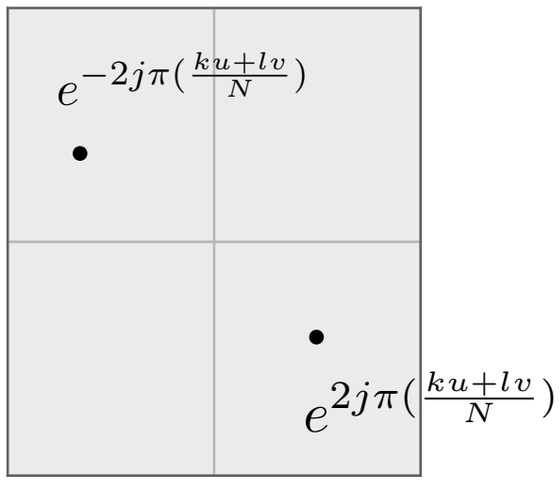
F[k,l]



v



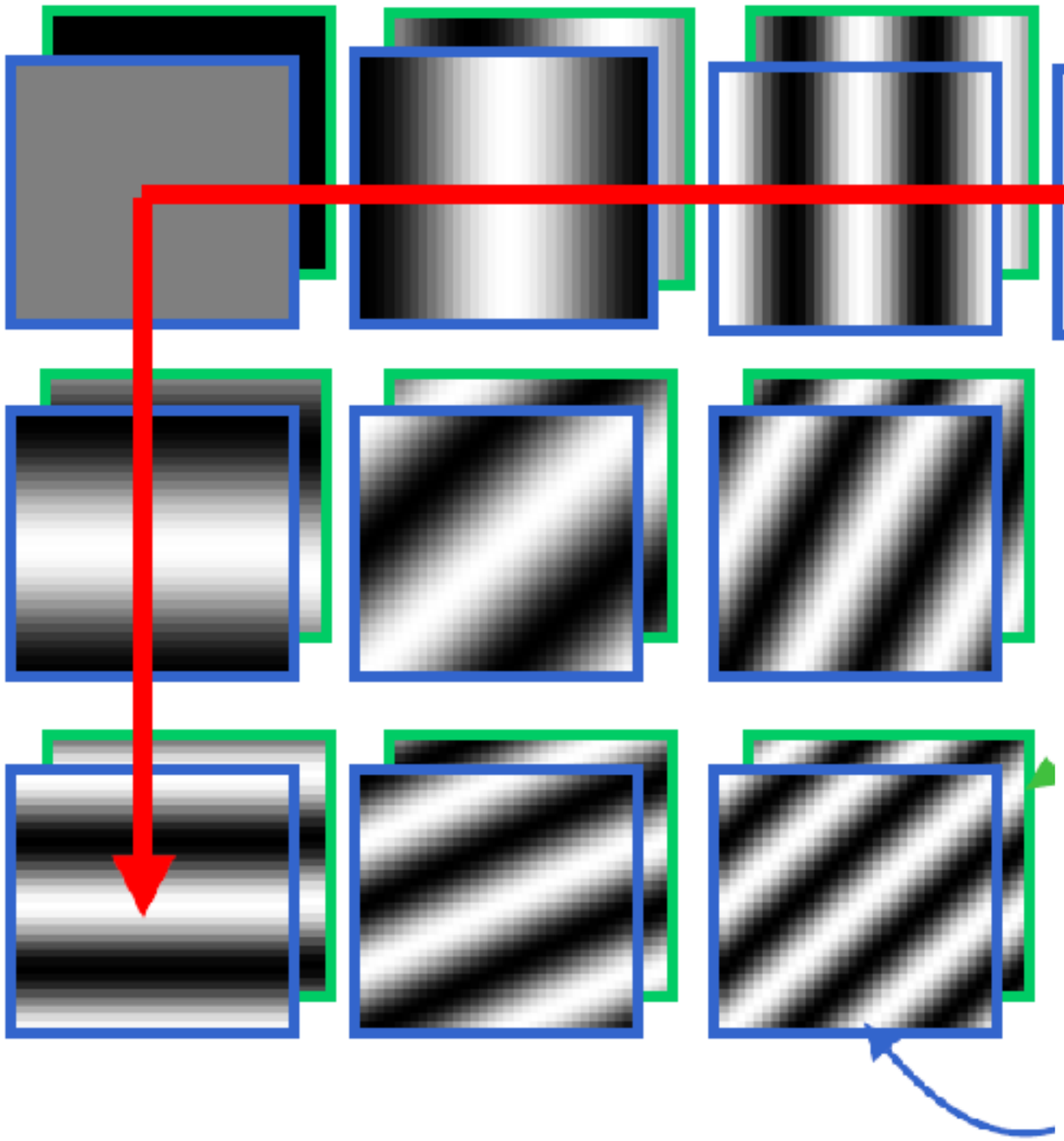
u



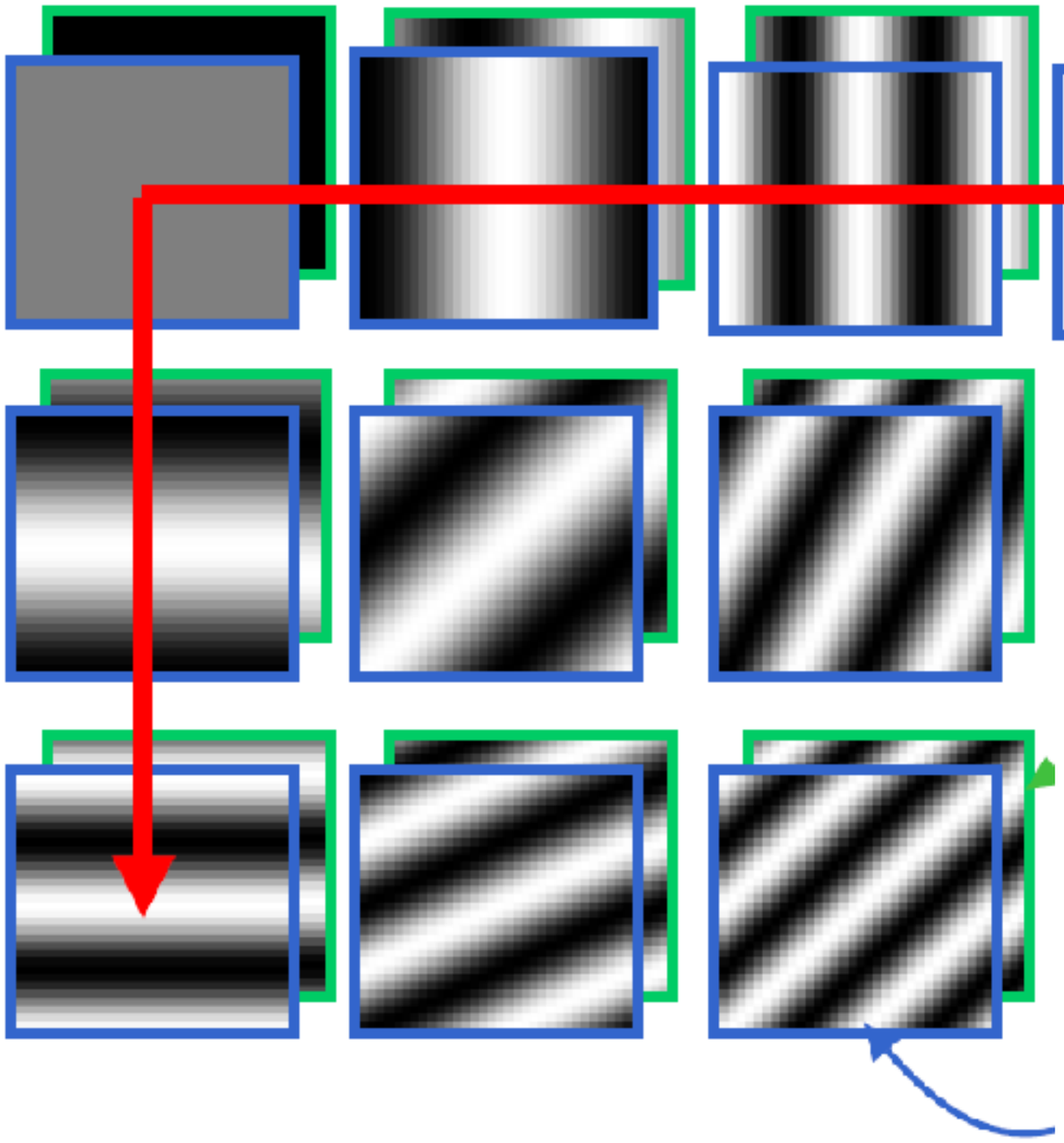
v

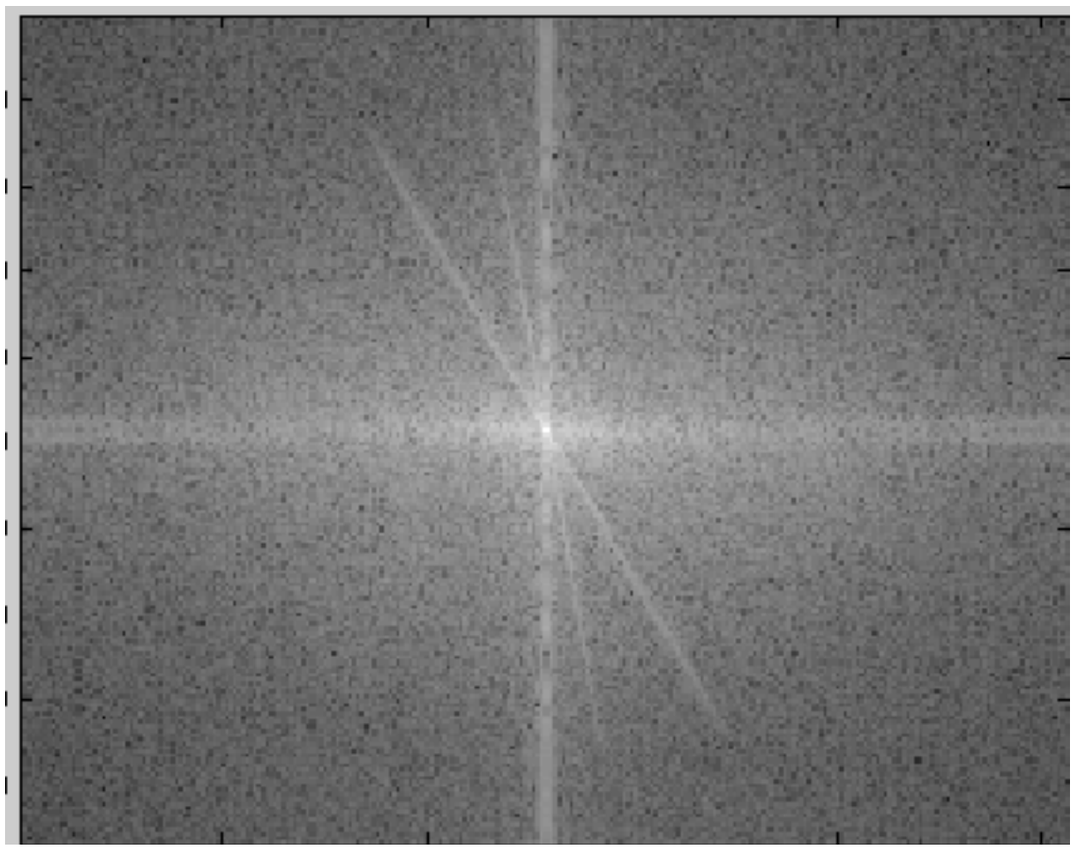
u

Extension to 2D



Extension to 2D

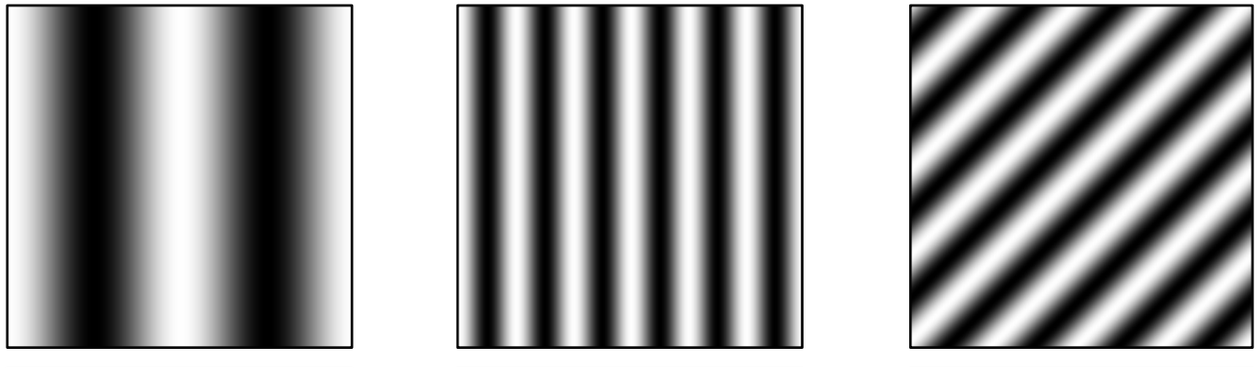




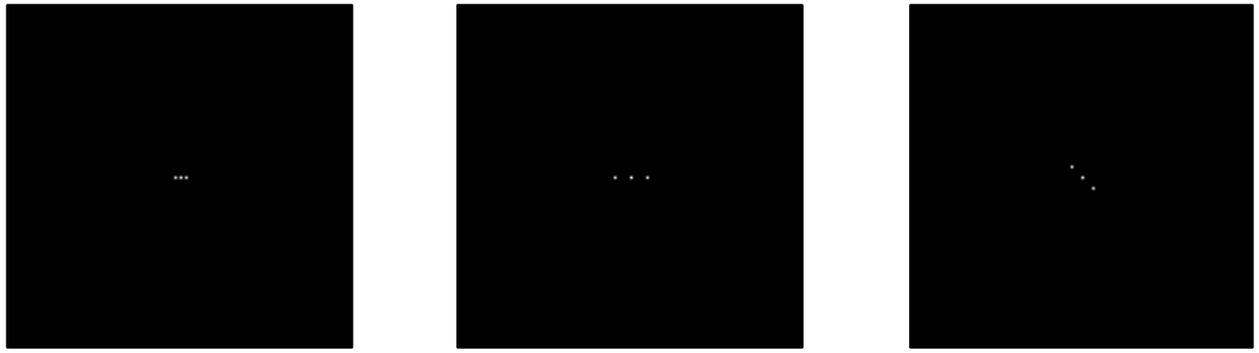
in Matlab, check out: imagesc(log(abs(fftshift(fft2(im)))));

Fourier analysis in images

Intensity Image

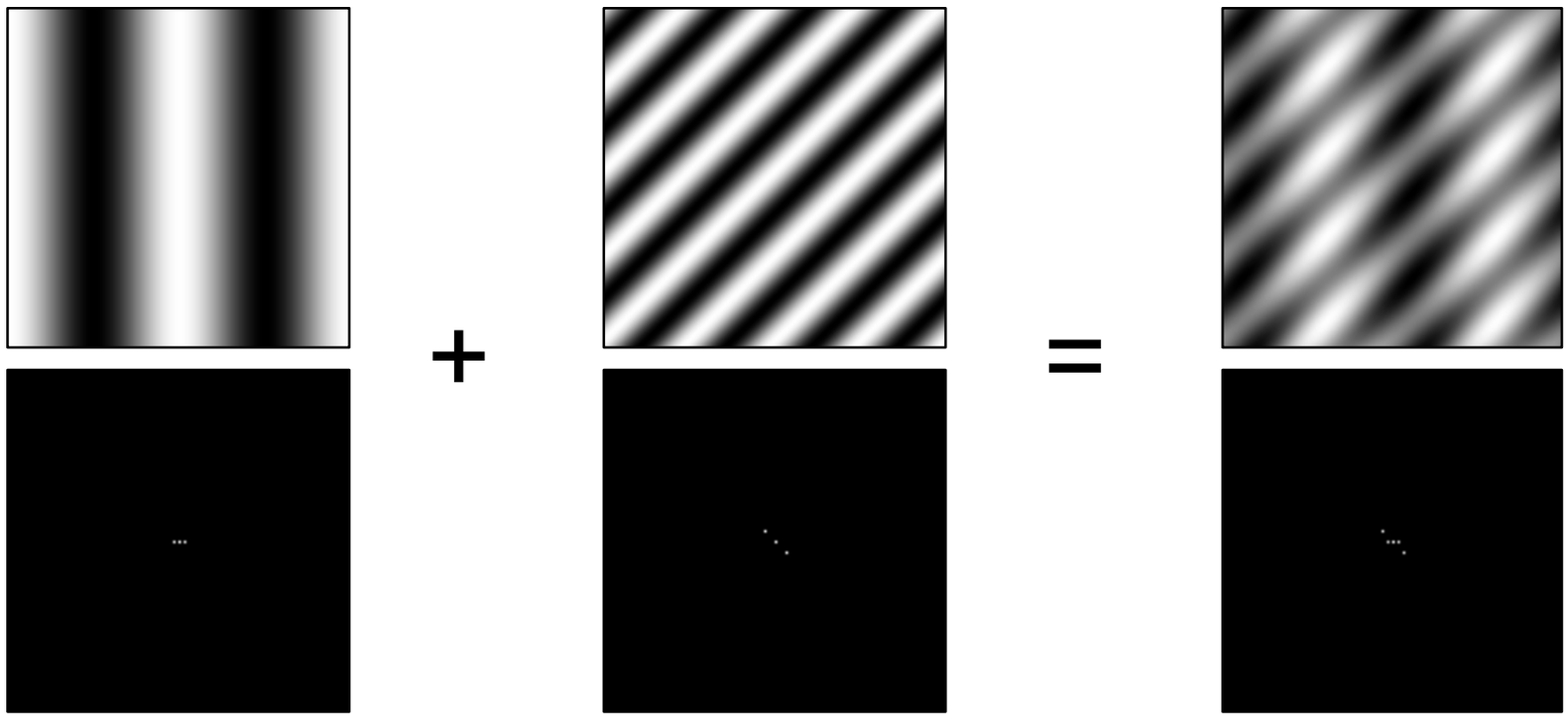


Fourier Image



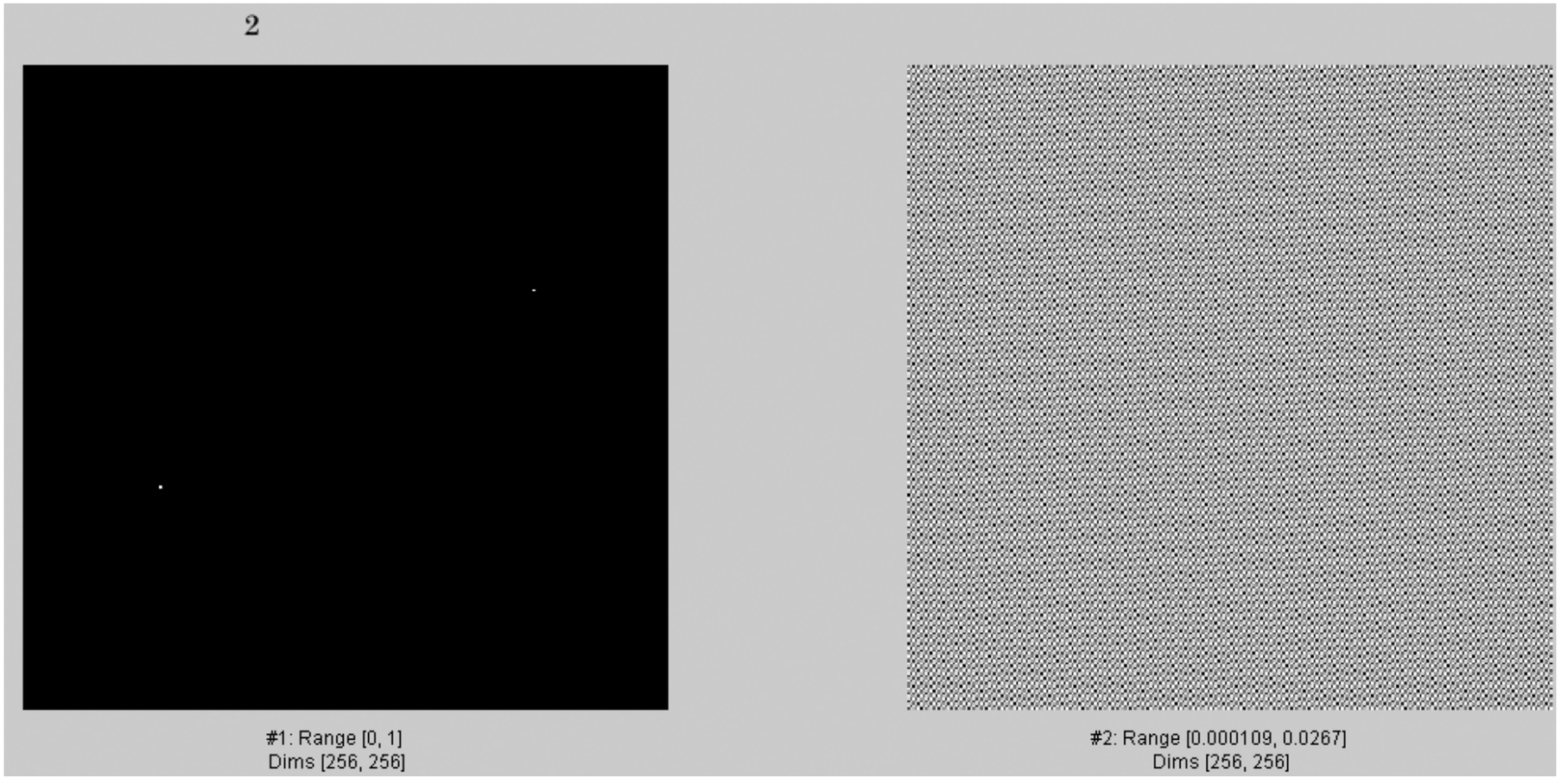
http://sharp.bu.edu/~slehar/fourier/fourier.html#filtering

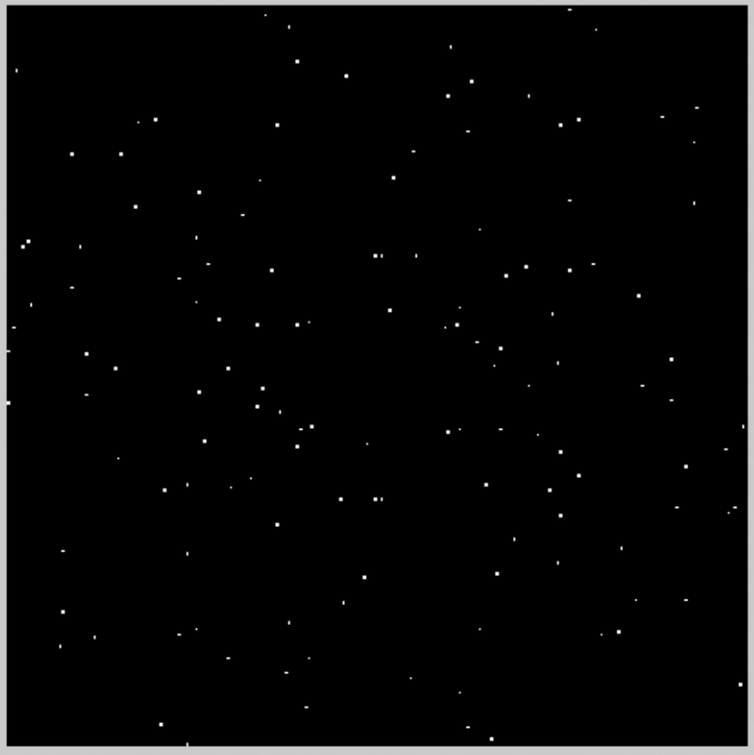
Signals can be composed



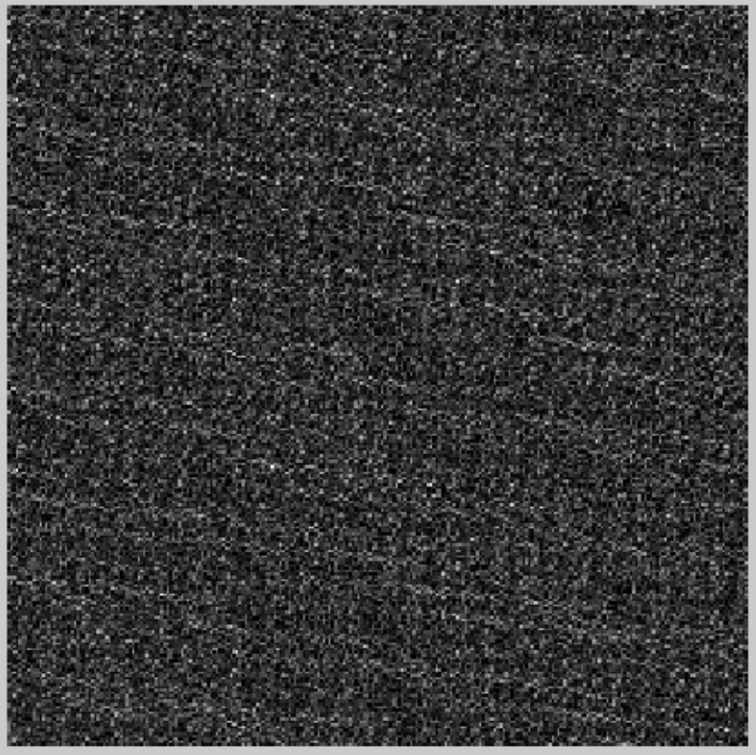
http://sharp.bu.edu/~slehar/fourier/fourier.html#filtering
More: http://www.cs.unm.edu/~brayer/vision/fourier.html

Let’s build up a signal by randomly adding in fourier transform coefficients

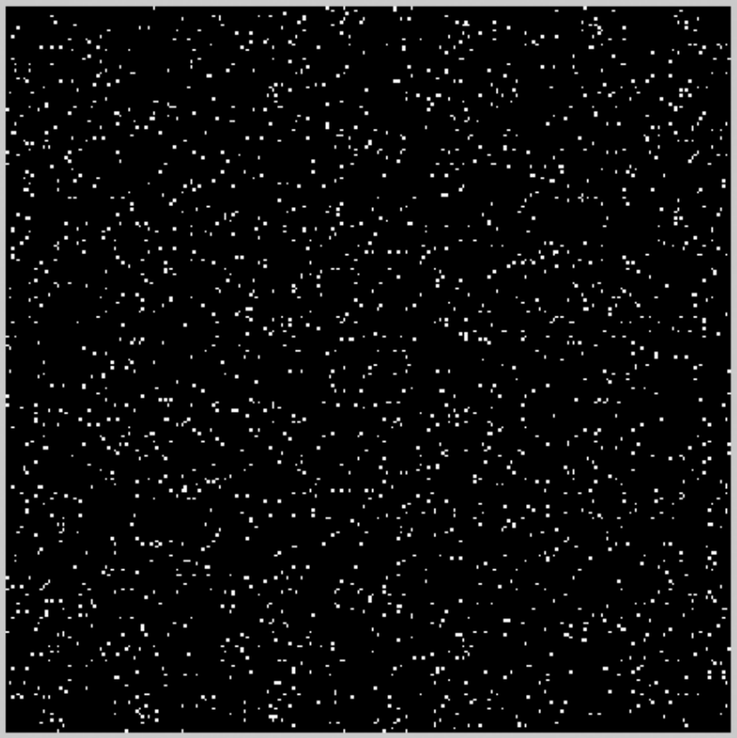




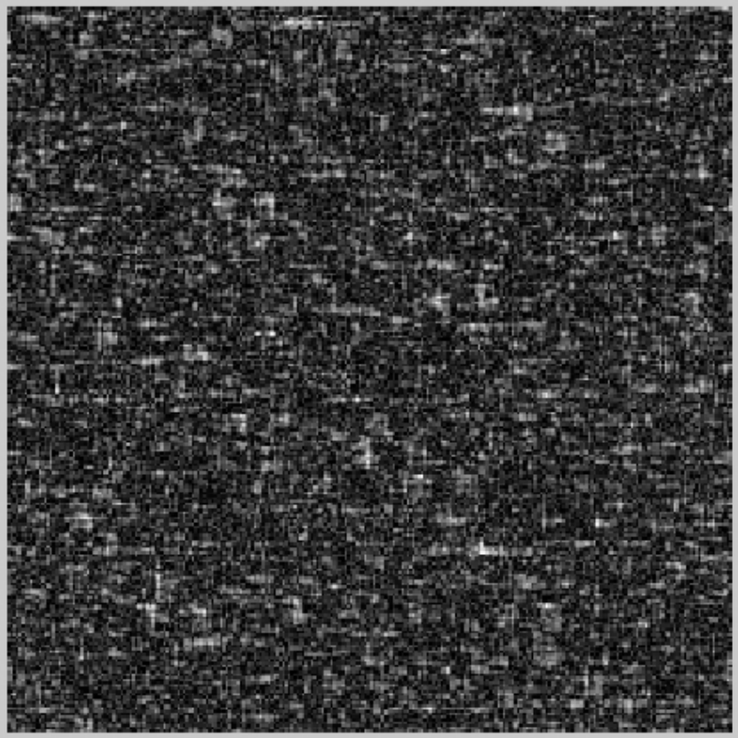
#1: Range [0, 1]
Dims [256, 256]



#2: Range [8.25e-006, 3.48]
Dims [256, 256]

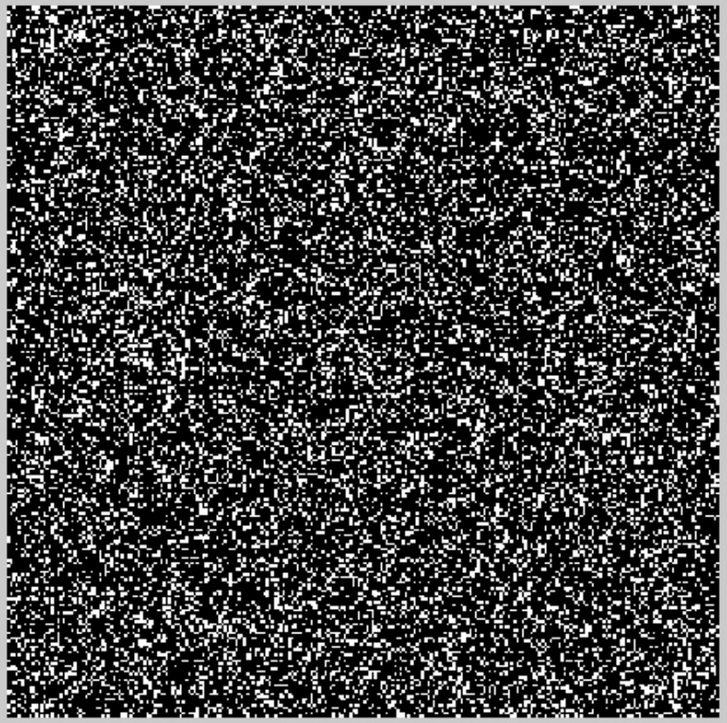


#1: Range [0, 1]
Dims [256, 256]

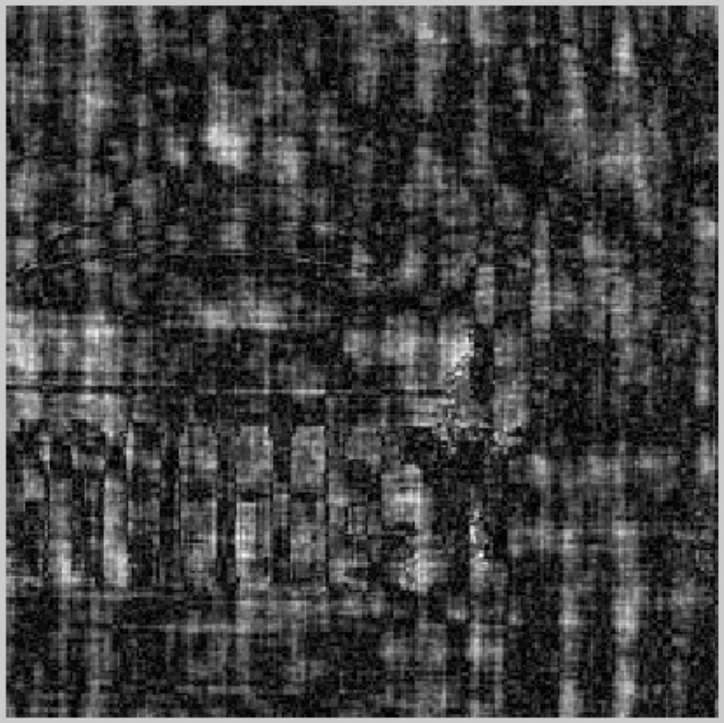


#2: Range [8.7e-005, 19]
Dims [256, 256]

15366

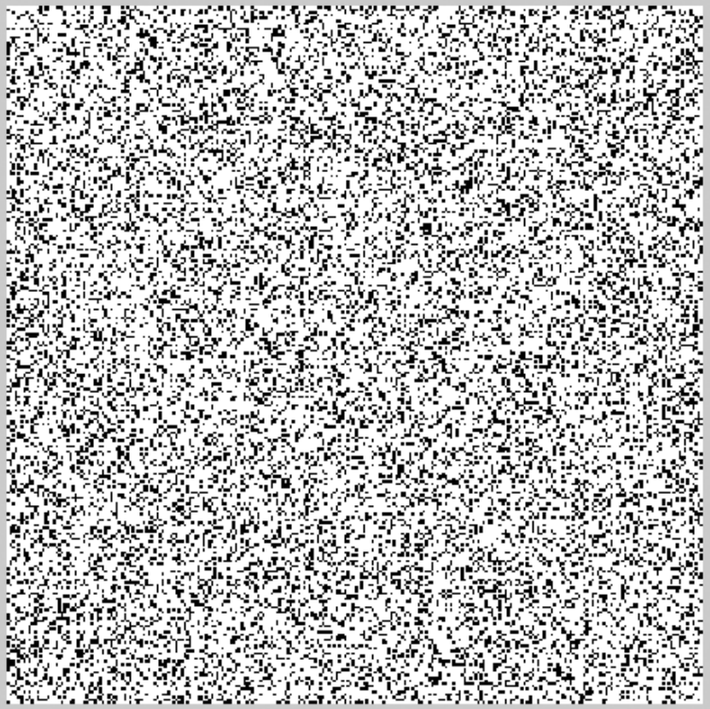


#1: Range [0, 1]
Dims [256, 256]



#2: Range [0.000231, 91.1]
Dims [256, 256]

49190



#1: Range [0, 1]
Dims [256, 256]

#2: Range [0.00758, 294]
Dims [256, 256]

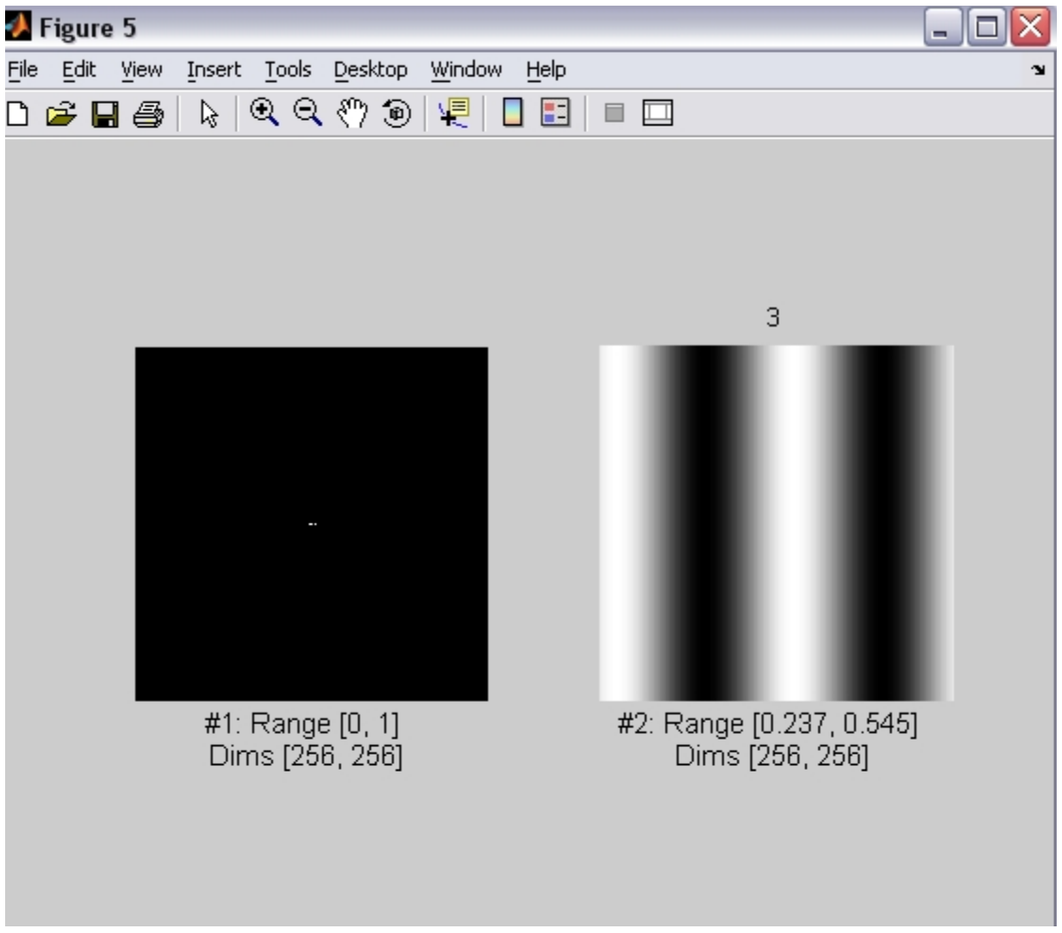


Figure 5
File Edit View Insert Tools Desktop Window Help
3
#1: Range [0, 1]
Dims [256, 256]
#2: Range [0.237, 0.545]
Dims [256, 256]

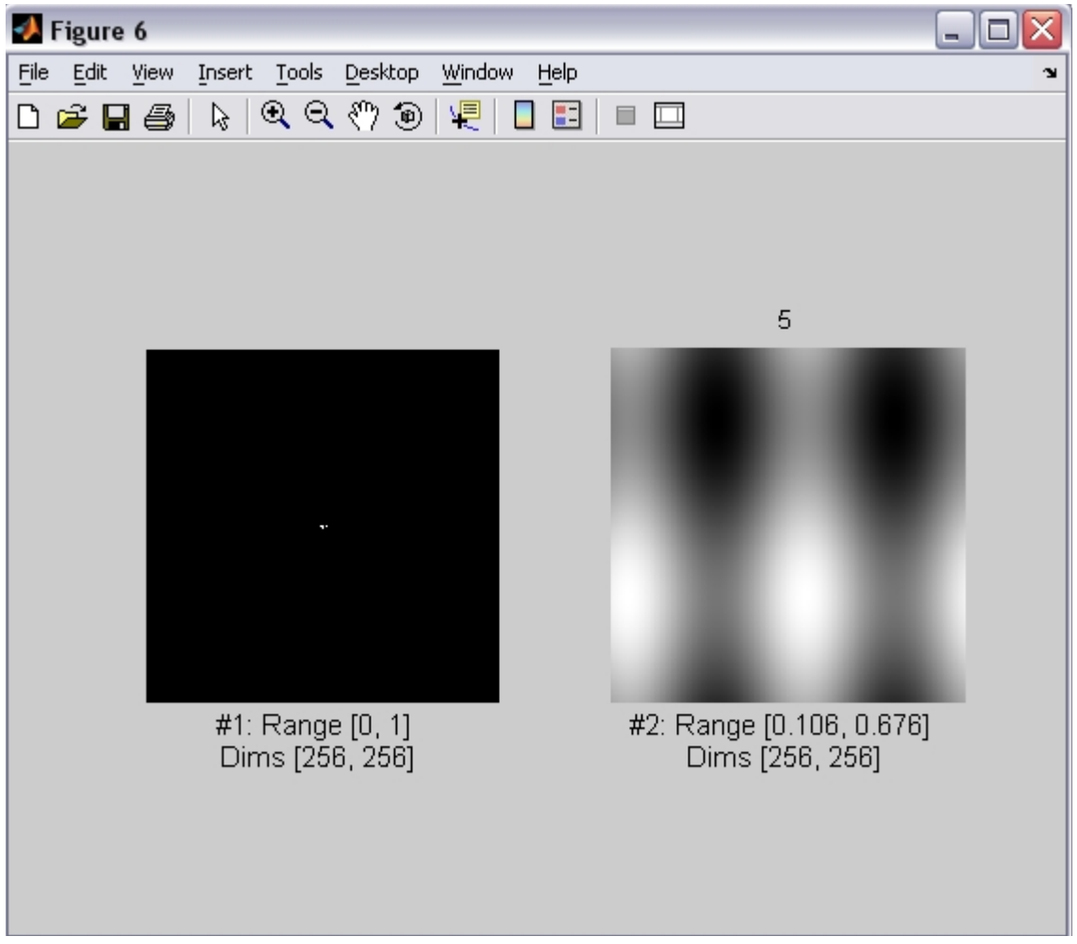


Figure 6
File Edit View Insert Tools Desktop Window Help
5
#1: Range [0, 1]
Dims [256, 256]
#2: Range [0.106, 0.676]
Dims [256, 256]

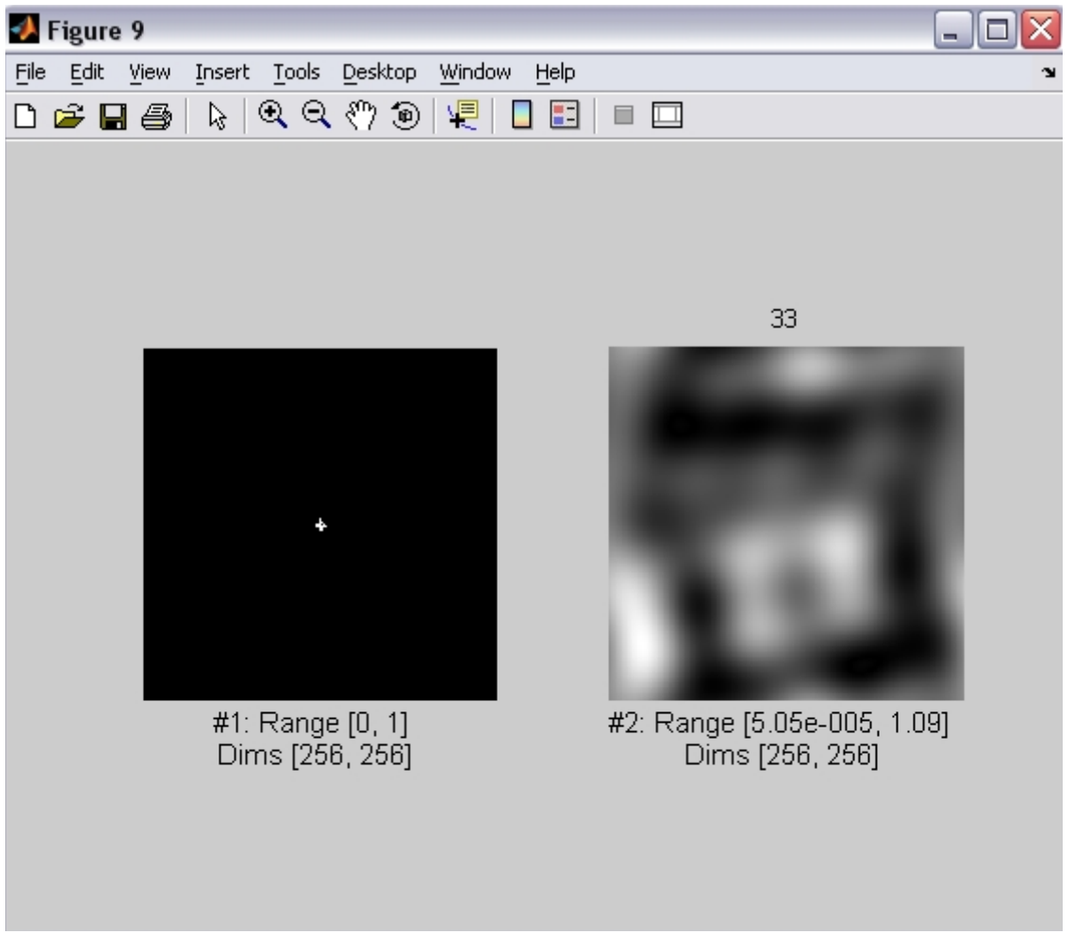


Figure 9
File Edit View Insert Tools Desktop Window Help
33
#1: Range [0, 1]
Dims [256, 256]
#2: Range [5.05e-005, 1.09]
Dims [256, 256]

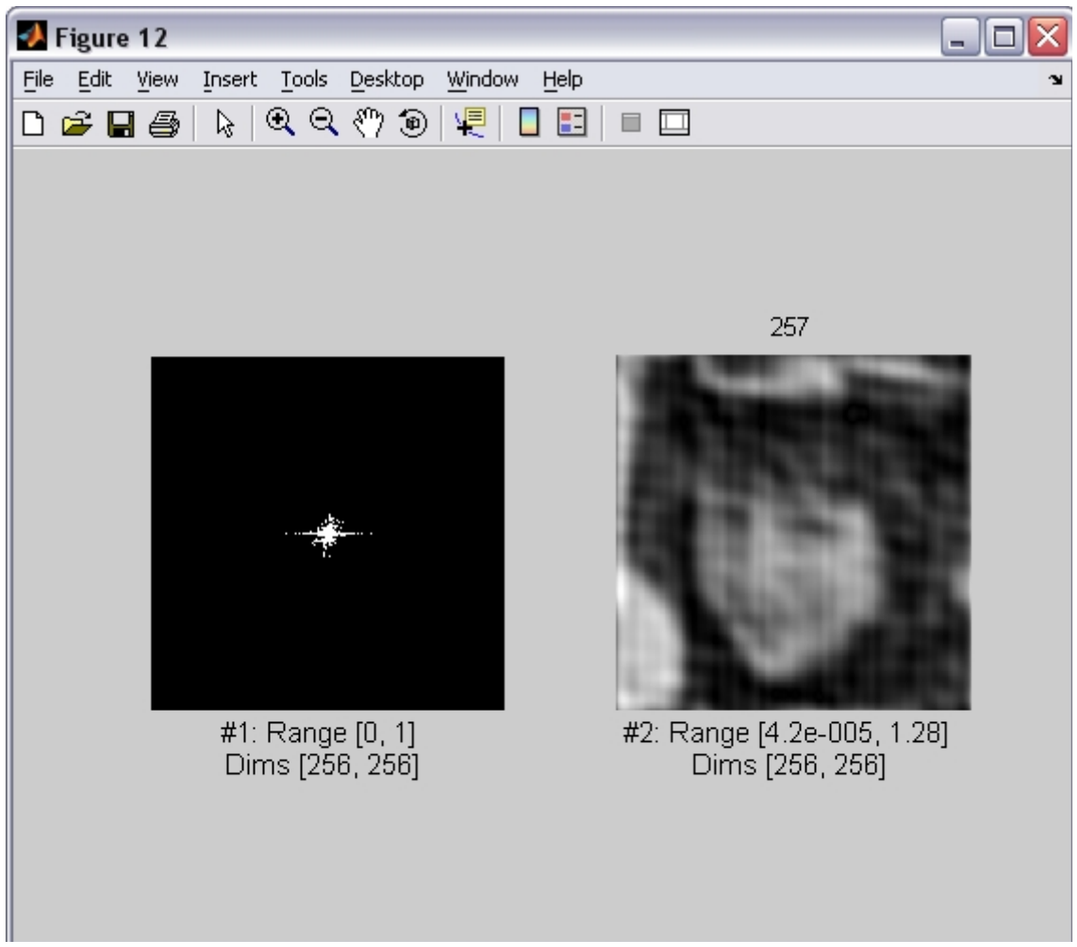


Figure 12
File Edit View Insert Tools Desktop Window Help
257
#1: Range [0, 1]
Dims [256, 256]
#2: Range [4.2e-005, 1.28]
Dims [256, 256]

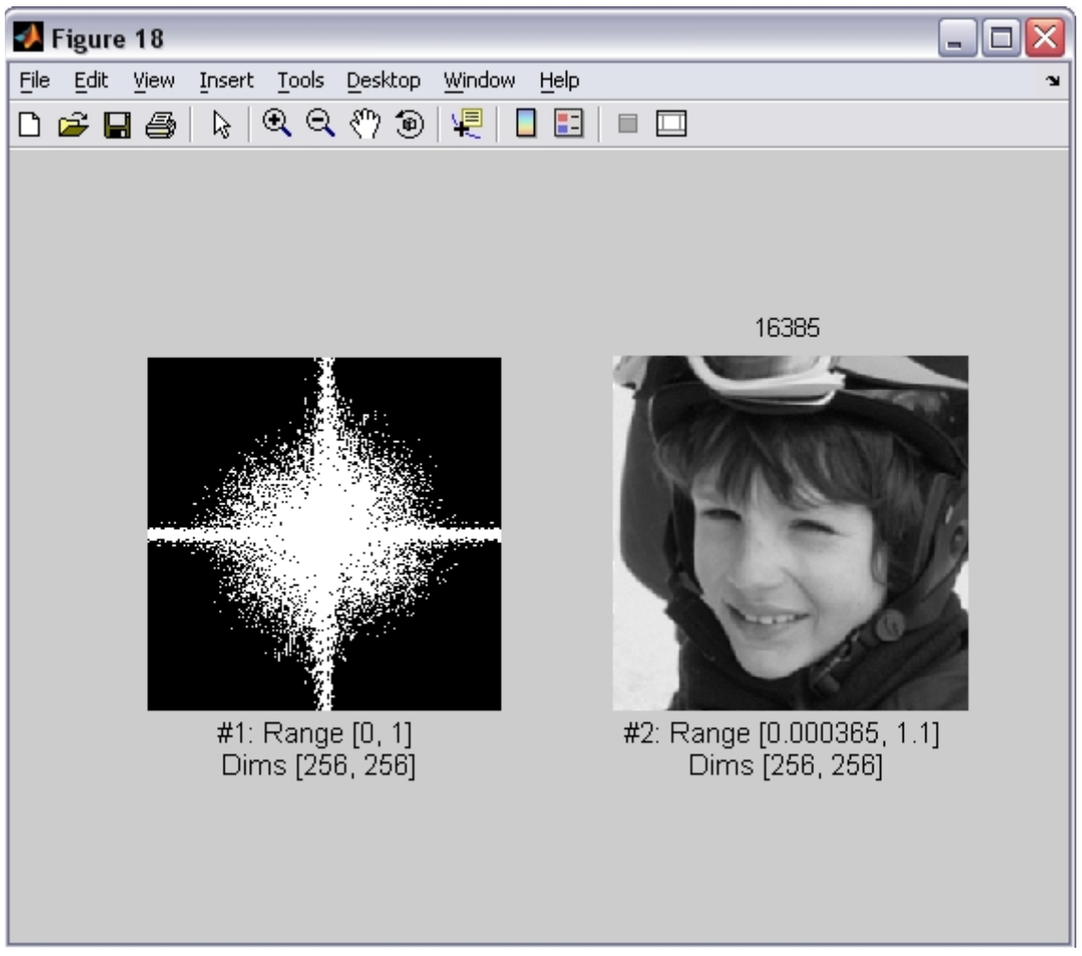


Figure 18
File Edit View Insert Tools Desktop Window Help
16385
#1: Range [0, 1]
Dims [256, 256]
#2: Range [0.000365, 1.1]
Dims [256, 256]

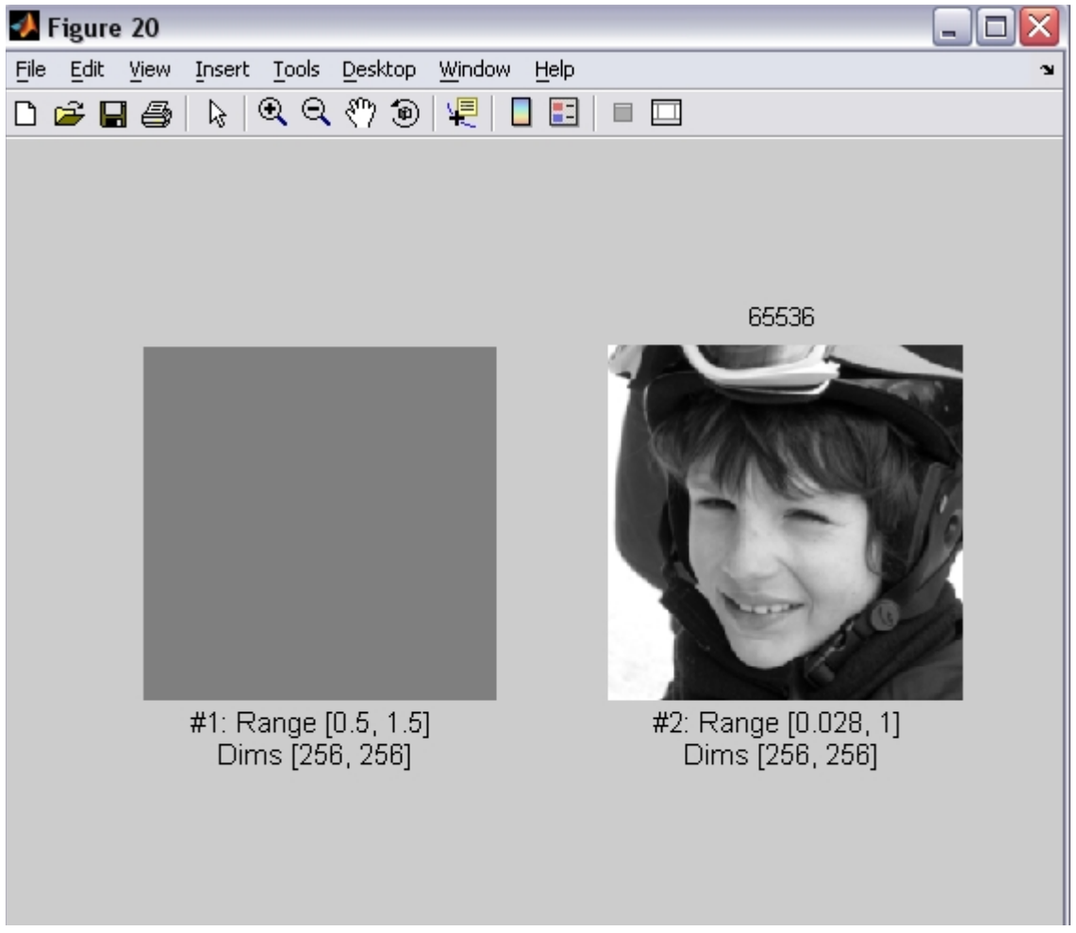
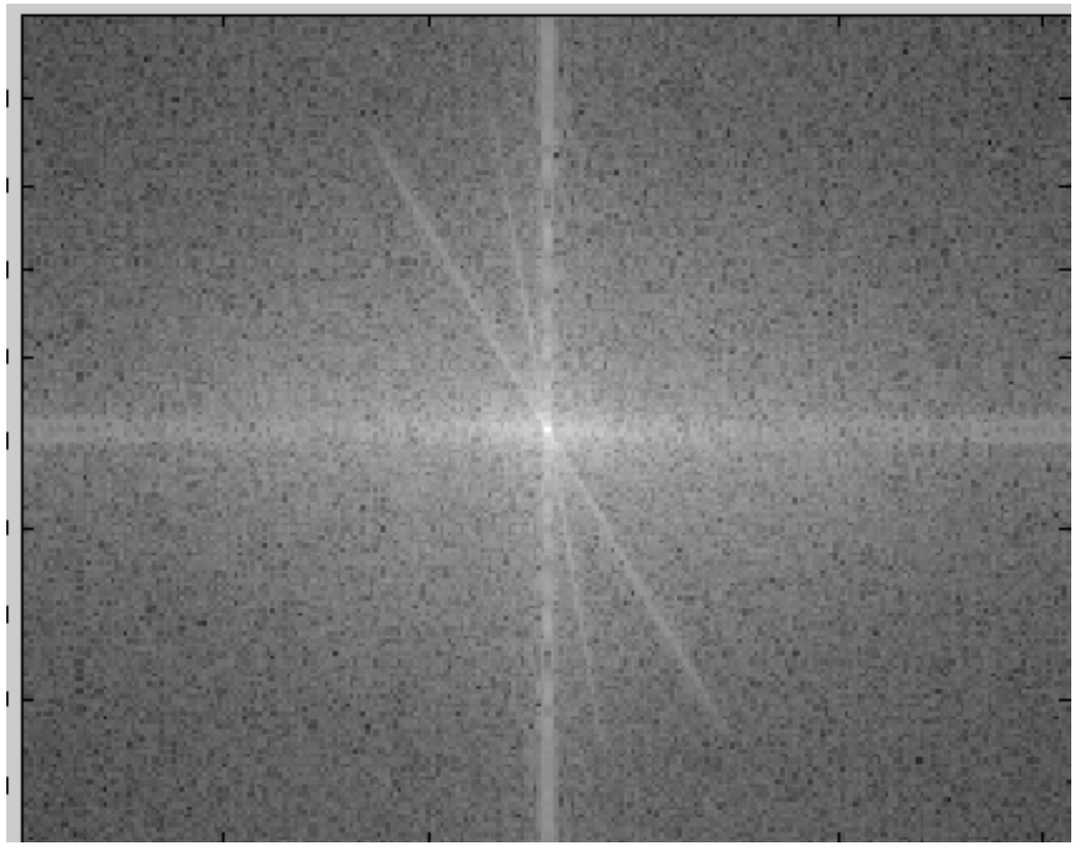
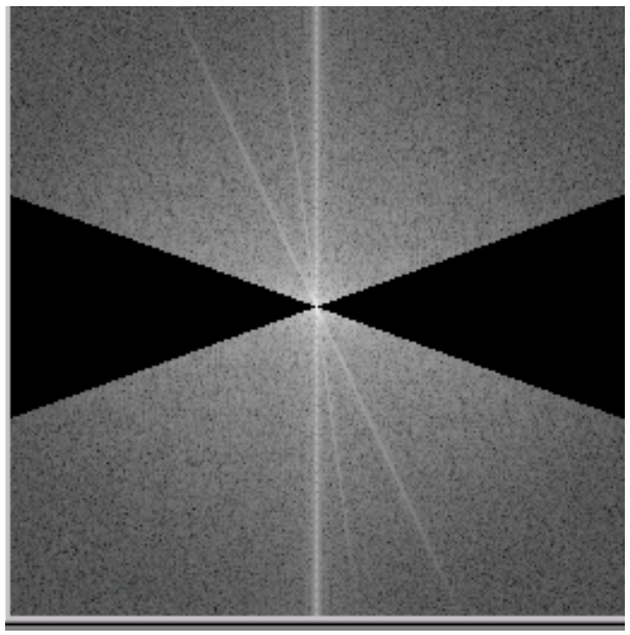


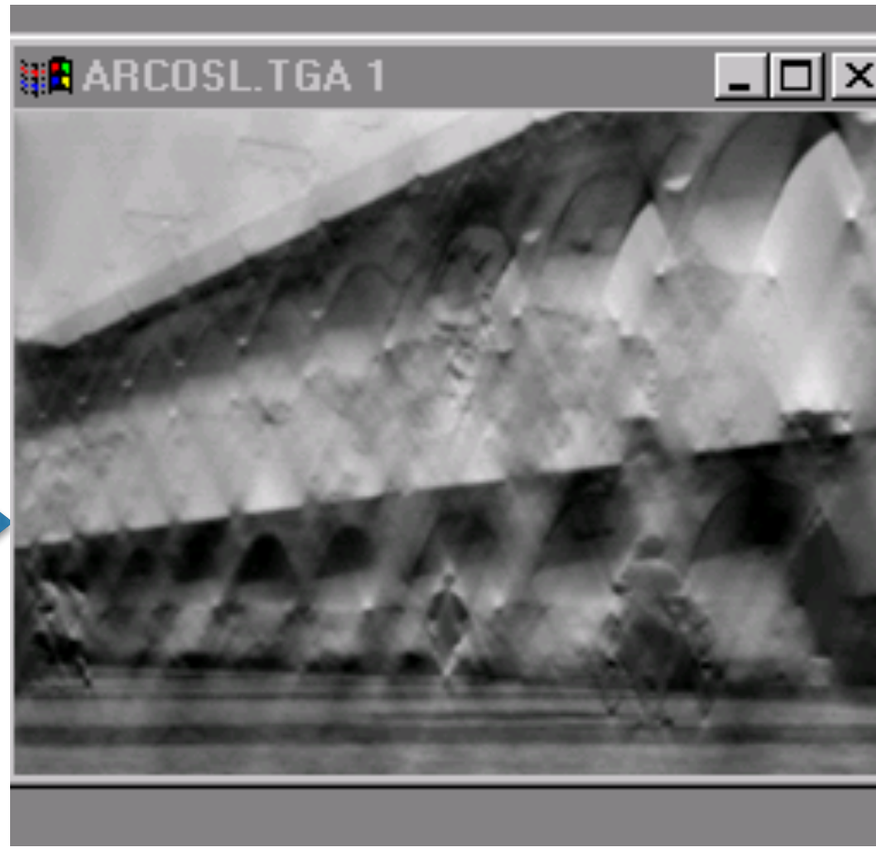
Figure 20
File Edit View Insert Tools Desktop Window Help
65536
#1: Range [0.5, 1.5]
Dims [256, 256]
#2: Range [0.028, 1]
Dims [256, 256]

Man-made Scene

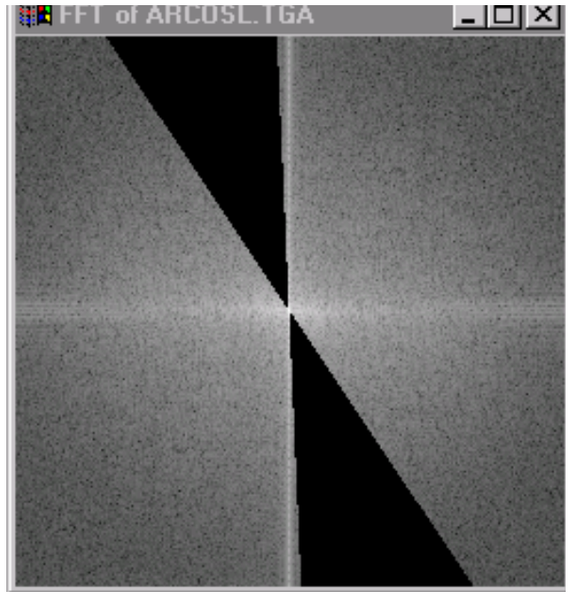


Man-made Scene

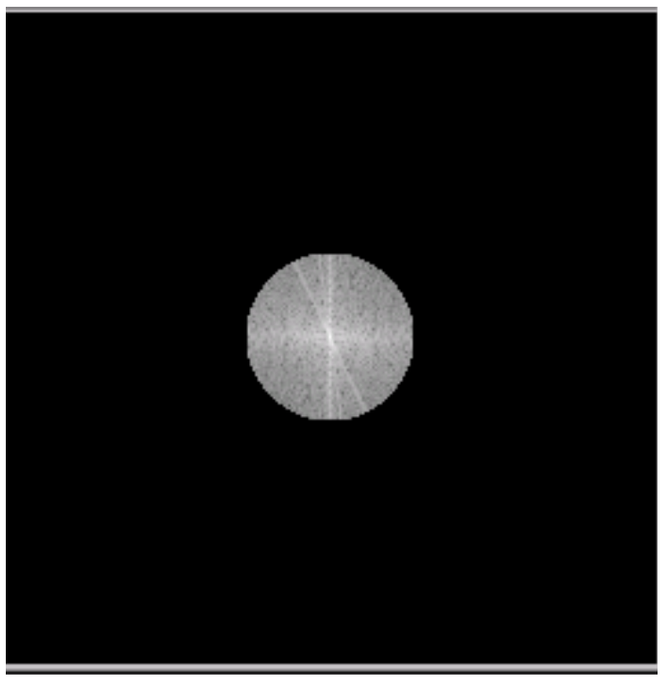




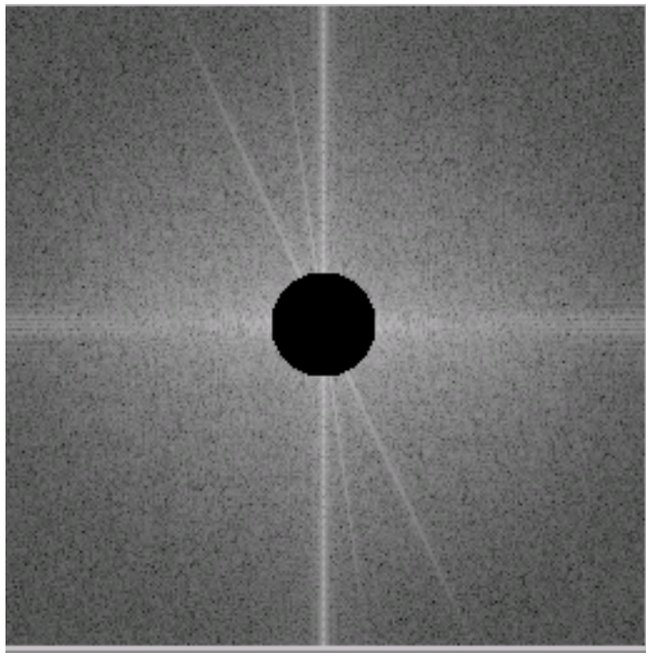
Man-made Scene

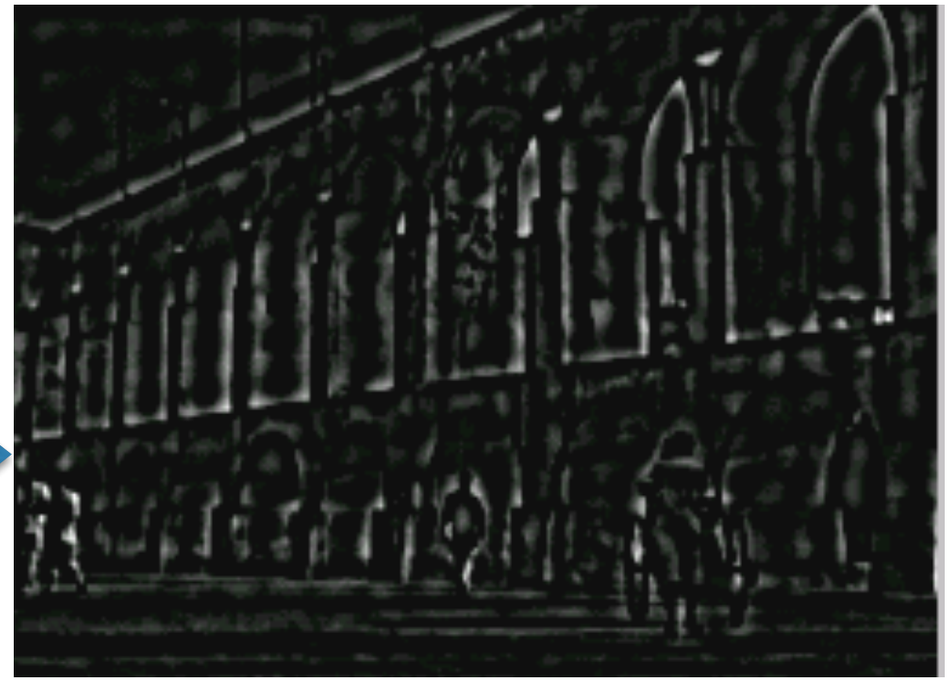


Man-made Scene



Man-made Scene

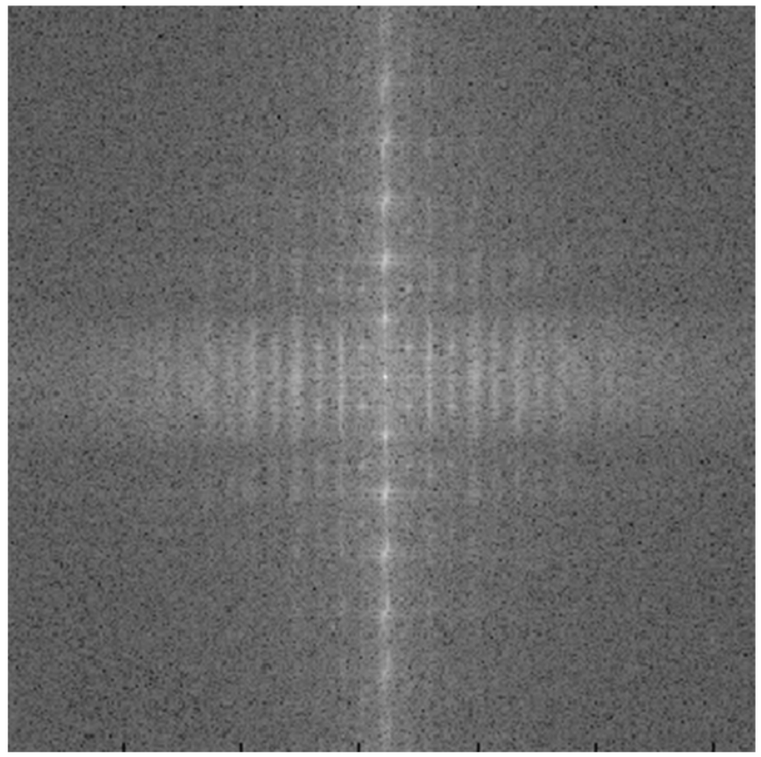




Example 2D Fourier transform

Image with periodic structure

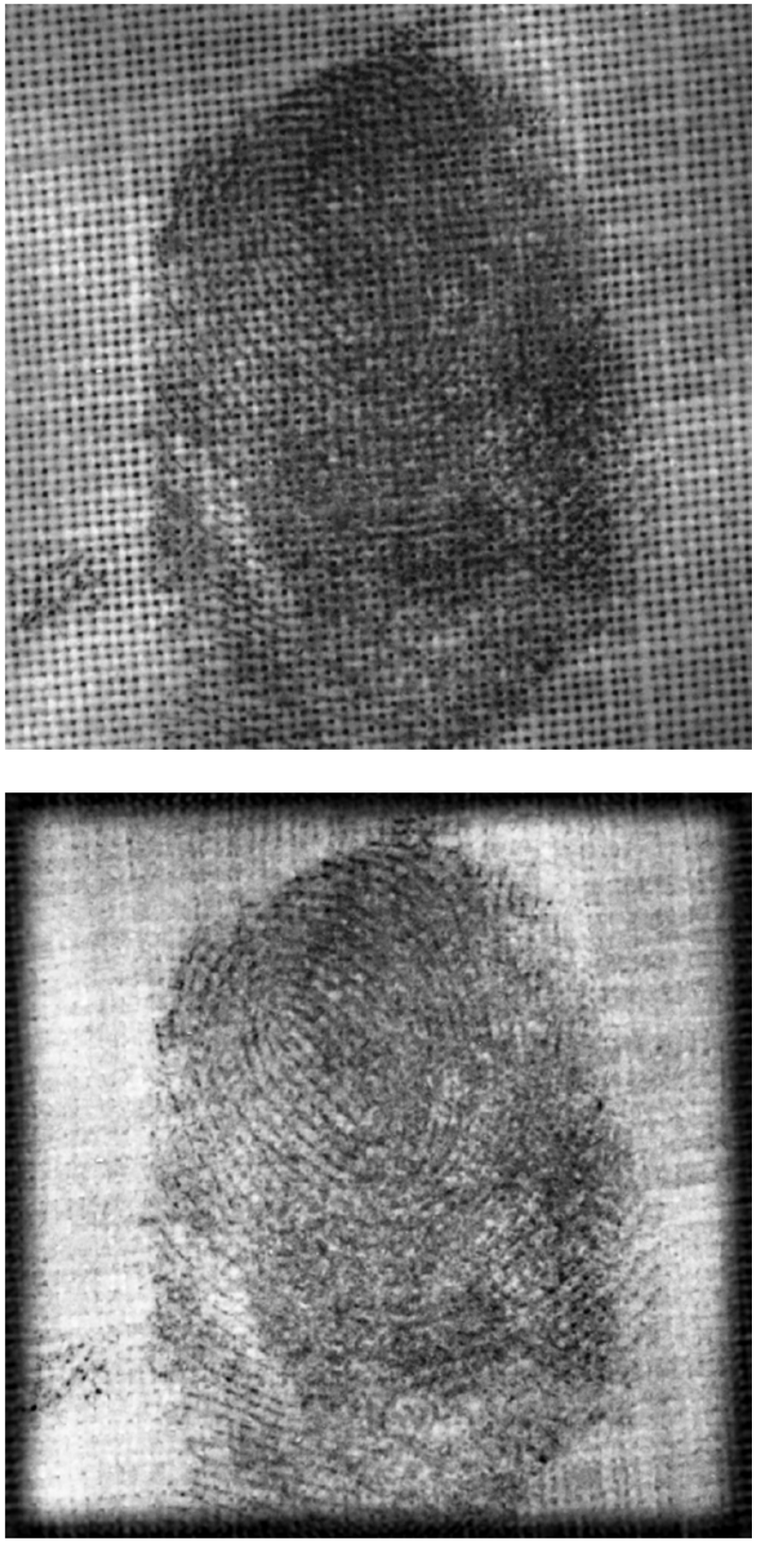
$f(x,y)$

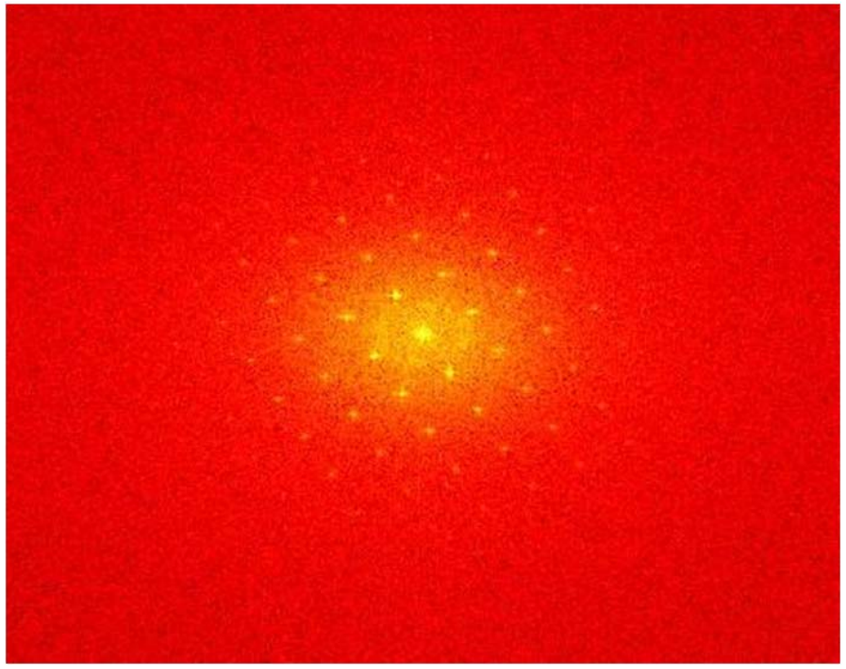


$|F(u,v)|$

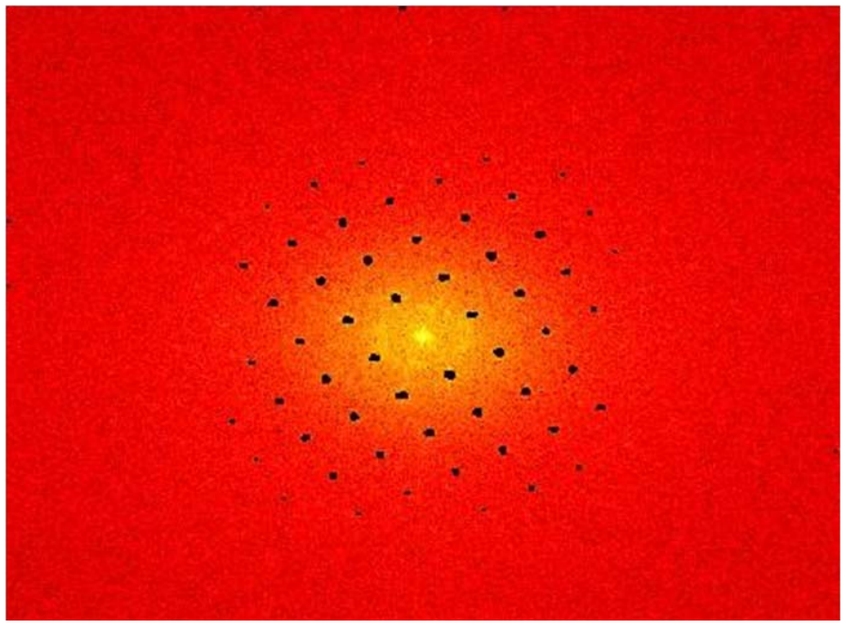
FT has peaks at spatial frequencies of repeated texture

Example – Forensic application



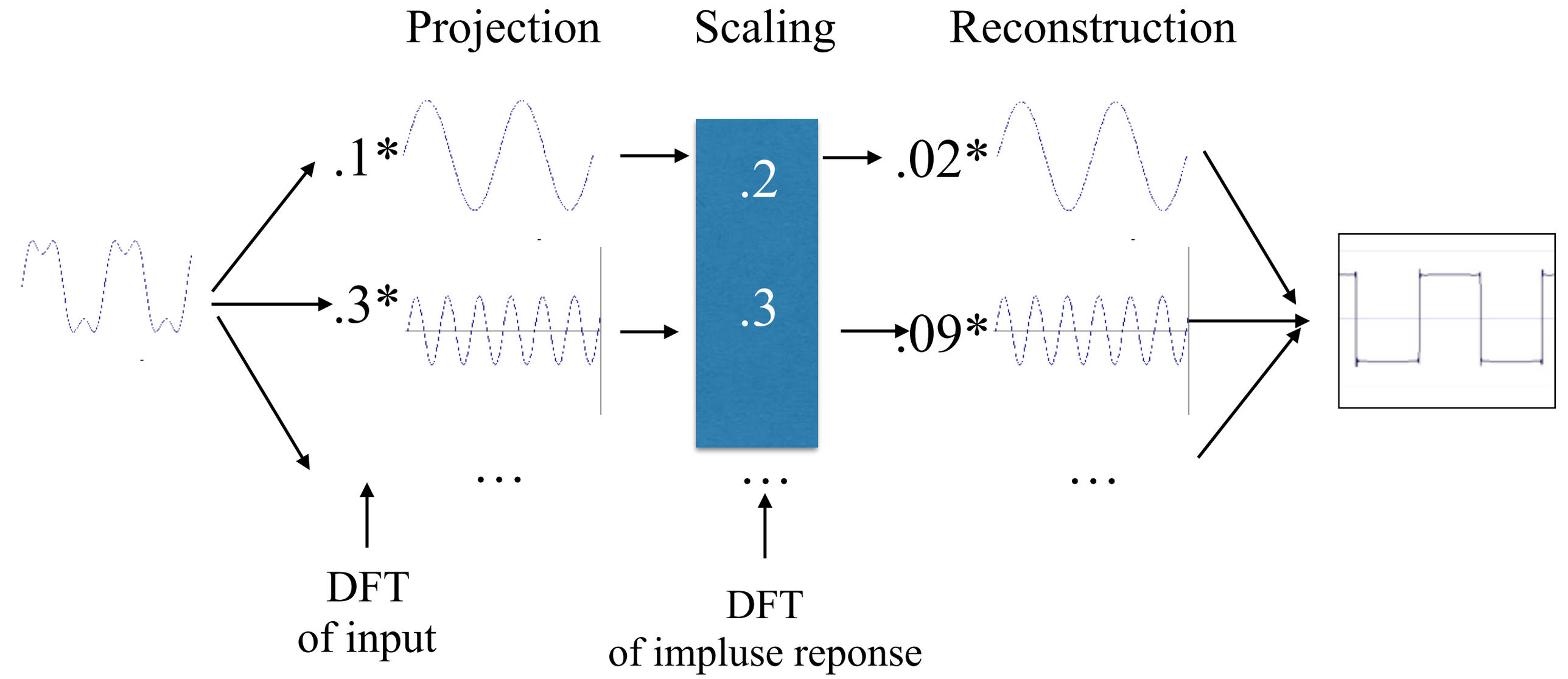


$|F(u,v)|$

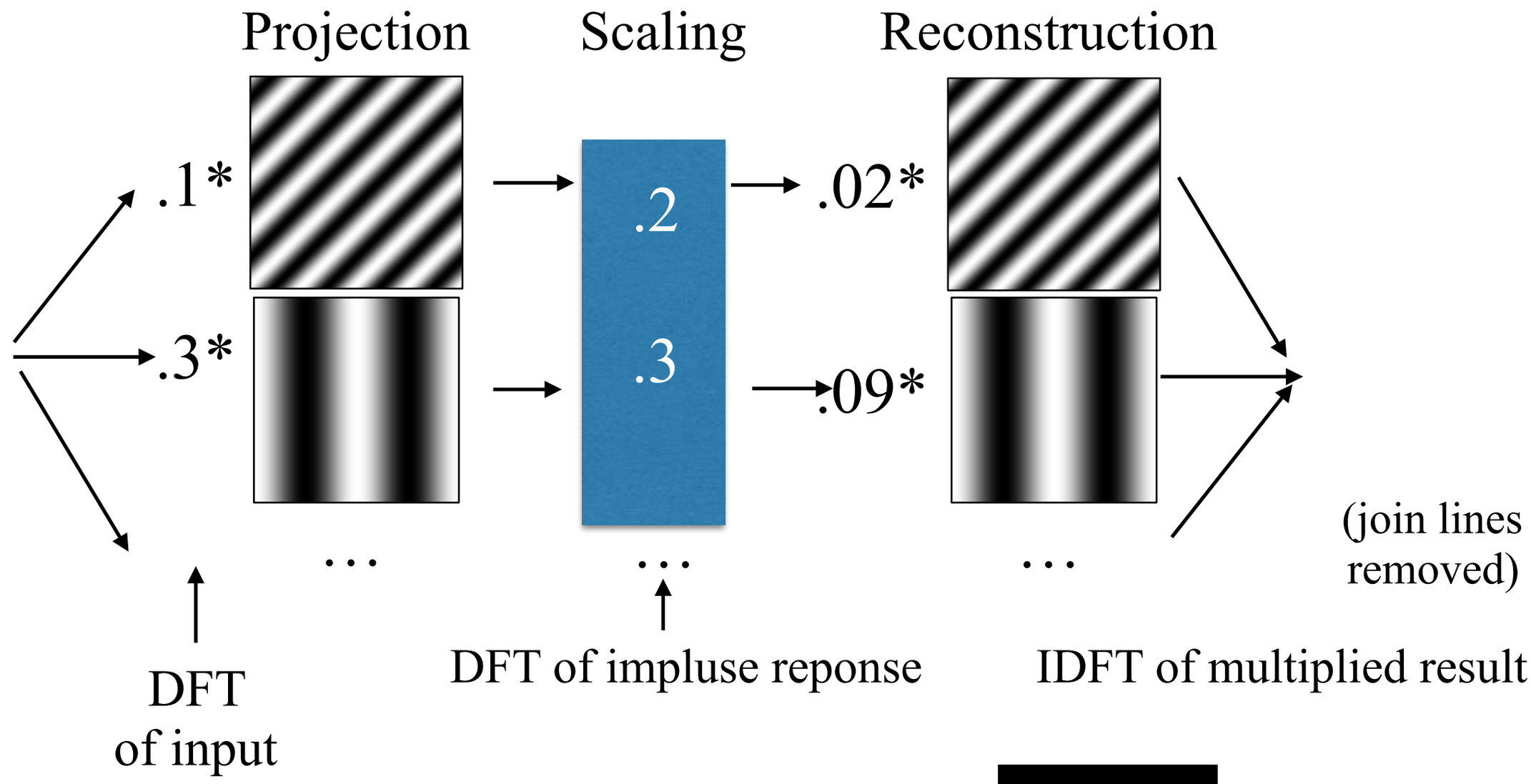


remove peaks

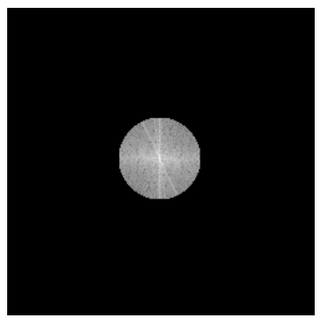
Recall: convolution is multiplication in frequency space



Recall: convolution is multiplication in frequency space



Implies that previous frequency manipulations can be implemented with convolutions!

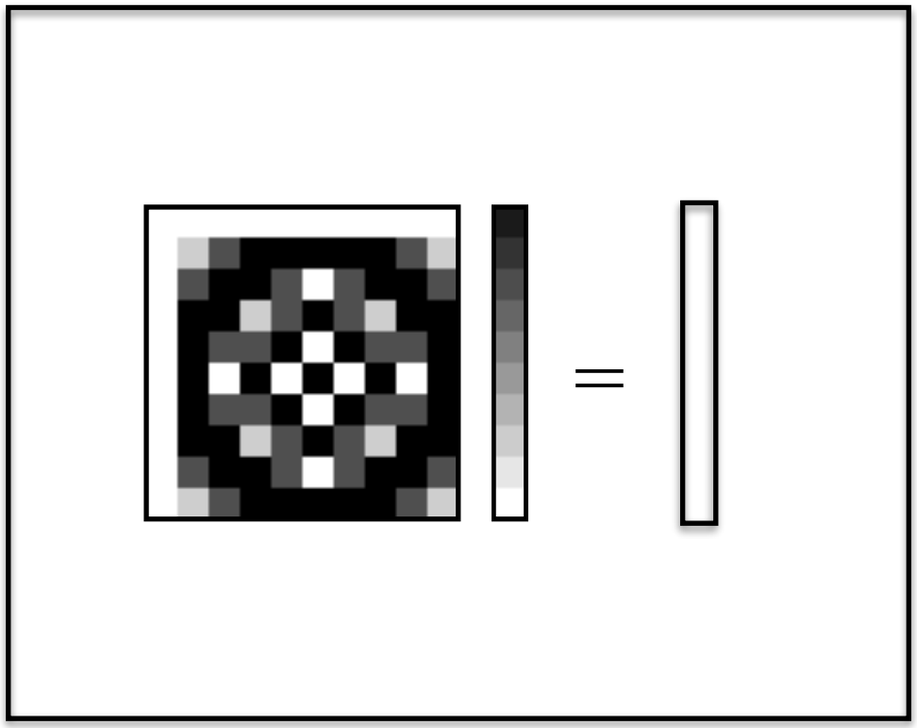


Fast Fourier transform (FFT)

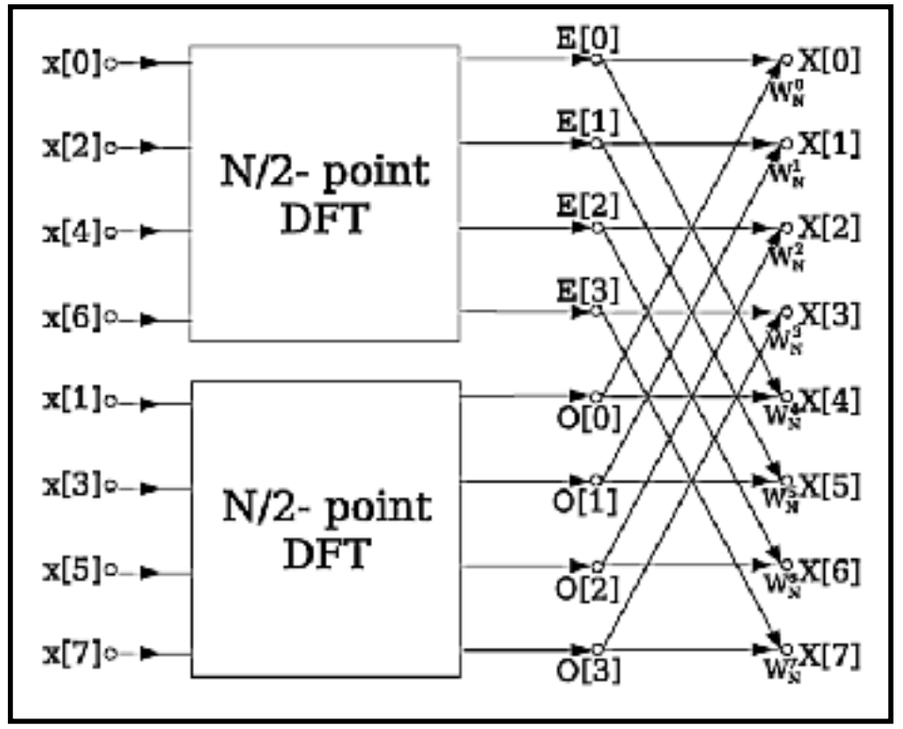
“The most important numerical algorithm of our lifetime”

Strang, Gilbert (May-June 1994). "Wavelets". *American Scientist* **82** (3): 253

Compute (I)DFTs in NlogN instead of N^2(standard matrix multiplication)



versus

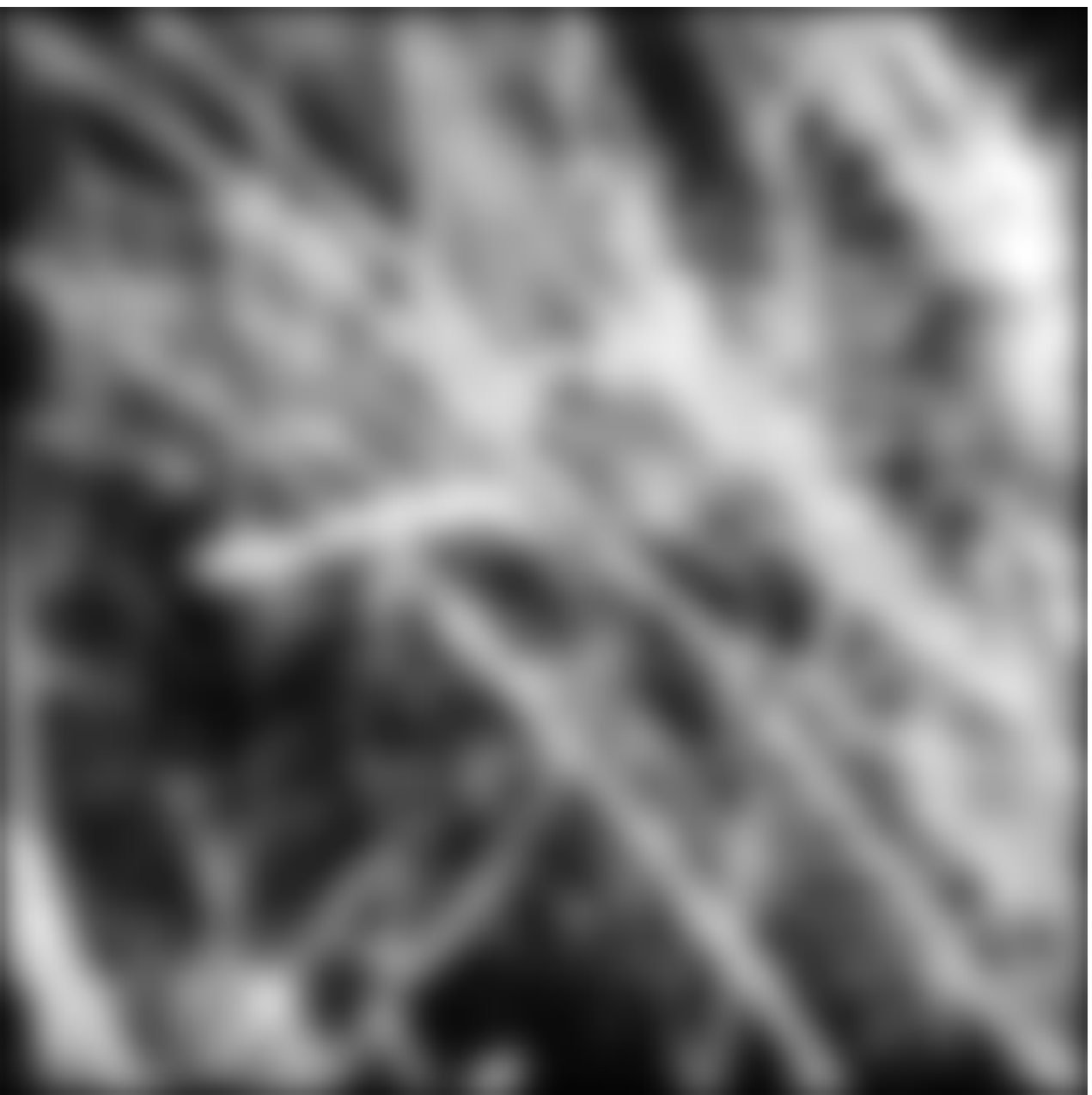


Recursively compute N-element DFT with 2 N/2-element DFT

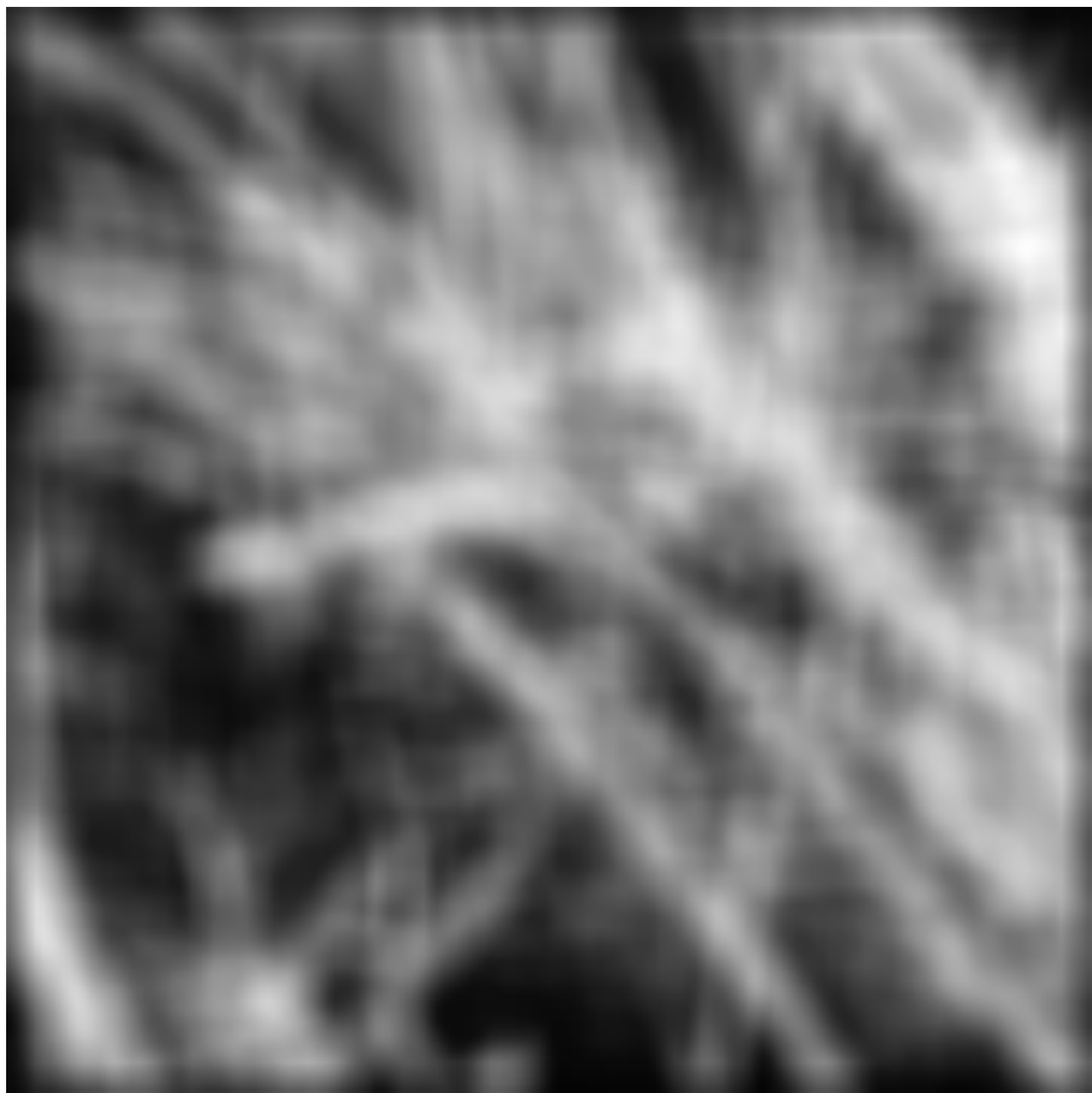
Filtering

Why does the Gaussian give a nice smooth image, but the square filter give edgy artifacts?

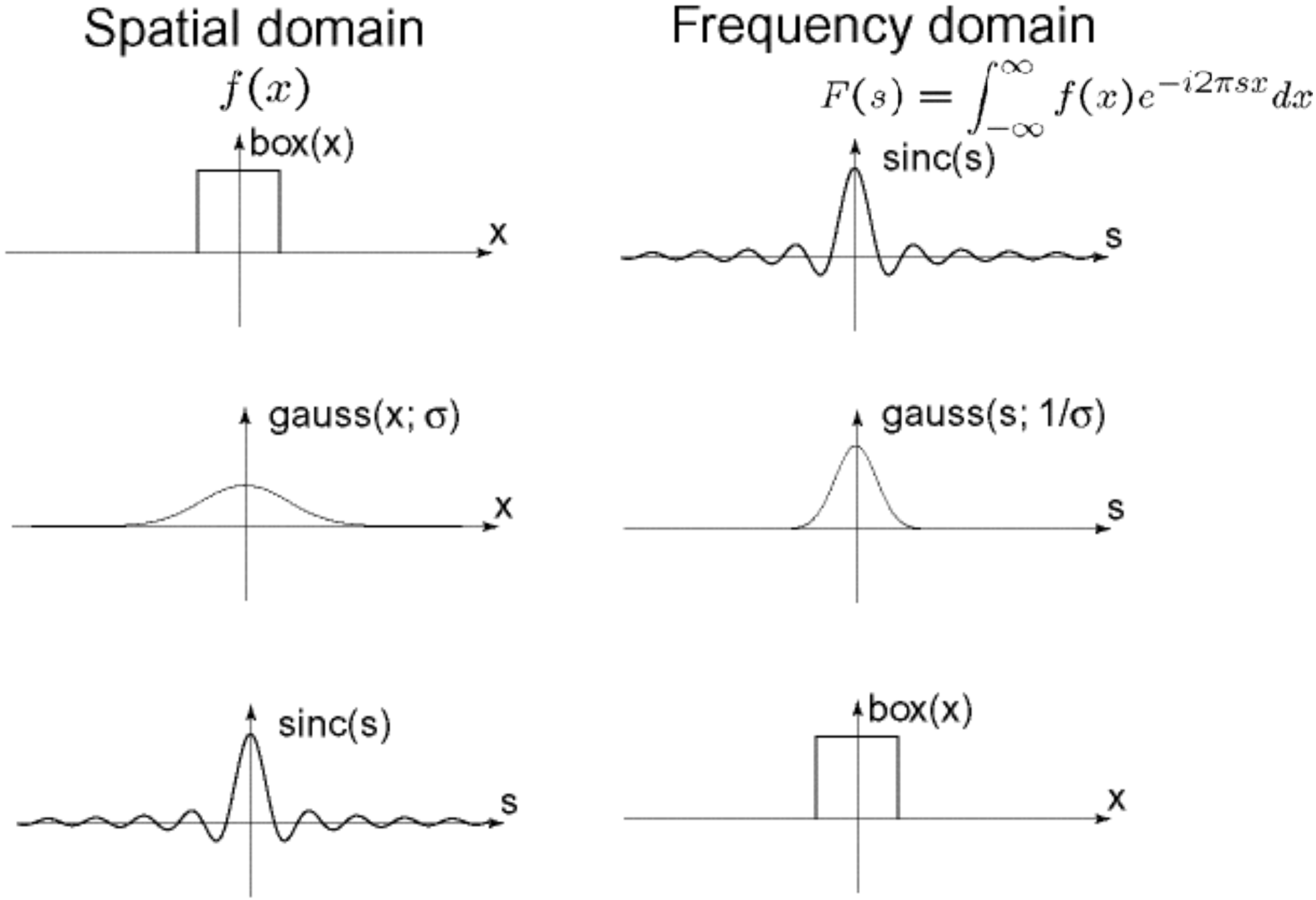
Gaussian 



Box filter 



Recall…


Spatial domain
$f(x)$
box(x)
x
gauss(x; σ)
x
sinc(s)
s
Frequency domain
$F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi sx} dx$
sinc(s)
s
gauss(s; 1/σ)
s
box(x)
x

Another look: blurring

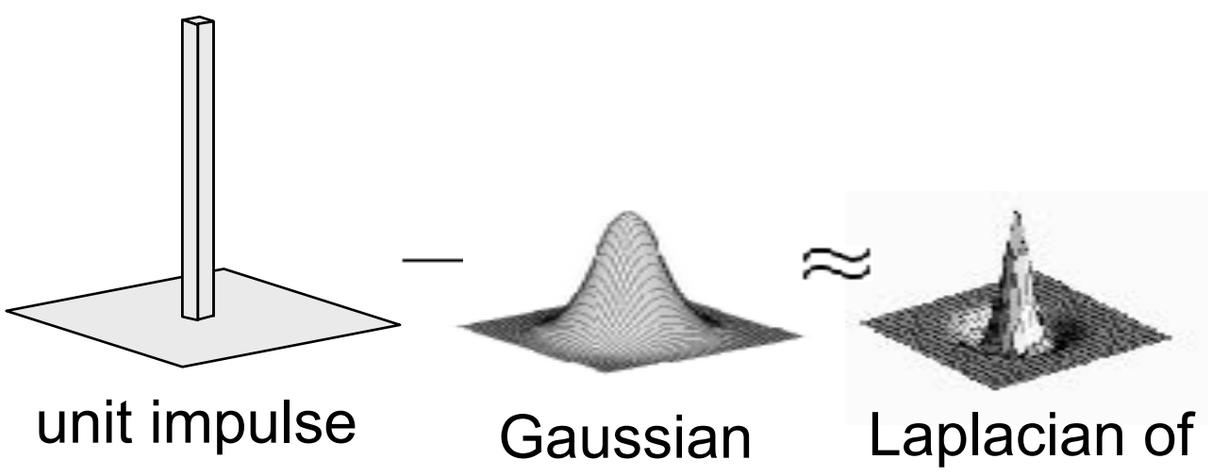
original

Another look: blurring

smoothed (5x5 Gaussian)

A “High-Pass” filter

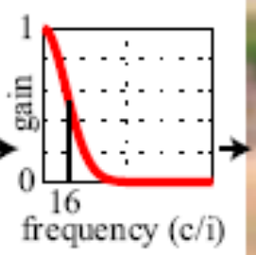
smoothed – original

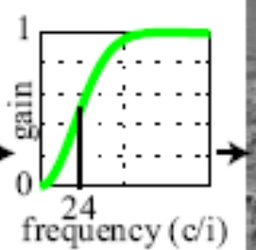


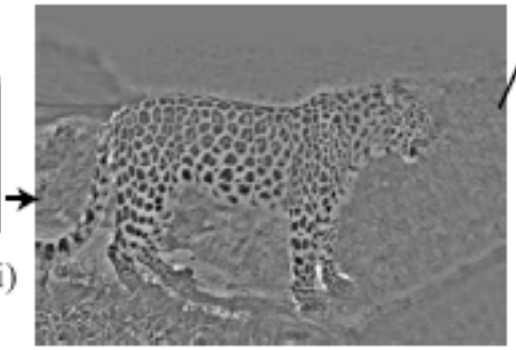
Hybrid Images

Gaussian Filter

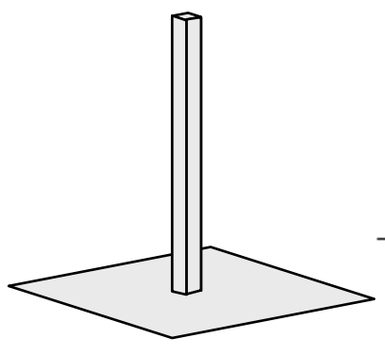
A. Oliva, A. Torralba, P.G. Schyns, [“Hybrid Images,”](#) SIGGRAPH 2006





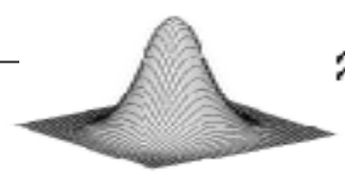


Laplacian Filter



unit impulse

—

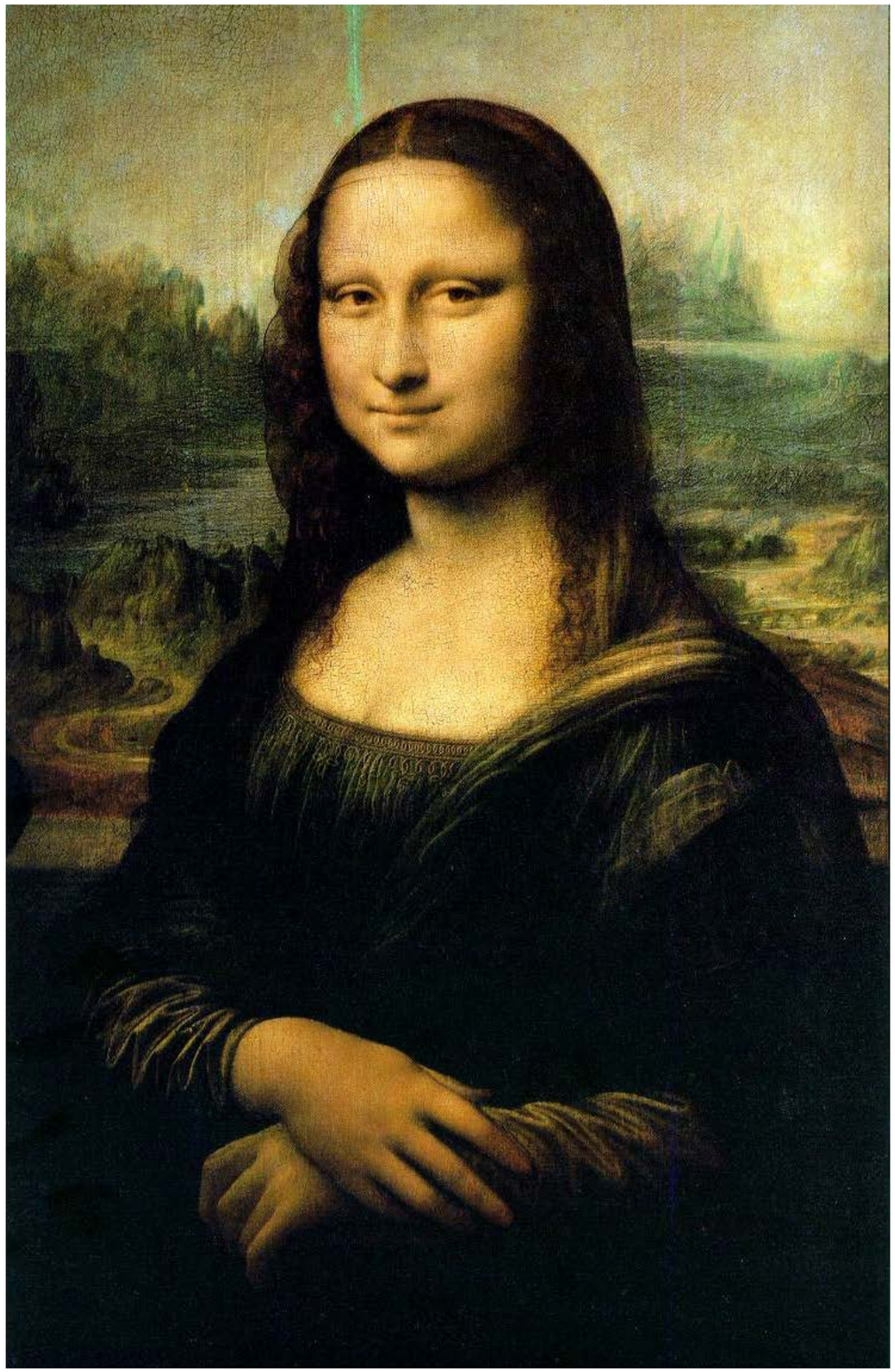


Gaussian

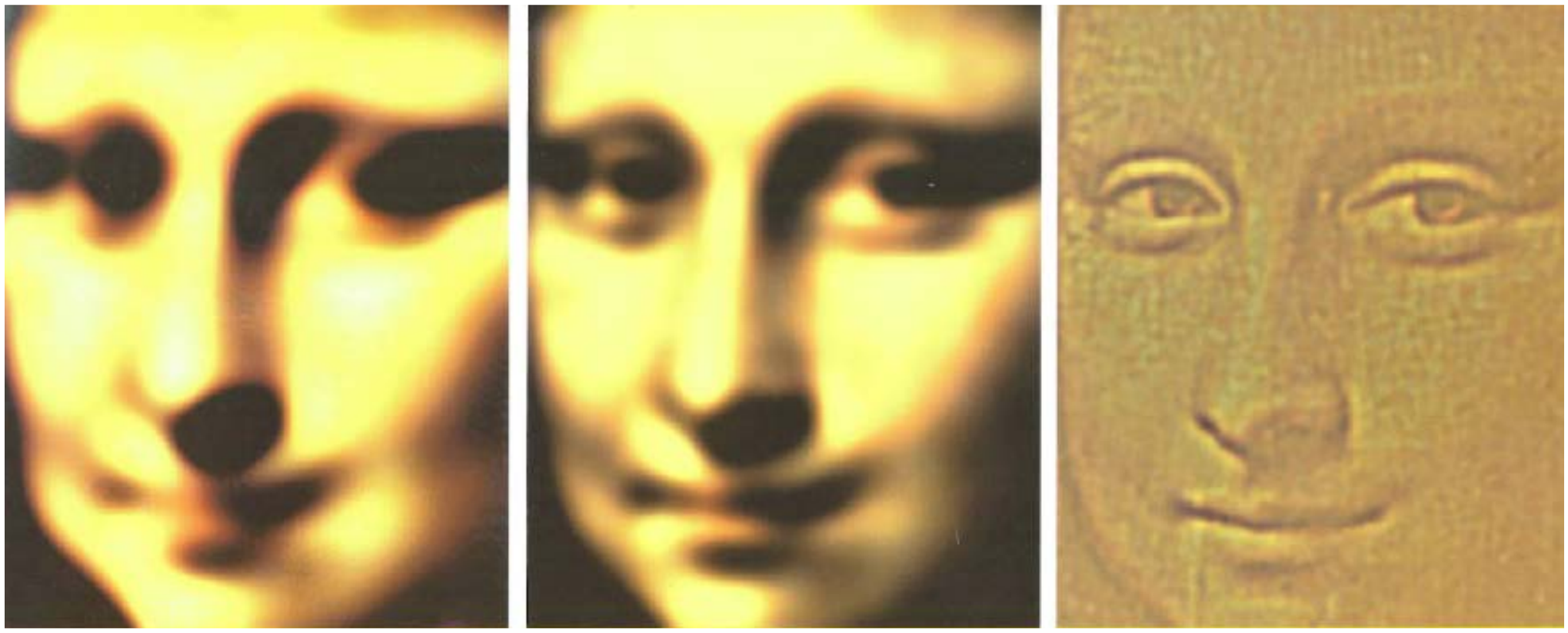
≈

Laplacian of Gaussian

DaVinci and Hybrid Images



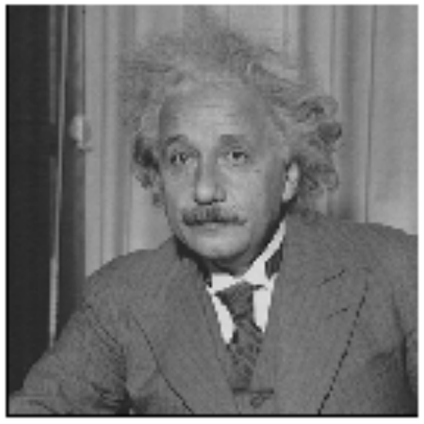
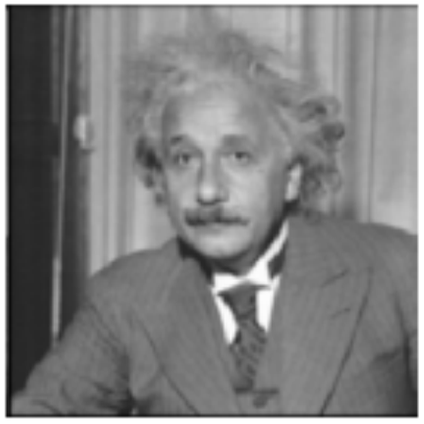
Humans have less resolution (fewer sensors) near periphery



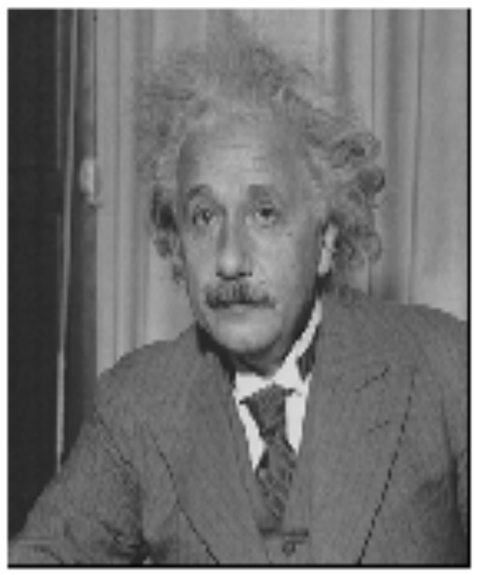
low frequencies middle frequencies high frequencies

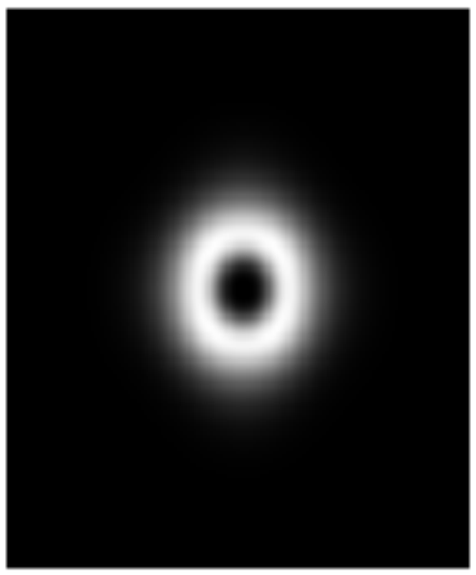
Low-pass, Band-pass, High-pass filters

low-pass:

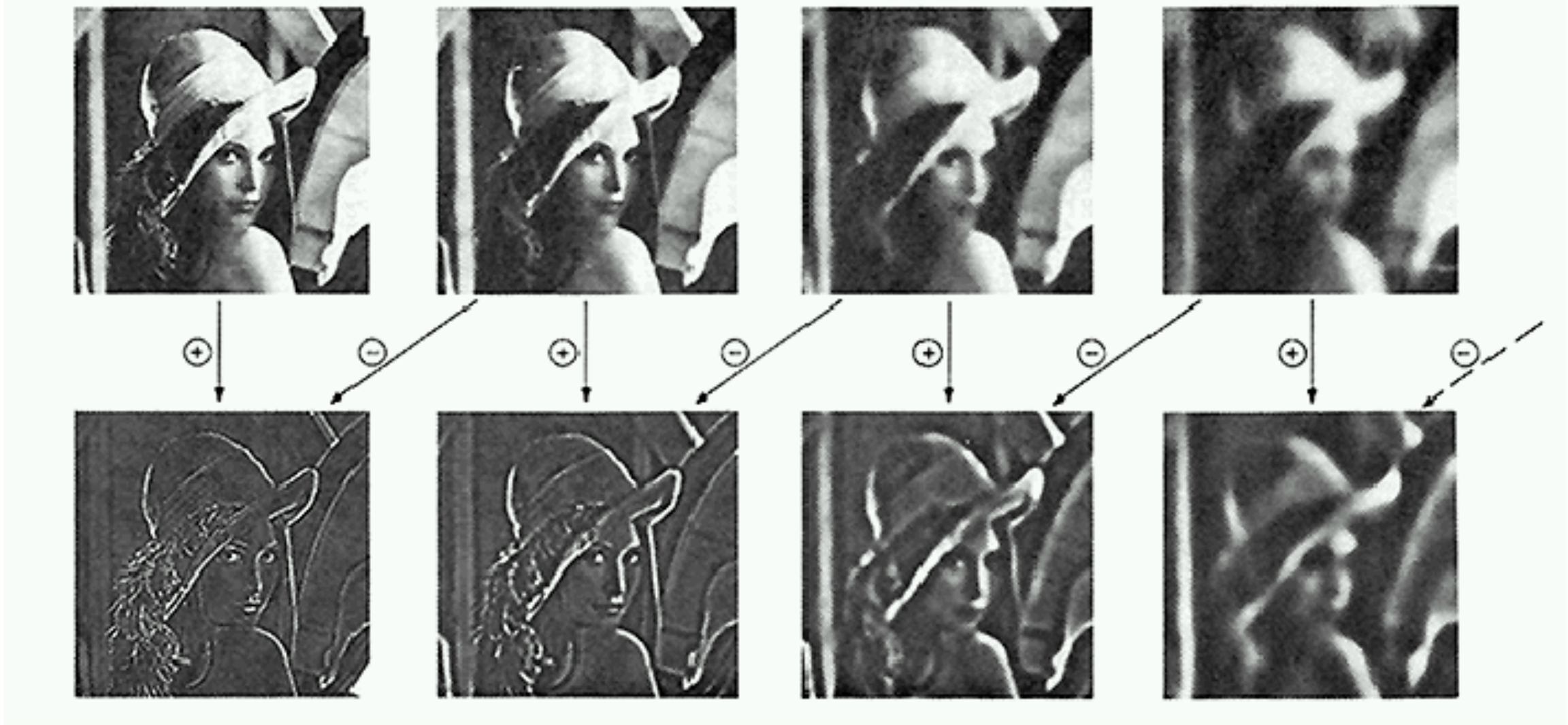

band-pass:





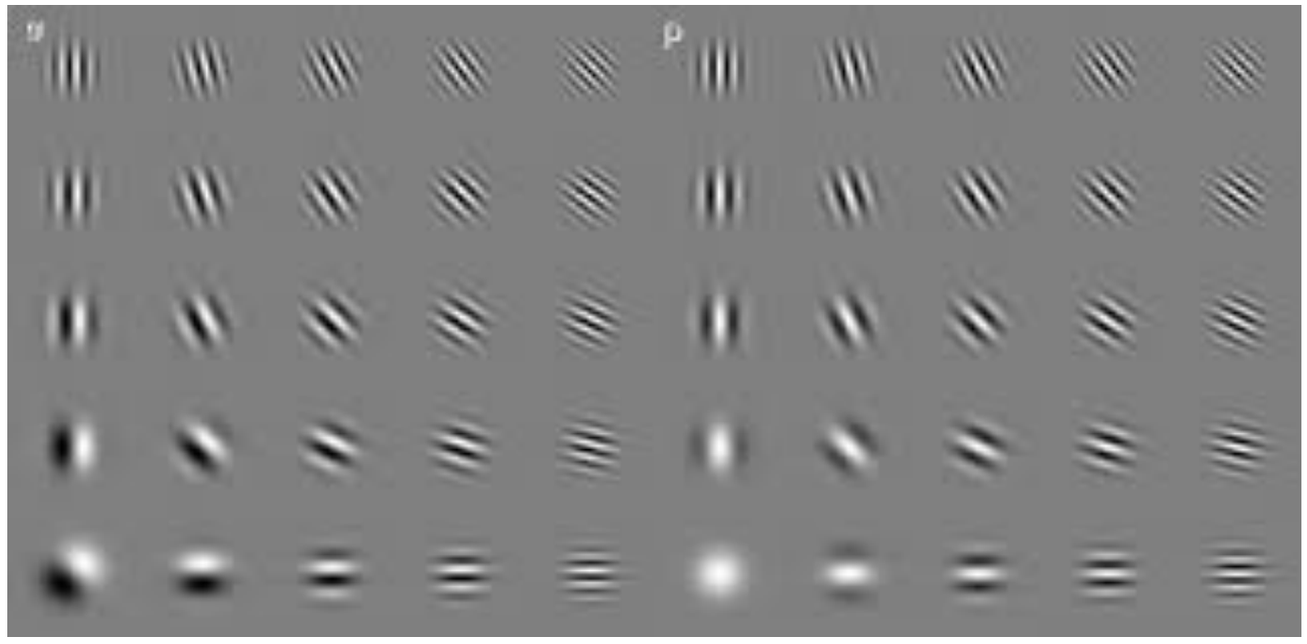
Efficient construction of a collection of bandpass images

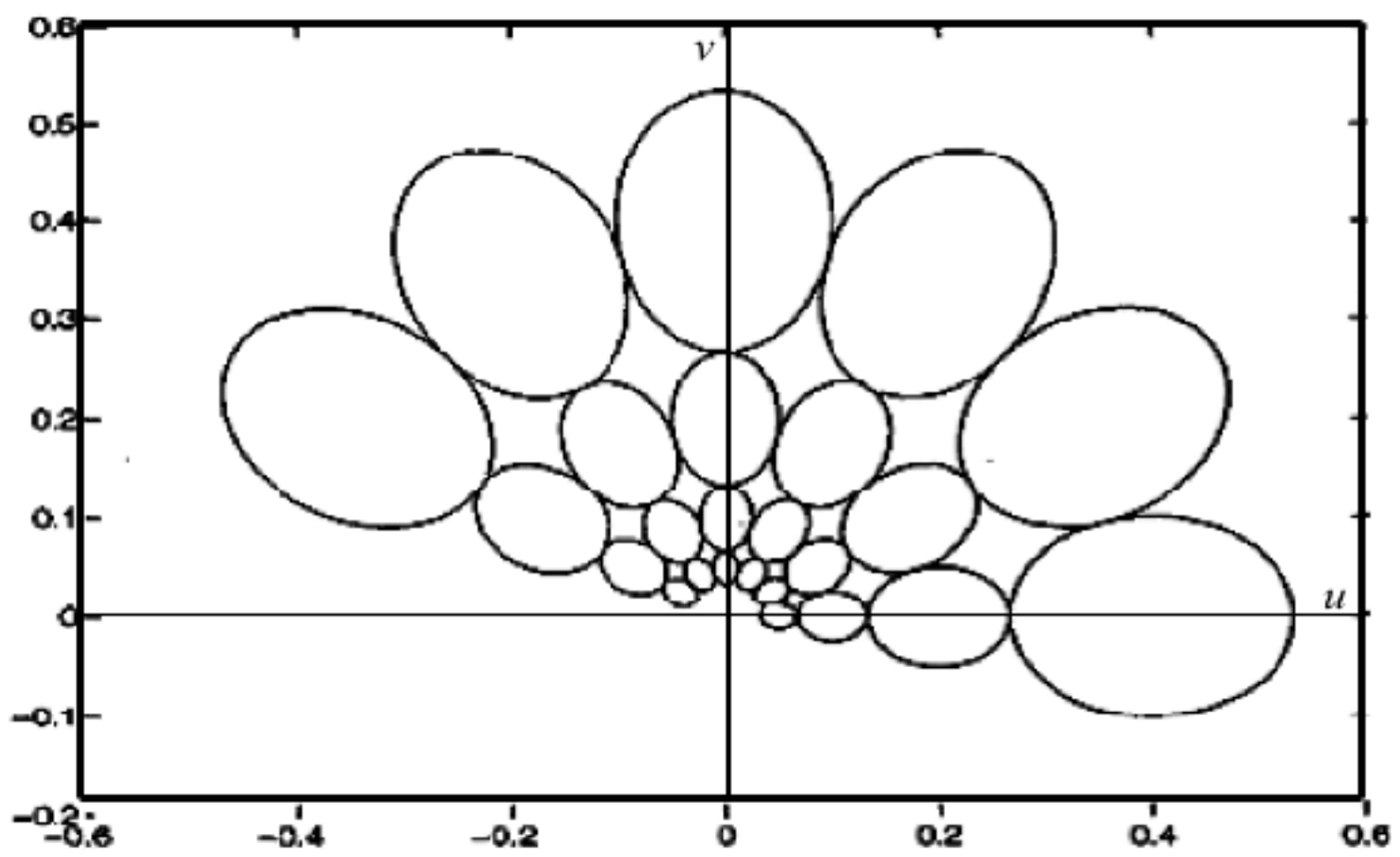
Lowpass Images



Bandpass Images

Gabor filters



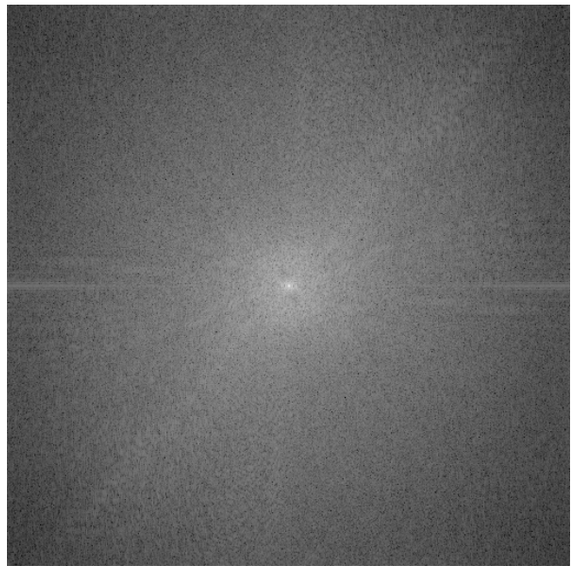


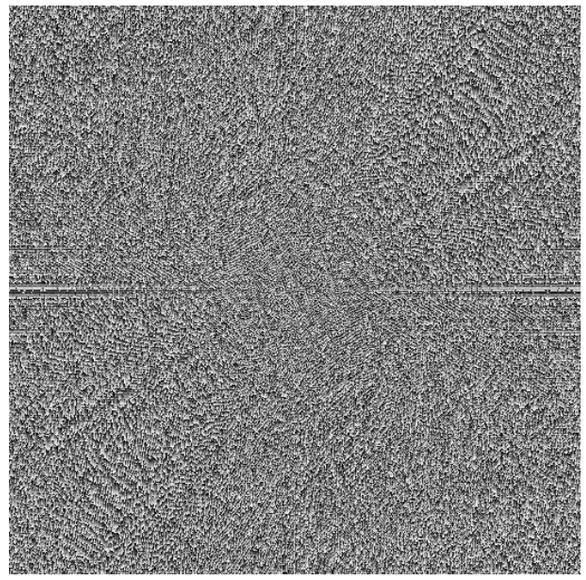
reason for a “quadrature pair”

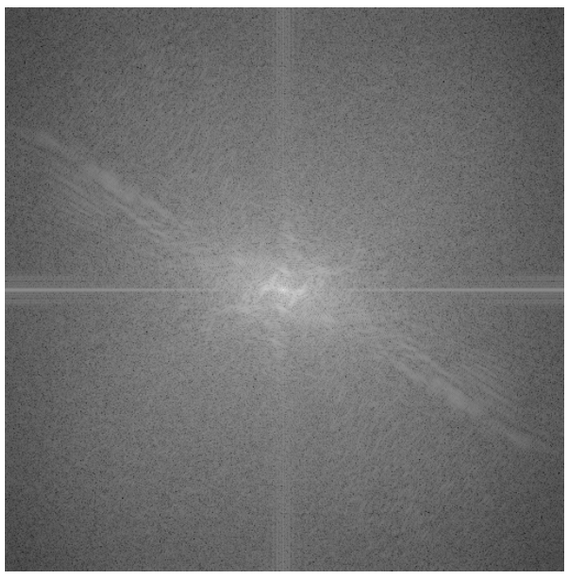
Phase and Magnitude

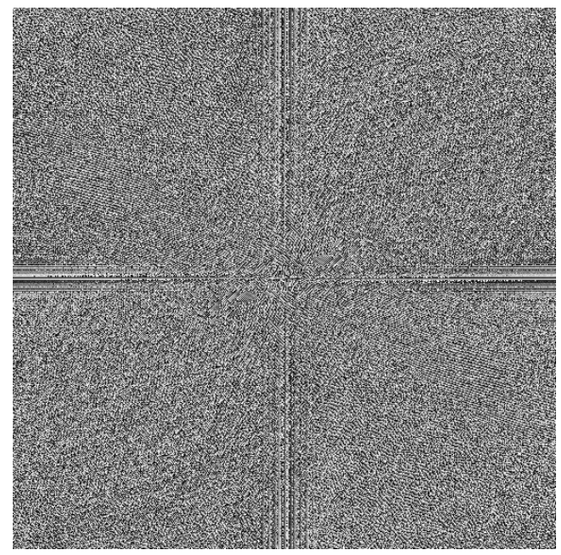
- Curious fact
  - all natural images have about the same magnitude transform
  - hence, phase seems to matter, but magnitude largely doesn't

- Demonstration
  - Take two pictures, swap the phase transforms, compute the inverse - what does the result look like?

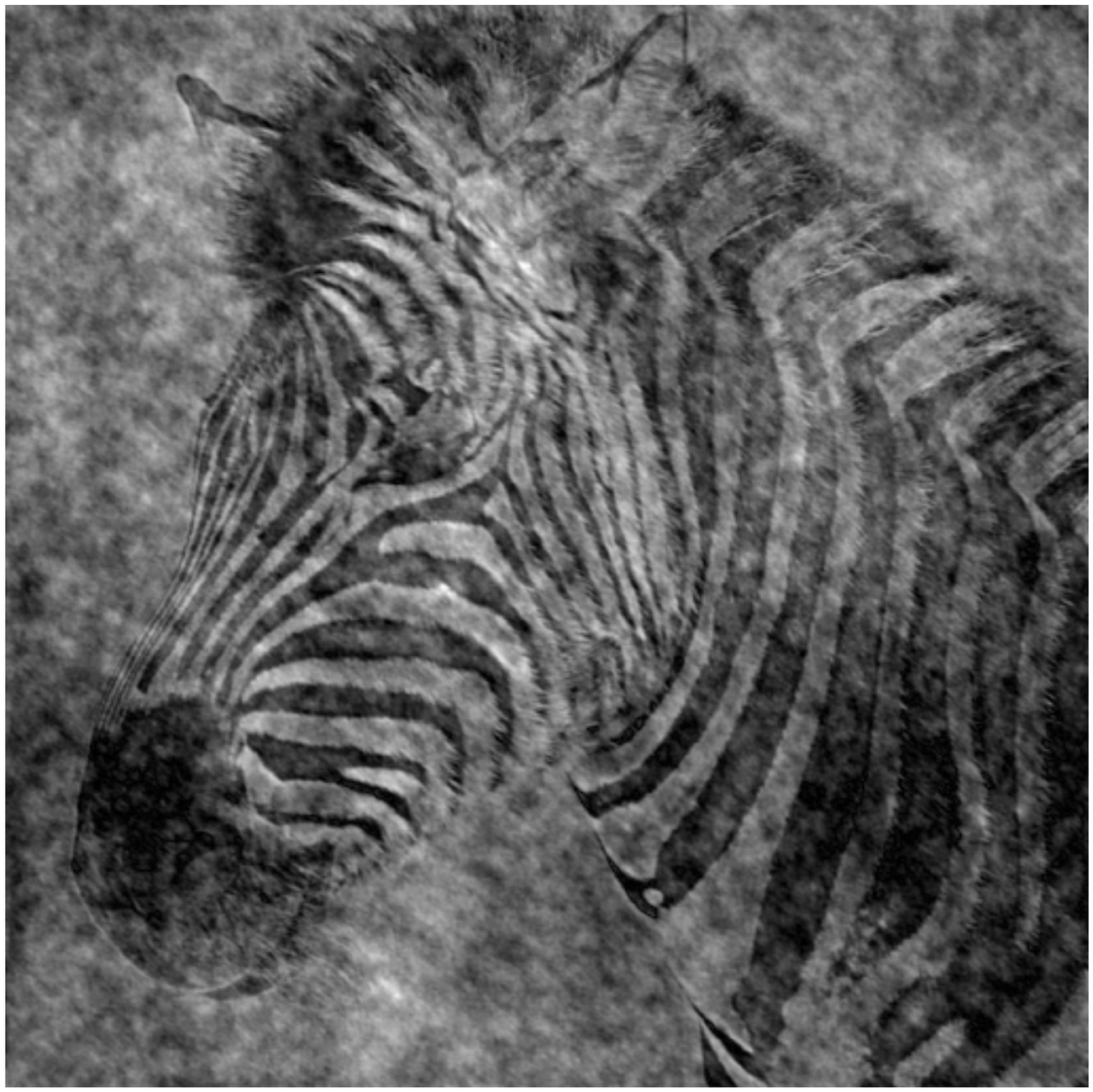




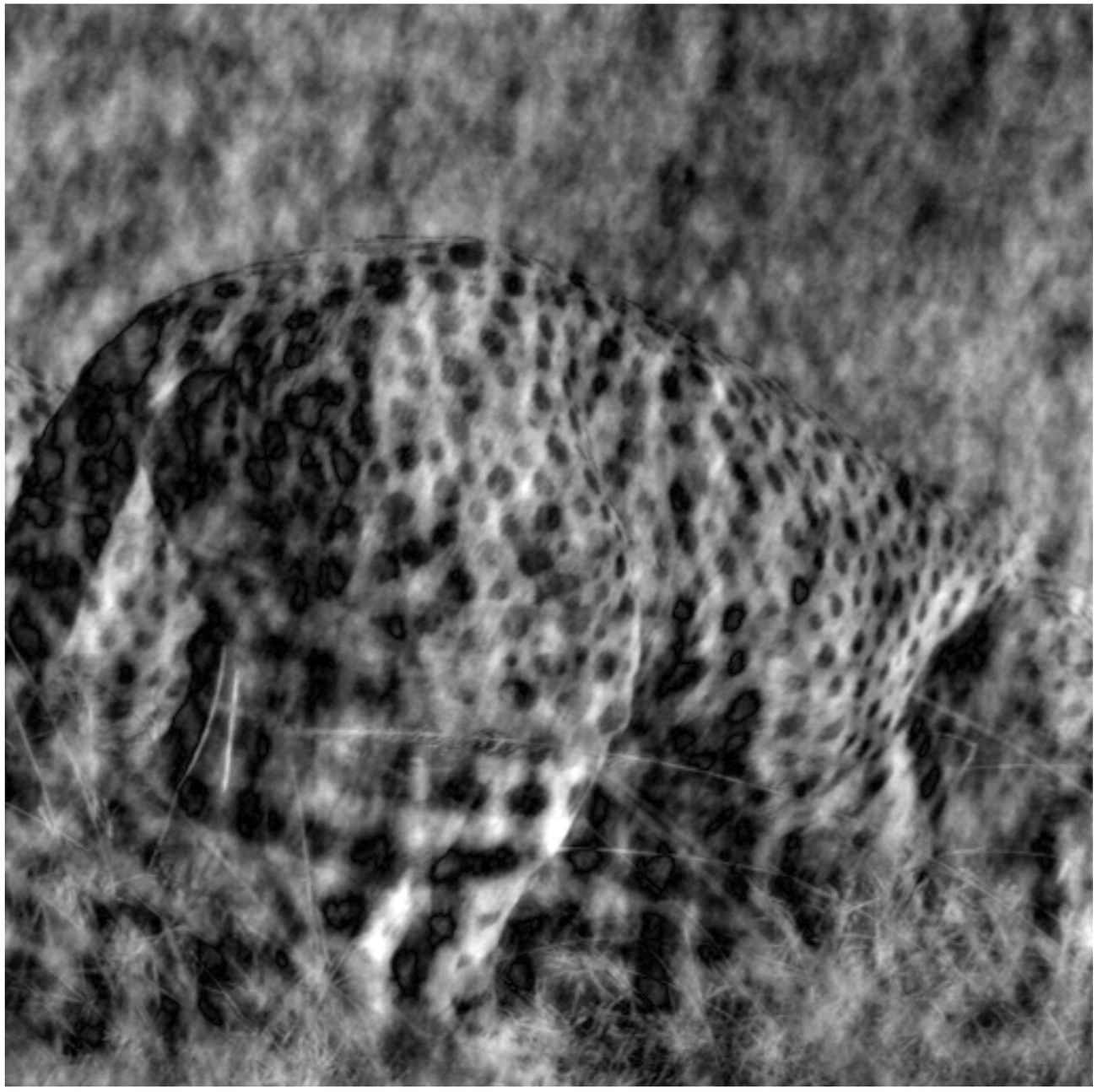




Reconstruction
with zebra phase,
cheetah
magnitude



Reconstruction
with cheetah phase,
zebra magnitude



Phase and Magnitude

Image with cheetah phase
(and zebra magnitude)

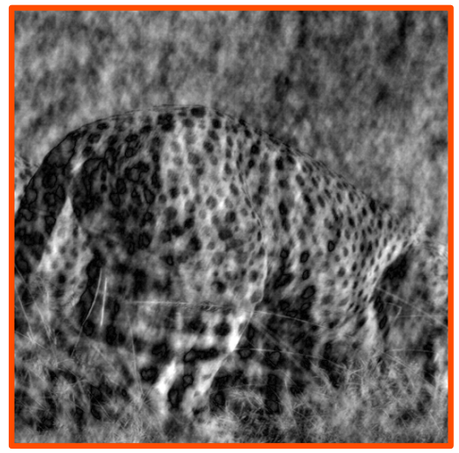
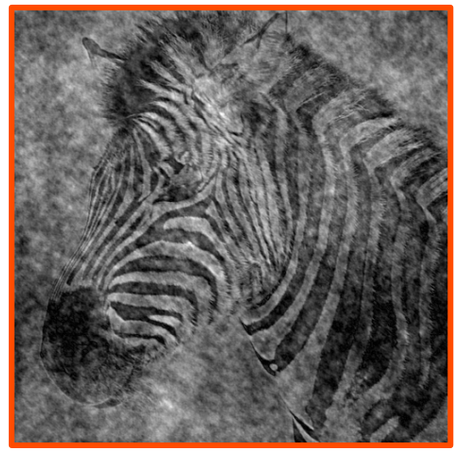


Image with zebra phase
(and cheetah magnitude)

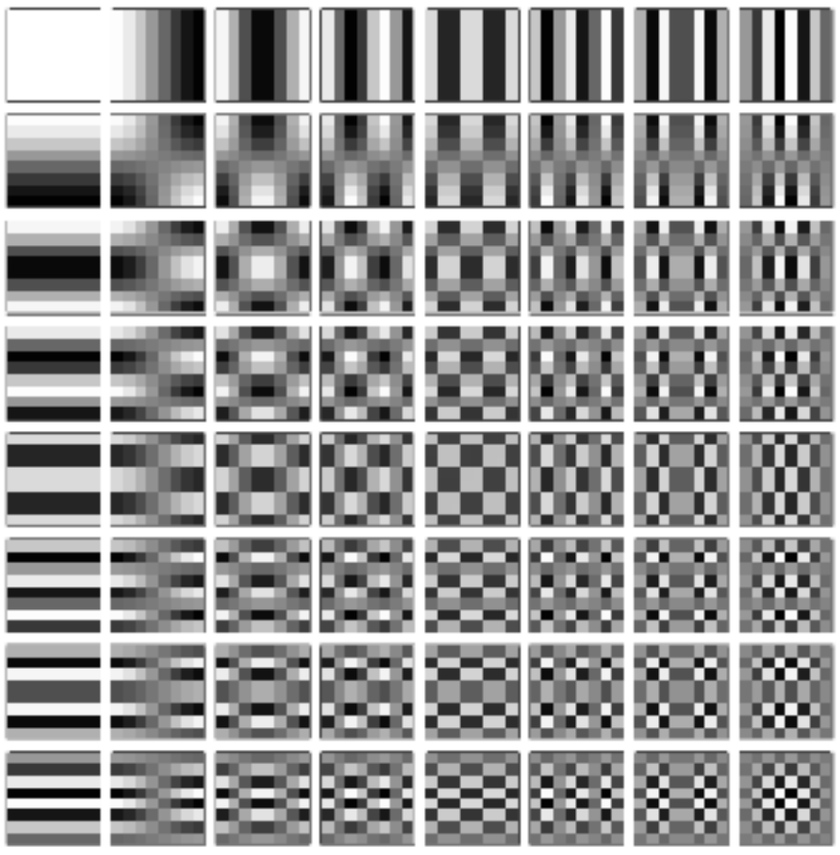


Application: JPEG compression

Optimize DFT for real images; we only need to encode half the values

Turns out we don't need complex exponentials, only cosine functions
= > discrete cosine transforms (DCT)

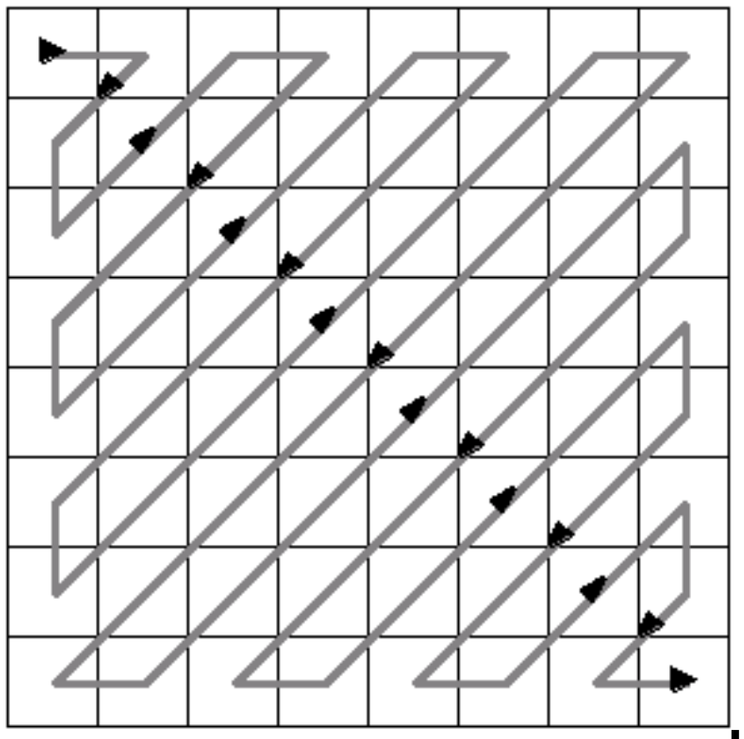
$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n+\frac{1}{2}\right)k\right] \qquad k = 0, \ldots, N-1.$$



Block-based DCT

Apply DCT on 8x8 pixel blocks

low frequencies



high frequencies

Quantize

- More coarsely for high frequencies (which also tend to have smaller values)
- Many quantized high frequency values will be zero

Encode

- Can decode with inverse dct

Filter responses

$$G = \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \downarrow v \end{matrix}$$

$u \longrightarrow$

Quantized values

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Quantization table

$$Q = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

89k

12k

Application:

“Style Transfer for Headshot Portraits” (SIGGRAPH ‘14)

Input | Example | Output

Step 2: multi-scale local transfer

Input

Example

Step 2: multi-scale local transfer

1. Construct Laplacian stacks for the input and the example

Input

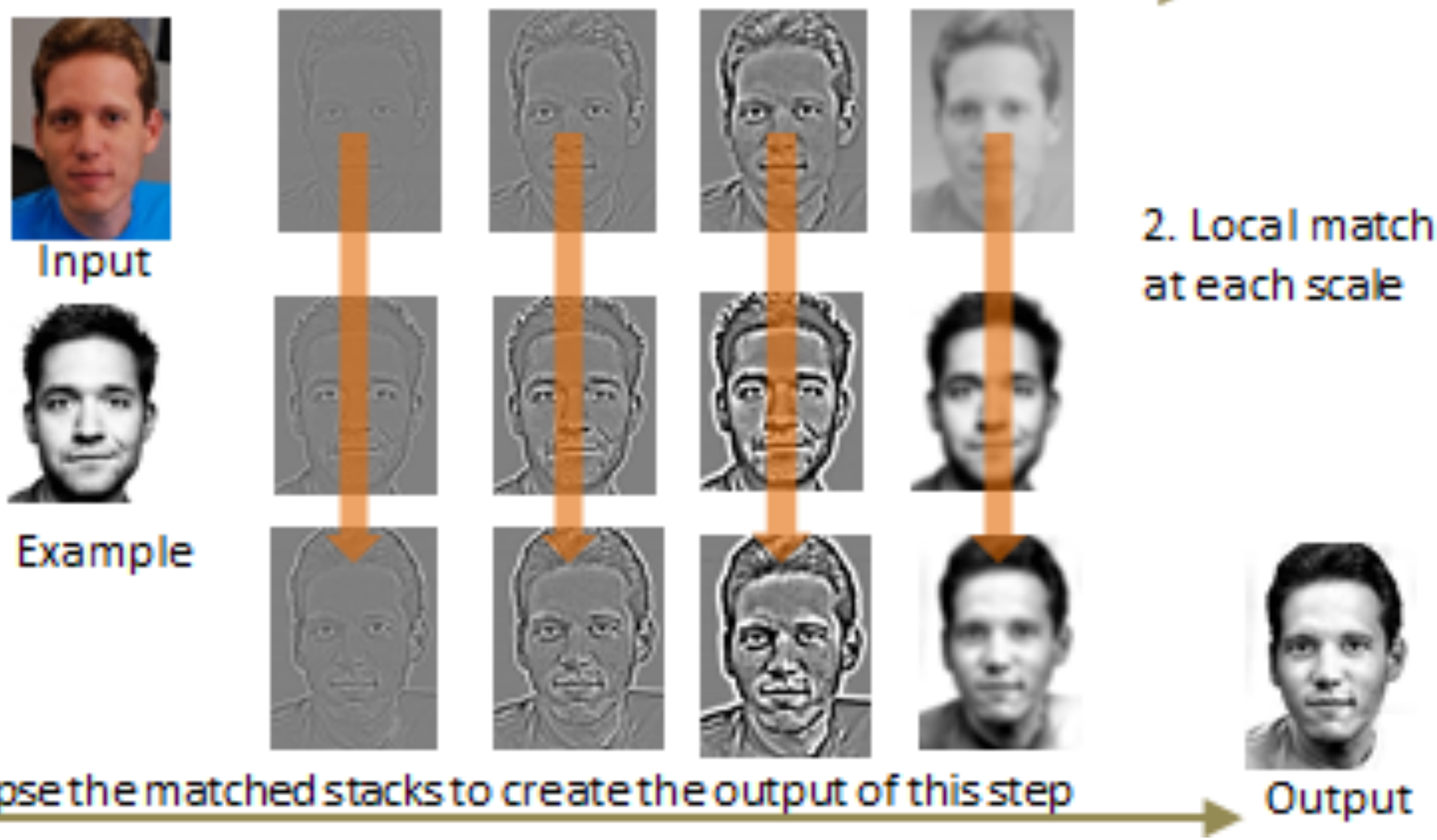
Example

Step 2: multi-scale local transfer

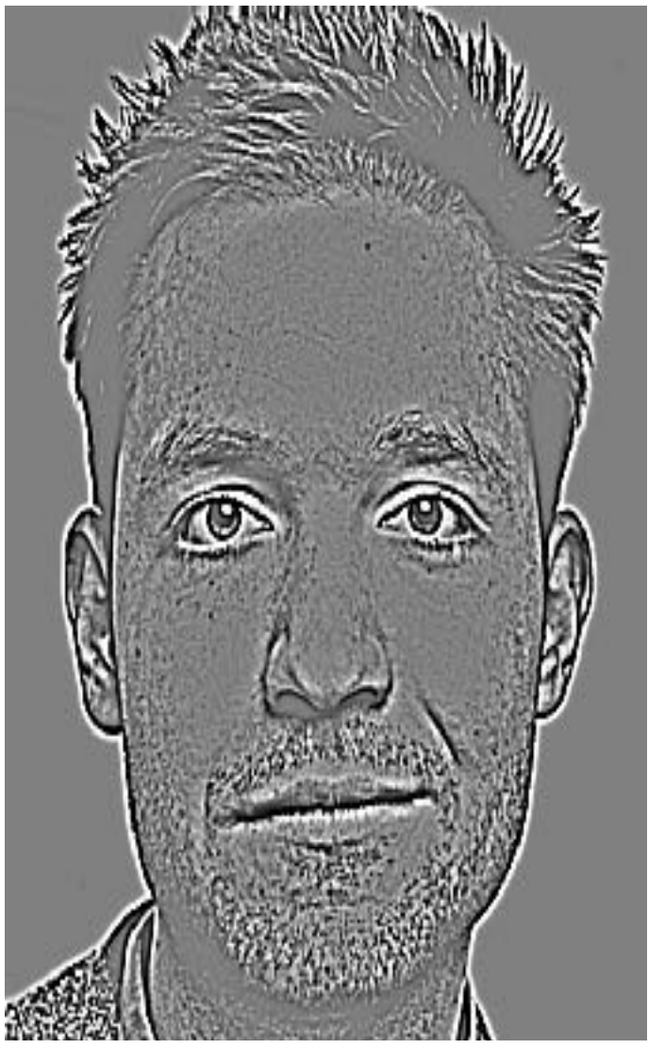
1. Construct Laplacian stacks for the input and the example

2. Local match at each scale

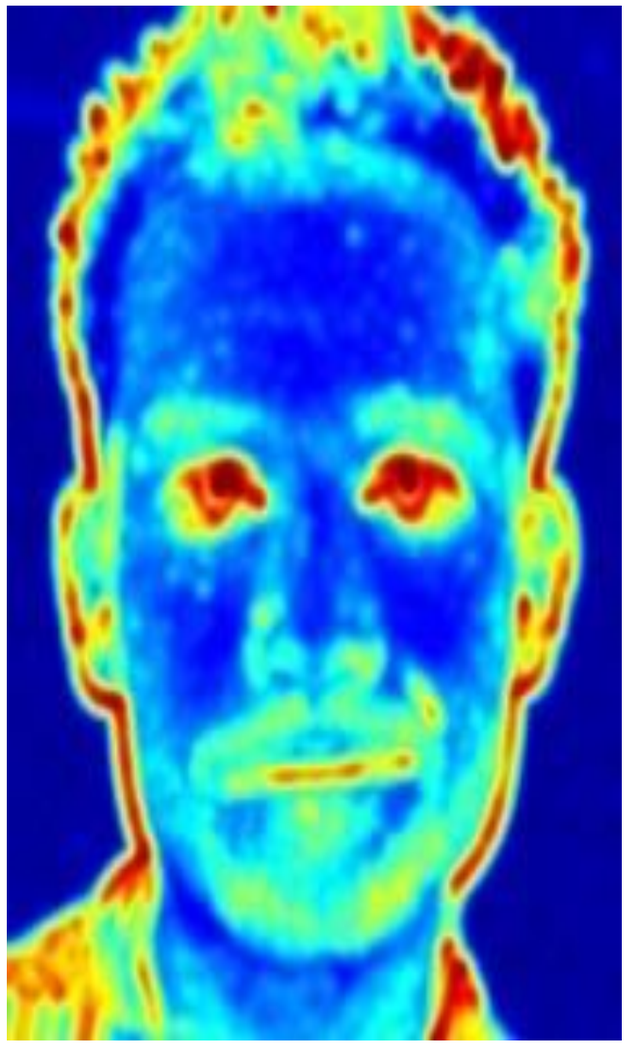
Step 2: multiscale transfer of local statistics



Local energy S

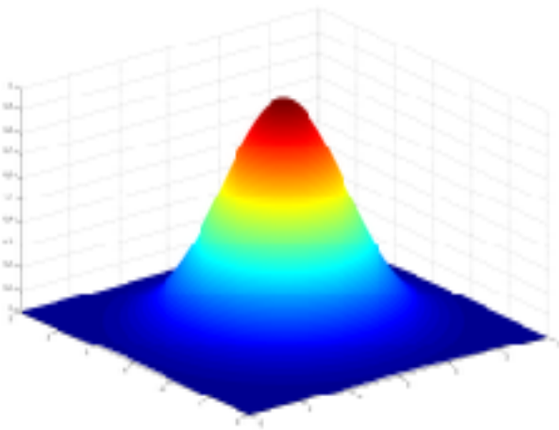


L



S

$$S = L^2 \otimes G_\ell$$



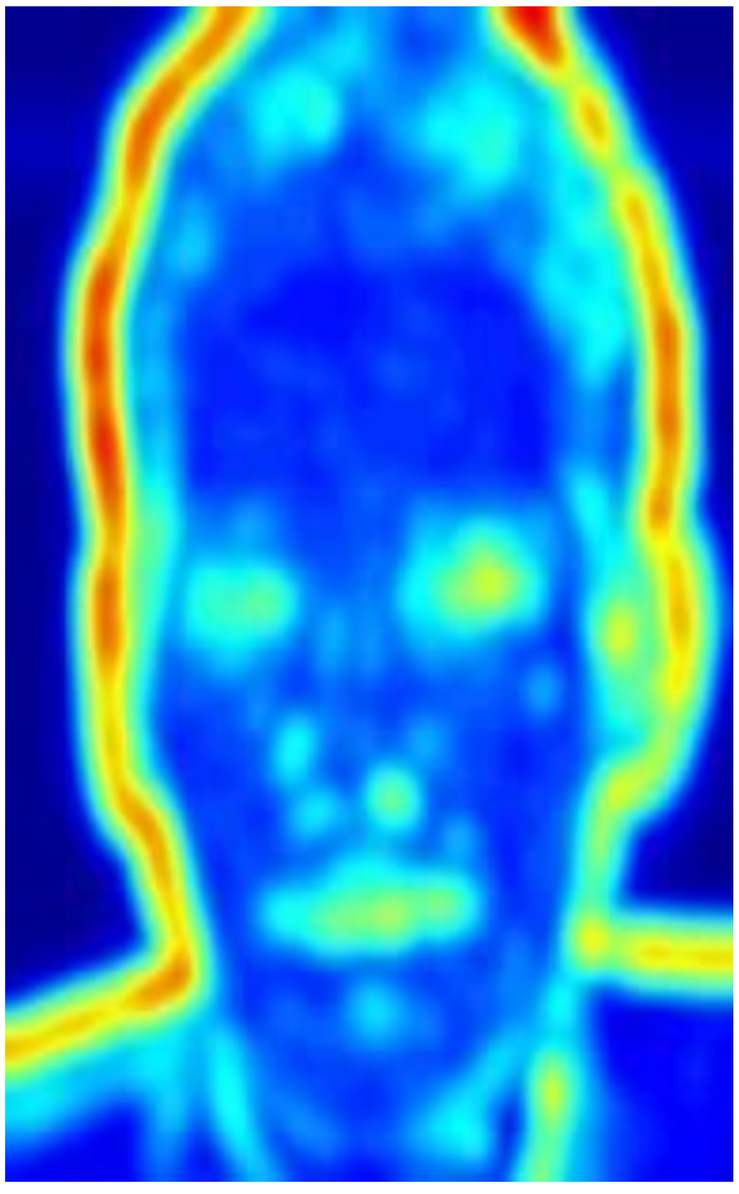
G_ℓ

Example Laplacian

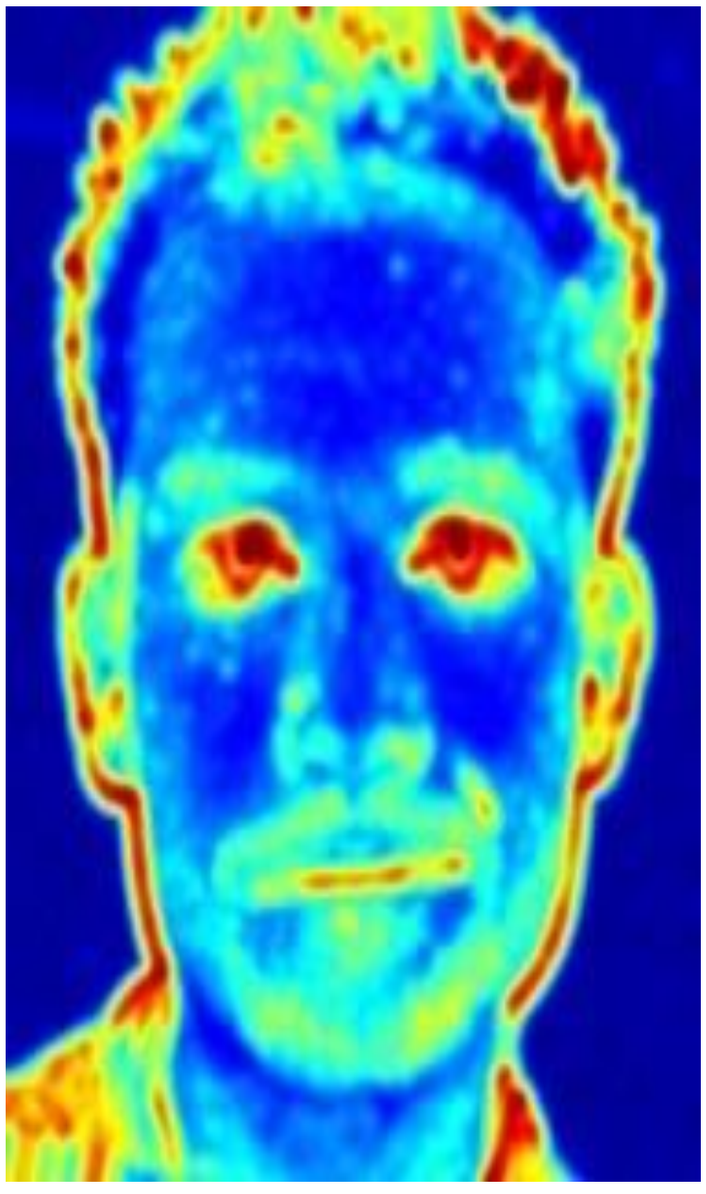
Local energy

Gaussian kernel at this scale

At each scale: match local energy



Input energy

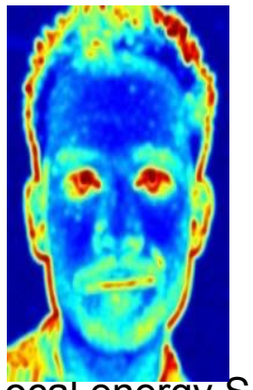


Example energy

At each scale: match local energy

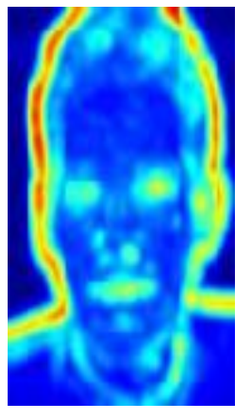
Compute
the gain map

Example Laplacian



Local energy S[E]

Input Laplacian



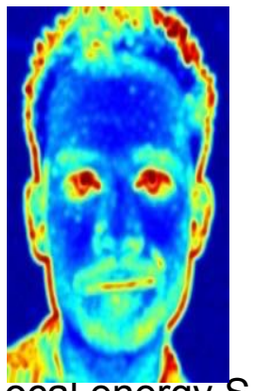
Local energy S[I]

Gain map = $S[E] / S[I]$

At each scale: match local energy

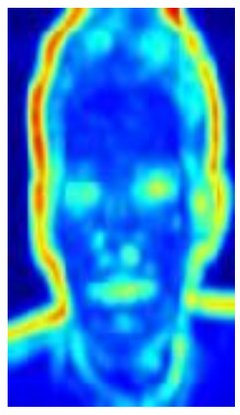
Compute
the gain map

Example Laplacian



Local energy S[E]

Input Laplacian



Local energy S[I]

Gain map = S[E] / S[I]

Modulate
the input Laplacian

Input Laplacian

×

Gain map

=

Output Laplacian

https://www.youtube.com/watch?v=Hj5lGFzlubU