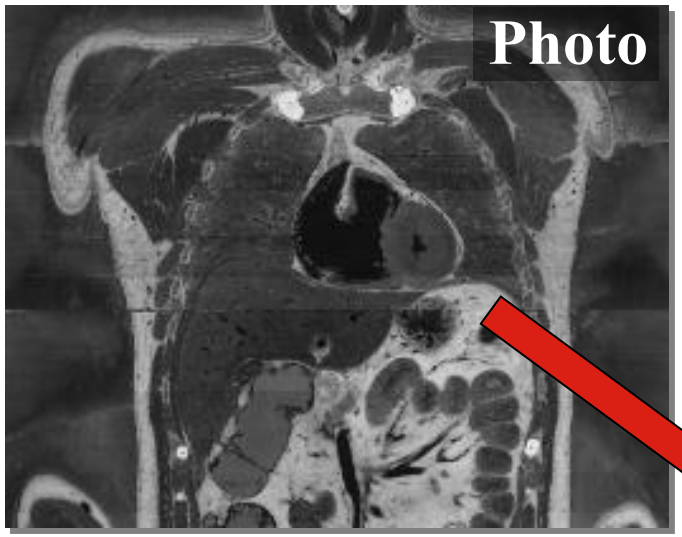
Medical Image Processing

CS 473

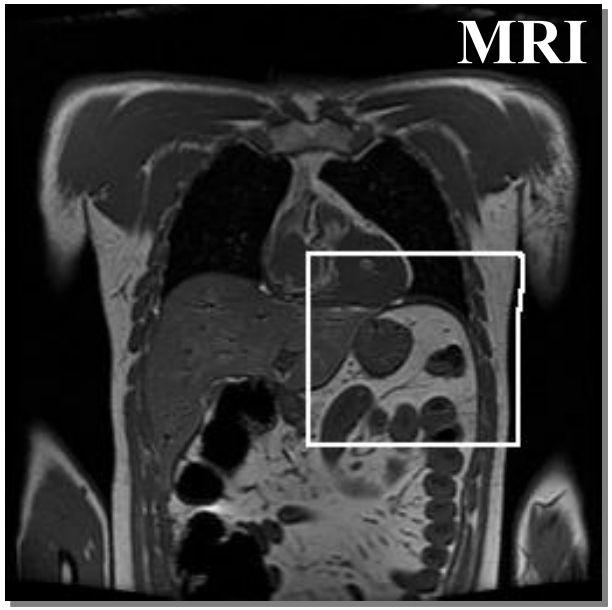
- Medical image sources (CT, MRI, PET, US)
- Image and Signal Processing
- Image Enhancement (denoising, deblurring)
- Registration (aligning images)
- Segmentation (tissue classification)
- Reconstruction

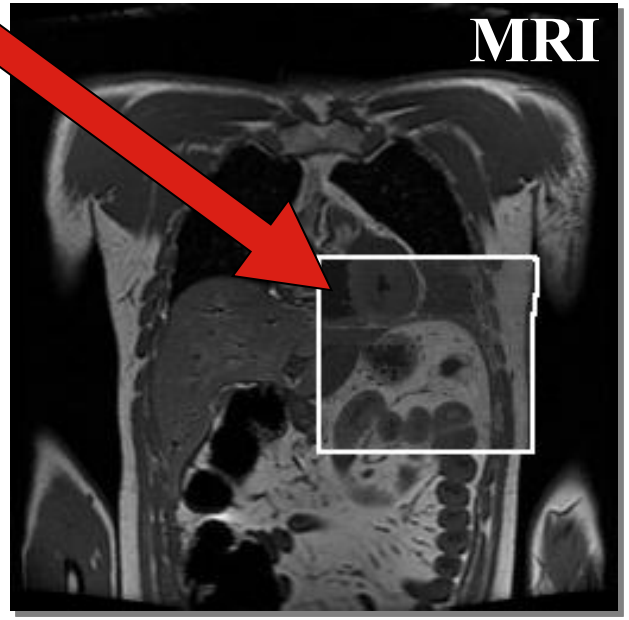
Registration

CS 473



Automatically align images, and insert the matching part from one image into the other.





Uses of Registration

CS 473

In general, when you want to compare 2 images (volumes) on a pixel-by-pixel basis

- Digital Subtraction Angiography

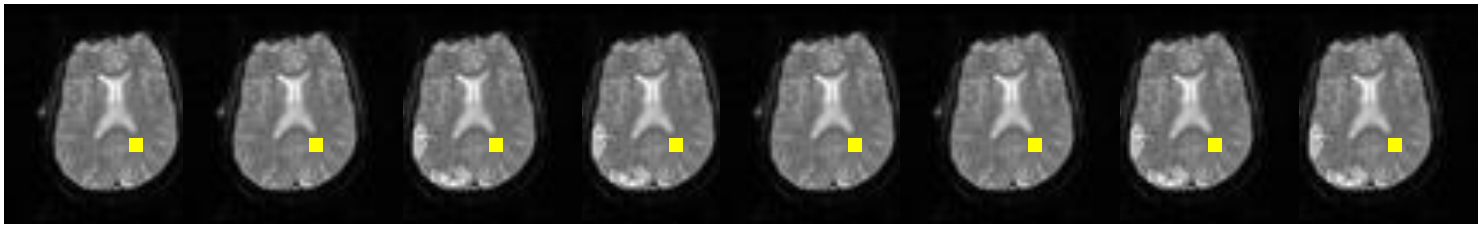
  http://www.strokecenter.org/radiology/browser2.aspx

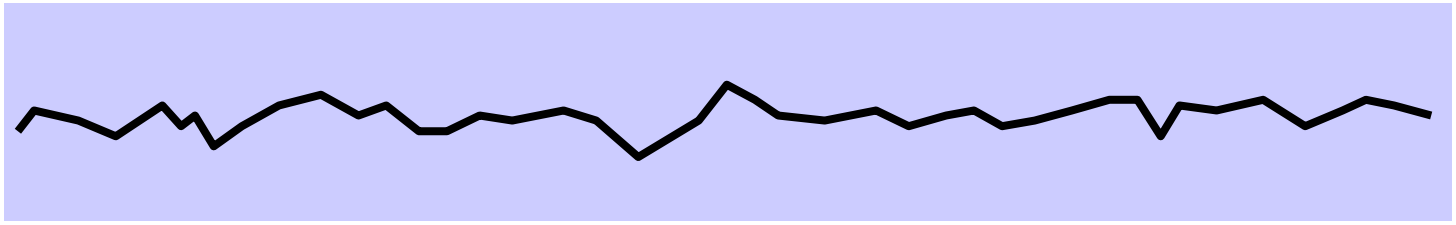
- Functional MRI
- Multimodal Fusion
- Forensic Identification

Functional MRI

CS 473

We monitor the time-course of a pixel.

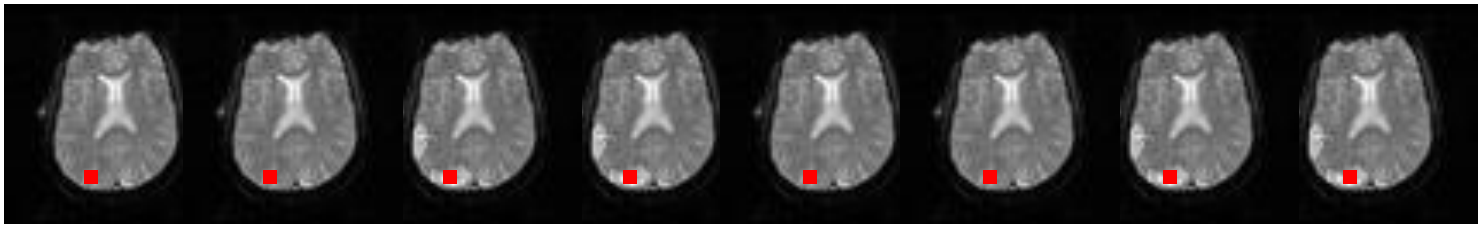


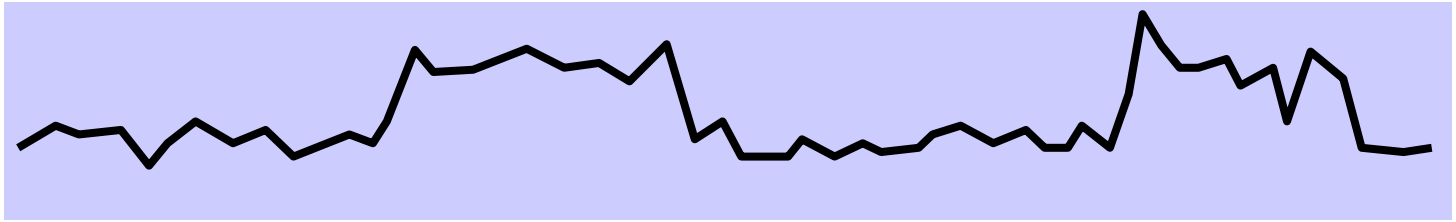


Functional MRI

CS 473

Regions of activity show signal changes.



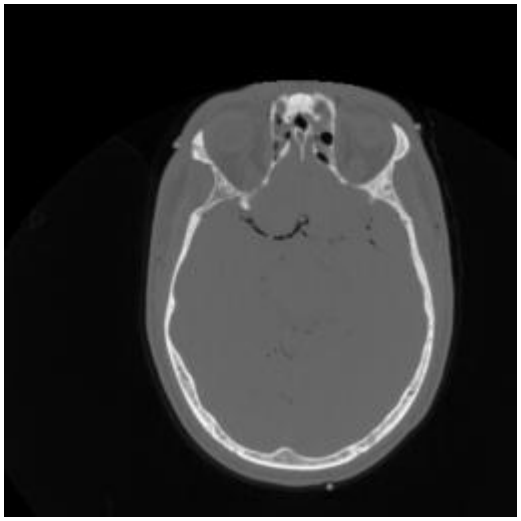


This only works if the images are properly registered.

Multimodal Fusion

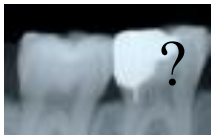
CS 473

- Some analyses require the information of multiple images.



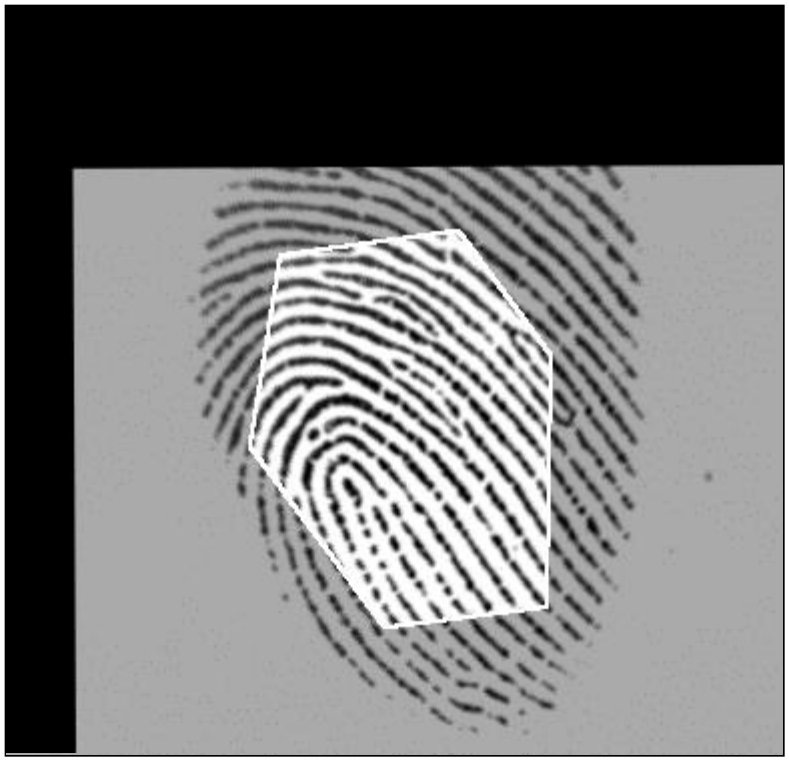
Forensic Identification

CS 473

Which is the correct match of  ?

Forensic Ident. 2

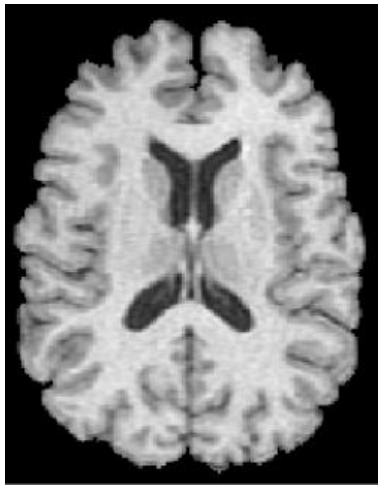
CS 473

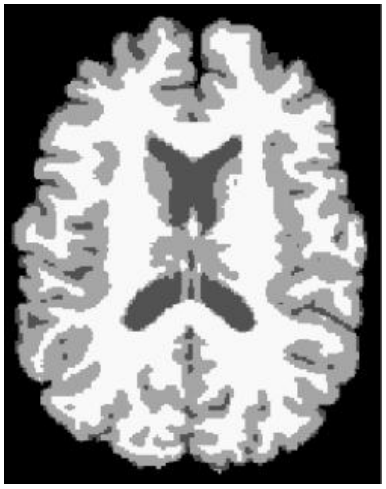


Segmentation

CS 473

Automatically classify tissue type (brain, liver, bone, muscle, etc.)





Uses of Segmentation

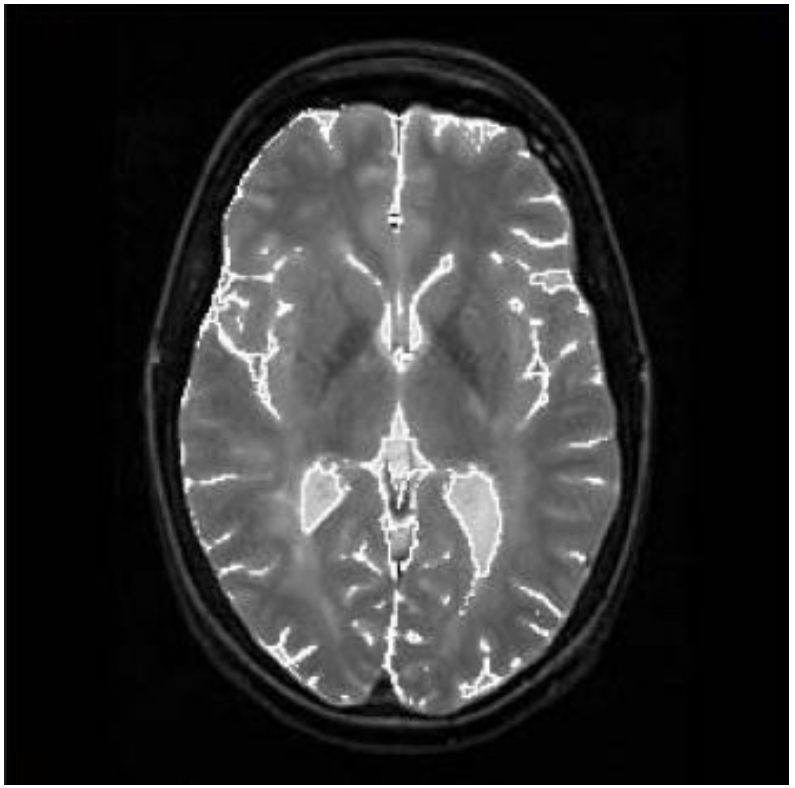
CS 473

- Volume studies (BPF)
- Rendering
- Region-Of-Interest (ROI) processing
- Shape analysis (Tim Lee, moles)

Volume Studies

CS 473

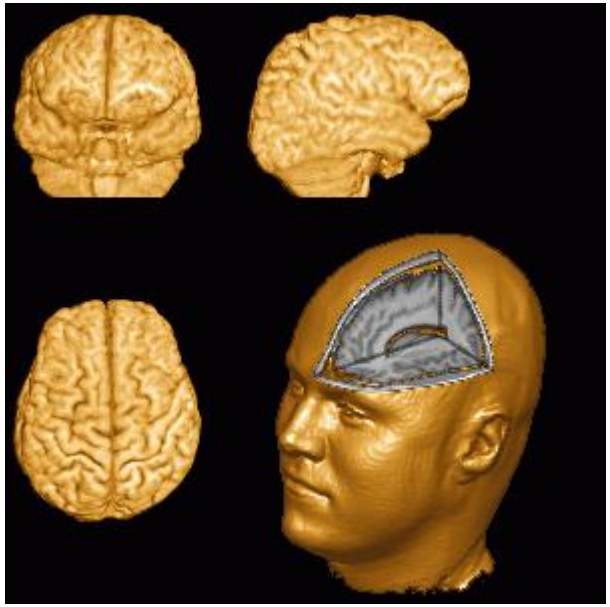
Brain Parenchymal Fraction (fraction of brain volume that is not cerebrospinal fluid) changes as degenerative diseases progress, such as Multiple Sclerosis, Alzheimers, and Parkinson's disease.

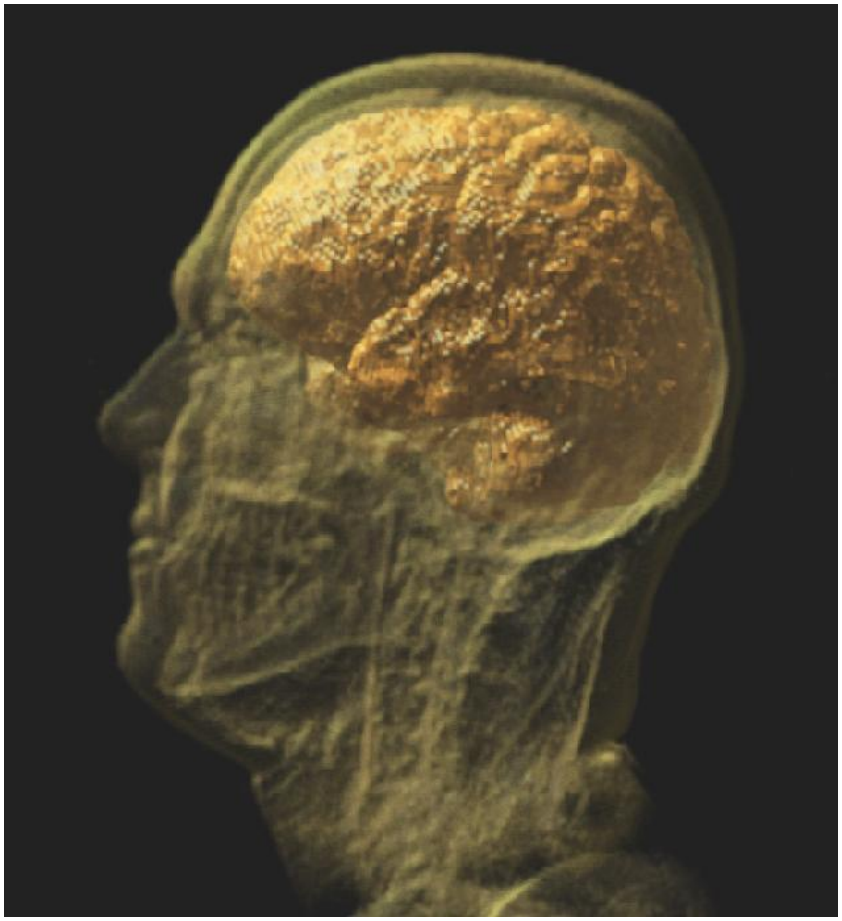


Segmented CSF

Rendering

CS 473





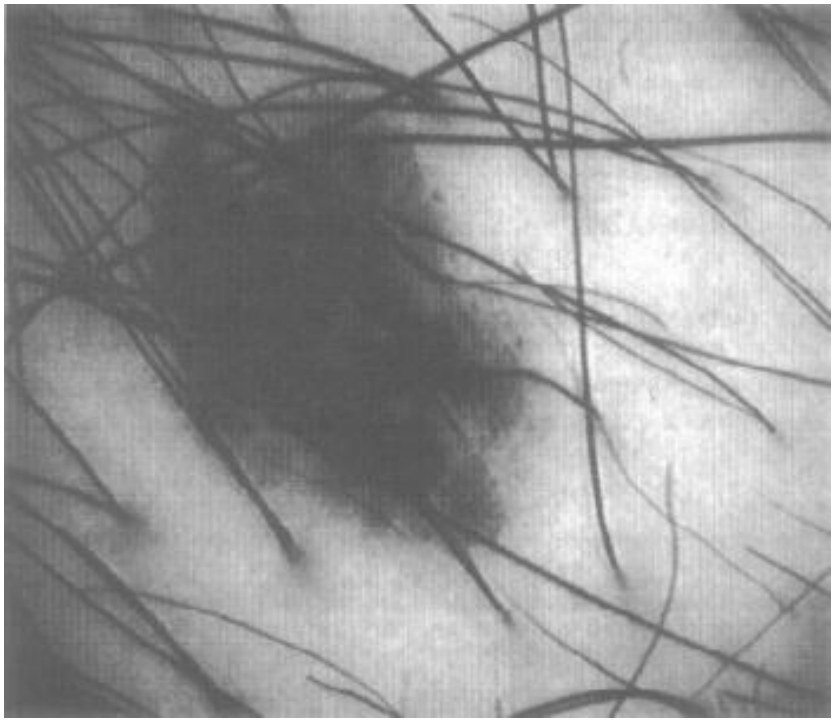
ROI Processing

CS 473

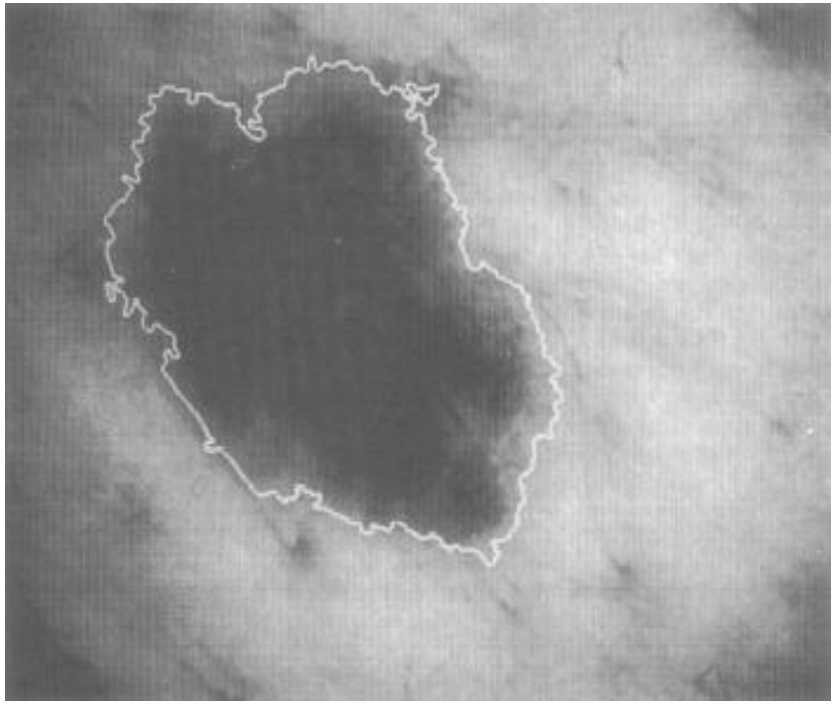
When looking for lung nodules, don't bother looking outside the lungs.

Shape Analysis

CS 473

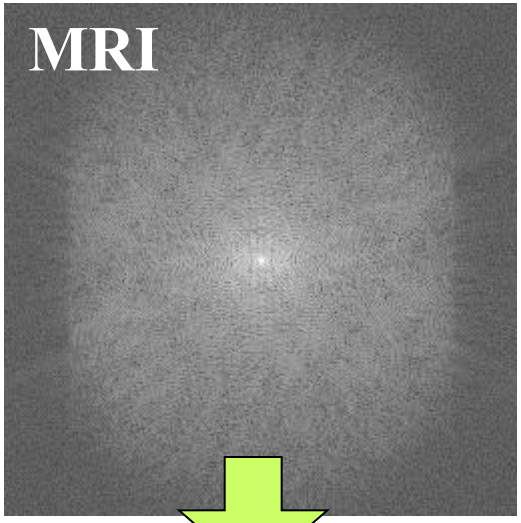


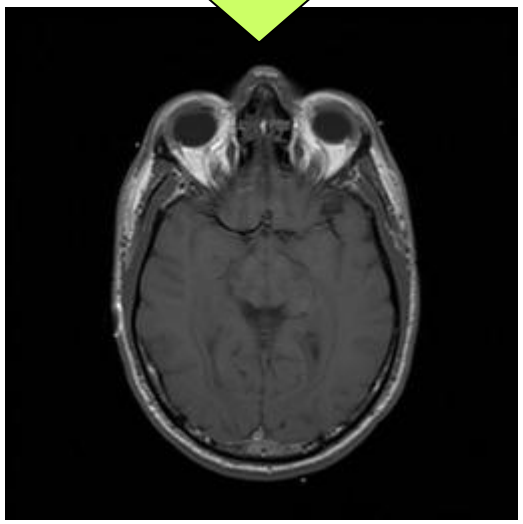
Mole border delineation
to detect skin cancer.

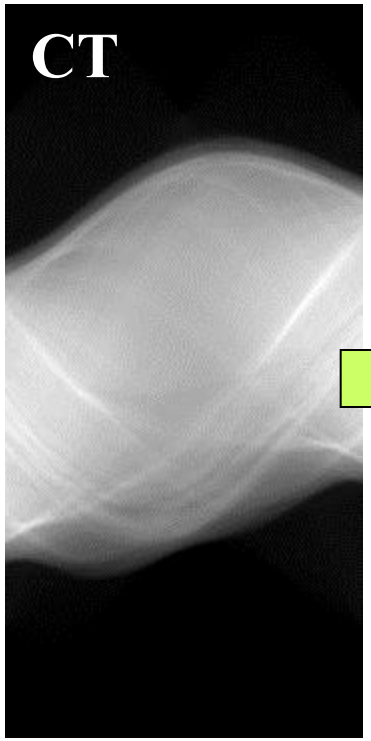


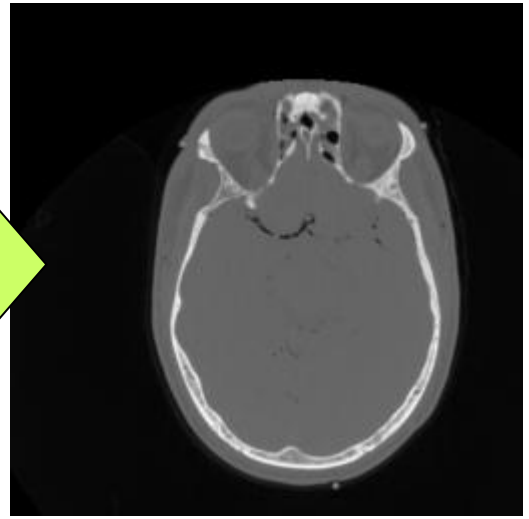
Reconstruction

CS 473









Images in Matlab

L02

Goal: To look at how images are stored, especially in Matlab.

Storing Digital Images

Images are just arrays of numbers

- 2D, 3D or higher.
- colour (3 images, one for each colour channel, Red, Green, Blue ... "R,G,B").

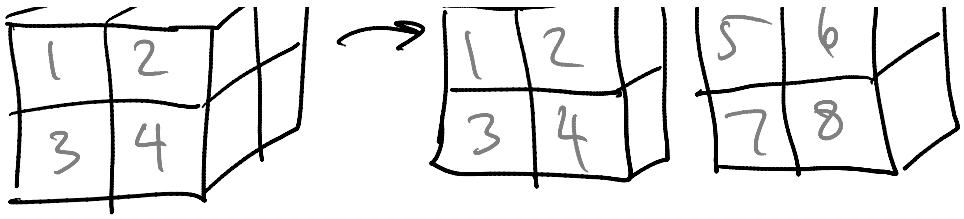
Each element of the array is called a "pixel", which comes from "picture element". For a 3D array (representing a volume), the elements are called voxels.

On disk, or in memory, the data is stored in a 1D array, so we need a convention to convert high-dim. arrays to 1D arrays

A common method:

Storing a volume is simply a matter of storing many consecutive 2D images (slices), one image after the next.

eg.

eg. 

Hence, to read a volume into memory...

```
for z=1 to Z
    for y=1 to Y
        for x=1 to X
            f(x,y,z) ⇐ read value
        next x
    next y
next z
```

Colour

We will deal almost entirely with graylevel images in this course (one value per pixel/voxel).

However, some applications save graylevel images as colour images with 3 identical colour channels.

Be aware of this, and make sure you aren't inadvertently using colour data.

Matlab Coordinates

Something that makes coordinats awkward is the fact that Matlab indexes its arrays using (row, col) format (like one would when specifying an element of a matrix).

eg $f = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$ element (2,3) is row 2, column 3 $\Rightarrow$ 6

But if we think of f as an image in the xy-plane using (x,y) coordinates, then (2,3) gives us 5.

Moreover, Matlab's indexing is base-1 ... the 1st element in an array has index 1. In C and C++ (among others), indexing starts at 0.

We can think of these differences as a coordinate system transformation.

Coordinate System Transformations

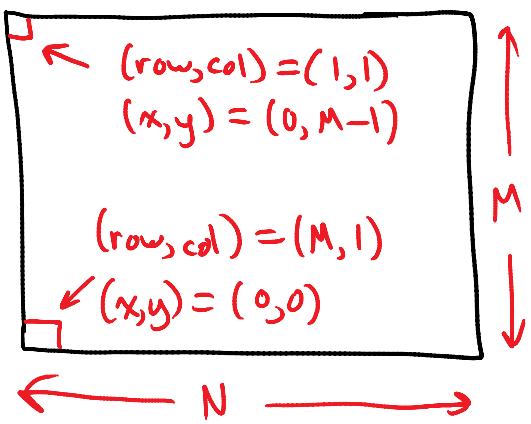
It's not hard to see that we can convert between

$$(row, col) \longleftrightarrow (x, y).$$

Suppose our image matrix has M rows and N cols.

$$\begin{cases} x = col - 1 \\ y = M - row \end{cases}$$

$$\begin{cases} row = M - y \\ col = x + 1 \end{cases}$$




N

L03

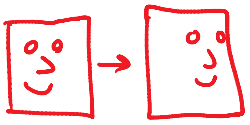
Spatial Transformations

Goal: To learn how to represent affine transformations in an efficient way.

Images (and volumes) can be thought of as fields (or functions) of space. We will talk a lot about transformations of these spatial domains. Some examples are: translation, rotation, scaling, etc.

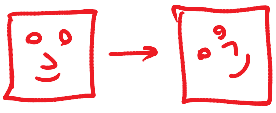
Translation

$$\begin{cases} u = x + a \\ v = y + b \end{cases}$$



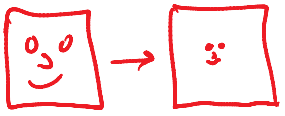
Rotation

$$\begin{cases} u = x\cos\theta + y\sin\theta \\ v = -x\sin\theta + y\cos\theta \end{cases}$$



Scaling

$$\begin{cases} u = s_1 x \\ v = s_2 y \end{cases}$$



Homogeneous Coordinates

Notice above that rotation & scaling can each be written as matrix-vector products.

$$\vec{u} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \vec{x}$$

Rotation

$$\vec{u} = \begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix} \vec{x}$$

Scaling

So called "affine" transforms.

However, translation cannot. But we can embed translations into a matrix-vector product if we bump up the dimension by 1.

i.e. $\begin{bmatrix} \quad \\ \quad \end{bmatrix} \begin{bmatrix} x \\ \quad \end{bmatrix} = \begin{bmatrix} x+a \\ \quad \end{bmatrix}$

we bump up the dimension by 1.

ie. $\begin{bmatrix} x \\ y \\ z \end{bmatrix} \Longleftrightarrow \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \Longleftrightarrow \begin{bmatrix} rx \\ ry \\ rz \\ r \end{bmatrix}$ for $r \neq 0$

We call this augmented coordinate system "Homogeneous coordinates".

Consider the transform

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$ in non-homogeneous coords

$$\Rightarrow \begin{bmatrix} u \\ v \\ r \end{bmatrix} = \begin{bmatrix} a & b & e \\ c & d & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + e \\ cx + dy + f \\ 1 \end{bmatrix}$$

Example:

Rotate by 30°, then translate by $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

Rotation matrix is $R_{30^\circ} = \begin{bmatrix} \sqrt{3}/2 & -\frac{1}{2} \\ \frac{1}{2} & \sqrt{3}/2 \end{bmatrix}$

Thus, $P = \begin{bmatrix} \sqrt{3}/2 & -\frac{1}{2} & 2 \\ \frac{1}{2} & \sqrt{3}/2 & 1 \\ 0 & 0 & 1 \end{bmatrix}$

Composition of Transforms

Why do we care about representing translation as a matrix-vector product?

Now we can do this

a matrix vector product.

But we can do this...

→ Two (or more) affine transforms can be combined into one affine transform by simple matrix multiplication.

Consider the composite transform:

1) $P_1 \equiv$ rotate by 30°

2) $P_2 \equiv$ translate by $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$

$$P_1 = \begin{bmatrix} \cos 30 & \sin 30 & 0 \\ -\sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad P_2 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Composite transform is

$$P_2 P_1 = \begin{bmatrix} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos 30 & \sin 30 & 0 \\ -\sin 30 & \cos 30 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos 30 & \sin 30 & 2 \\ -\sin 30 & \cos 30 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

Now suppose we add a 3rd transform to the sequence:

3) $P_3 \equiv$ rotate by 45°, then translate by $\begin{bmatrix} -1 \\ 1 \end{bmatrix}$

$$P_3 = \begin{bmatrix} \cos 45 & \sin 45 & -1 \\ -\sin 45 & \cos 45 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

The transform resulting from the sequential composition of all 3 can be written

The transform resulting from the sequential composition of all 3 can be written

$$P = P_3 P_2 P_1$$

[Pause... Matlab exercise L03_transforms.m]

→ your end result should be

P_all_three =

0.2588	0.9659	1.1213
-0.9659	0.2588	0.2929
0	0	1.0000

Decomposing Affine Transforms

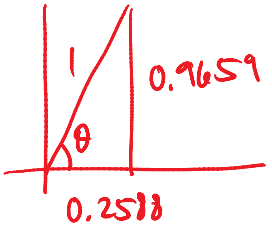
We can decompose a transformation matrix into a single rotation & translation (& scale, skew, etc.)

In the above case

$$\begin{bmatrix} \cos\theta & \sin\theta & t_1 \\ -\sin\theta & \cos\theta & t_2 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0.2588 & 0.9659 & 1.1213 \\ -0.9659 & 0.2588 & 0.2929 \\ 0 & 0 & 1.0000 \end{bmatrix}$$

$\Rightarrow \cos\theta = 0.2588$

$\sin\theta = 0.9659$



$\Rightarrow \theta = 75°$

$t_1 = 1.1213$

$t_2 = -1.7071$

Thus, $P_3 P_2 P_1$ is equivalent to

- Rotation by 75°, followed by
- Translation of $\begin{bmatrix} 1.1213 \\ -1.7071 \end{bmatrix}$

It is often helpful to think of these transforms

It is often helpful to think of these transforms in terms of their affine part & translation part.

$$P = \left[\begin{array}{c|c} A & T \\ \hline 0 & 1 \end{array}\right] \qquad A \in \mathbb{R}^{d \times d} \qquad T \in \mathbb{R}^{d \times 1}$$

$d = 2$ for 2D
$d = 3$ for 3D
etc.

Combining Simple Transforms

Translations:

$$\left[\begin{array}{c|c} I & T_1 \\ \hline 0 & 1 \end{array}\right]\left[\begin{array}{c|c} I & T_2 \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} I & T_1+T_2 \\ \hline 0 & 1 \end{array}\right]$$

Thus, combining translations simply adds the translations.
Inverse of transl. T_1 is $-T_1$

Rotations:

$$\left[\begin{array}{c|c} R_{\theta_1} & 0 \\ \hline 0 & 1 \end{array}\right]\left[\begin{array}{c|c} R_{\theta_2} & 0 \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R_{\theta_1}R_{\theta_2} & 0 \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R_{\theta_1+\theta_2} & 0 \\ \hline 0 & 1 \end{array}\right]$$

Thus, combining rotations simply adds their angles.
Inverse of rotation by θ is rotation by $-\theta$.

Scaling:

$$\left[\begin{array}{c|c} S_1 & 0 \\ \hline 0 & 1 \end{array}\right]\left[\begin{array}{c|c} S_2 & 0 \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} S_1 S_2 & 0 \\ \hline 0 & 1 \end{array}\right]$$

Thus, combining scales simply multiplies them.
Inverse of scaling by S is scaling by $\frac{1}{S}$

... multiplies them.

Inverse of scaling by S is scaling by $\frac{1}{S}$.

CAUTION: Be aware of the complexities that arise when you are combining different kinds of simple transforms.

Example: transl. by $\begin{bmatrix} 2 \\ 1 \end{bmatrix}$, rotate by 30°, transl. by $\begin{bmatrix} -2 \\ -1 \end{bmatrix}$.

$$P = \left[\begin{array}{c|c} I & -T \\ \hline 0 & 1 \end{array}\right]\left[\begin{array}{c|c} R & 0 \\ \hline 0 & 1 \end{array}\right]\left[\begin{array}{c|c} I & T \\ \hline 0 & 1 \end{array}\right] = \left[\begin{array}{c|c} R & RT-T \\ \hline 0 & 1 \end{array}\right]$$

Notice that the translations do not undo each other because there is a rotation between them.

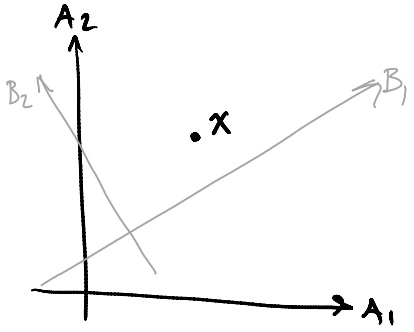
Converting Between Coordinate Systems

Suppose we have a point x and two different coordinate systems (CS) labeled A and B.

$x_A \equiv$ coords of x w.r.t. CS A

$x_B \equiv$ coords of x w.r.t. CS B

eg.



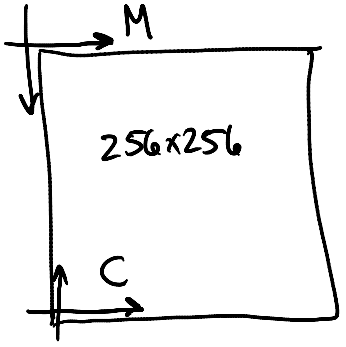
We want to be able to convert x_A to x_B (and vice versa).

$$P_{A \to B} x_A = x_B$$

Magic Rule: To get the transform $P_{A \to B}$, we can simply use the same transform that moves the B axes onto the A axes, using A's coord. system.

That is, suppose the B axes is an object in A's coordinate space. Compose the transform that will overlay the B axes on the A axes.

Example: Cartesian to Matlab index coords.



To convert from x_c (Cartesian) to x_m (Matlab index), we move M axes onto C.

1) shift M by $\begin{bmatrix} 1 \\ -256 \end{bmatrix}$ $\longrightarrow$ $P_1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & -256 \\ 0 & 0 & 1 \end{bmatrix}$

2) reverse M's vertical axis $\longrightarrow$ $P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

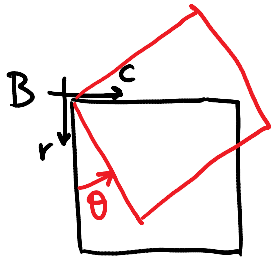
3) swap M's axes $\longrightarrow$ $P_3 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$P_{C \to M} = P_3 P_2 P_1 = \begin{bmatrix} 0 & -1 & 256 \\ 1 & 0 & 1 \\ 0 & 0 & 1 \end{bmatrix}$$

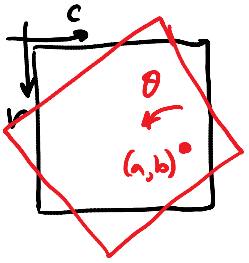
Check: $\begin{bmatrix} 0 \\ 0 \end{bmatrix}_C \longrightarrow \begin{bmatrix} 256 \\ 1 \end{bmatrix}_M$ $\quad$ $\begin{bmatrix} 255 \\ 255 \end{bmatrix}_C \longrightarrow \begin{bmatrix} 1 \\ 256 \end{bmatrix}$

Converting Transforms Between Coordinate Systems

Suppose you want to rotate an image around a specified point with coords (a,b). You already know how to build a rotation matrix, but it rotates about the origin.



Action of R_θ



Rotation of θ about pt. (a,b)

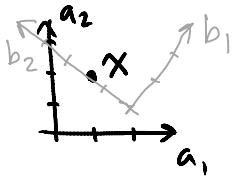
The Theory

Given two coord systems A and B, and a transform matrix M_A, expressed w.r.t. CS A, find the equivalent transform matrix w.r.t. CS B, M_B.

Let P_{AB} be the change-of-basis matrix that converts from A coords to B coords.

i.e. $x_B = P_{AB}\, x_A$

eg.



$x_A = [1 \;\; 2]^T$

$x_B = [0 \;\; \sqrt{2}]^T$

Note that $P_{AB}^{-1} = P_{BA}$

Thus, $x_B = P_{AB}\, x_A$ and $x_A = P_{BA}\, x_B$.

The transform M_A is applied to vectors in CS A.

i.e.

$$y_A = M_A x_A$$

$$P_{BA}\, y_B = M_A\, P_{BA}\, x_B \Rightarrow y_B = P_{AB}\, M_A\, P_{BA}\, x_B$$

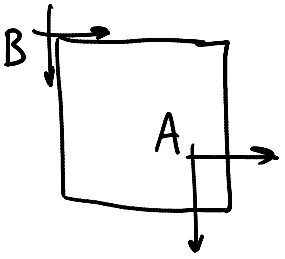
$$y_B = M_B\, x_B$$

Hence, $\boxed{M_B = P_{AB}\, M_A\, P_{BA} \left(= P_{BA}^{-1} M_A P_{BA}\right)}$

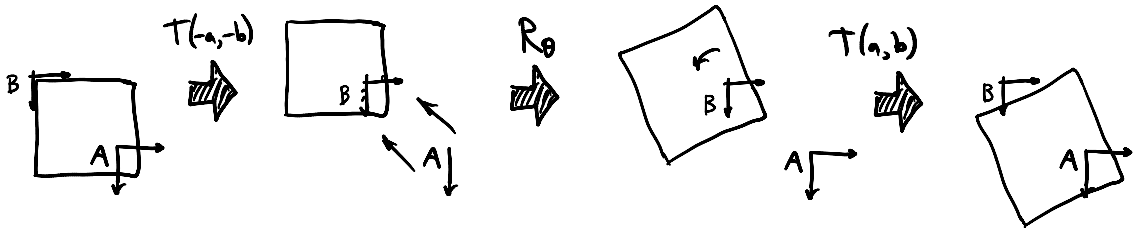
Back to our rotation example:

$M_A = R_\theta$ (rotation about A's origin)

$P_{AB} = T(a,b)$ (translation by (a,b))



Hence, $M_B = T(a,b)\, R_\theta\, T(-a,-b)$



Fourier Series

L04

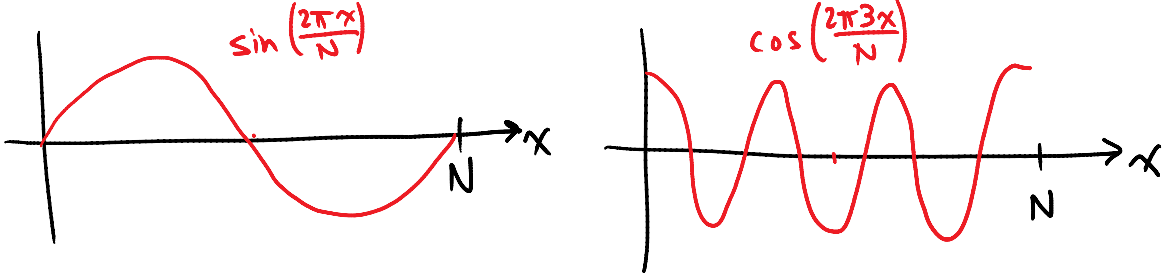
Goal: To introduce the Fourier transform & review complex numbers. The Fourier transform is fundamental not only to image processing, but it also plays a special role in medical imaging.

Consider the trigonometric functions of n,

$$\sin\left(\frac{2\pi k x}{N}\right) \quad \text{and} \quad \cos\left(\frac{2\pi k x}{N}\right) \qquad k \in \mathbb{Z}$$

They repeat when x increases by $\frac{N}{k}$.

Or, they repeat k times in $0 \leq \leq N$.



Theorem: Suppose f is a "nice" N-periodic function. There exist coefficients a_k & b_k such that

$$f(x) = a_0 + \sum_{k=1}^{\infty}\left[a_k \cos\left(\frac{2\pi k x}{N}\right) + b_k \sin\left(\frac{2\pi k x}{N}\right)\right]$$

This is known as a Fourier Series.

In practice, we approximate f with a truncated Fourier Series,

$$f(x) = a_0 + \sum_{k=1}^{m}\left[a_k \cos\left(\frac{2\pi k x}{N}\right) + b_k \sin\left(\frac{2\pi k x}{N}\right)\right] \quad (*)$$

Instead of treating the a's and b's separately, we can use the more sophisticated and compact complex notation,

$$f(x) = \sum_{k=-m}^{m} C_k \left(\cos\left(\frac{2\pi k x}{N}\right) + i \sin\left(\frac{2\pi k x}{N}\right) \right)$$

Notice the sum is now from $-m$ to m. Here's why. If $f(x) \in \mathbb{R}$, then we need to make sure all the imaginary parts cancel out.

$$f(x) = a_0 + \sum_{k=1}^{m} \left[C_k \cos\tfrac{2\pi k x}{N} + i C_k \sin\tfrac{2\pi k x}{N} + C_{-k} \cos\tfrac{-2\pi k x}{N} + i C_{-k} \sin\tfrac{-2\pi k x}{N} \right]$$

$$= a_0 + \sum_{k=1}^{m} \left[(C_k + C_{-k}) \cos\tfrac{2\pi k x}{N} + i (C_k - C_{-k}) \sin\tfrac{2\pi k x}{N} \right]$$

Comparing to ⊛ above...

$$\begin{cases} a_k = C_k + C_{-k} & ① \\ b_k = i(C_k - C_{-k}) & ② \end{cases}$$

$① + i②$ $\Rightarrow$ $a_k + i b_k = 2C_{-k}$ $\Rightarrow$ $C_{-k} = \dfrac{a_k + i b_k}{2}$

$① - i②$ $\Rightarrow$ $a_k - i b_k = 2C_k$ $\Rightarrow$ $C_k = \dfrac{a_k - i b_k}{2}$

Notice, then, that $C_k = \overline{C_{-k}}$ (complex conjugates)

Task: Do the associated quiz in D2L.

Fourier Transform

LO5

Goal: To introduce some of the basic methods for the Fourier transform.

Fourier Transform (Continuous Domain)

Let $f: \mathbb{R} \to \mathbb{R}$ (i.e. $f(x)$)

Its Fourier transform (FT) is defined as

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

$F(\omega)$ is a frequency decomposition... a different way of representing the same signal f.

The FT is invertible

$$f(x) = \mathcal{F}^{-1}\{F(\omega)\}(x) = \int_{-\infty}^{\infty} F(\omega) e^{2\pi i \omega x} d\omega$$

Discrete Fourier Transform (DFT)

When both $f(x)$ and $F(\omega)$ are sampled,

i.e. $f_n, n = 0, \ldots, N-1 \Rightarrow [f_0 \; f_1 \; f_2 \; \cdots \; f_{N-1}]$

$F_k, k = 0, \ldots, N-1 \Rightarrow [F_0 \; F_1 \; F_2 \; \cdots \; F_{N-1}]$

then we have the Discrete Fourier Transform (DFT) and its Inverse:

$$F_k = \sum_{n=0}^{N-1} f_n e^{\frac{-2\pi i n k}{N}} \qquad f_n = \frac{1}{N} \sum_{k=0}^{N-1} F_k e^{\frac{2\pi i n k}{N}}$$

$k = 0, \ldots, N-1$ $\qquad$ $n = 0, \ldots, N-1$

Fast Fourier Transform (FFT)

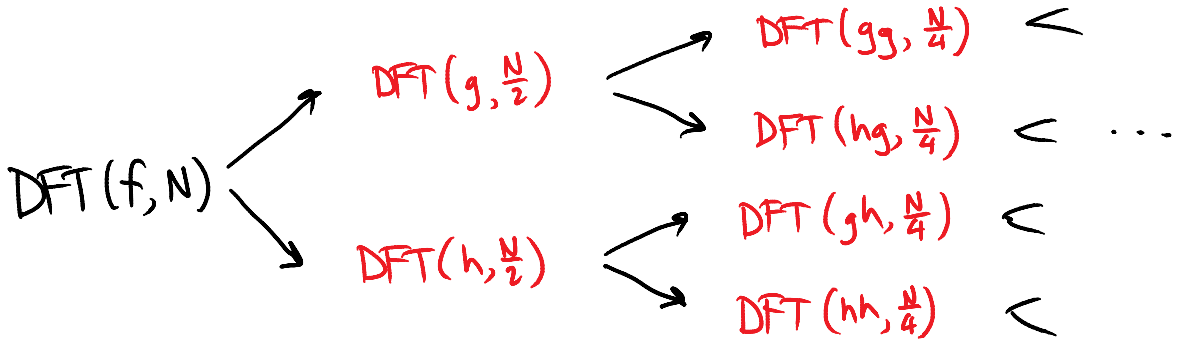
Computing all N Fourier coefs. directly using the formula for F_k above would take $O(N^2)$ flops.

A divide-and-conquer method turns out to be faster. It's called the Fast Fourier Transform.

Briefly, it decomposes the length-N DFT into two $\frac{N}{2}$-length DFTs. This process recursively decomposes the DFT until it arrives at arrays of length 1 ... those are easy. The whole process takes $O(N \log N)$ flops for a 1D array of length N.

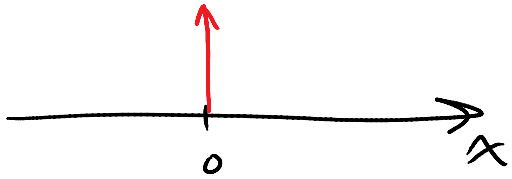
(what about for 2D or 3D?)

Given f_n, $n = 0, \ldots, N-1$, we recombine the elements to get g_n and h_n, $n = 0, \ldots, \frac{N}{2} - 1$.



Def^n: Dirac delta function

$$\delta(x) = \begin{cases} \infty & \text{if } x = 1 \\ 0 & \text{otherwise} \end{cases}$$



s.t. $\int_{-\infty}^{\infty} \delta(x)\,dx = 1$

and $\int_{-\infty}^{\infty} \delta(x-c)\,f(x)\,dx = f(c)$

Theorem: $\int_{-\infty}^{\infty} e^{2\pi i w x}\, dx = \delta(w)$

Pf: Not in THIS course.

We can use the Dirac delta function to prove that $\mathcal{F}$ and $\mathcal{F}^{-1}$ are inverses.

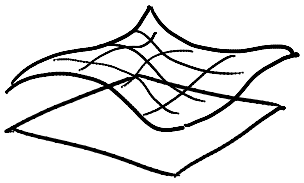
Pf: $\mathcal{F}^{-1}\{\mathcal{F}\{f(x)\}(w)\}(s)$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x) e^{-2\pi i w x}\, dx\, e^{2\pi i w s}\, dw$$

$$= \int_{-\infty}^{\infty} f(x)\left[\int_{-\infty}^{\infty} e^{2\pi i w (s-x)}\, dw\right] dx$$

$$= \int_{-\infty}^{\infty} f(x)\, \delta(s-x)\, dx = f(s)$$

2D Fourier Transform



Consider the function $f: \mathbb{R}^2 \to \mathbb{R}$

The 2D FT is defined as

$$F(w,\lambda) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y) e^{-2\pi i (wx+\lambda y)}\, dx\, dy$$

$(w,\lambda)\cdot(x,y)$

Notice that the FT is separable:

$$F(w,\lambda) = \iint f(x,y) e^{-2\pi i w x}\, dx\, e^{-2\pi i \lambda y}\, dy$$

$$= \int \underbrace{\left[\int f(x,y)\, e^{-2\pi i \omega_x x}\, dx \right]}_{\text{Apply FT along } x\text{-dim}} e^{-2\pi i \omega_y y}\, dy$$

Apply FT along y-dim

Thus, an N-D FT can be done using 1D FTs along each of the N dimensions.

TASK: Check out the associated Matlab script. There are some lines for you to fill in, and a number of important concepts to learn.

Convolution

L06

Goal: To define convolution, see what it does, and find out how it can be done using the Fourier transform.

Def'n: The convolution between two functions $f(x)$ and $g(x)$ is an integral of the form

$$(f*g)(x) = \int_{-\infty}^{\infty} f(\tau)\, g(x-\tau)\, d\tau$$

For discrete signals f_n and g_n,

$$(f*g)_n = \sum_{k=0}^{N-1} f_k\, g_{n-k} \qquad n = 0, \ldots, N-1$$

Theorem: (continuous-domain version)

Let $f(x)$ and $g(x)$ be functions, and let

$$F(\omega) = \mathcal{F}\{f(x)\}(\omega) \quad \text{and} \quad G(\omega) = \mathcal{F}\{g(x)\}(\omega)$$

be their Fourier transforms. Then

$$\mathcal{F}\{(f*g)(x)\}(\omega) = \mathcal{F}\{f(x)\}(\omega) \cdot \mathcal{F}\{g(x)\}(\omega)$$

$$= F(\omega)\, G(\omega)$$

Pf:

$$\mathcal{F}\{(f*g)(x)\}(\omega) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(\tau)\, g(x-\tau)\, d\tau\, e^{-2\pi i \omega x}\, dx$$

Change of variables: Let $y = x - \tau \Rightarrow x = y + \tau$

$dy = dx$

$$= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(\tau)\, g(y)\, d\tau\, e^{-2\pi i w(y+\tau)}\, dy$$

$$= \int_{-\infty}^{\infty} f(\tau)\, e^{-2\pi i w\tau}\, d\tau \int_{-\infty}^{\infty} g(y)\, e^{-2\pi i w y}\, dy$$

$$= F(w)\; G(w)$$

∎

Note: One can just as easily prove the theorem

$$\mathcal{F}^{-1}\{(F*G)(w)\}(x) = f(x)\, g(x)$$

∴ convolution in one domain is equivalent to element-wise multiplication in the other domain.

$$\mathcal{F}\{f*g\}(w) = F(w)\, G(w)$$

Conv. in spatial domain ⇔ mult. in freq. domain

$$\mathcal{F}^{-1}\{F*G\}(x) = f(x)\, g(x)$$

Conv. in freq. domain ⇔ mult. in spatial domain.

Theorem: (discrete-domain version)

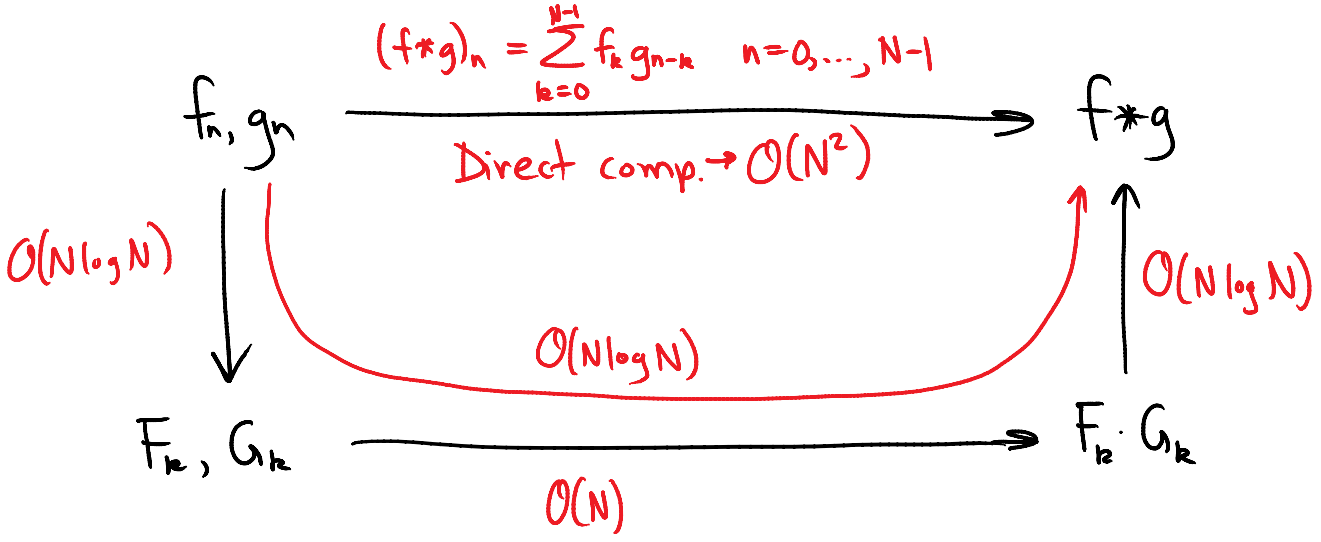
Let f_n and g_n, $n = 0, \ldots, N-1$, be two discrete functions (signals).

$$\mathcal{F}\{(f*g)_n\}_k = F_k\, G_k$$

TASK: Check out the Matlab script
L06_Convolution.m

Question: Why bother using the FT to compute convolution? Sure, it may be cool, but isn't it more work?

Consider the # of flops (floating-point operations) to compute (f*g) using the 2 methods.



Thus, as N gets large, using the DFT is more efficient.
What about convolving NxN images, f_{mn} and g_{mn}, where $m=0,\dots,N-1$ and $n=0,\dots,N-1$?

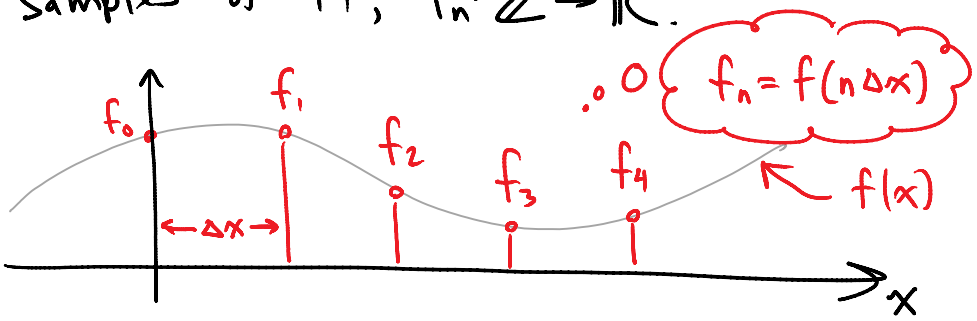
→ $O(N^2 \log N)$ flops

Sampling Theory

L07

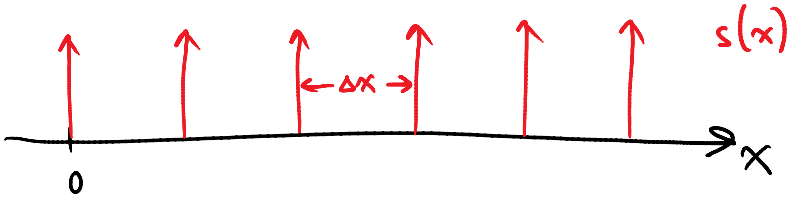
Goal: Images can be thought of as sampled versions of 2D functions. We want to develop a framework for understanding the relationship between a function and its samples.

Consider a continuous-domain function $f: \mathbb{R} \to \mathbb{R}$, and samples of it, $f_n: \mathbb{Z} \to \mathbb{R}$.



To sample f, we multiply by the Shah function. (or "comb")

$$s(x) = \sum_{n \in \mathbb{Z}} \delta(x - n\Delta x)$$



Then, the sampled version of f can be written

$$\bar{f}(x) = f(x)\, s(x) = f(x) \sum_{n \in \mathbb{Z}} \delta(x - n\Delta x)$$

Consider the FT of $\bar{f}(x)$.

$$\mathcal{F}\{f(x)\}(\omega) = \mathcal{F}\{s(x)f(x)\}(\omega) = (S*F)(\omega)$$

What is $S(\omega)$?

$$S(\omega) = \mathcal{F}\{s(x)\}(\omega)$$

$$= \int_{-\infty}^{\infty} s(x)\, e^{-2\pi i \omega x}\, dx$$

$$= \int_{-\infty}^{\infty} \sum_n \delta(x - n\Delta x)\, e^{-2\pi i \omega x}\, dx$$

$$= \sum_n \int_{-\infty}^{\infty} \delta(x - n\Delta x)\, e^{-2\pi i \omega x}\, dx \quad \text{(after swapping } \Sigma \text{ and } \int\text{)}$$

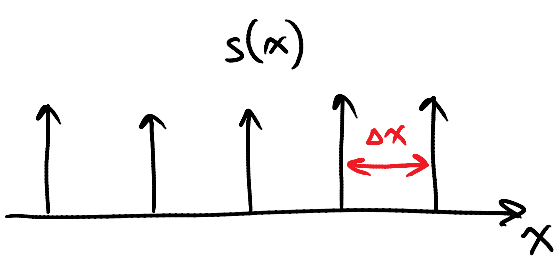
$$= \sum_{n \in \mathbb{Z}} e^{-2\pi i \omega (n\Delta x)}$$

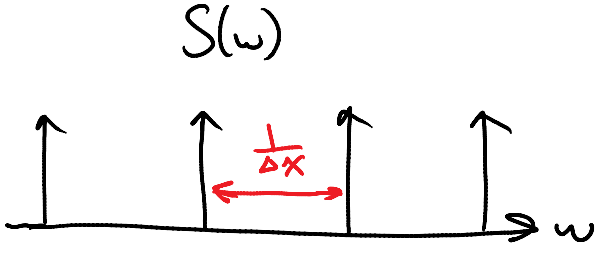
$= 1$ when $\omega \Delta x = k \in \mathbb{Z} \Rightarrow \omega = \frac{k}{\Delta x}, k \in \mathbb{Z}$

POOF

$$= \sum_{k \in \mathbb{Z}} \delta\left(\omega - \frac{k}{\Delta x}\right)$$

Thus, the FT of the Shah function is also a Shah function, but with different spacing.

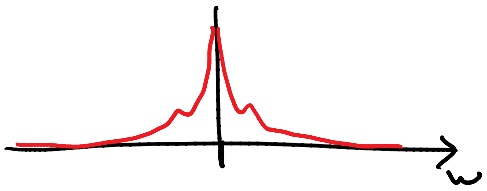


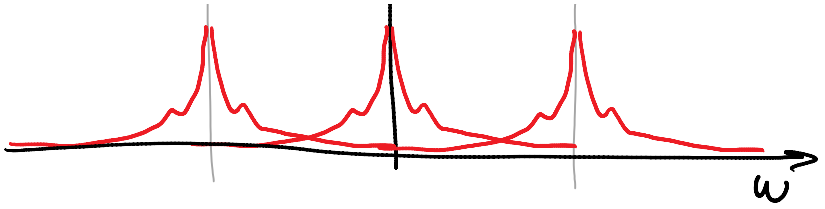


Ok, back to $\mathcal{F}\{f(x)s(x)\}(\omega) = (S*F)(\omega)$

$F(\omega) = \mathcal{F}\{f(x)\}(\omega)$

$(S*F)(\omega)$





So, sampling f gives us a periodic FT.

Likewise, a similar derivation can be used to show that a periodic f yields a discrete FT.

f(x)	F(w)
sampled	periodic
periodic	sampled
periodic & sampled	periodic & sampled

TASK: Do the short quiz in D2L.

Aliasing

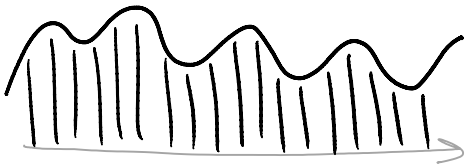
L08

Goal: To observe and understand the issues when sampling an image.

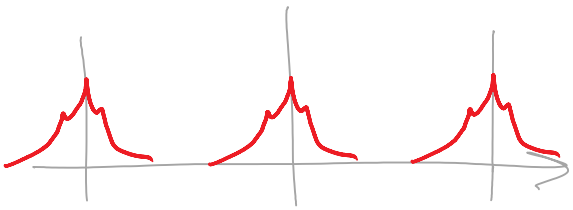
Demo: Moiré video

What's going on here?

Suppose we have this signal:

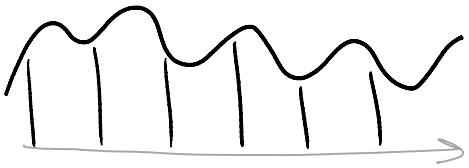


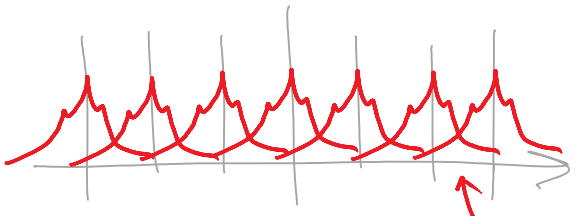
f(x)



F(ω)

Suppose we sub-sample it by a factor of 3:



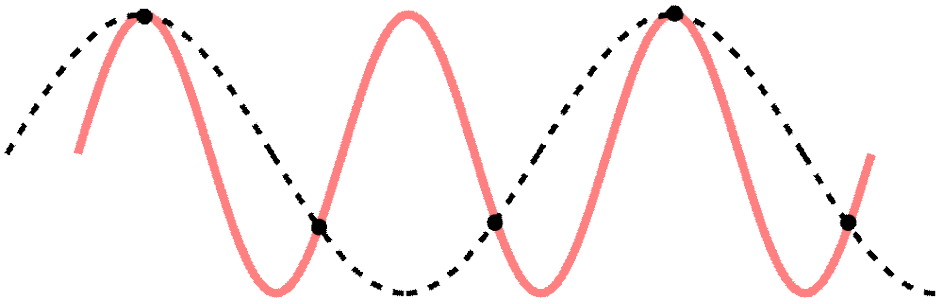


Lots of overlap.

The overlap between the copies of the Fourier repetitions is called aliasing. The coefs. are added & can't be separated to get the correct FT.

Nyquist-Shannon Sampling Theorem

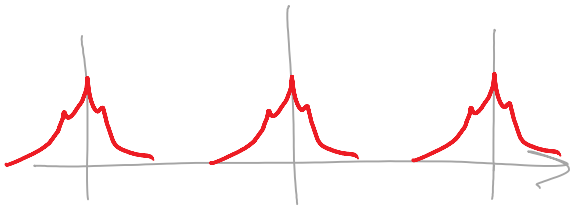
To avoid aliasing, your sampling freq. must be at least double the highest freq. in your signal.



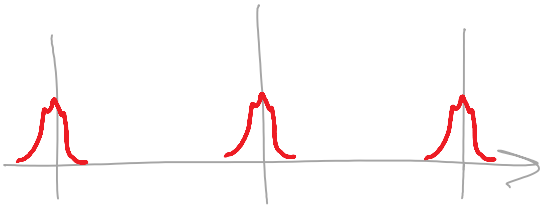
http://en.wikipedia.org/wiki/Nyquist-Shannon_sampling_theorem

Filtering

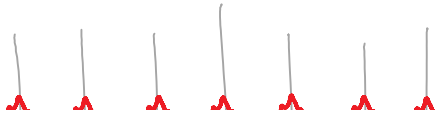
If we want to subsample an image, but avoid Moiré artifacts, we can filter the image first. Filtering is multiplying the Fourier coefs. to adjust the relative contributions of the different frequencies. In our case, we need to dampen the higher frequencies so they don't overlap so much.



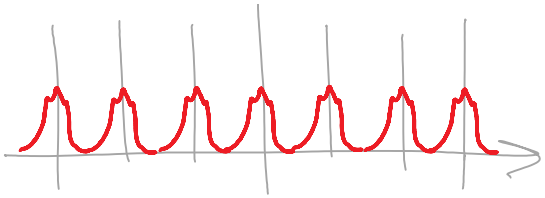
Original FT



Filtered to dampen higher frequencies



FT of subsampled



FT of subsampled image ... not much overlap.

Demo: Moiré script.

Strobe Effect

(a related phenomenon)

http://youtu.be/ltMPMz37VPk

Properties of the FT

L09

Goal: To survey some of the properties of the Fourier transform that are useful in image processing (beyond the convolution theorem).

But first, a quick look at what a 2D Fourier coefficient means.

Demo: L09_wavefront.m

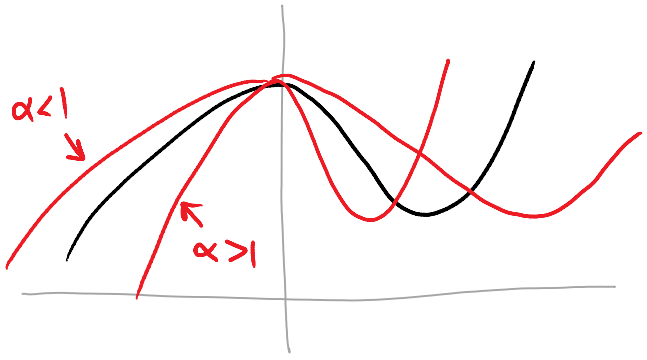
Scale Property

What happens to F(w) if we scale f?

Let $\bar{f}(x) = f(\alpha x)$

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

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



Change of variables: $y = \alpha x \Rightarrow dy = \alpha dx$

Note: Lots of Fourier proofs involve a change of variables.

$$\frac{dy}{\alpha} = dx$$

$$\bar{F}(\omega) = \frac{1}{\alpha} \int_{-\infty}^{\infty} f(u) e^{-2\pi i \frac{\omega u}{\alpha}} du = \frac{1}{\alpha} F\left(\frac{\omega}{\alpha}\right)$$

$$\bar{F}(\omega) = \frac{1}{\alpha}\int_{-\infty}^{\infty} f(y)\, e^{-2\pi i \frac{\omega y}{\alpha}}\, dy = \frac{1}{\alpha} F\left(\frac{\omega}{\alpha}\right)$$

Thus, $\mathcal{F}\{f(\alpha x)\}(\omega) = \frac{1}{\alpha}\mathcal{F}\{f(x)\}\left(\frac{\omega}{\alpha}\right)$

So, as $f(x)$ stretches, $F(\omega)$ contracts & shrinks (and vice versa).

Example: Consider the Shah function with spacing Δx, $s(x)$. Recall the spacing of $S(\omega) = \mathcal{F}\{s(x)\}(\omega)$.

Exercise:

What is the spacing of $s(\alpha x)$, $\alpha \neq 0$?

What is the spacing of $S(\alpha x)$?

There is also a scale property for the DFT, though the interpretation is different. Suppose f_n and F_k are each an array of N numbers. What do those numbers mean? What points in the spatial/frequency domain do those values correspond to?

$$F_k = \sum_{n=0}^{N-1} f_n\, e^{-2\pi i \frac{nk}{N}}$$

One period... $0 \le \frac{nk}{N} < N$ (*)

Notice we can introduce an arbitrary spatial scale, $0 \le \frac{n}{N} < 1 \Rightarrow 0 \le \frac{nL}{N} < L$

$\Rightarrow 0 \le x < L$

We compensate by scaling the frequency variable,

(*) $\Rightarrow 0 \le \frac{nL}{N}\frac{k}{L} < N$

i.e. $x = \frac{nL}{N}$ for $n = 0, \ldots, N-1$

$w = \frac{k}{L}$ for $k = 0, \ldots, N-1$

Thus, if $x \in \left\{ \frac{0L}{N}, \frac{L}{N}, \frac{2L}{N}, \ldots, \frac{(N-1)L}{N} \right\}$ ← L ← "Field of view" (FOV) or period

then $w \in \left\{ \frac{0}{L}, \frac{1}{L}, \frac{2}{L}, \ldots, \frac{N-1}{L} \right\}$ ← $\frac{N}{L}$

In general, $(FOV_x)(\Delta w) = (FOV_w)(\Delta x) = 1$

$(FOV_x)(\Delta w) \to 1$ $\qquad$ $(FOV_w)(\Delta x) \to 1$

$(L)\left(\frac{1}{L}\right) = 1$ $\qquad$ $\left(\frac{N}{L}\right)\left(\frac{L}{N}\right) \to 1$

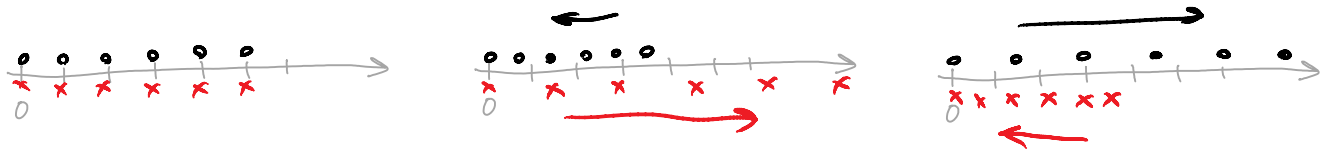
Equivalently, we could use

$x \in \left\{ \frac{0}{L}, \frac{1}{L}, \frac{2}{L}, \ldots, \frac{N-1}{L} \right\}$

$w \in \left\{ \frac{0L}{N}, \frac{L}{N}, \frac{2L}{N}, \ldots, \frac{(N-1)L}{N} \right\}$

Exercise: Verify that this sampling obeys the discrete scaling rule.

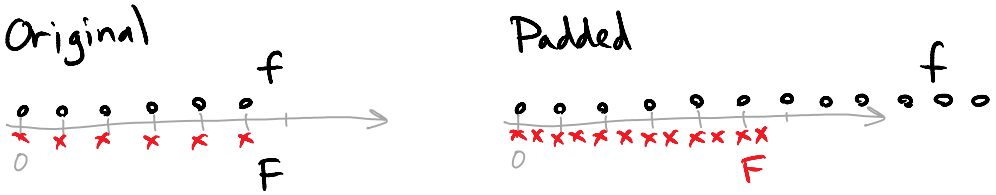
Visual Examples: For 6-vectors ($N=6$)...



Side-Effect: Scaling using padding & cropping, & the DFT

Consider this situation...

- Pad f (add samples), then apply DFT.
  Where will the samples be in the freq. domain?



In other words, padding in one domain can be thought of as subsampling in the other.

Demo: Image scaling using the DFT.

Shift Property

Consider f_n and its DFT F_n, $n,k=0,\ldots,N-1$.

Let $g_n = f_{n-d}$ (a shifted version of f_n).

$$G_k = \sum_{n=0}^{N-1} g_n e^{-2\pi i \frac{nk}{N}} \qquad k=0,\ldots,N-1$$

$$= \sum^{N-1} f_{n-d}\, e^{-2\pi i \frac{nk}{N}} \qquad \text{Change of vars: Let } m=n-d$$

$$= \sum_{n=0}^{N-1} f_{n-d}\, e^{-2\pi i \frac{nk}{N}}$$

Change of vars: Let $m = n-d$

$n = m+d$

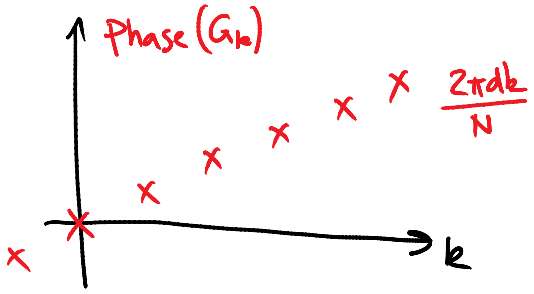
$$= \sum_{m=-d}^{N-1-d} f_m\, e^{-2\pi i \frac{(m+d)k}{N}}$$

Since f & e are N-periodic

$$= \sum_{m=0}^{N-1} f_m\, e^{-2\pi i \frac{mk}{N}}\, e^{-2\pi i \frac{dk}{N}}$$

$$= e^{-2\pi i \frac{dk}{N}} F_k$$

That is, shifting f by d samples is equivalent to multiplying F by a "phase ramp".



Slope is dictated by d.

d larger ⇒ steeper

"Phase" because it only influences the phase of the Fourier coefs.

This property is also used quite often in MRI.

Demo: Fourier Shift

Rotational Invariance

If R is a rotation matrix, then $\bar{f}(\vec{x}) \equiv f(R\vec{x})$ is a rotated version of f. What does $\mathcal{F}\{\bar{f}(\vec{x})\}(\omega)$ look like?

$\mathcal{F}\{f(\vec{x})\}(\omega)$ look like?

As it turns out

$$\mathcal{F}\{f(R\vec{x})\}(\omega) = F(R\vec{\omega})$$

We won't prove this here, but it's quite simple. Thus, rotating f is equivalent to rotating F by the same amount.

This works for the FT, and also for the DFT with a few minor caveats.

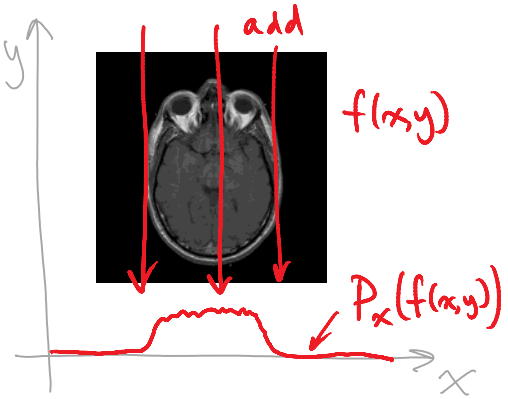
Demo: Fourier rotation

Fourier Projection Theorem

Suppose you have $f: \mathbb{C}^2 \to \mathbb{C}$, and $F(\omega, \lambda) = \mathcal{F}\{f(x,y)\}(\omega,\lambda)$.

Consider the Radon projection of f onto the x-axis,

$$\mathcal{P}_x\{f(x,y)\}(x) = \int_{-\infty}^{\infty} f(x,y)\, dy$$



For no apparent reason, let's take the 1D FT of $\mathcal{P}_x$.

$$\int_{-\infty}^{\infty} \mathcal{P}_x\{f(x,y)\}(x)\, e^{-2\pi i \omega x}\, dx = \int\int f(x,y)\, e^{-2\pi i \omega x}\, dy\, dx$$

$$\int_{-\infty}^{\infty} \mathcal{P}_x\{f(x,y)\}(x)\, e^{-2\pi i(\omega x + \lambda \cdot 0)}\, dx = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\, e^{-2\pi i(\omega x + \lambda y)}\, dy\, dx$$

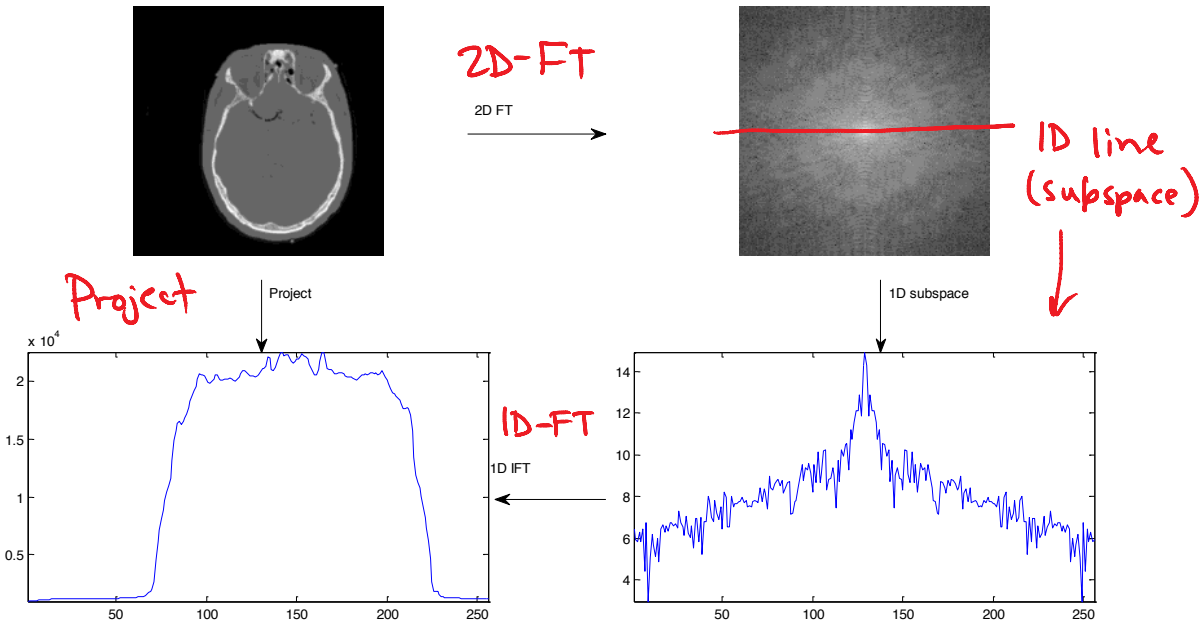
but $\lambda = 0$.

$$= \int\int f(x,y)\, e^{-2\pi i(\omega x + \lambda y)}\, dy\, dx$$

$$= F(\omega, 0)$$

Thus, the 1D-FT of the projection is the same as the 1D line of Fourier coefs taken from the 2D-FT.

In a picture...



This works for a projection along any direction, as long as the subspace in the freq. domain has the same orientation.

Interpolation & Resampling

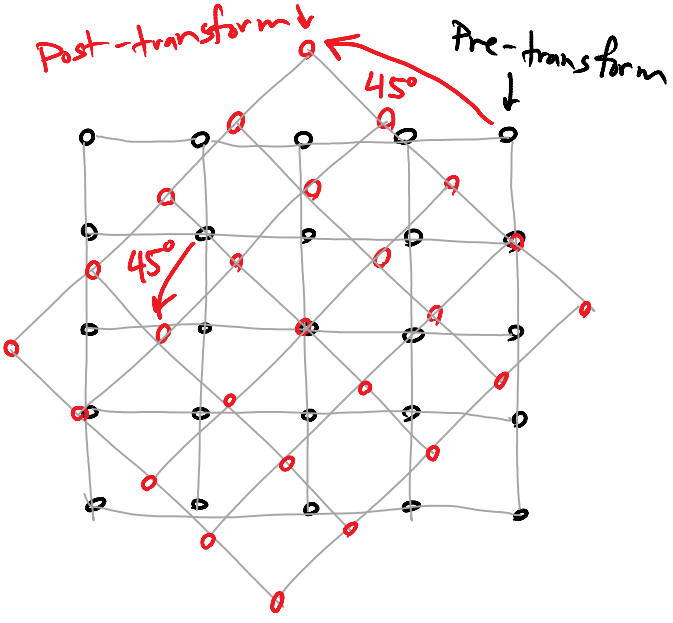
L10

Goal: To learn when & how to resample an image. We also want to have a way to understand imperfections in our resampling.

Resampling

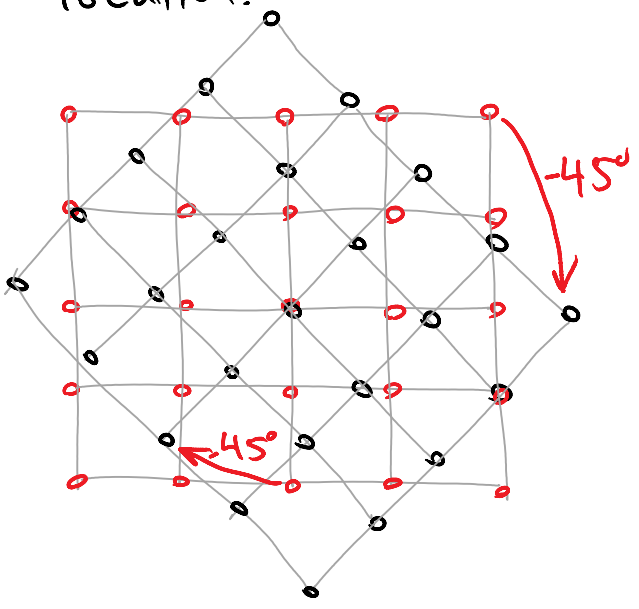
Suppose we want to rotate an image by 45°.

Method 1: For each pre-transform pixel, rotate it 45° to its post-transform location.



Problem: Pixels tend to land between post-transform pixels.

Method 2: For each post-transform pixel, counter-rotate it 45° to its pre-transform location.



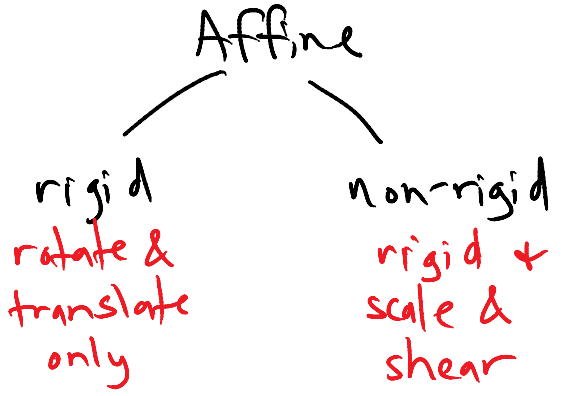
Problem: Pixels tend to come from locations between pre-transform pixels.

The vast majority use Method 2. The reason becomes more obvious when you consider non-affine transformations.

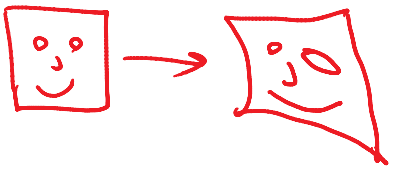
Pseudocode: to rotate an image f by 45°

```
for each (r,c) in g (post-transf. image)
    rotate (r,c) by -45° → (r',c')
    sample f at (r',c') → g(r,c)
next (r,c)
```

Types of Spatial Transforms



Non-Affine (everything else)
"warp", "deform"

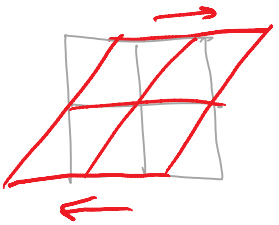


Scale

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Shear

eg. $\begin{bmatrix} 1 & a & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$



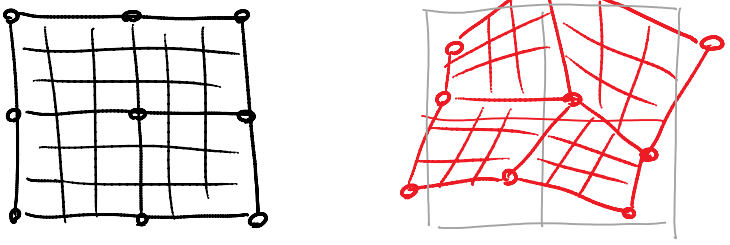
$x' = x + ay$
$y' = y$
$z' = z$

There are many ways to represent non-affine (a.k.a. "non-linear") transformations. Ultimately, one needs to define a displacement

Ultimately, one needs to define a displacement map... a set of vectors defining how each pixel moved. To implement it, we need to know the inverse of the deformation. For each post-transform pixel, we need a displacement vector that tells us where it came from.

Piecewise Linear

The displacement of a grid of control points defines the transform. Between those ctrl. points, we use a linear transformation.



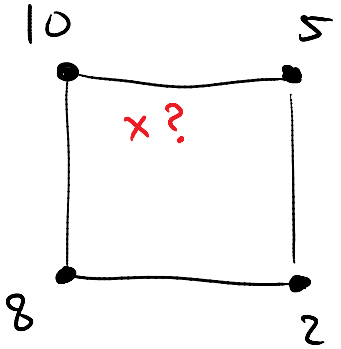
Basis Functions

Use a linear combination of mathematical functions to define the x- and y-displacements.

eg. trig, polynomials (splines, etc), Gaussians, etc.

Interpolation

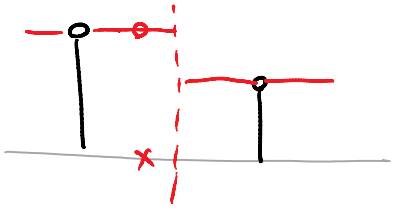
This is the operation of estimating the value of an image between its pixels.

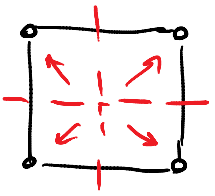


Nearest Neighbour

The simplest method is to round to the nearest sample, and use its value. The problem is that this creates discontinuities in the result.

eg.

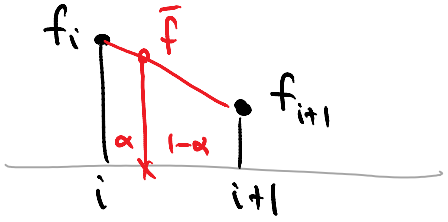




Linear Interpolation

You've heard of this before.

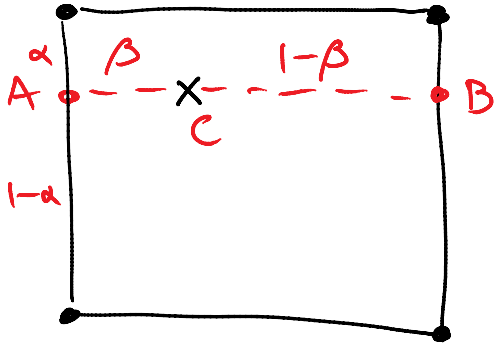
i.e.



$$\bar{f} = \alpha f_{i+1} + (1-\alpha) f_i$$

It's a weighted sum ("convex combination" in fact) of the two nearest values.

What about in 2D, for images?



Use 1D linear interp. along each dimension sequentially.

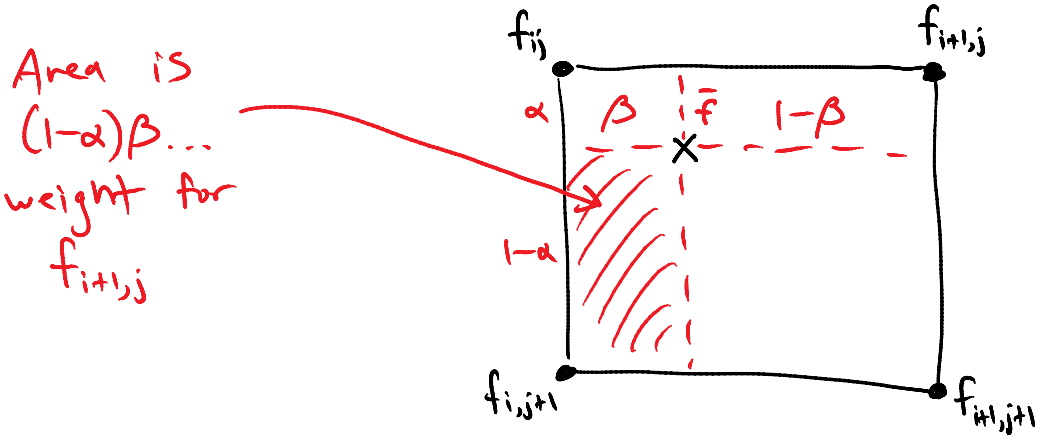
- compute A & B
- compute C

This is called "bilinear interpolation".

In 3D, "trilinear..."

Trick: Weight each sample by the area (length, volume) of its opposite region.

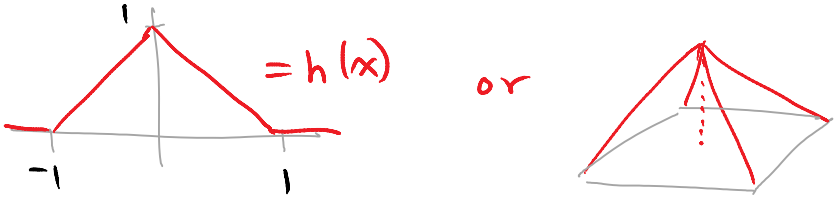
volume) of its opposite region.



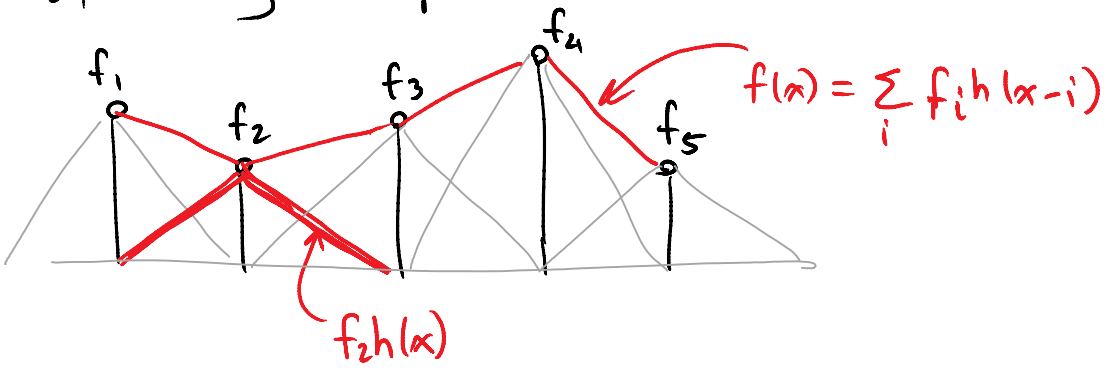
$$\bar{f} = f_{ij}(1-\alpha)(1-\beta) + f_{i+1,j}(1-\alpha)\beta + f_{i,j+1}\alpha(1-\beta) + f_{i+1,j+1}\alpha\beta$$

Convolution

Another way to think of (bi) linear interpolation is as convolving the samples with a continuous-domain interpolation kernel.



This is equivalent to placing a weighted copy of h(x) at every sample.



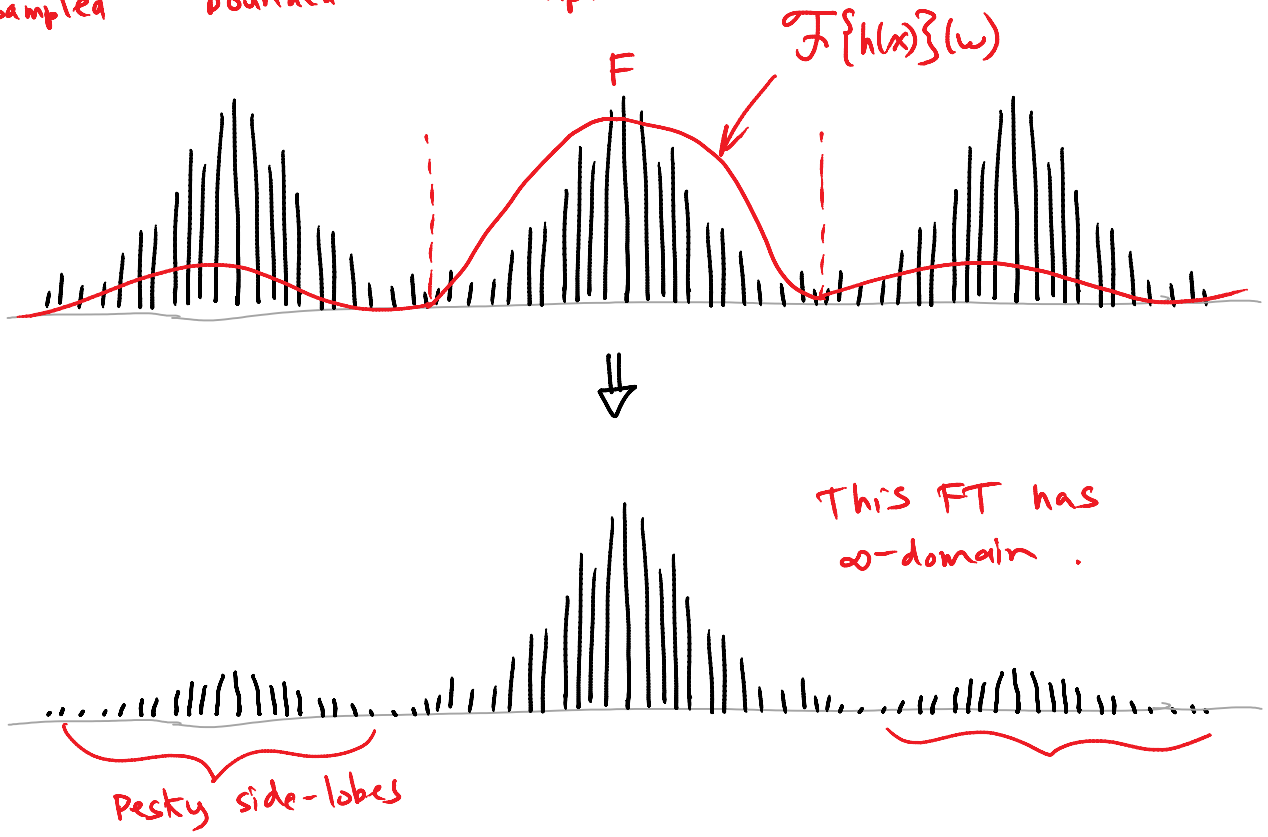
Sampling Theory (revisited)

Hence, interp. can be thought of as

Hence, interp. can be thought of as

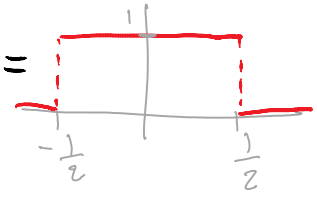
$$(f * h(x))(x) \iff F\,H$$

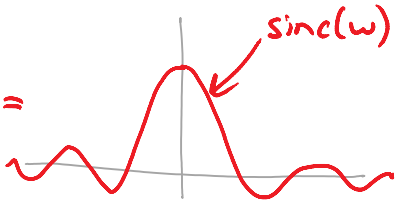
periodic, sampled — continuous, bounded — periodic, sampled — continuous, ∞-domain



But, since we sample $(f * h(x))$, its FT becomes periodic, and those pesky side-lobes interfere with adjacent copies. This is another example of aliasing, and results in a fundamental loss of information.

For nearest-neighbour, the side-lobes are even bigger than for linear interp.

kernel $h(x) =$ 

$H(\omega) =$ 

There are lots of other kernels we can use. But there is a trade-off:

There are lots of other kernels we can use. But, there is a tradeoff:

- You want a large kernel because the scaling property says that spatially wide functions have a tighter FT, & the side lobes will be smaller. (accuracy)
- You want a small kernel so it is cheaper to interpolate (efficiency)