13 - Filtering

Dr. Hamish Carr & Dr. Rafael Kuffner dos Anjos

Agenda

- Images
- Sampling
- Filtering
- FFT

What Is an Image?

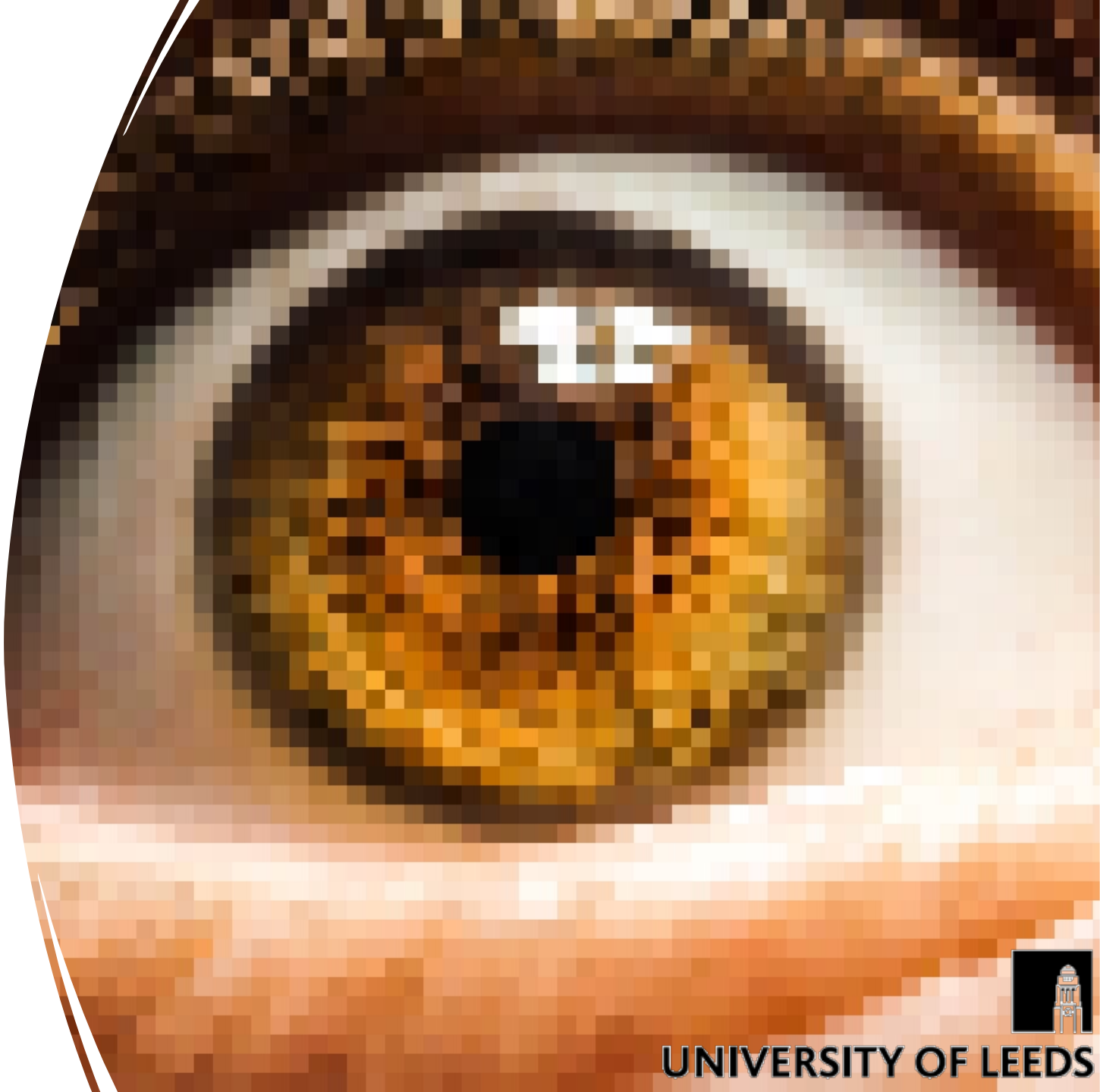
- An *ideal* or *continuous* image is a function
  - $I(x, y)$: $\mathbb{R}^2 \rightarrow \mathbb{R}$
- That maps location on the image to intensity
- We should also represent *wavelength*
  - $I(x, y, \lambda)$: $\mathbb{R}^2 \times \mathbb{R} \rightarrow \mathbb{R}$
- Or even irradiance:
  - $L(X, \omega, \lambda)$: $\mathbb{R}^3 \times S^2 \times \mathbb{R} \rightarrow \mathbb{R}$

On the Retina

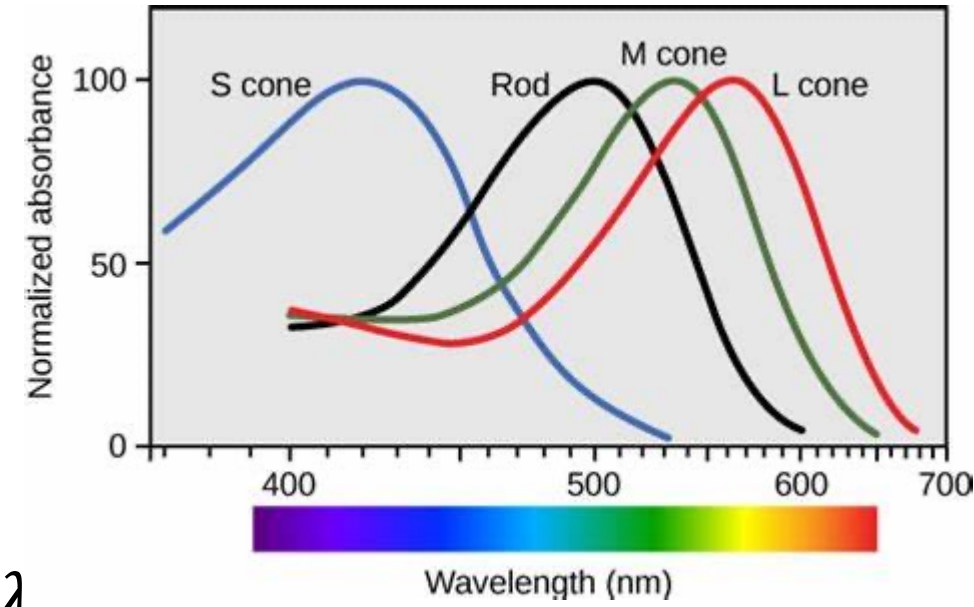
- Consider the irradiance on the retina
  - $I_{retina}(x, y, \lambda) = \int_{S^2} L(X, \omega, \lambda) dS$
  - Integrates all light at that wavelength
  - Over all incoming directions
- If lens is perfect, only one direction counts
  - But the retina is *not* continuous
  - It is made of cells of varying area

Discrete Retina / Image

- Retinal cells are blobs
- We will assume a nice square array
  - i.e. the pixels on the image plane
  - indexed on i,j
- So we can transform this to:
  - $p_{i,j} = \int_{x_i-\frac{1}{2}\Delta x,}^{x_i+\frac{1}{2}\Delta x} \int_{y_j-\frac{1}{2}\Delta y}^{y_j+\frac{1}{2}\Delta y} \int_{S^2} L(X, \omega, \lambda) dS dy dx$



Colour Response



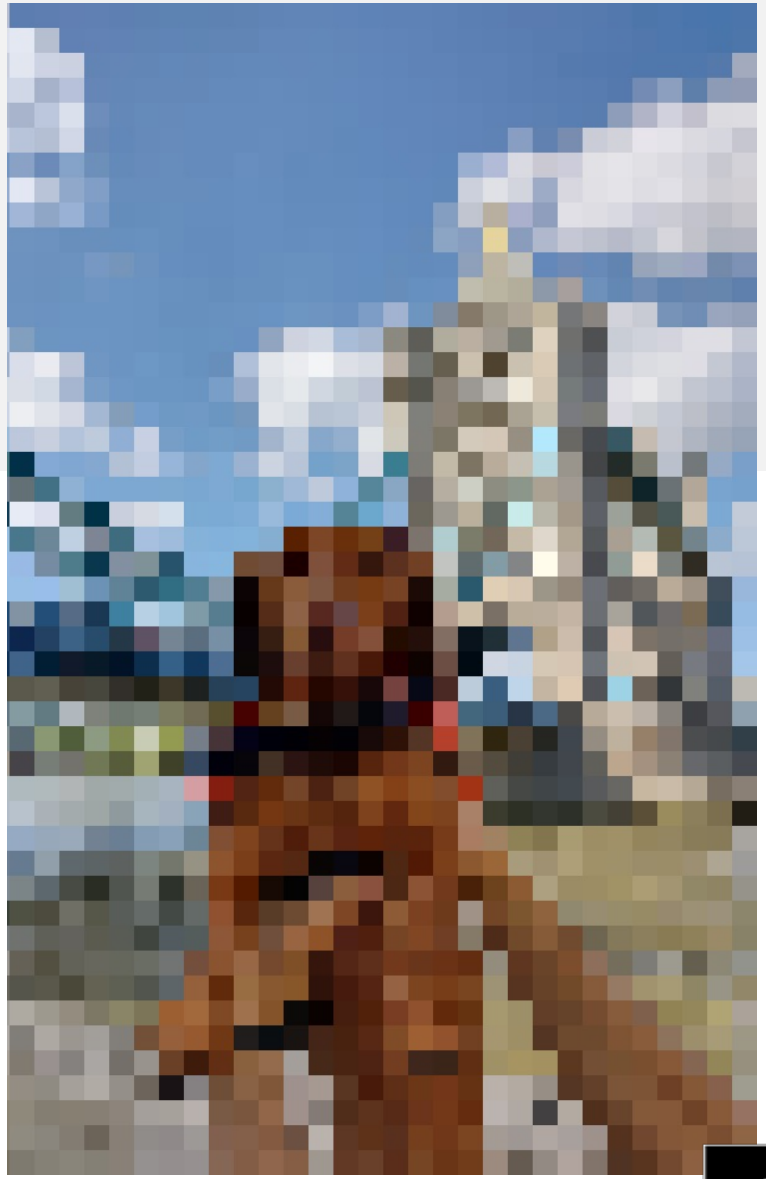
- Each cell has its own response function
  - $r(\lambda)\colon \mathbb{R} \to \mathbb{R}$, $g(\lambda)\colon \mathbb{R} \to \mathbb{R}$, or $b(\lambda)\colon \mathbb{R} \to \mathbb{R}$
- How strongly it responds to wavelength λ
- It may actually vary between cells
- So assume we have $r_{i,j}(\lambda)$:
- $p_{i,j} = \int_{x_i-\frac{1}{2}\Delta x,}^{x_i+\frac{1}{2}\Delta x} \int_{y_j-\frac{1}{2}\Delta y}^{y_j+\frac{1}{2}\Delta y} \int_{350nm}^{750nm} \int_{S^2} r_{i,j}(\lambda) L(X, \omega, \lambda) d\lambda dS dy dx$
- We could even parameterise with time t

Assumptions

- Assume a perfect lens (only one ω matters)
- Assume square pixels (retinal cells)
- Assume one wavelengt
  - $p_{i,j} = \int_{x_i-\frac{1}{2}\Delta x,}^{x_i+\frac{1}{2}\Delta x} \int_{y_j-\frac{1}{2}\Delta y}^{y_j+\frac{1}{2}\Delta y} I(x,y)dydx$
- And assume we are storing as floats
  - not integers

Sampling

- We never actually compute this
  - although perhaps we should
- Instead, we sample the function
  - measure the light at the centre of the pixel
- Then display the same value across the pixel
- And this leads immediately to errors
  - In particular, *aliasing*

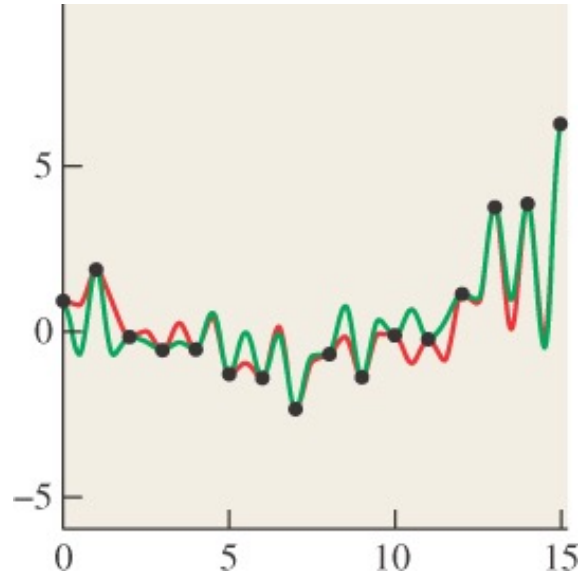


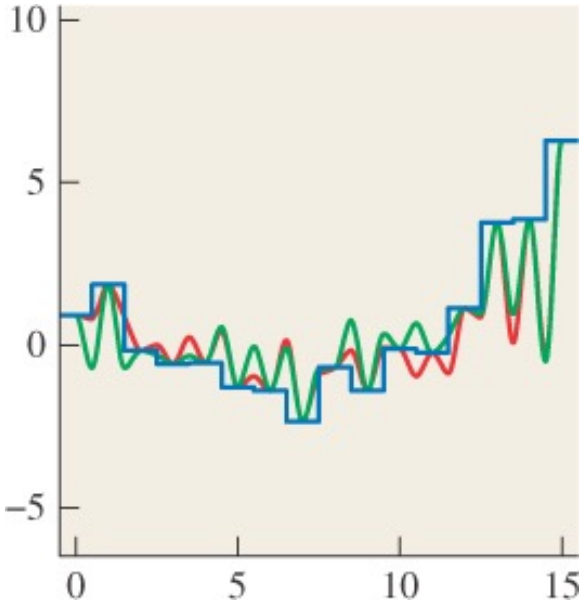
Reconstruction

- Suppose we measure a function at 10 points
- Can we *reconstruct* the function?
  - i.e. recover the continuous version
- Only if there is a unique representation
- And it is easy to show there isn't

1-D Example

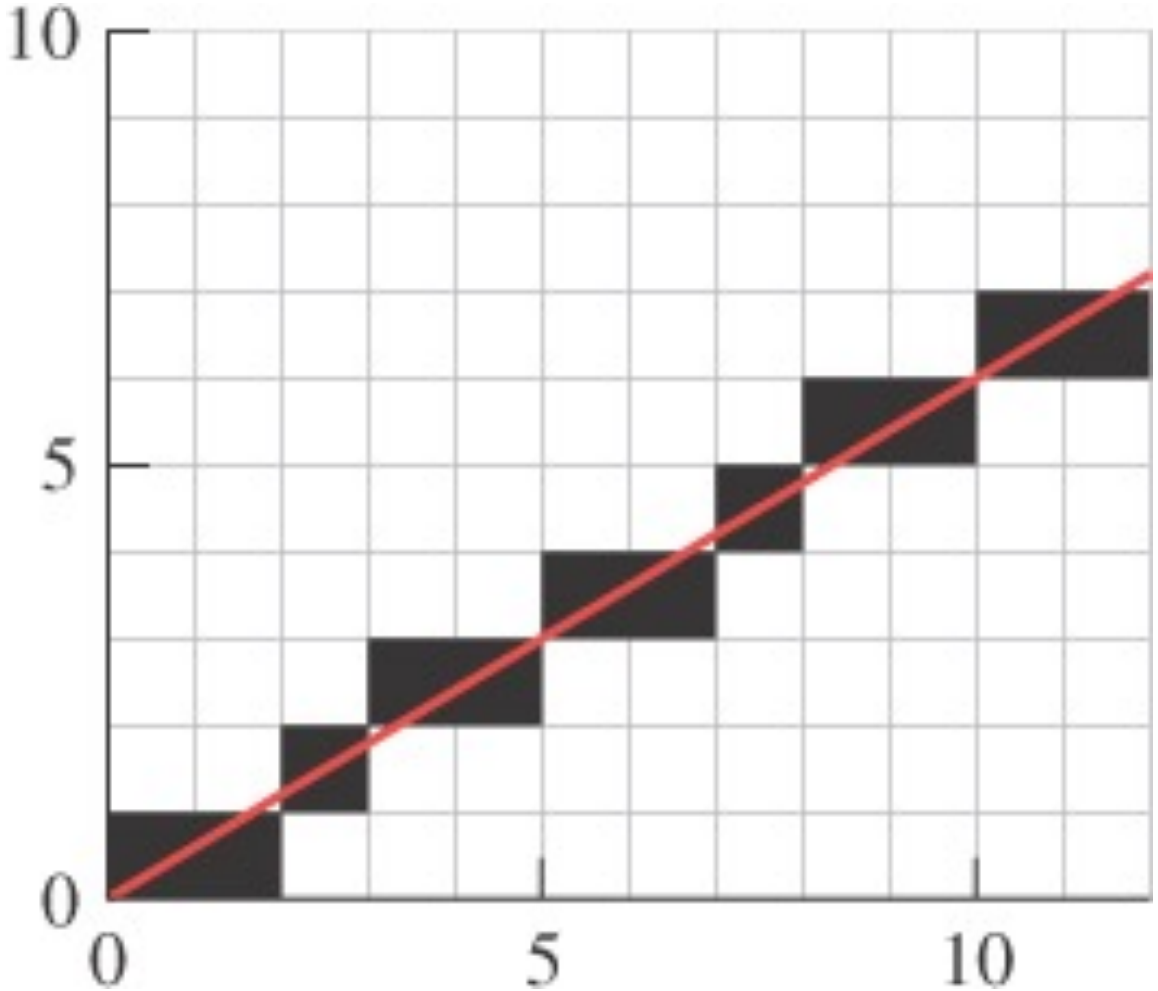
- Both green and red have the same samples
- Which one is right?
- The screen does a third thing
  - Because of hardware



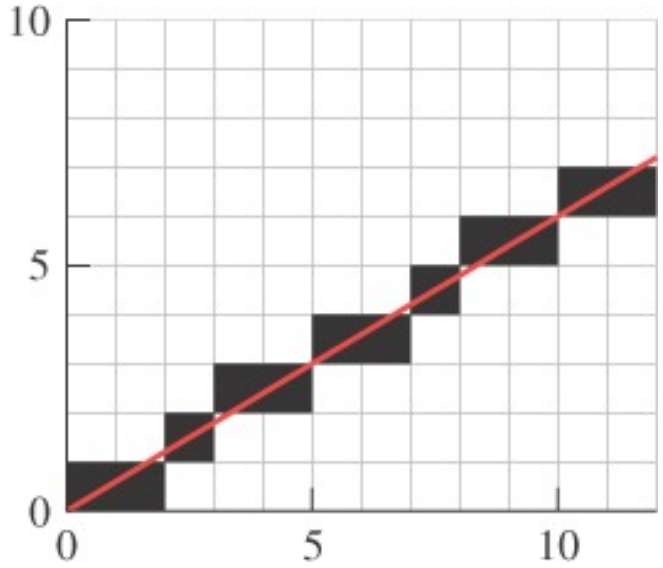


Aliasing

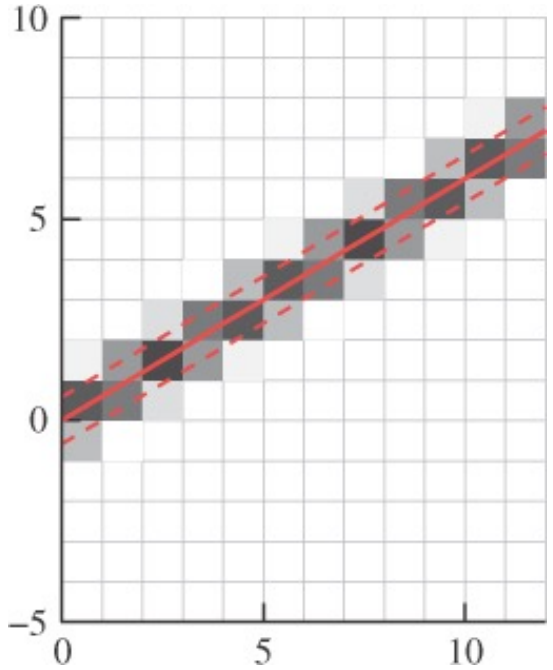
- A general phenomenon
- Caused by not enough samples
- Reconstruction dominated by error
- We *know* this should be a straight line
- But it is jaggy when discretised



Solution (Goal)



Aliased



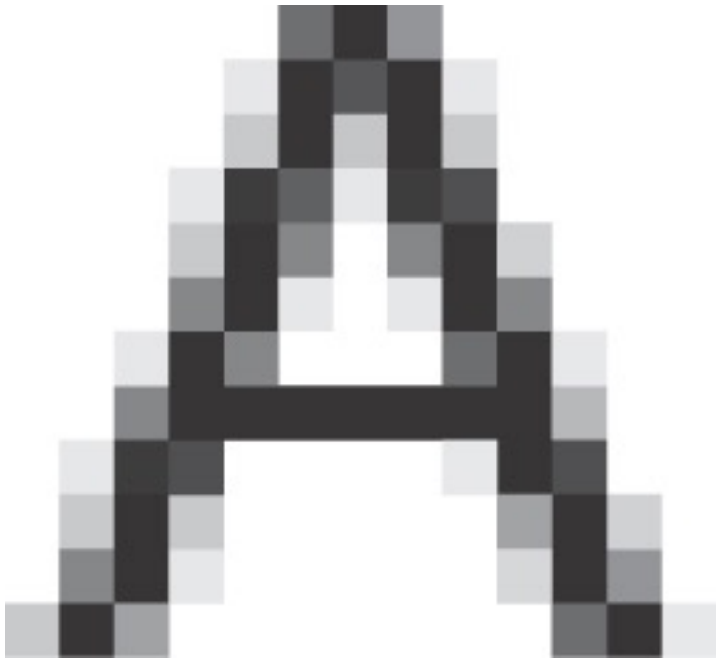
Anti-aliased

Far away

- Replace it with a grey-level
- Overlap between line of width 1 & pixel

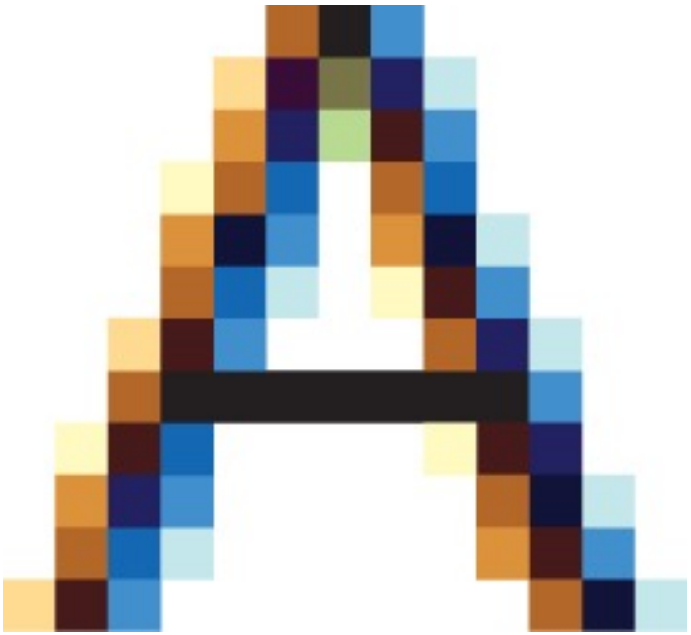
Immediate Use: Fonts

Binary Font



Anti-aliased

Using Greyscale

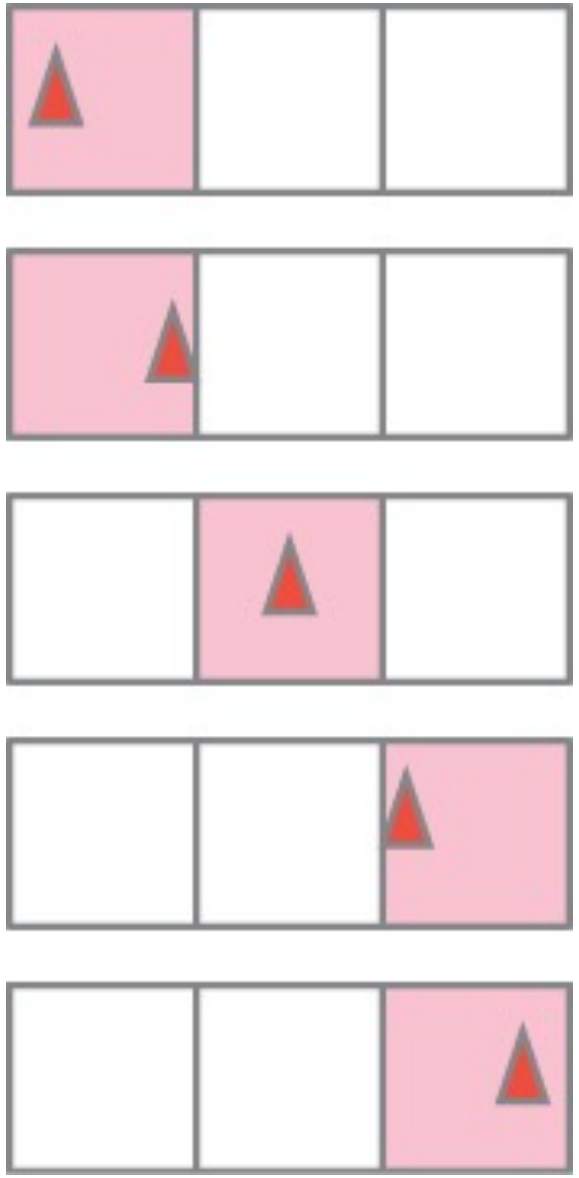


TrueType

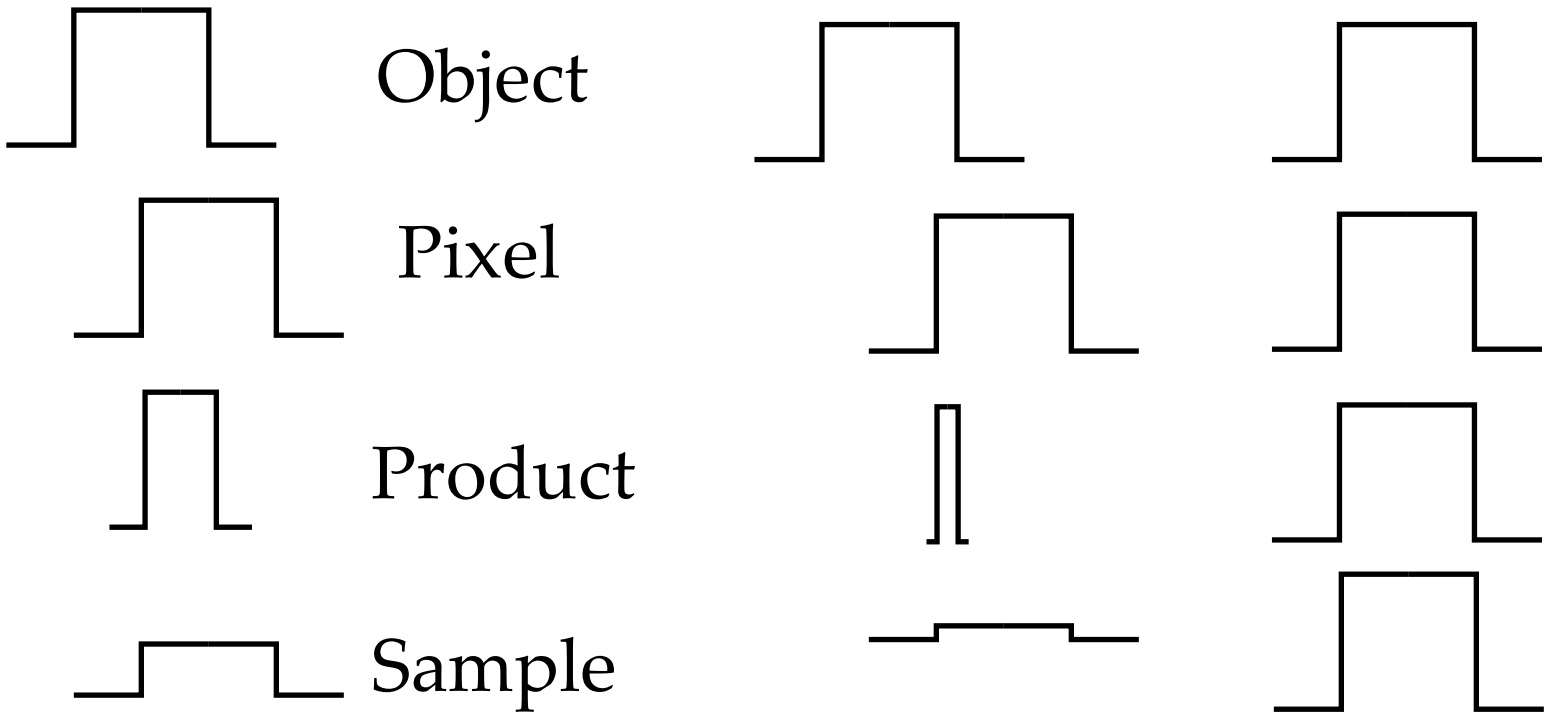
(RGB in different columns, eg LCD)

Aliased Motion

- Object moves 2 pixels in 4 time steps
- But it looks like it speeds up in the middle
- Really just the same problem



Solution

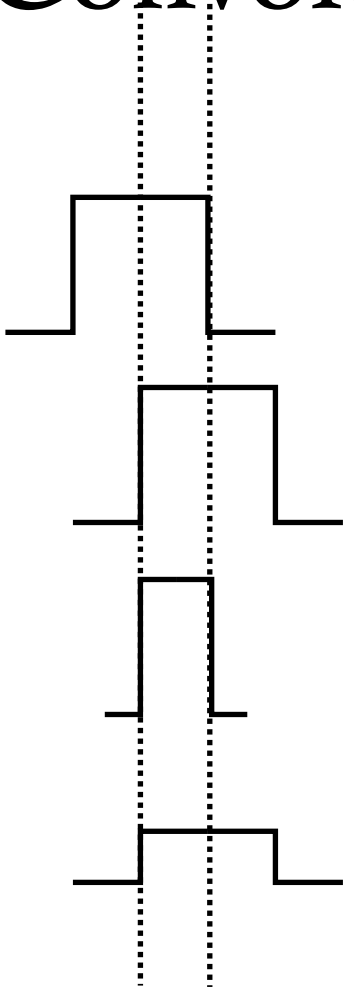


- Treat the object as a function itself
- Combine the function with the pixel

Mathematically

- Clearly, this varies with the *offset*
  - between object and pixel
- We express this as a function (of course)
- And an operation called *convolution*

Convolution



Object $f(x)$, centred at 0

Pixel $g(x)$, centred at h

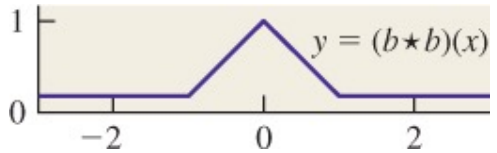
Product $f(x)g(h-x)$

Sample $\int_{-\infty}^{\infty} f(x)g(h-x)dx$

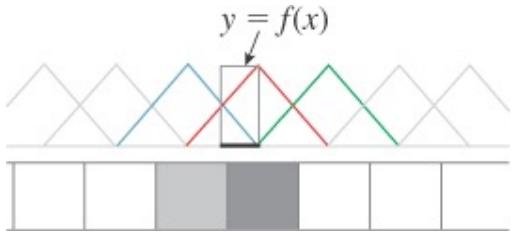
- Convolution works with any functions
  - and has nice mathematical properties

Filtering

- Depends on distance from pixel to object



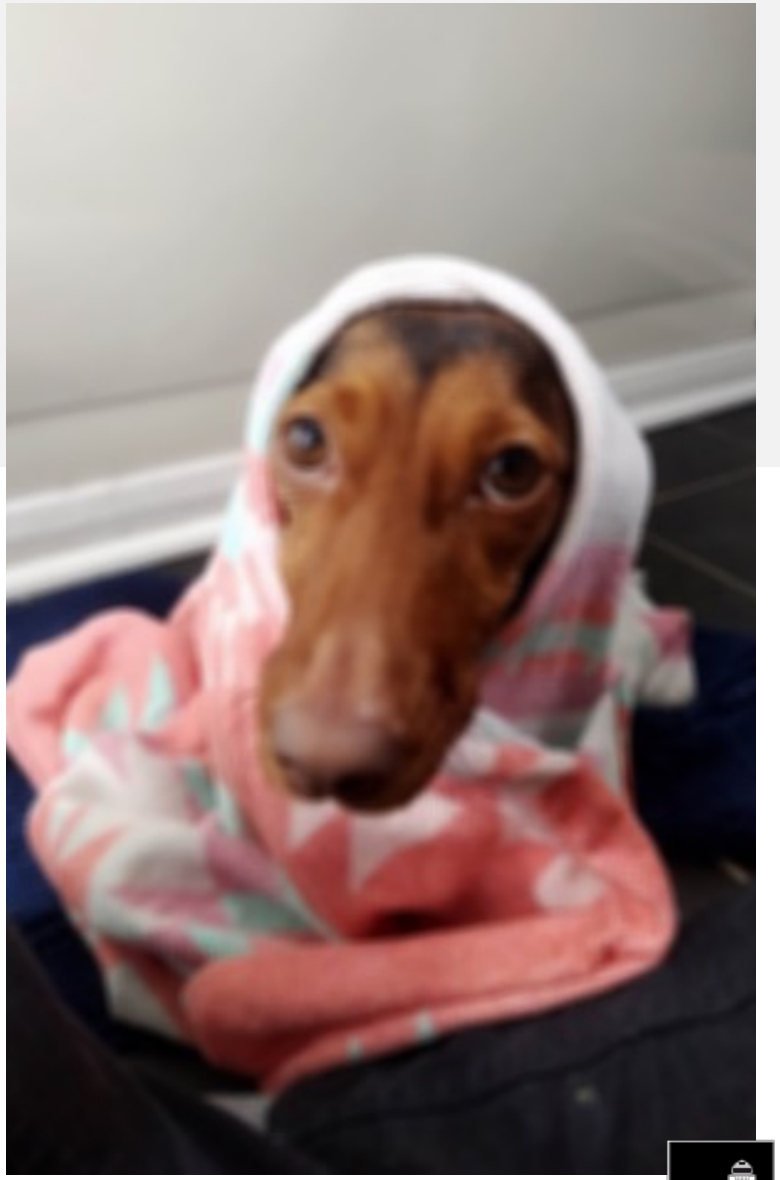
- So the overlap varies as the pixel moves



- But cost over an image is $O(NM)$
  - N is the number of pixels
  - M is the size of the filter

Blurring

- Simple trick: filter with a slightly larger area
  - average all adjacent pixels
- But this is just a 3x3 filter!
- $\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$
- Larger filter, more blur.

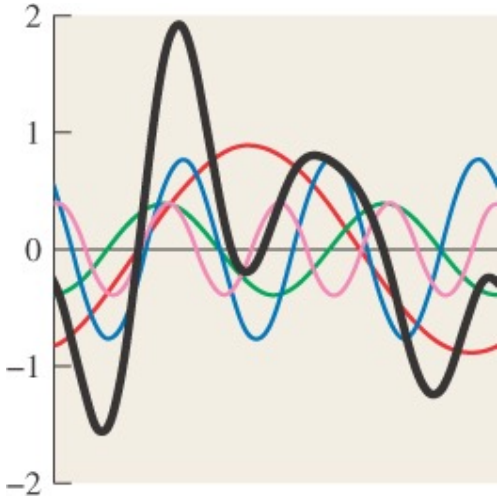


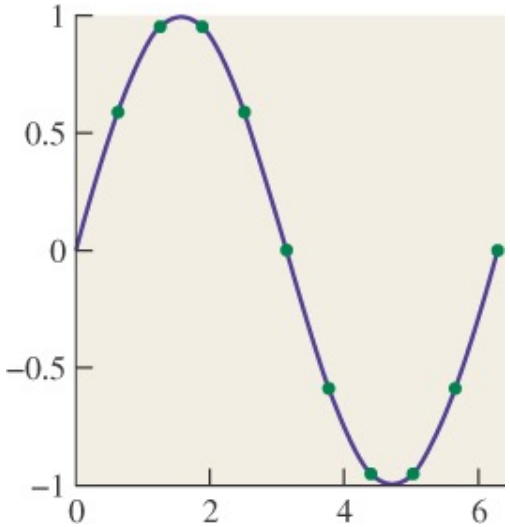
Sampling Theory

- The next piece of the puzzle
- Treats the image like an electrical signal
- Or an audio signal
- Breaks it down into *frequencies*
  - i.e. how rapidly it changes
- Expressed with Fourier analysis

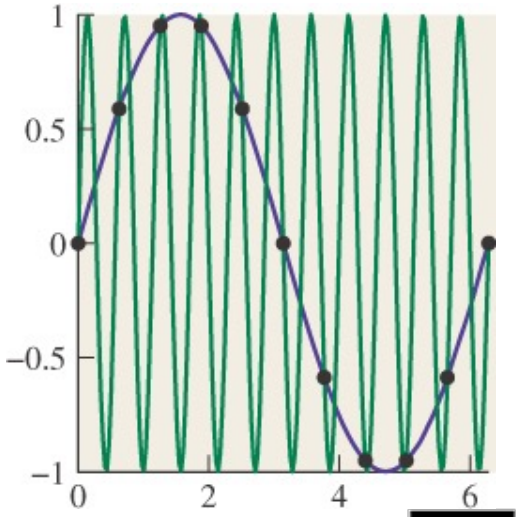
Fourier Analysis

- Any *periodic* function can be broken down
- Into a sum of sines & cosines
- In the complex plane
- Based on our samples
- We will have to assume a *band limit*
- I.e. a maximum spatial frequency





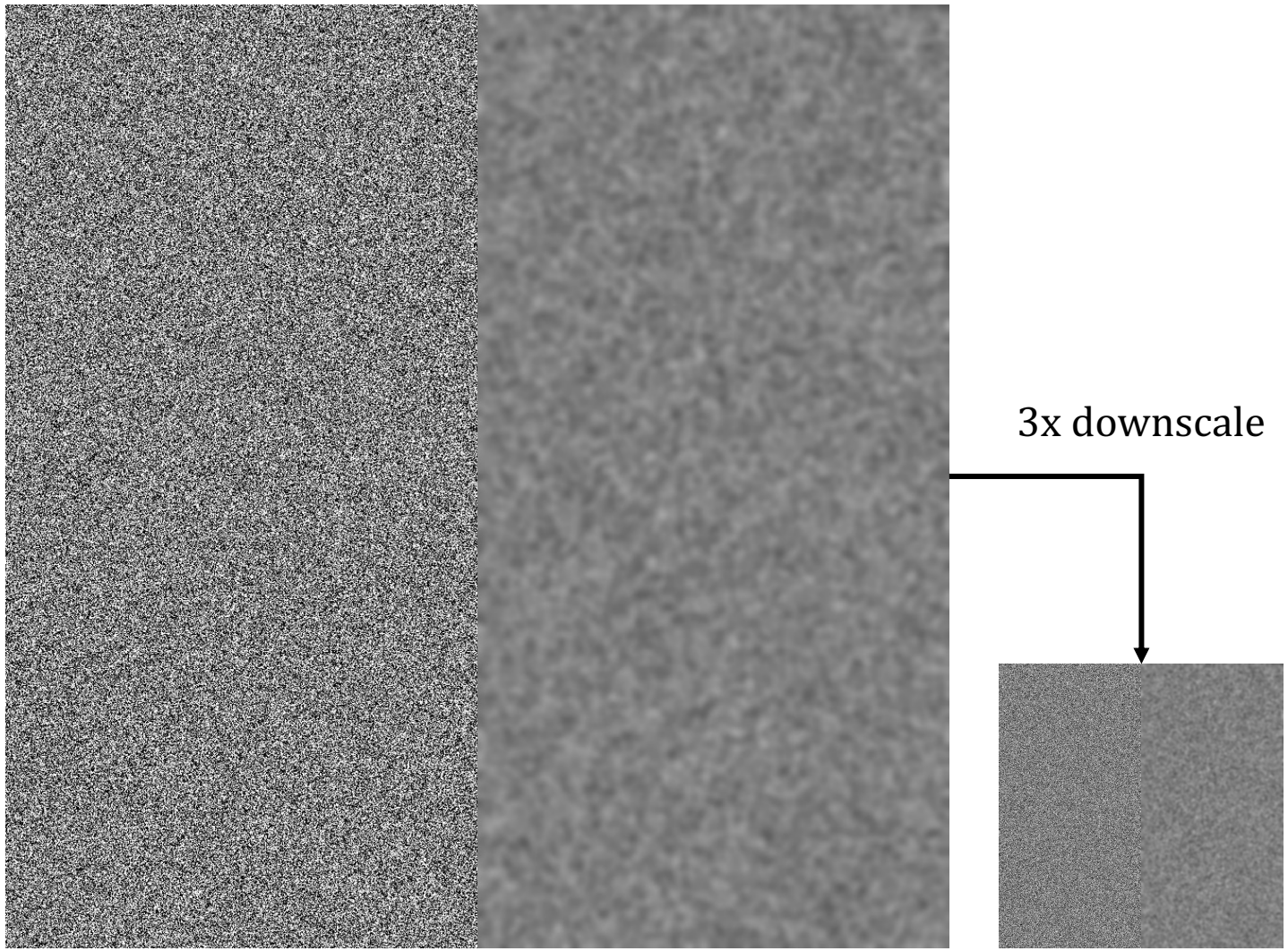
y = sin(x)

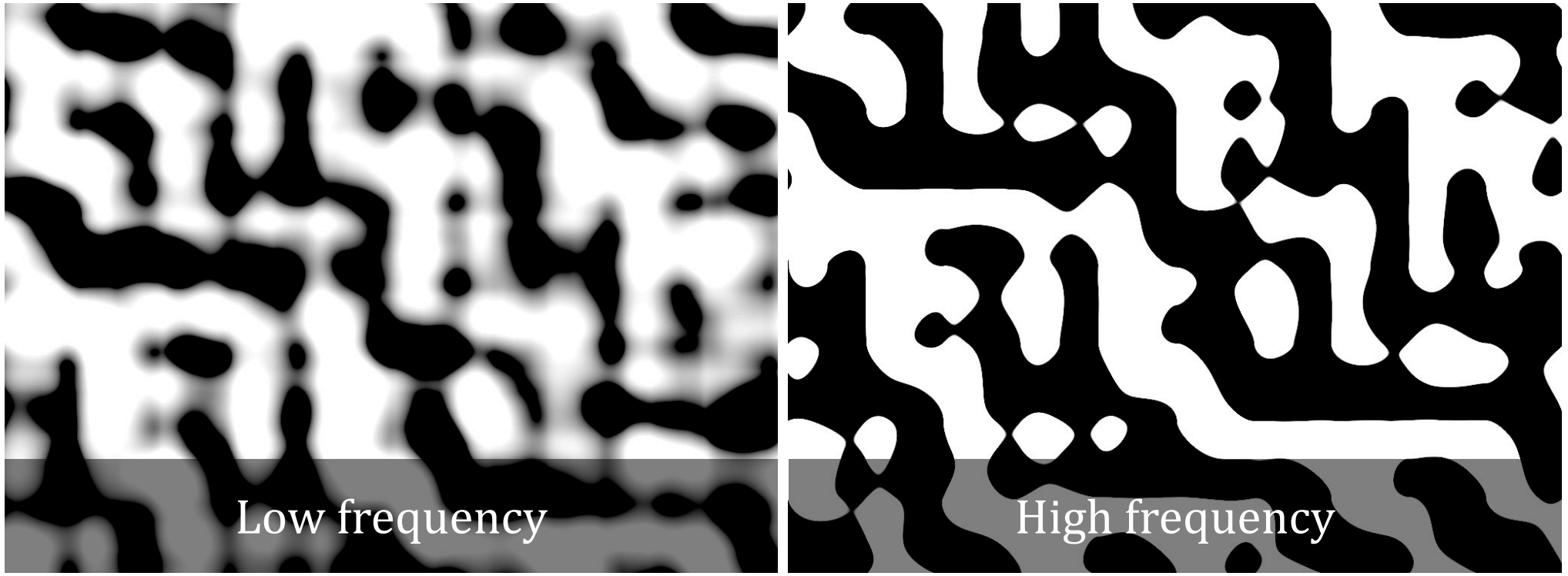


y = sin(11x)

How can we see that in images?

- Each pixel is a sample of your signal
- Low resolution images:
  - band limited to low frequencies
- High resolution images:
  - Low frequency changes happen over several pixels
  - High frequency changes happen over fewer pixels





Example (Perlin noise)

Fourier Transform

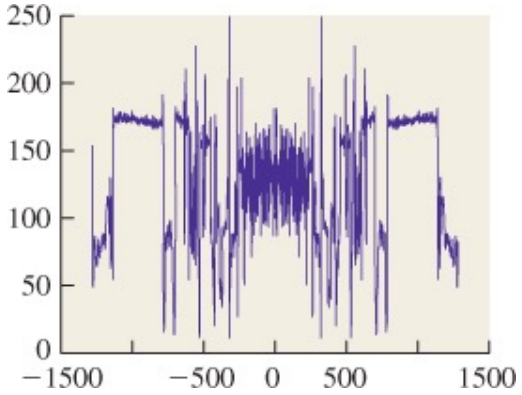
- Rewrites our function
  - from a function of x
  - to a function of ξ: spatial frequency
- Expressed as samples:

$$a_k = \int_{-\frac{1}{2}}^{\frac{1}{2}} f(x)\cos(kx)\,dx \text{ and} \qquad (18.40)$$

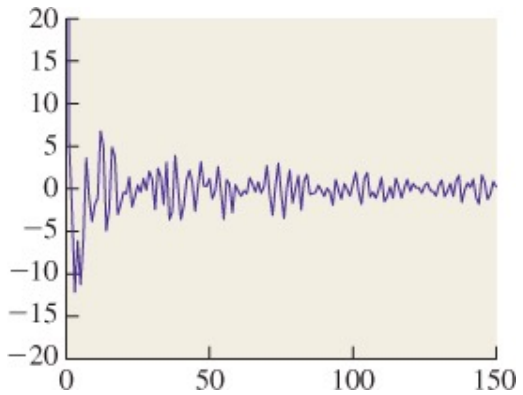
$$b_k = \int_{-\frac{1}{2}}^{\frac{1}{2}} f(x)\sin(-kx)\,dx. \qquad (18.41)$$

- actually a complex number $a_k + b_k i$

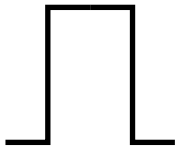
Frequency Filtering: 1D



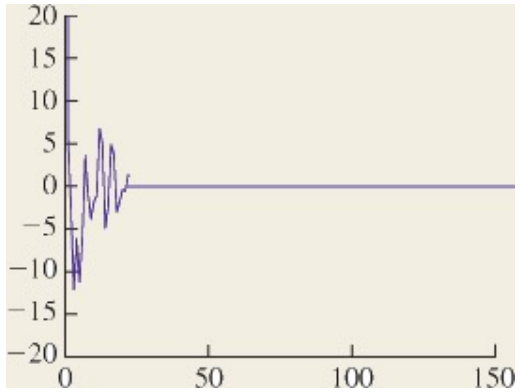
One Row



Fourier Transform



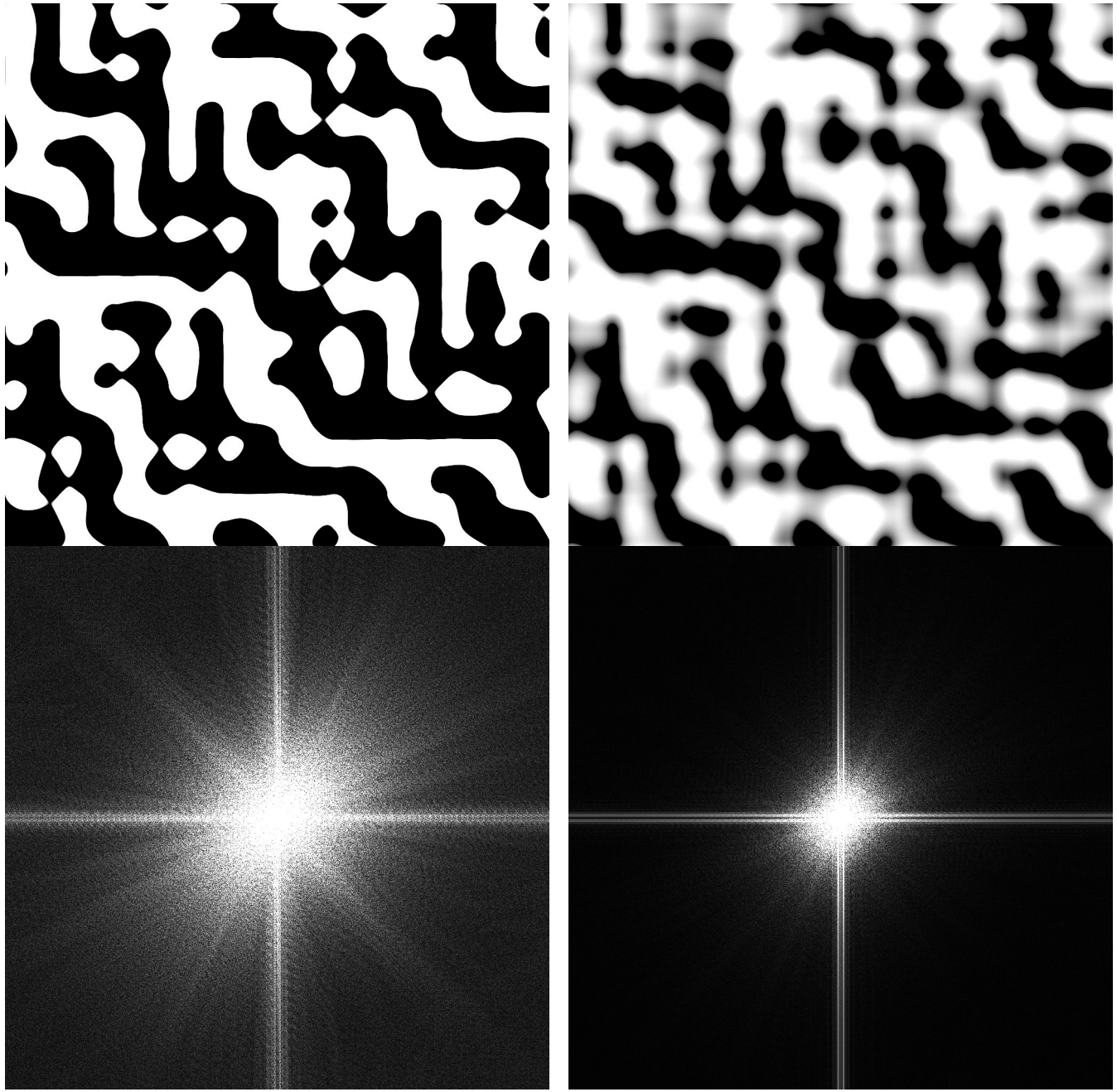
Box Filter



After Filtering

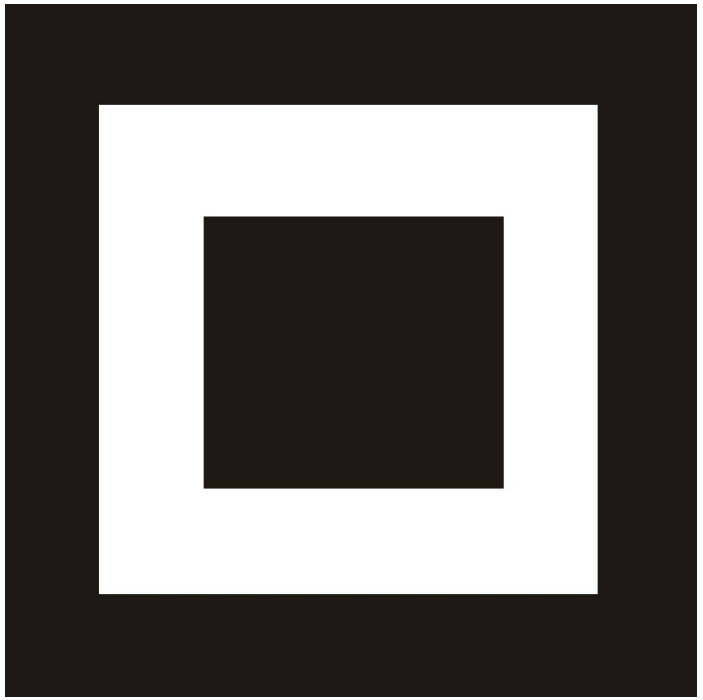
2D FFT

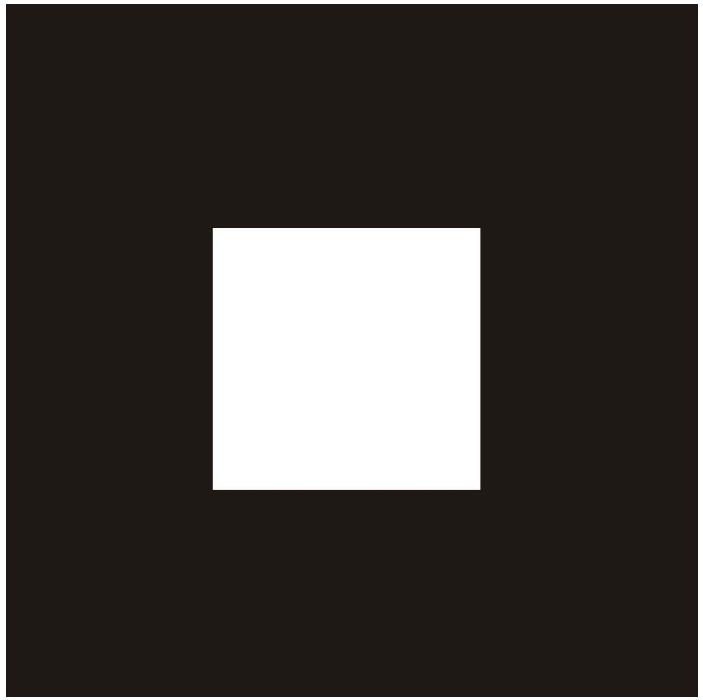
- We can plot the 2D fft to an image.
  - Typically shifted to the center
  - Lower frequencies in the center
  - Higher frequencies going towards the edges
  - Oriented lines mean frequencies of that orientation



Filtering 2D

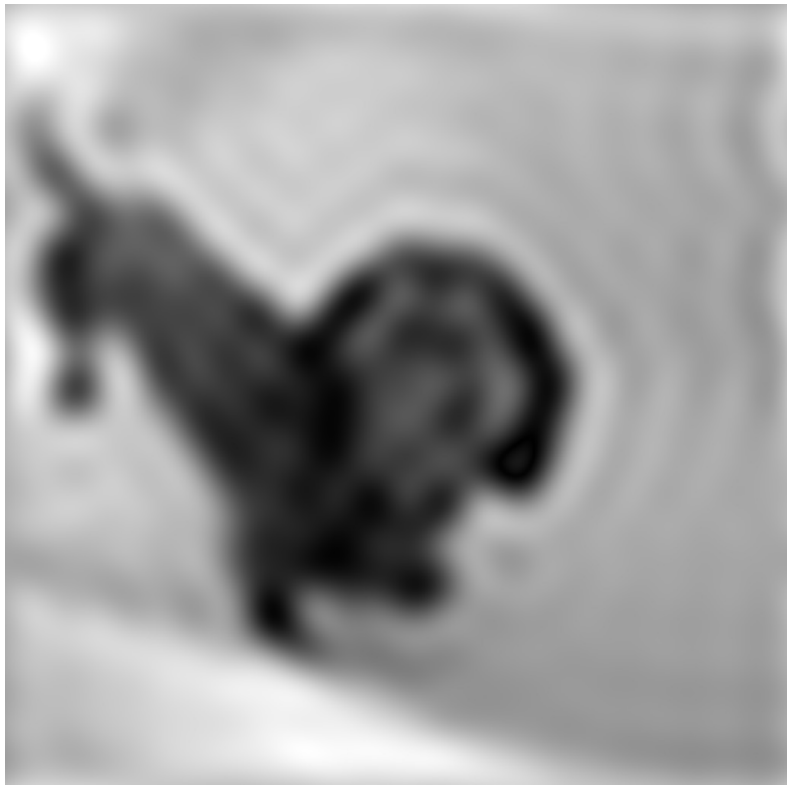
- Filters would look like this
- Can easily separate oriented frequencies as well
- Convolution = multiplication in Fourier domain.



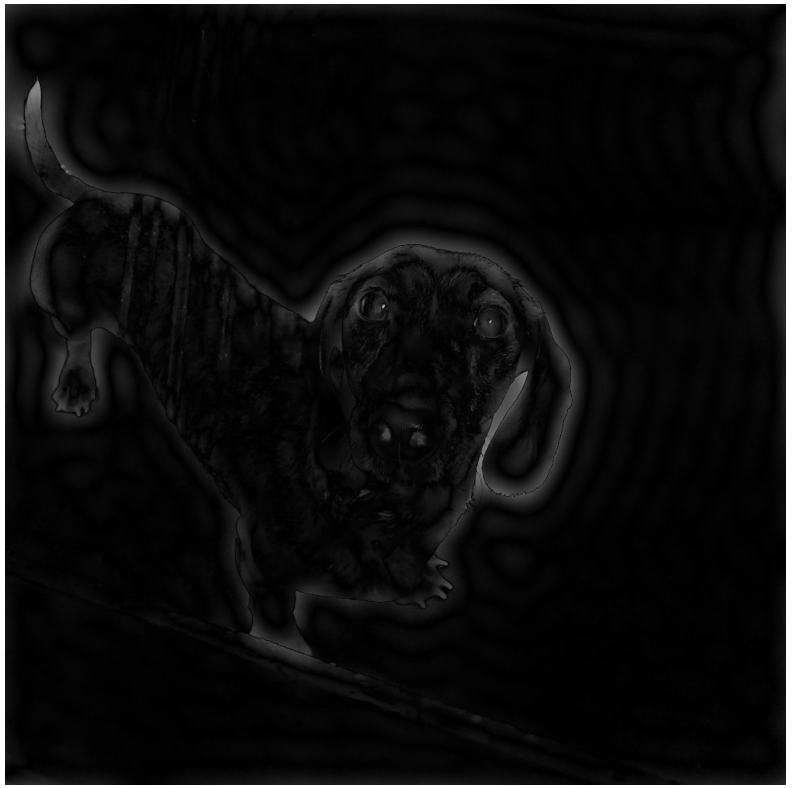


Results

Lola



Low frequency

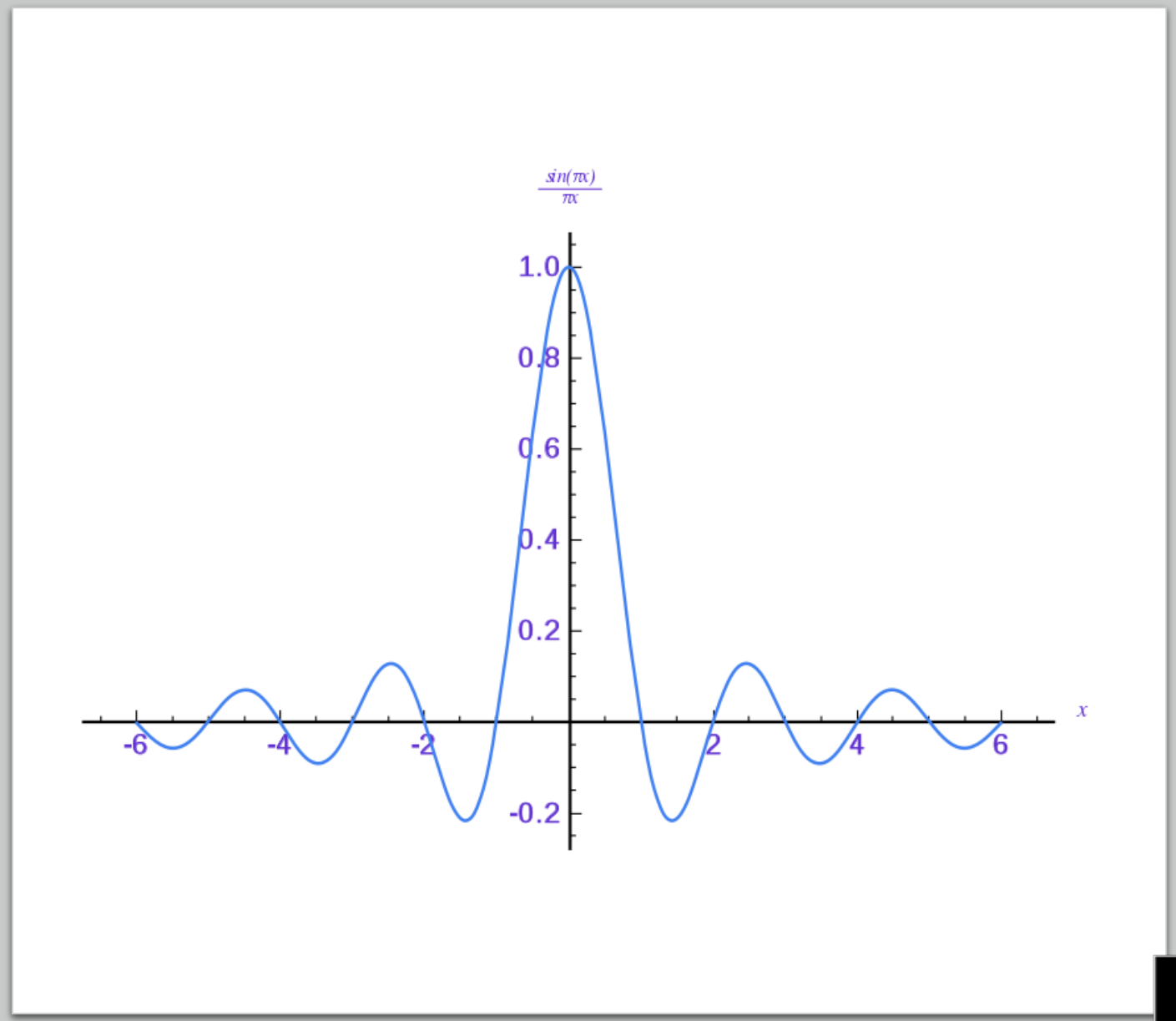


High frequency

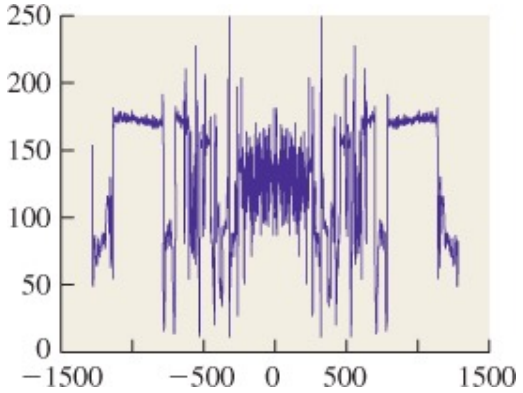
Ringing artifacts!

Sinc Filter

- What does this filter look like?
- In the spatial domain
- It becomes the *sinc filter*
- Ideal filter that removes frequencies above a certain cutoff, without affecting lower ones.

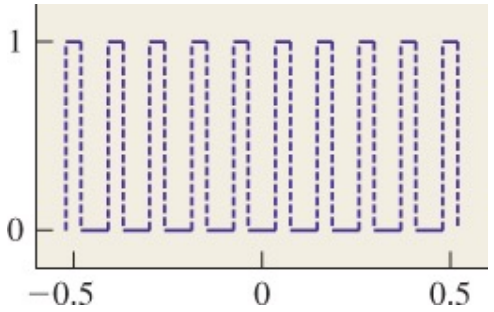


Sampling



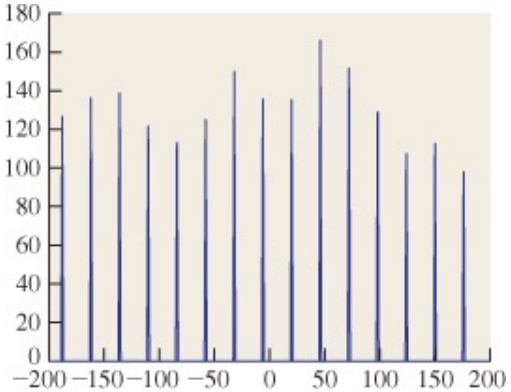
Taj Mahal

×



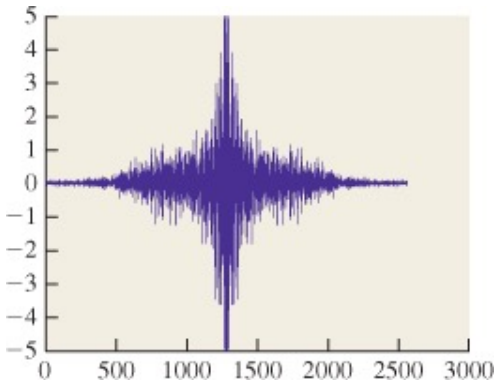
Sampling Function

=



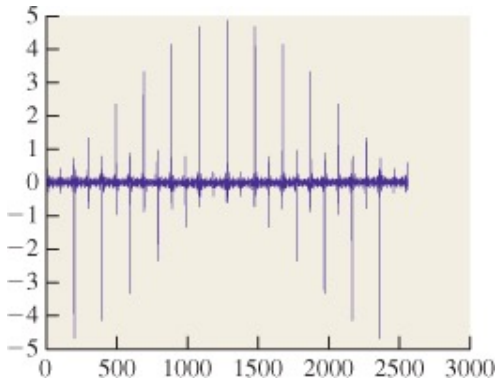
Sampled

Function



Fourier

Transform



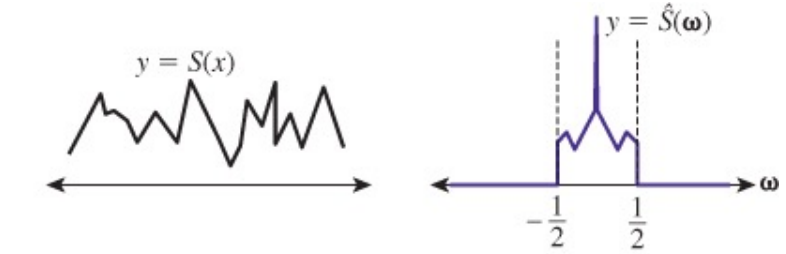
Fourier

Transform

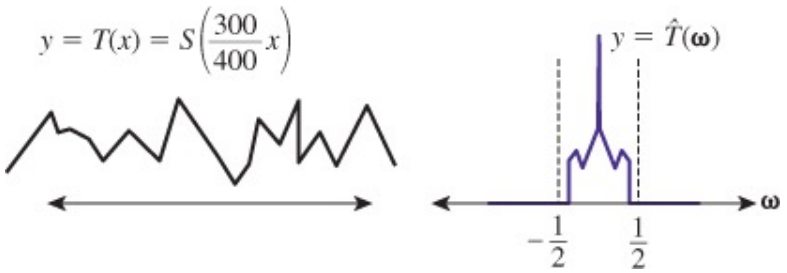
More generally

- Mipmapping is a special case
- We need a more general solution
- But scaling up & scaling down are different
  - Scaling up – use the sinc filter if possible
  - Scaling down – increases the frequency
    - And no longer band-limited

Scaling Up



Original Signal



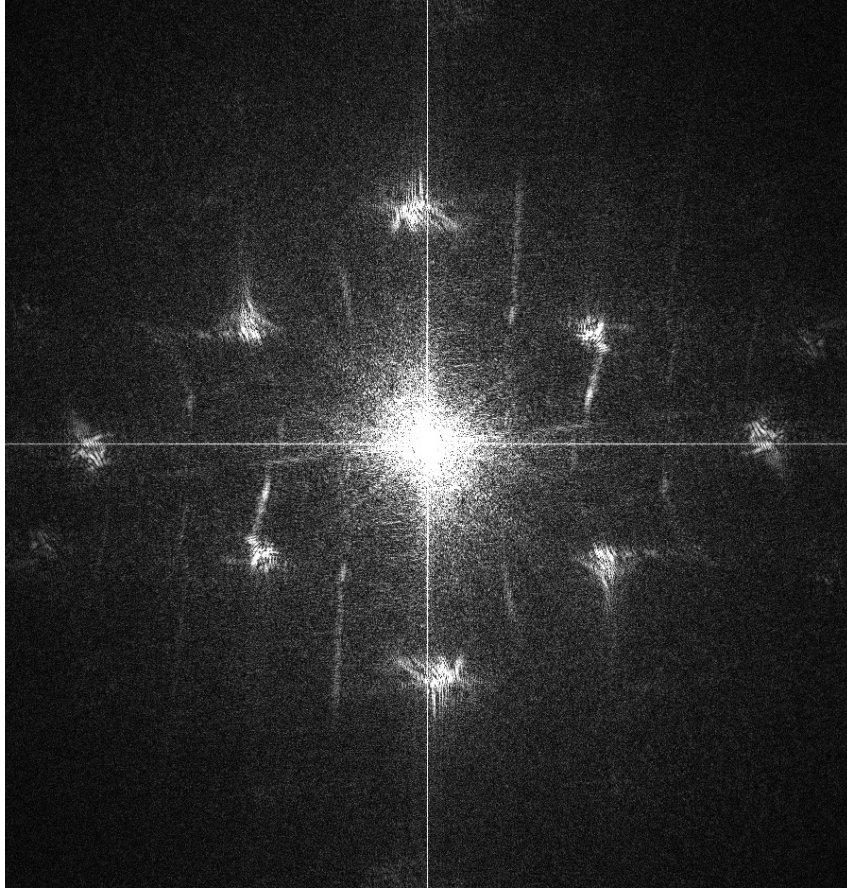
Rescaled Signal

Spatial

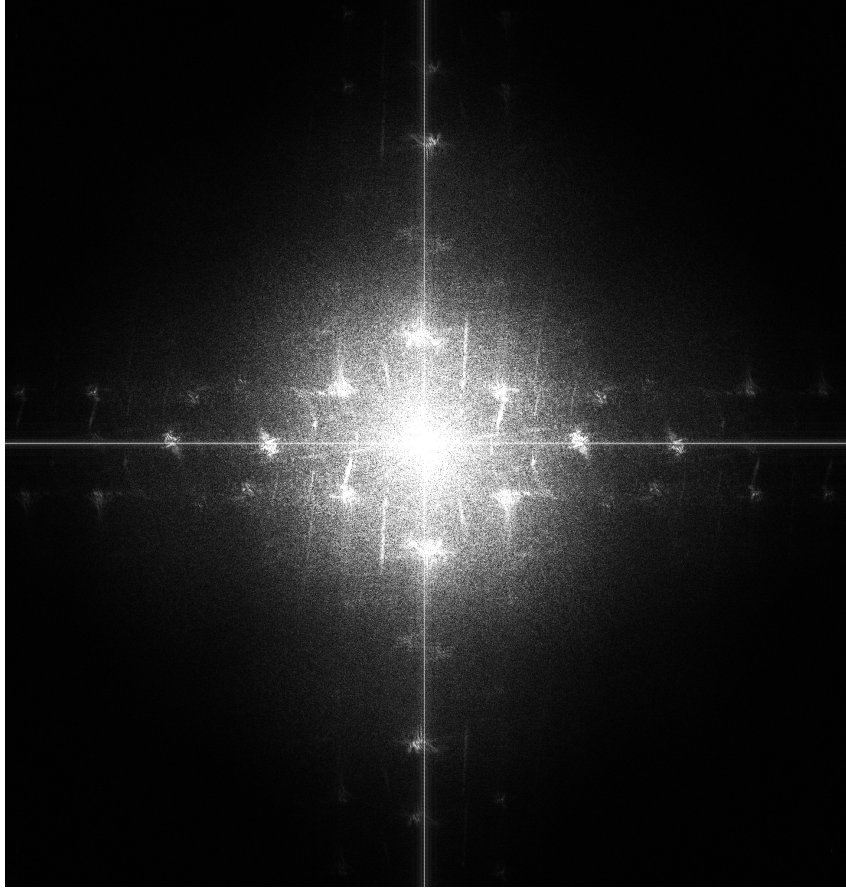
Frequency

- Increasing spatial resolution
- Decreasing frequency resolution
- Bandwidth limit is preserved

Image



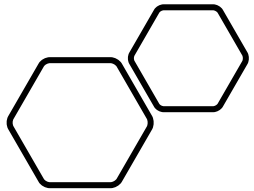
FFT

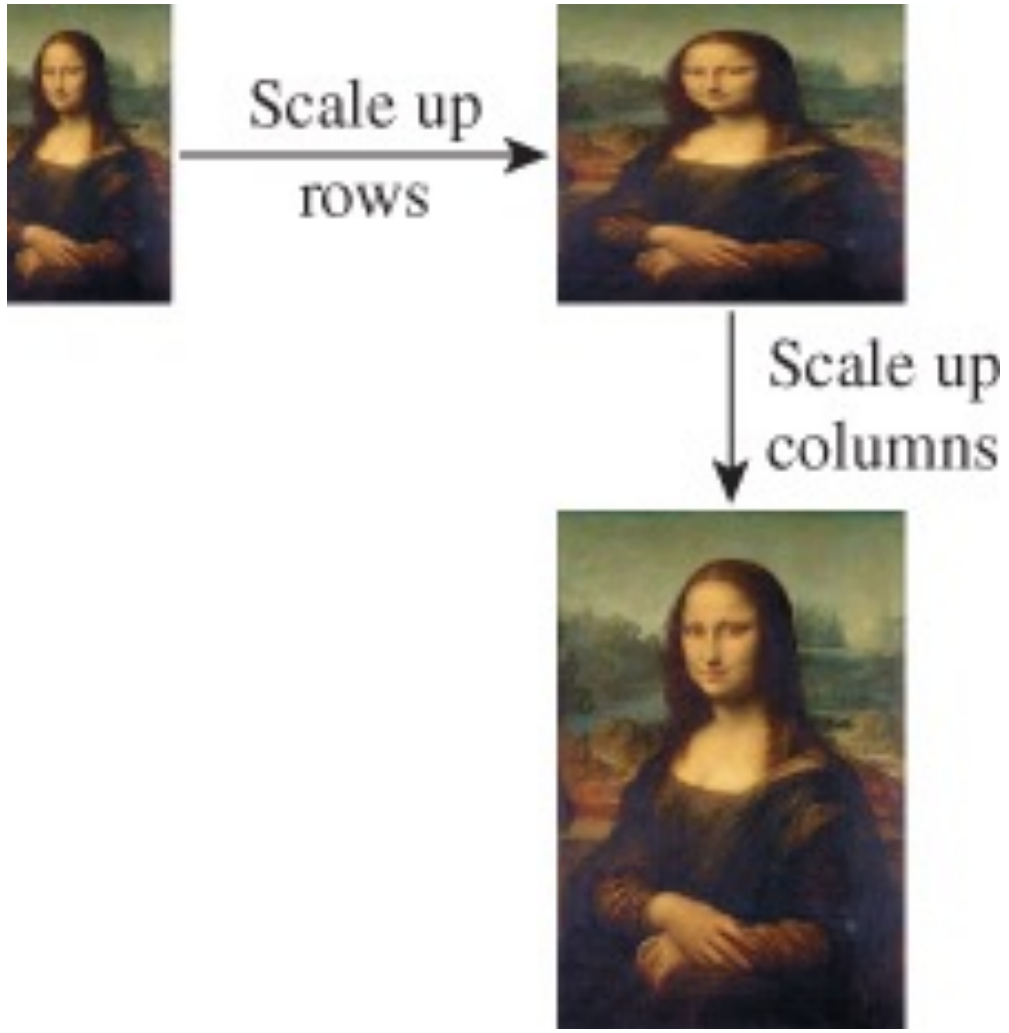


FFT of upscale

Example of upscale

- FFT is “shrinked”



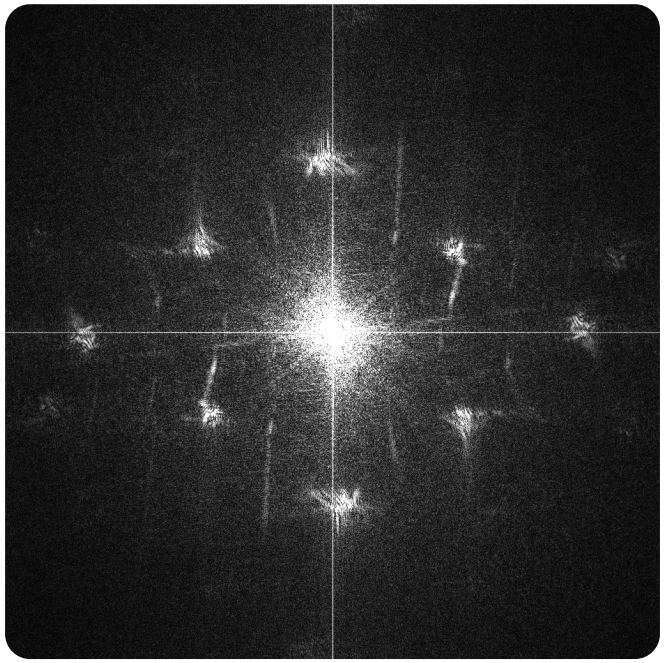


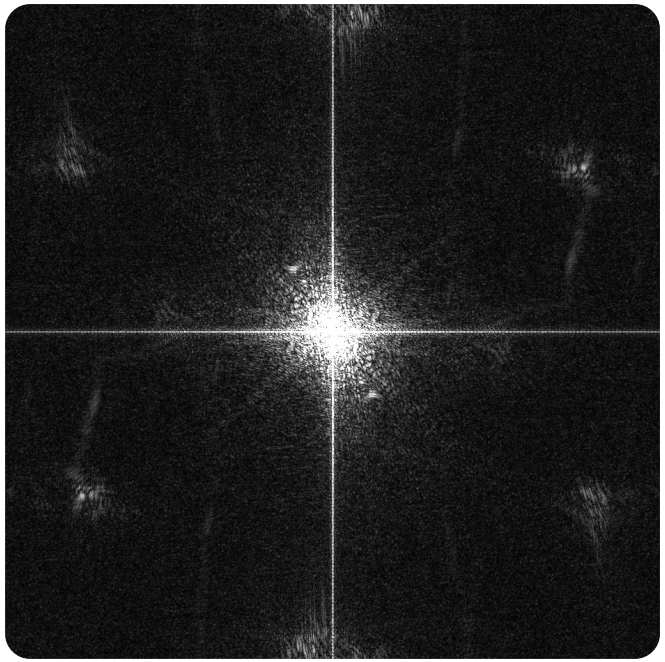
Scaling in Two Dimensions

- Just do one dimension at a time
- Either order produces same result (separable)

Terminology

- Support of f is interval where it is nonzero
  - Can be finite (e.g. box) or infinite (e.g. sinc)
- Images typically have finite support
  - So scaling up isn't usually a problem
  - And sinc filters *preserve* detail
- Bilinear filtering isn't as good at this
  - But textures are more often scaled *down*

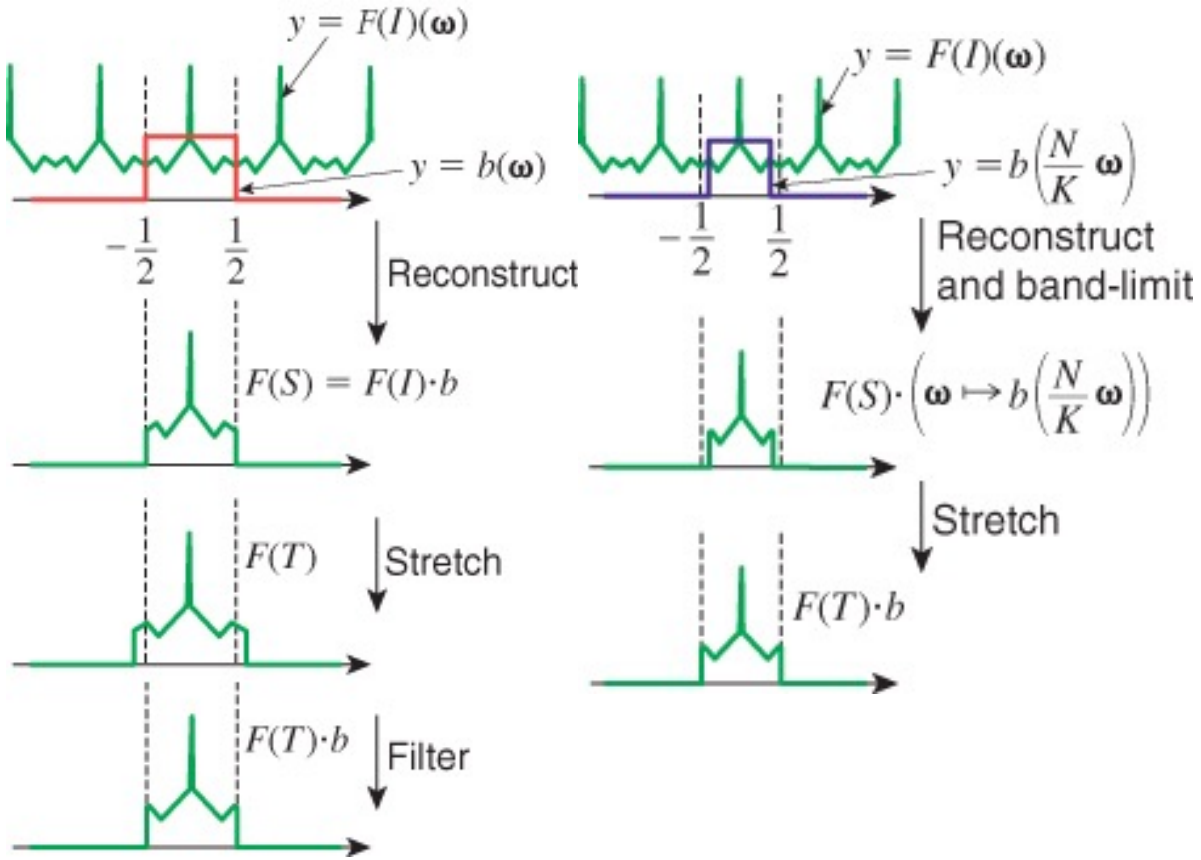




Scale down

- Depends on the method of re-scale
- Limited our band to show higher frequencies
- But can introduce new sharp details

Scaling Down



Ideal

Optimised

- We have to add an extra box filter
- In order to preserve the bandlimit

Boundary Effects

- What happens at the image boundary?
- Not enough pixels for filter
- So we have to invent / assume extra pixels
- And each method has it's side effects

Boundary Assumptions

- We can also limit the reconstruction filter
- Or we can cheat & re-weight

Extend by reflection

Original

Extend by constants

Extend by zeroes

Side-Effects

- Extend by Zeroes:
  - Leads to darkening around edge of image
- Extend by Reflection:
  - Not necessarily realistic
- Extend by Constants:
  - Tends to smear boundary inwards
- Finite Support Filter
  - Impossible to do in both domains at once

Band-Limited Filters

- We want a wider filter than linear
- But not as wide as sinc
- Iterate the box (tent) filter multiple times
- Leads to cubic B-spline filters
  - See 5821M for discussion

The Standard Hack

- Each filter assigns weights to pixels
- And *should* guarantee they sum to 1
  - (otherwise we lose/gain brightness overall)
- Omit filter pixels outside the image
  - This drops the sum below 1
  - So divide by the sum of the weights kept
- Result: interpolation only uses valid pixels

Images by:

- S2: Matt moloney
- S3: Kalea Jerielle
- S6: Zachary Keimig
- S40: Nathan dumlao