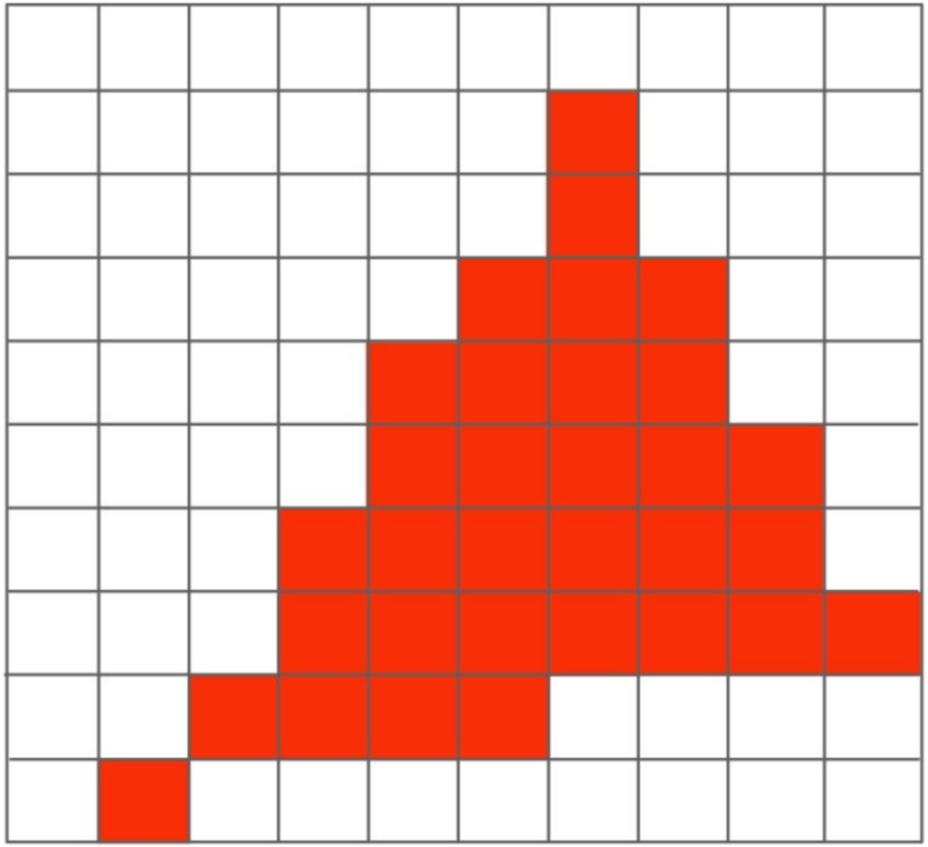
Sampling Theory - CSE160 – Nov 24

- What aliasing looks like
- Sampling a function
- Reconstructing a function
- Supersampling
- Representing a function as sines and cosines
- Filtering (frequency domain)
- Pre-filtering for anti-aliasing
- Convolution Theorem
- Administrative
- Q&A
- (last time ended 20 min early)

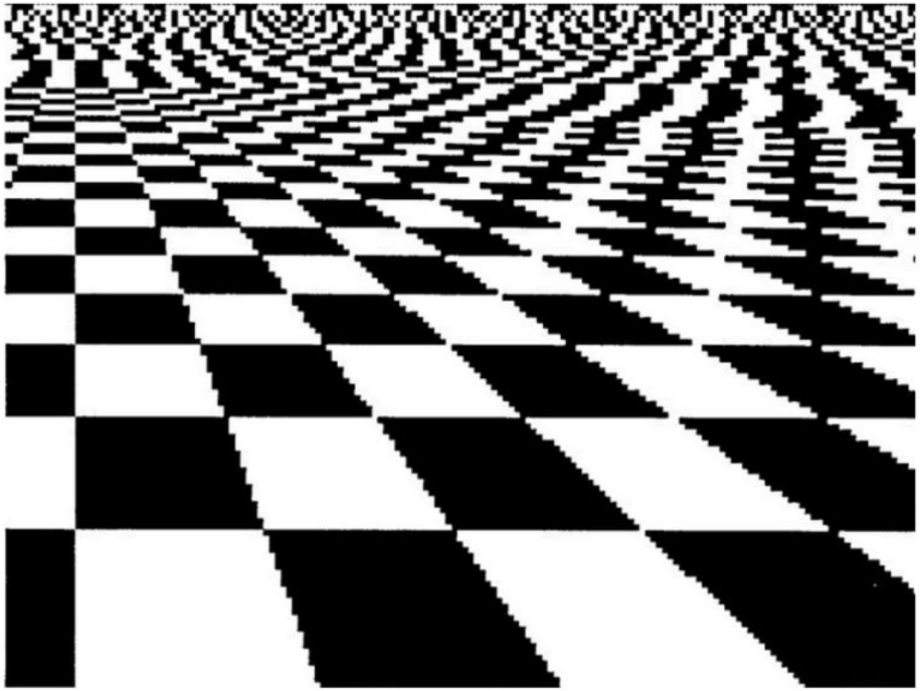
What aliasing looks like

What's wrong with this picture?

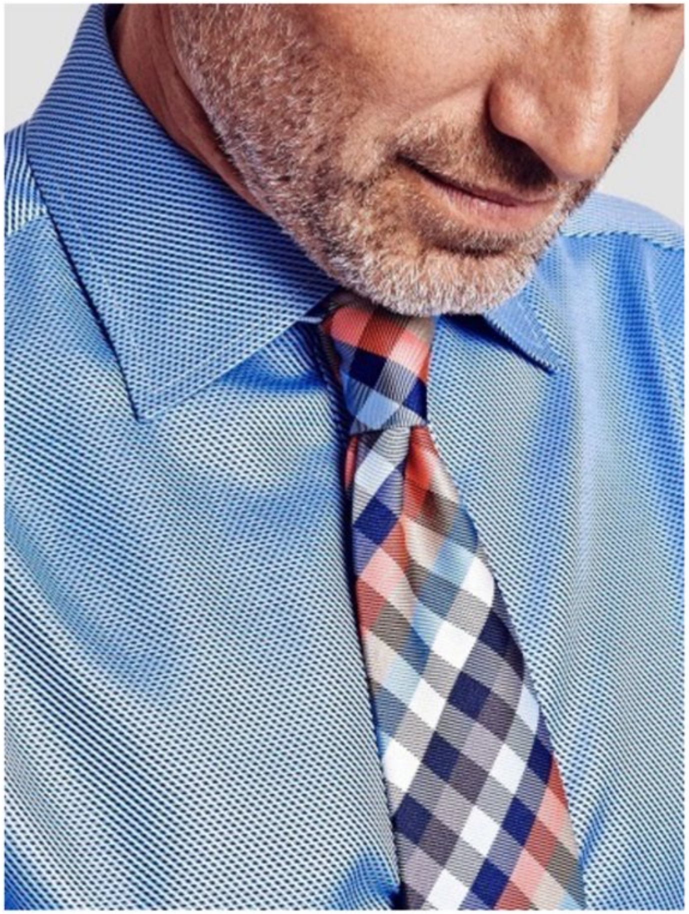


Jaggies!

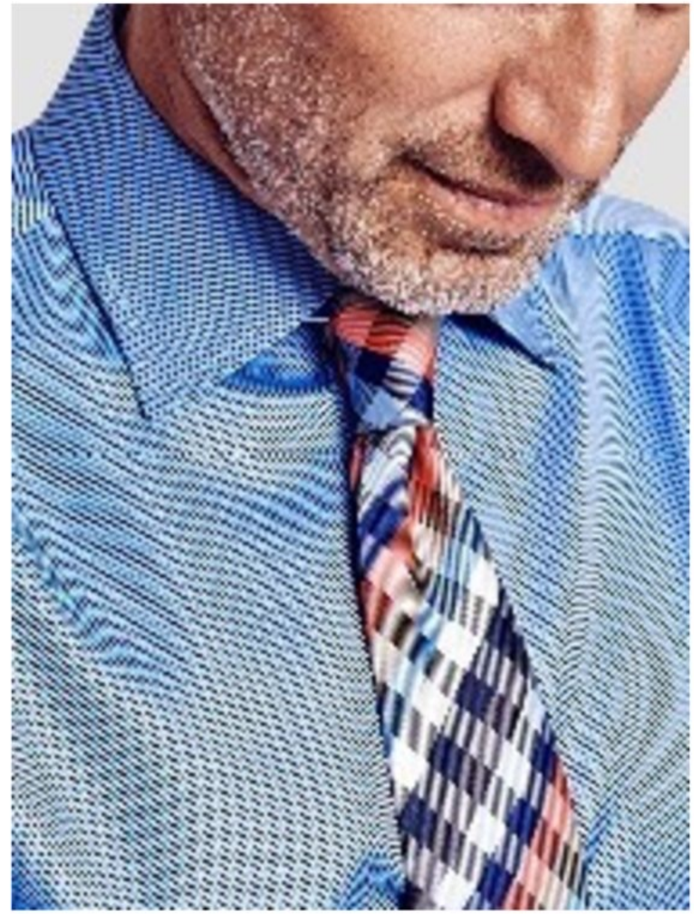
Jaggies (staircase pattern)



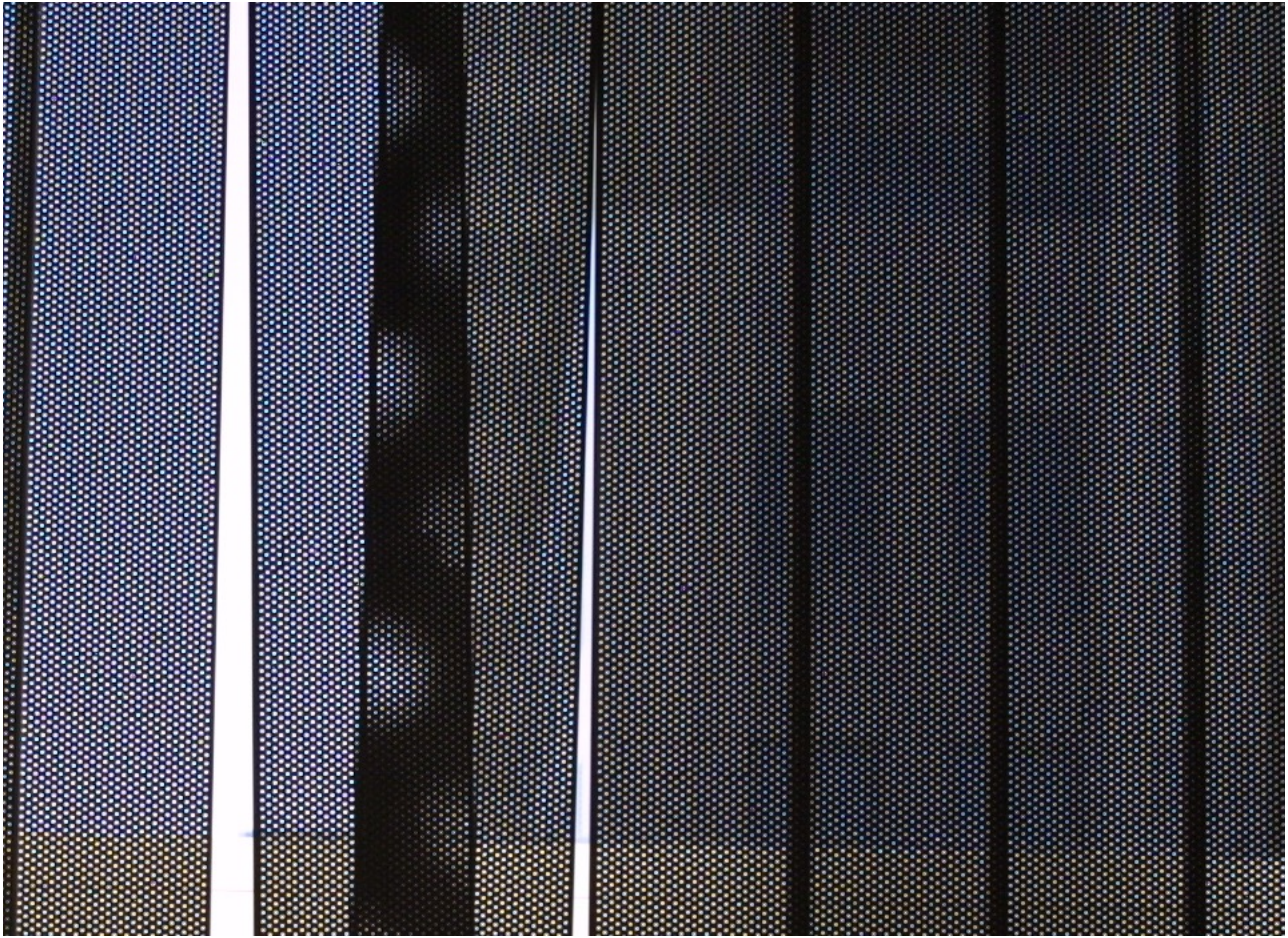
Moiré patterns in imaging



Full resolution image

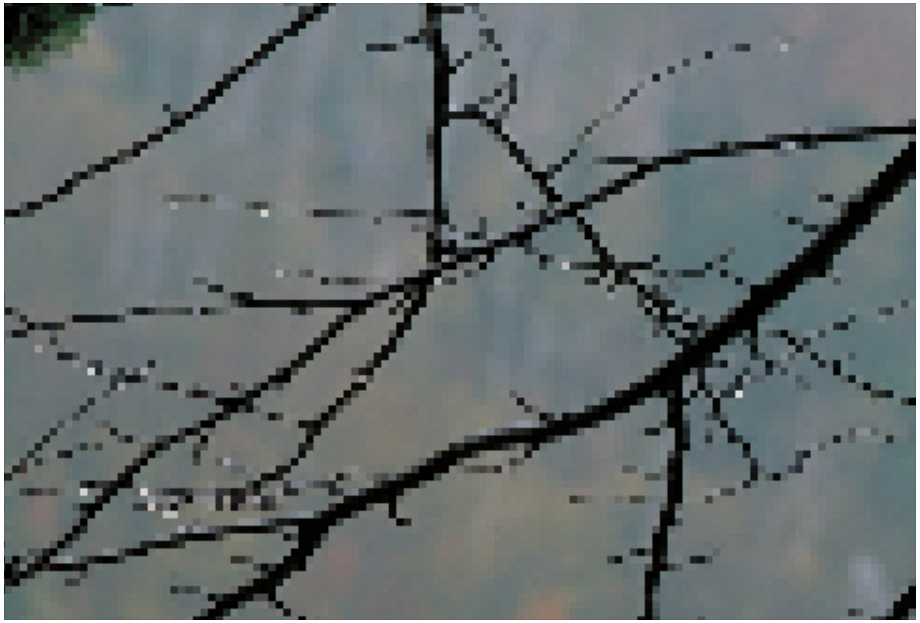


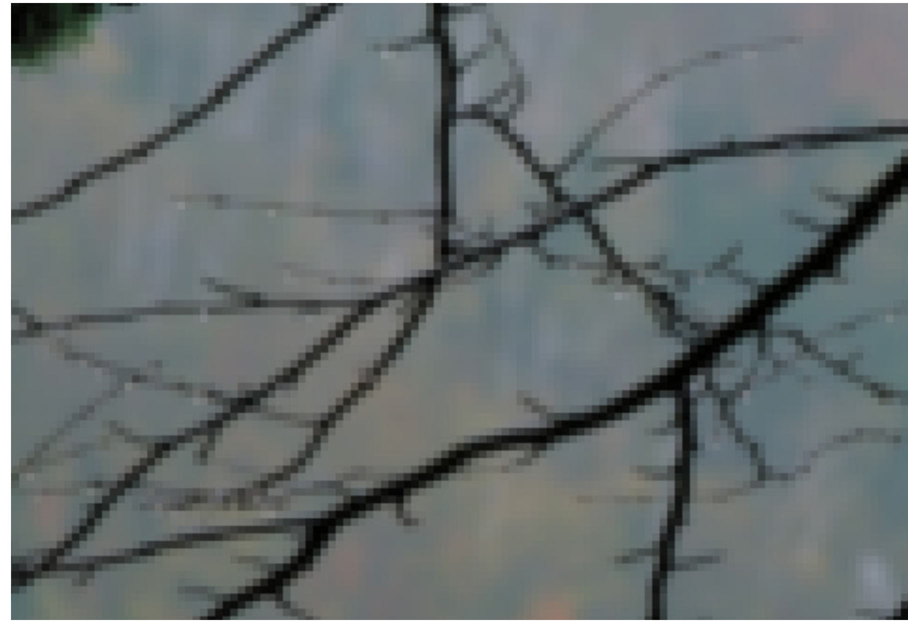
1/2 resolution image:
skip pixel odd rows and columns



[Philip Greenspun]

1000 pixel width





[Philip Greenspun]

by dropping pixels

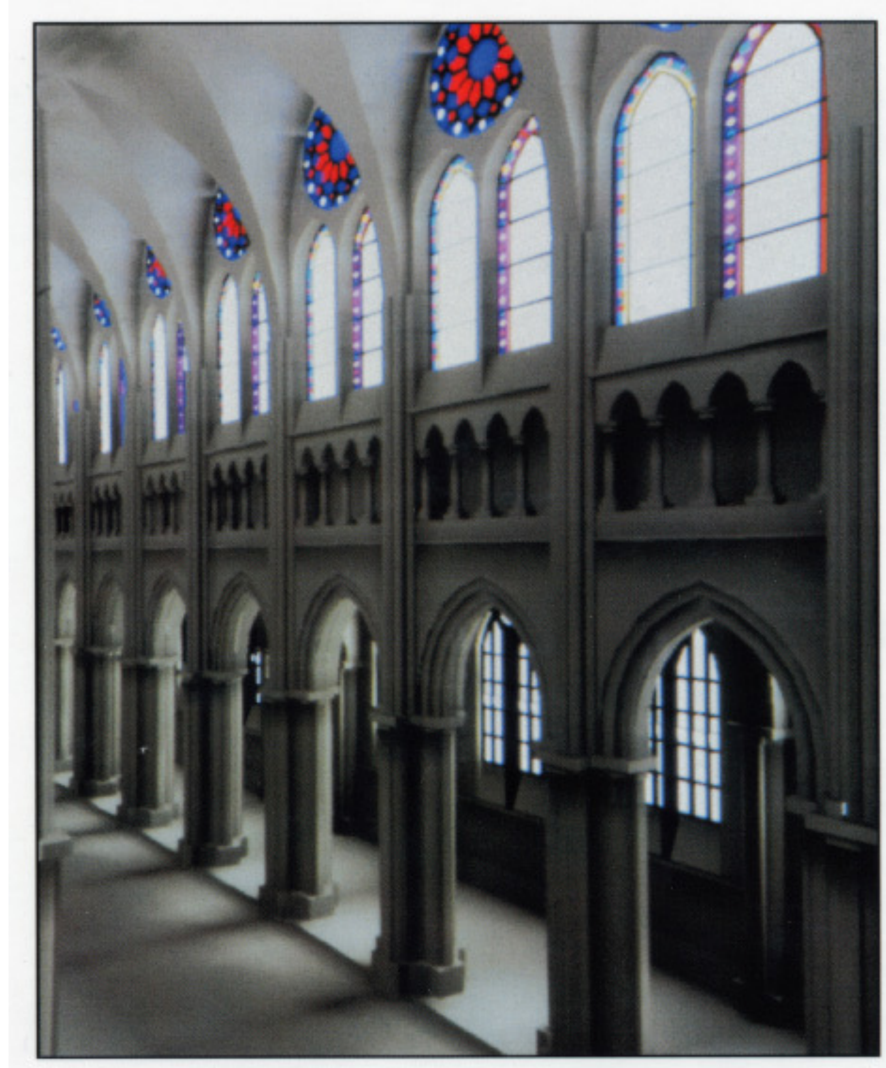
gaussian filter

250 pixel width

[Hearn & Baker cover]

600ppi scan of a color halftone image

by dropping pixels



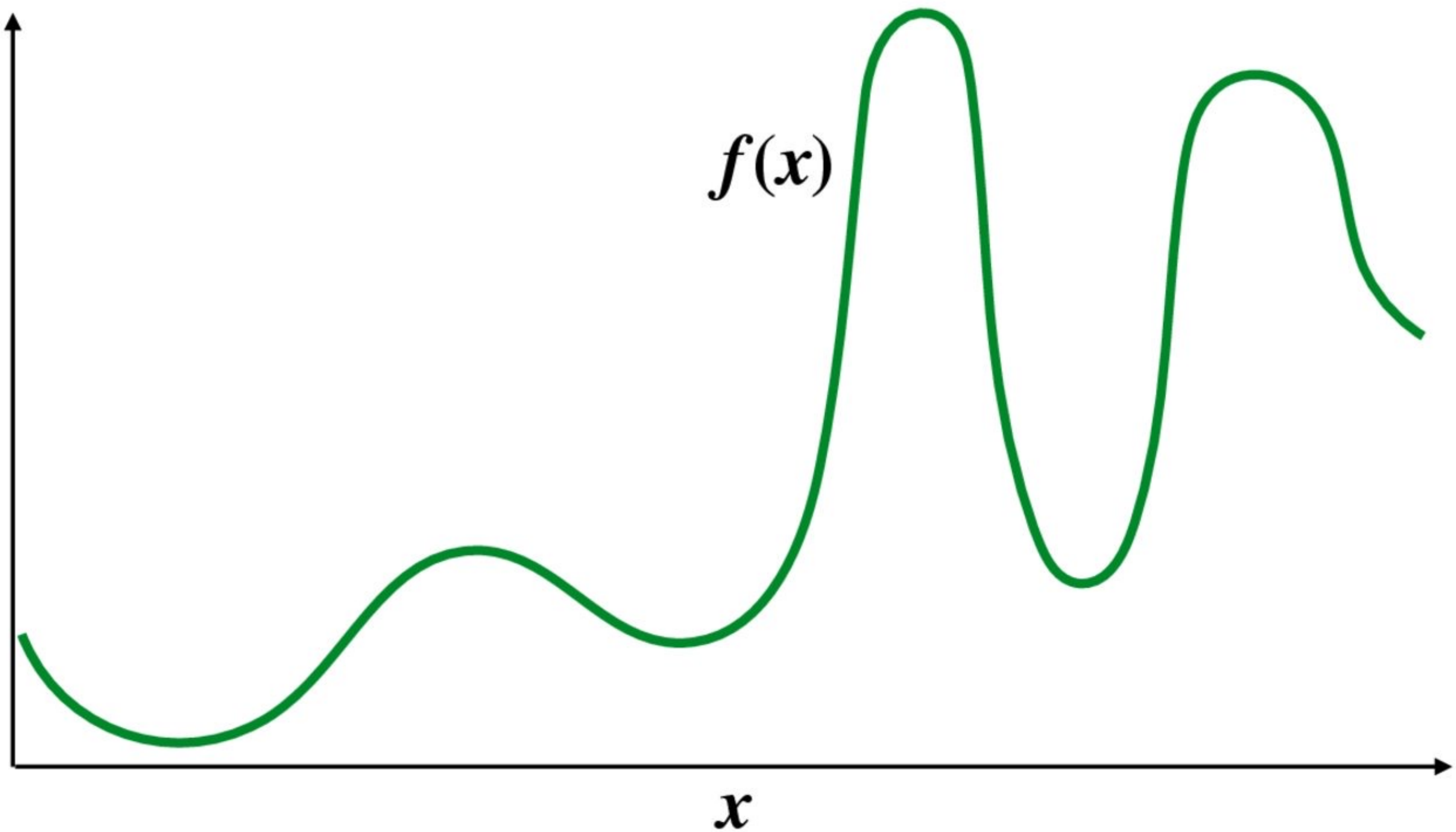
gaussian filter

[Hearn & Baker cover]

downsampling a high resolution scan

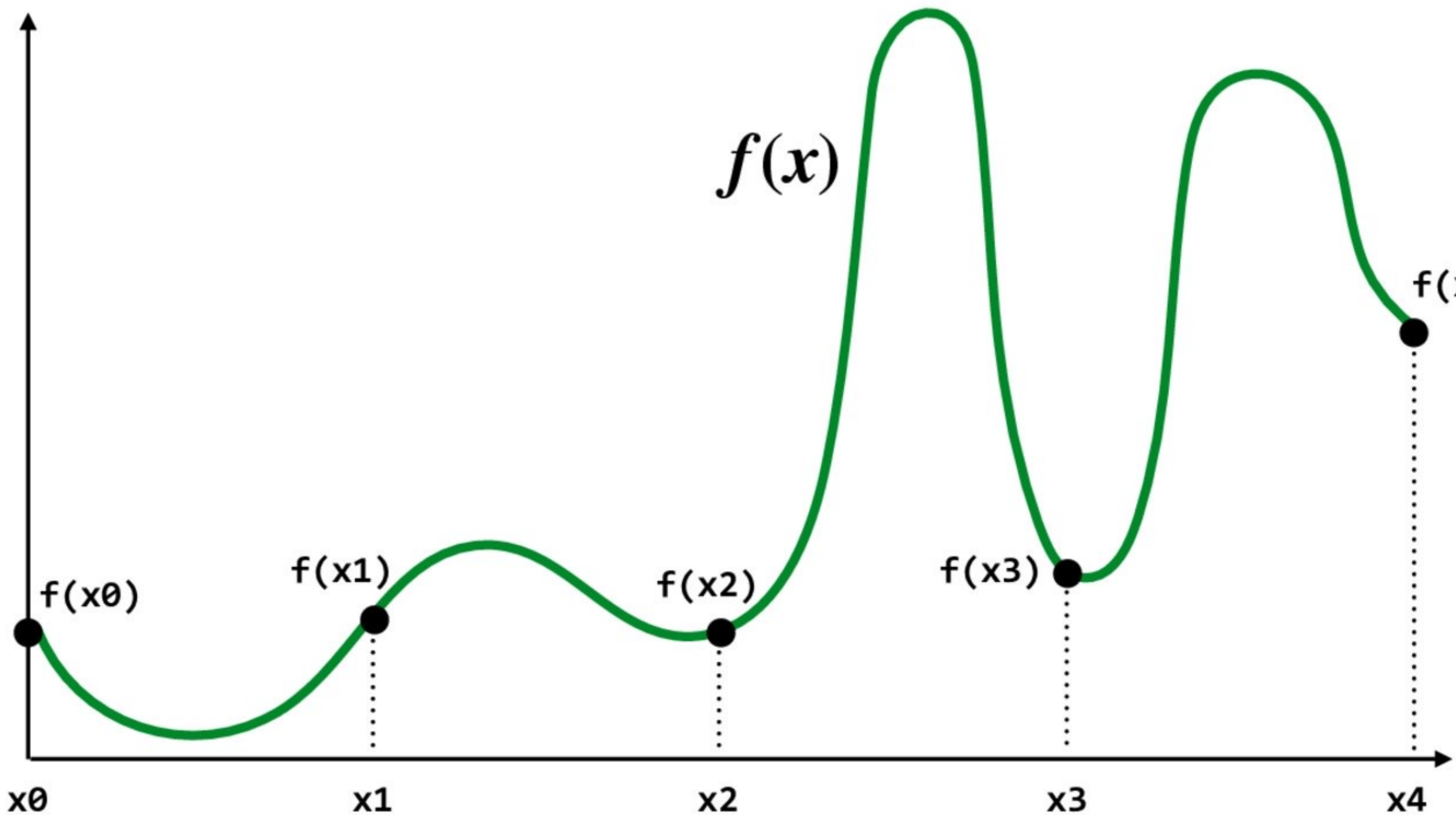
Sampling a function

Consider a 1D signal: f(x)



Sampling: taking measurements of a signal

Below: five measurements ("samples") of $f(x)$



Audio file: stores samples of a 1D signal

Audio is often sampled at 44.1 KHz

Sampling a function

- Evaluating a function at a point is sampling the function's value

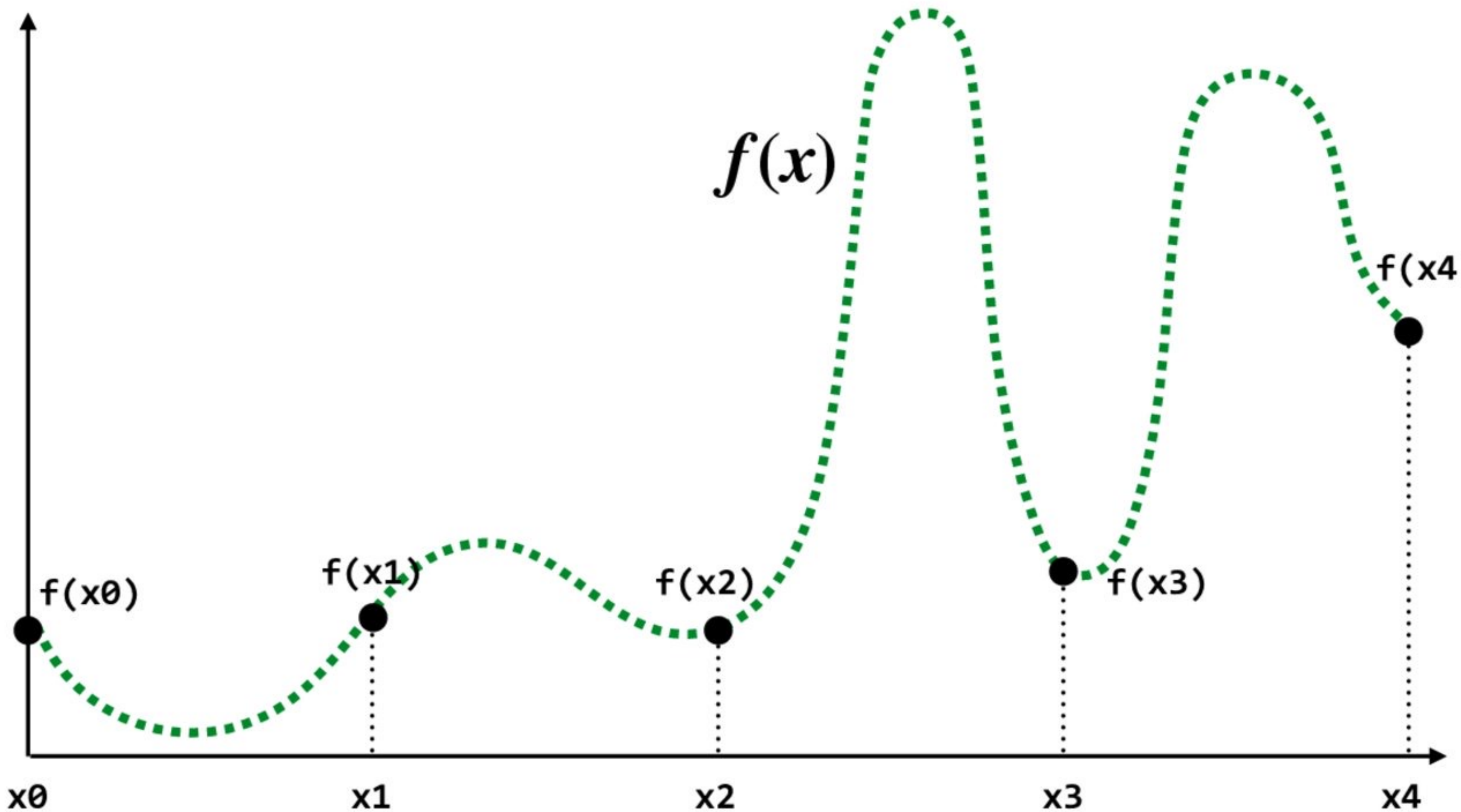
- We can discretize a function by periodic sampling

```
for(int x = 0; x < xmax; x++)
    output[x] = f(x);
```

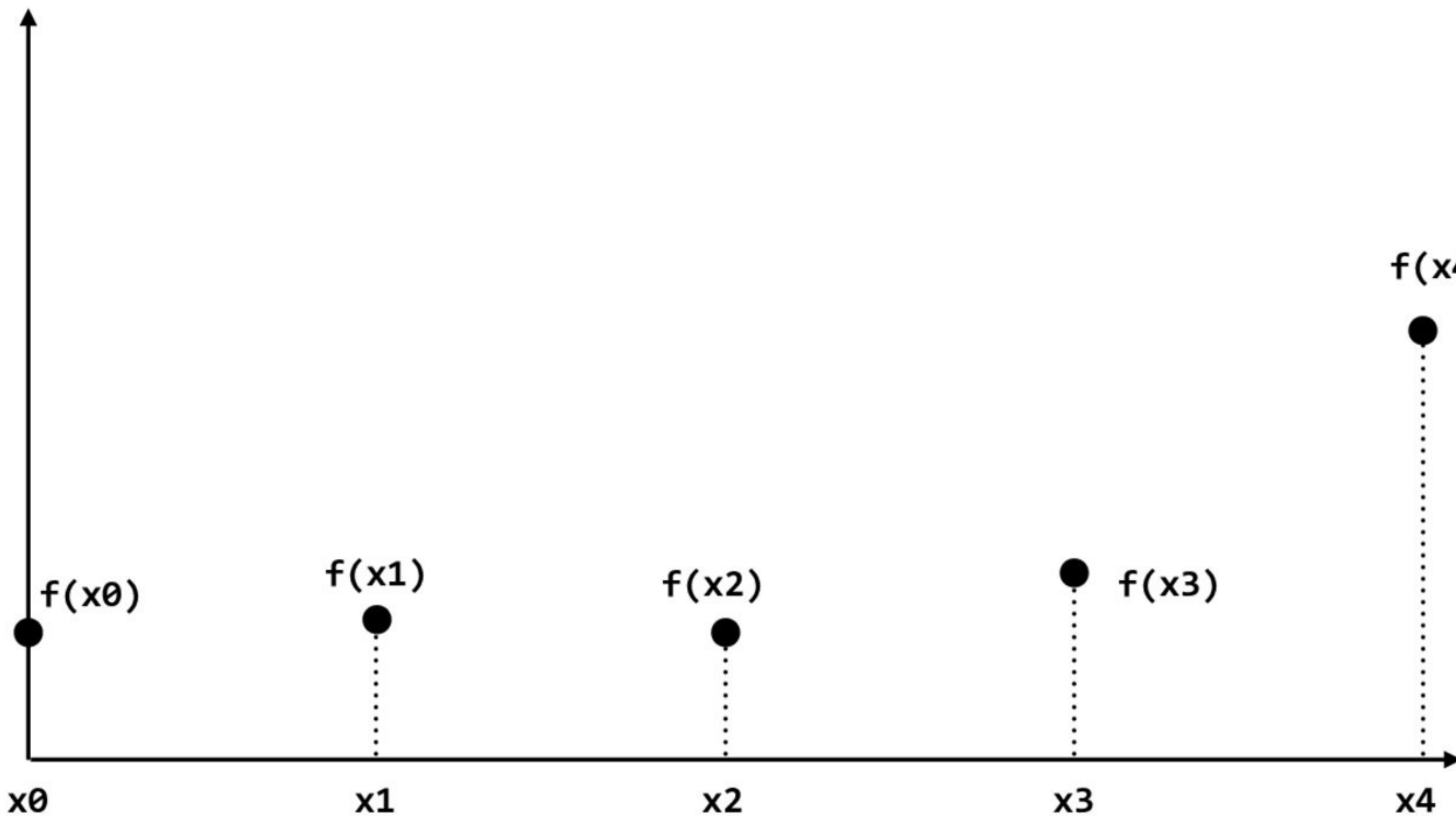
- Sampling is a core idea in graphics. In this class we'll sample time (1D), area (2D), angle (2D), volume (3D), etc ...

Reconstructing a function

Reconstruction: given a set of samples, how might we attempt to reconstruct the original signal $f(x)$?



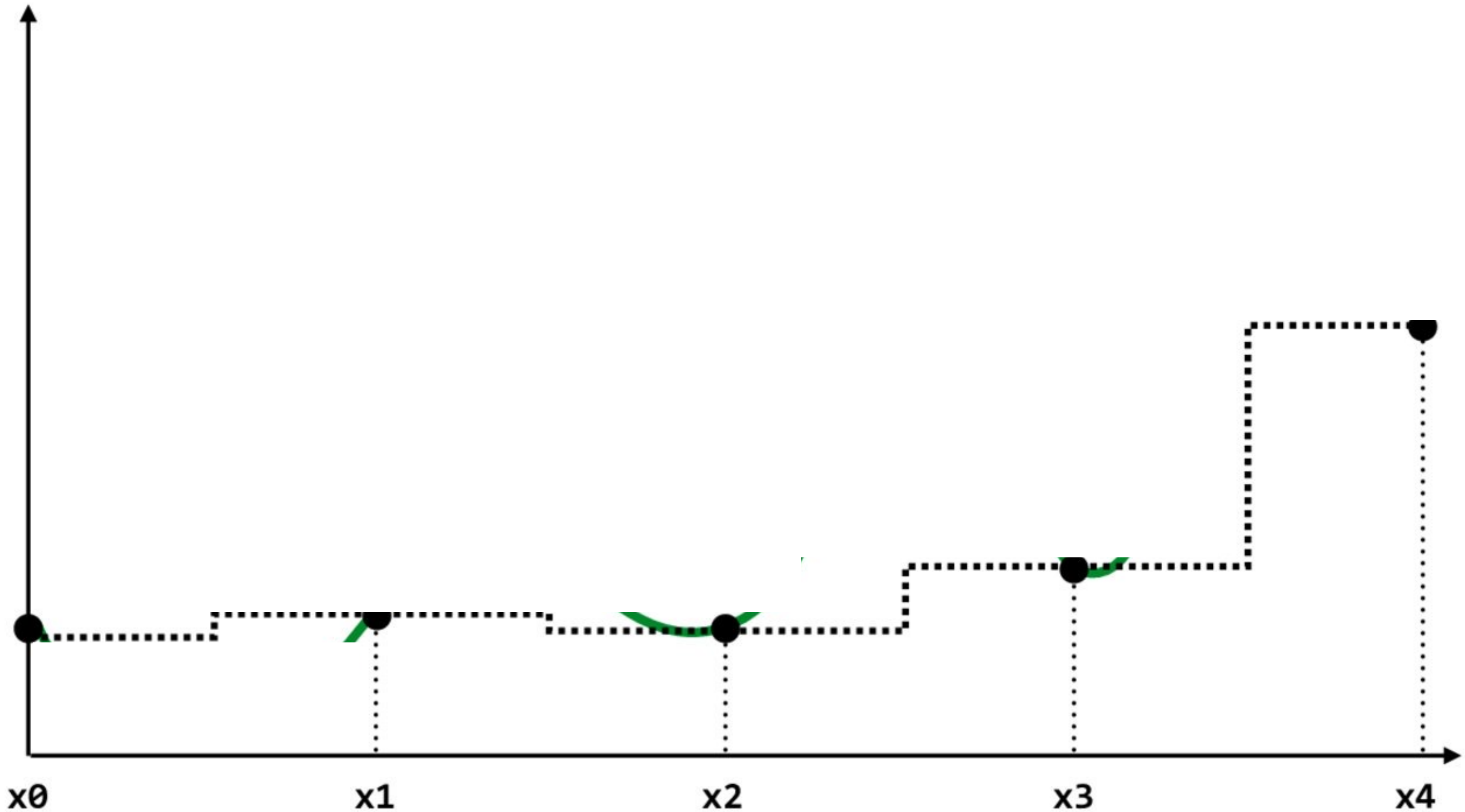
Reconstruction: given a set of samples, how might we attempt to reconstruct the original signal $f(x)$?



Piecewise constant approximation

$f_{recon}(x) =$ value of sample closest to x

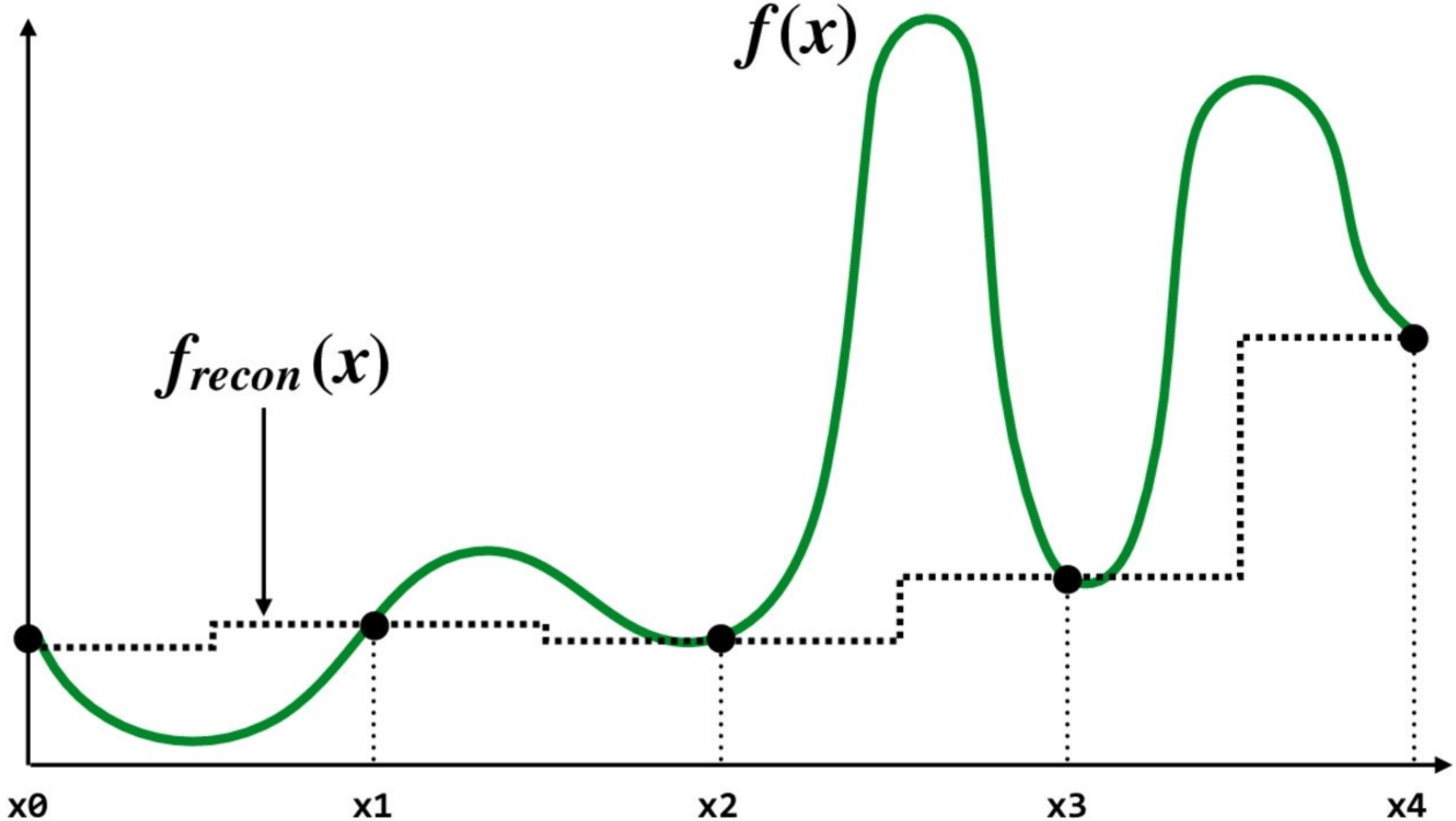
$f_{recon}(x)$ approximates $f(x)$



Piecewise constant approximation

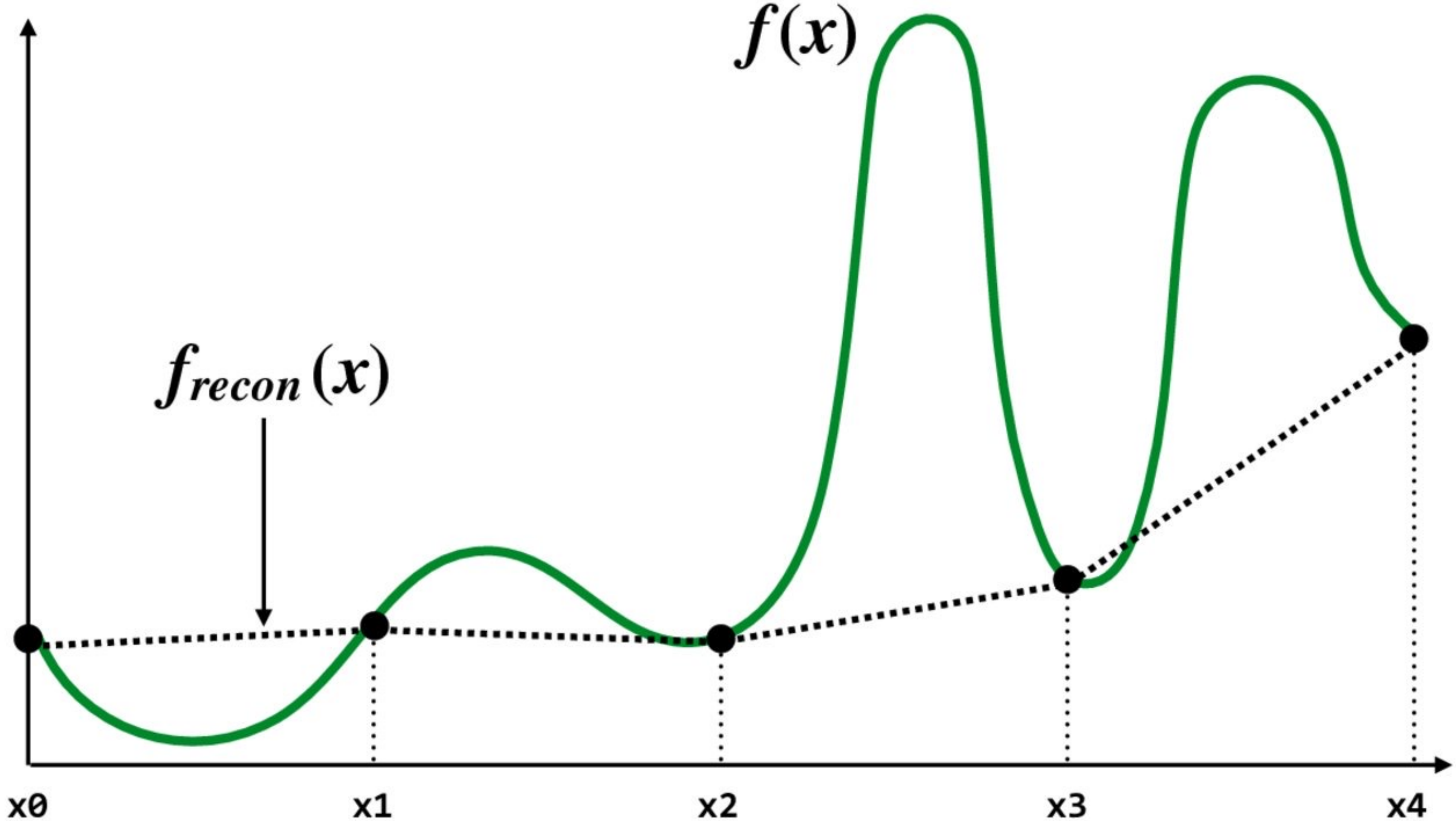
$f_{recon}(x)$ = value of sample closest to x

$f_{recon}(x)$ approximates $f(x)$

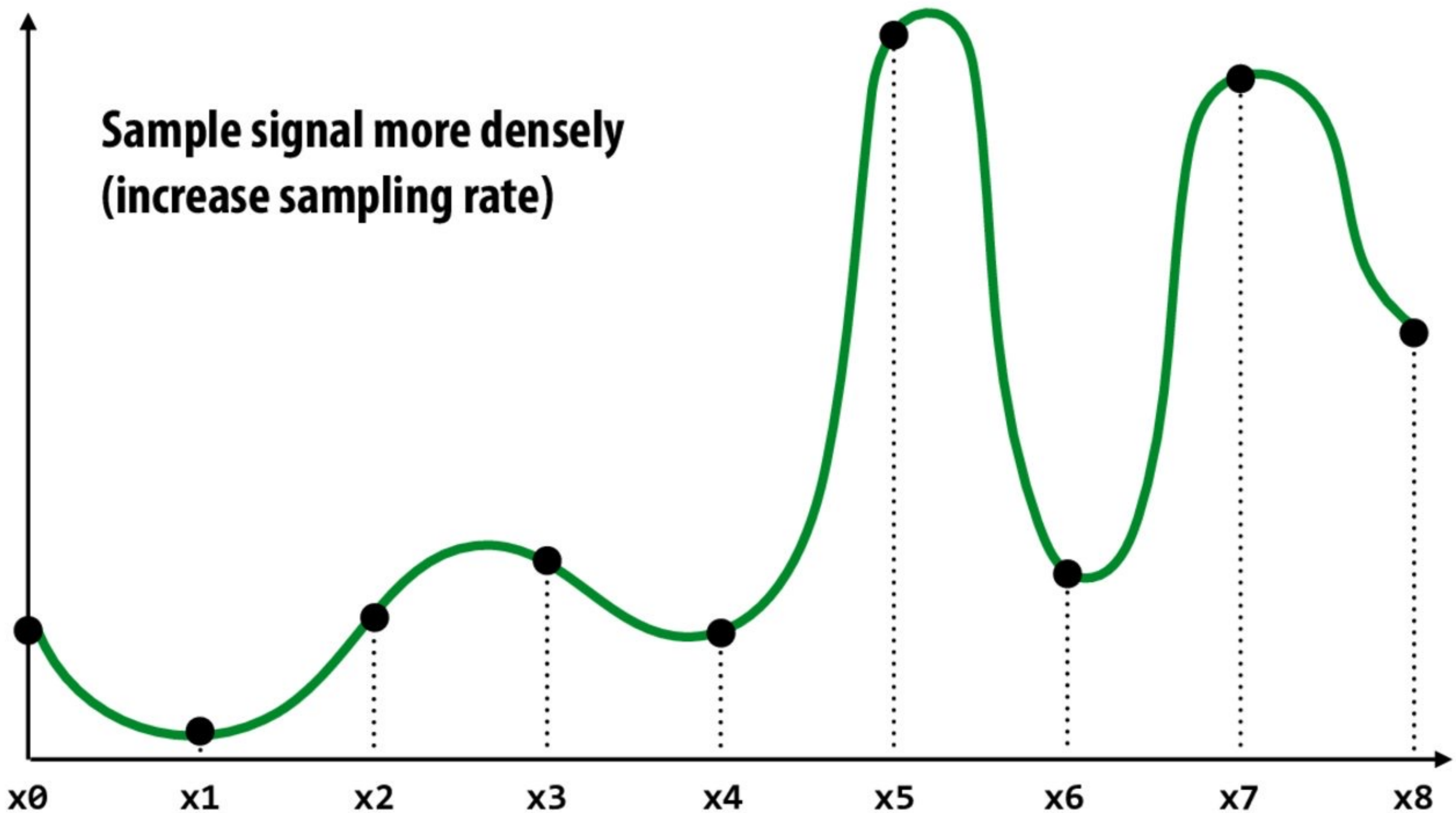


Piecewise linear approximation

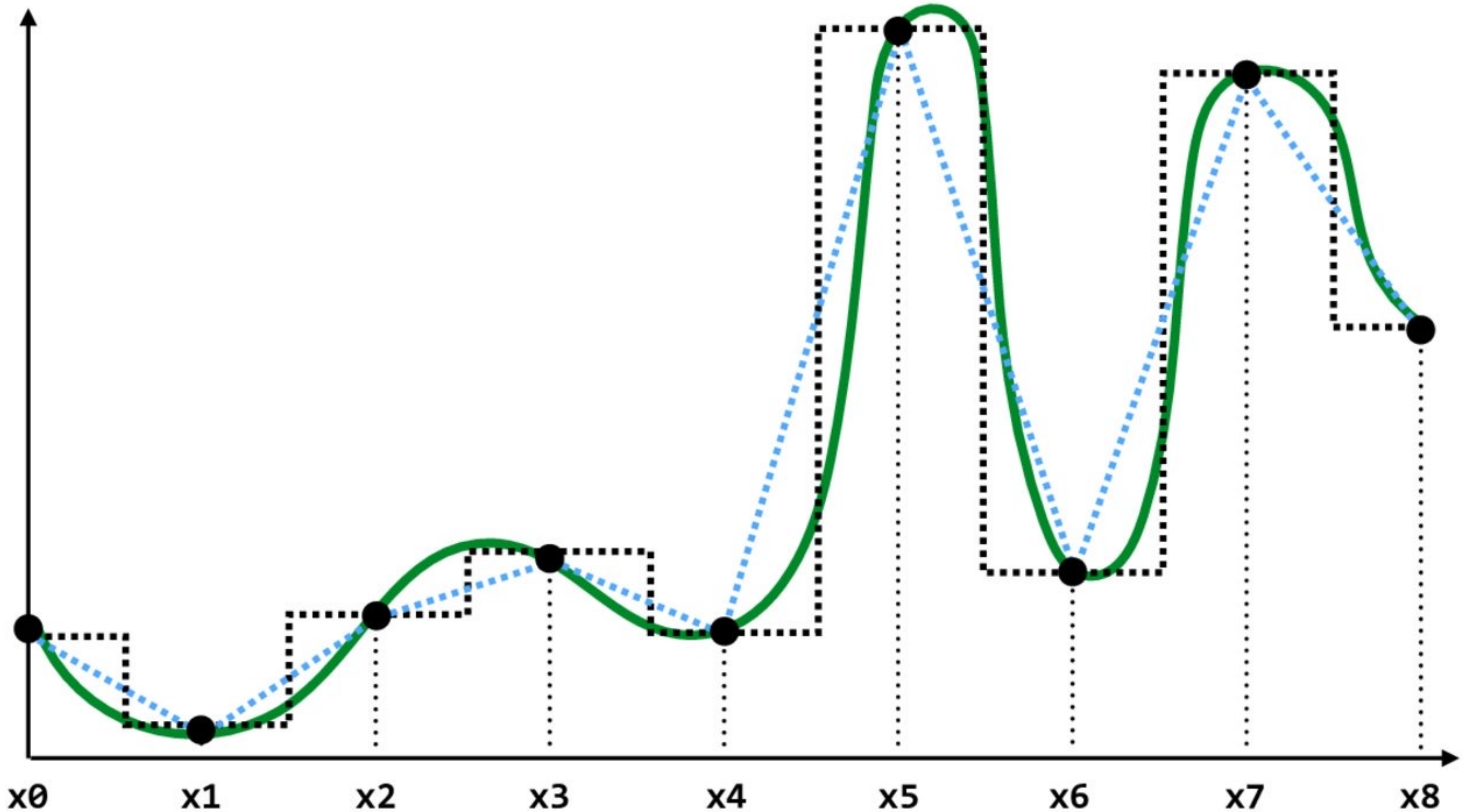
$f_{recon}(x)$ = linear interpolation between values of two closest samples to x



How can we represent the signal more accurately?



More accurate reconstructions result from denser sampling



= reconstruction via nearest neighbor

= reconstruction via linear interpolation

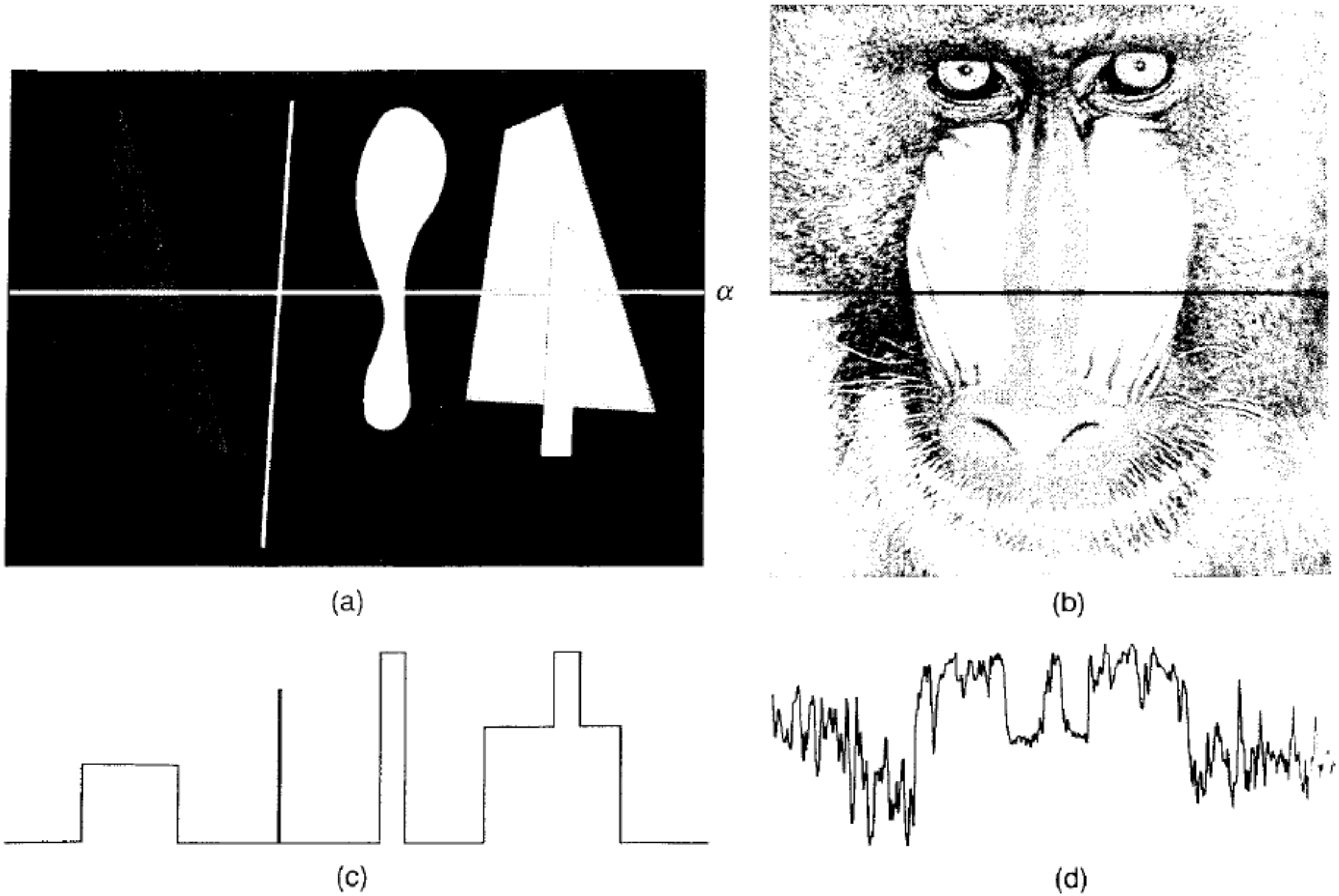


Fig. 14.8 Image. (a) Graphical primitives. (b) Mandrill. (c) Intensity plot of scan line α in (a). (d) Intensity plot of scan line α in (b). (Part d is courtesy of George Wolberg, Columbia University.)

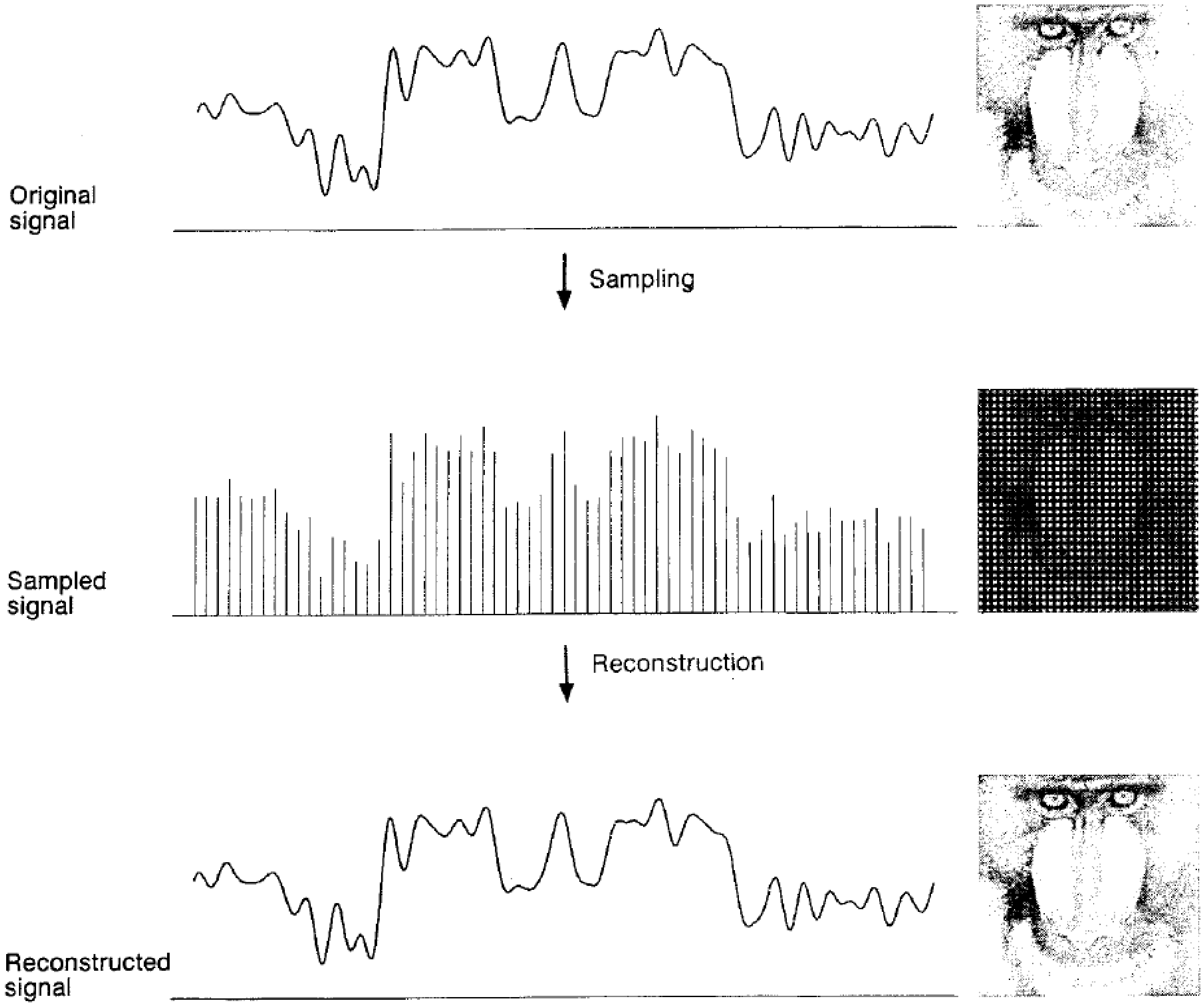
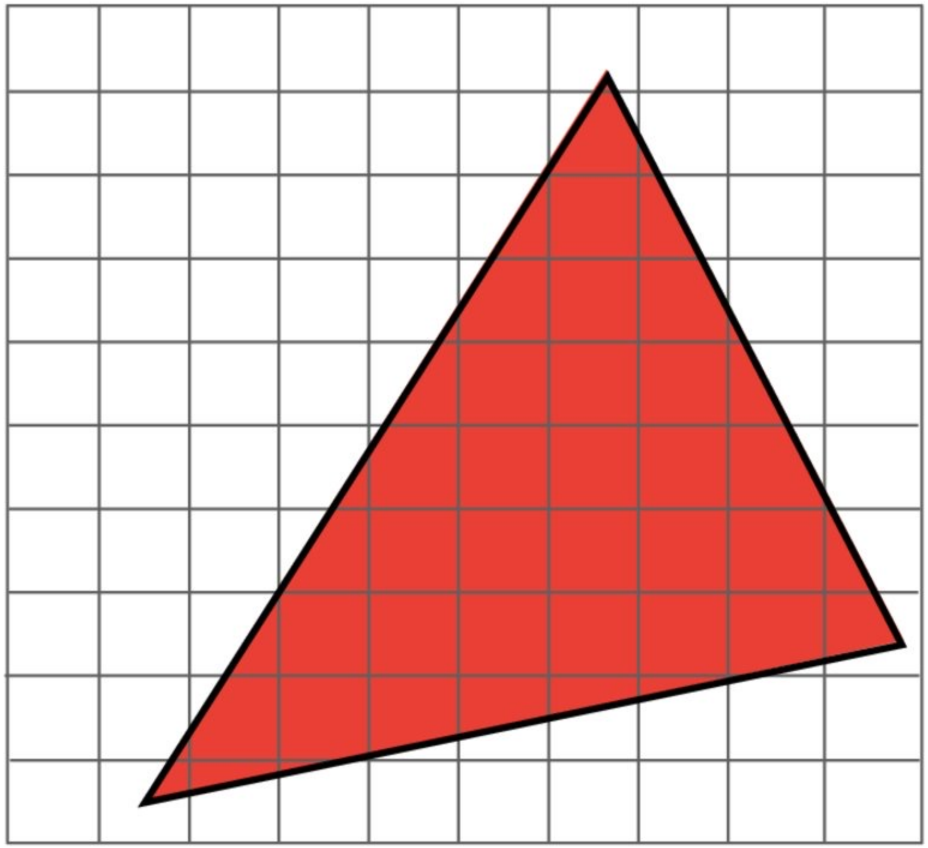
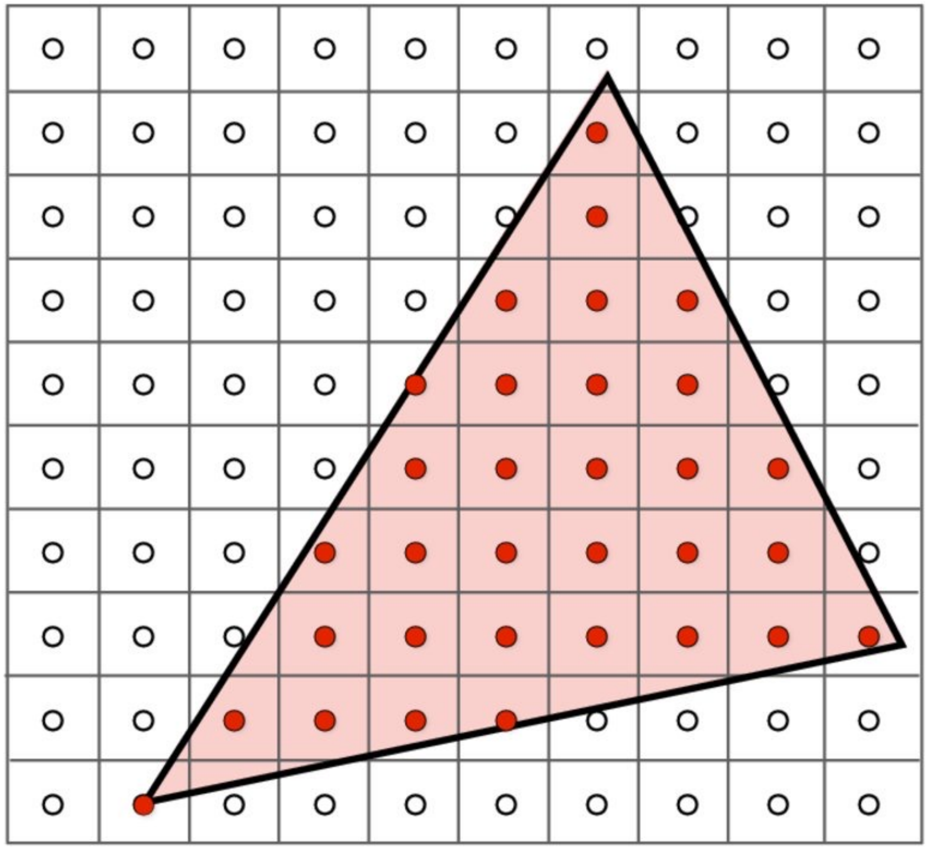


Fig. 14.9 The original signal is sampled, and the samples are used to reconstruct the signal. (Sampled 2D image is an approximation, since point samples have no area.) (Courtesy of George Wolberg, Columbia University.)

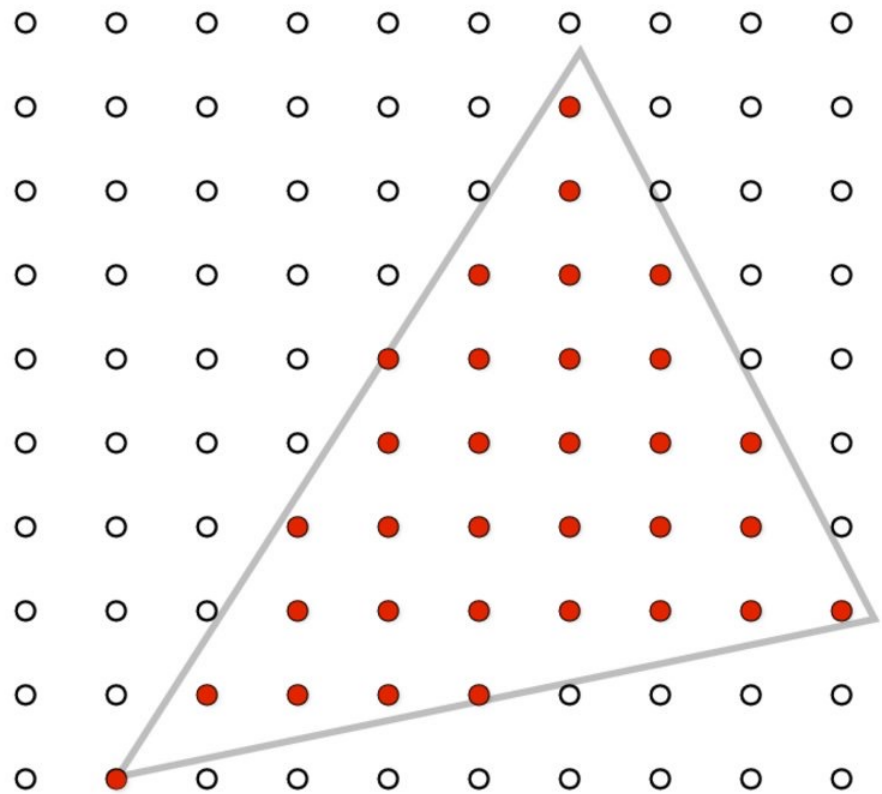
Drawing a triangle by 2D sampling



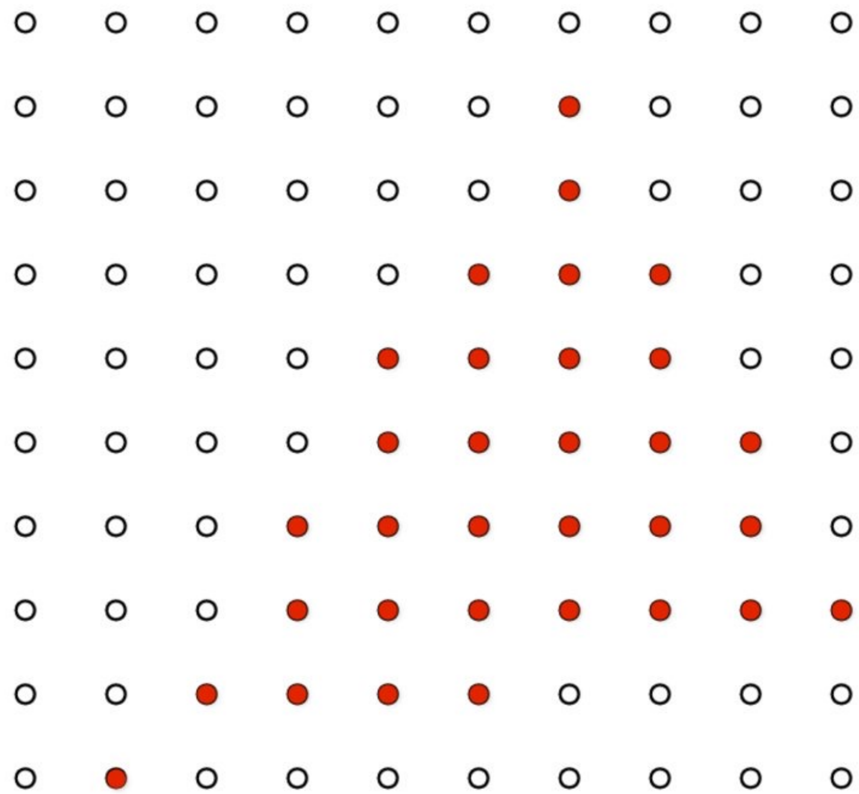
Sample coverage at pixel centers



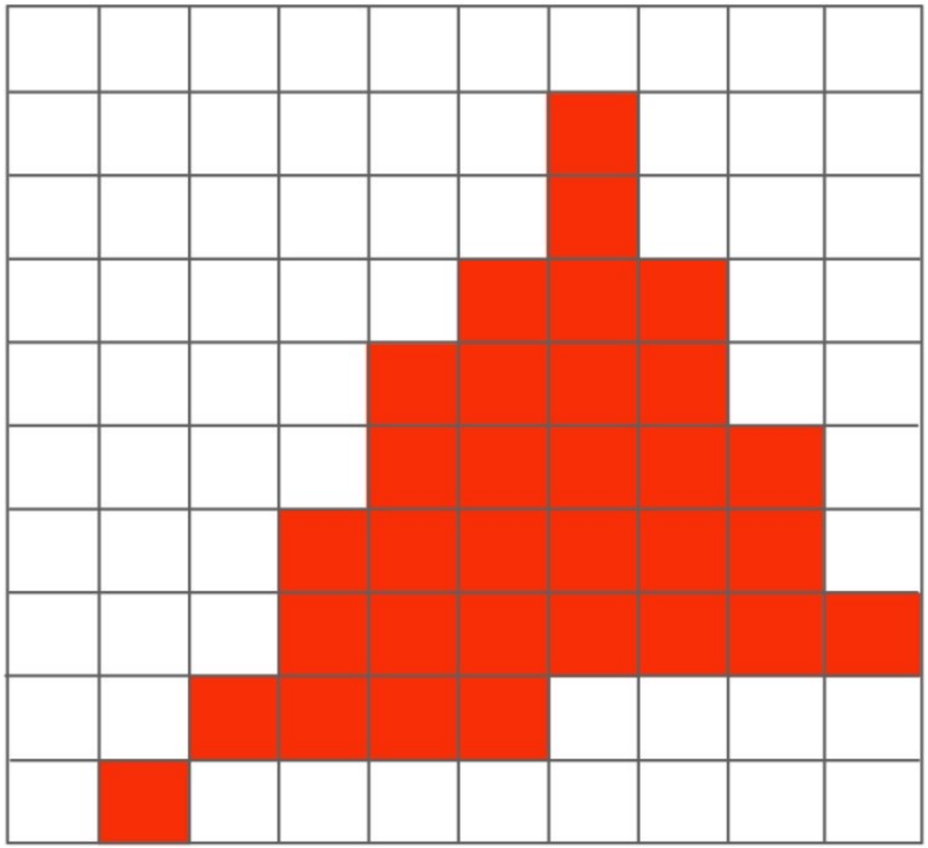
Sample coverage at pixel centers



So, if we send the display this sampled signal

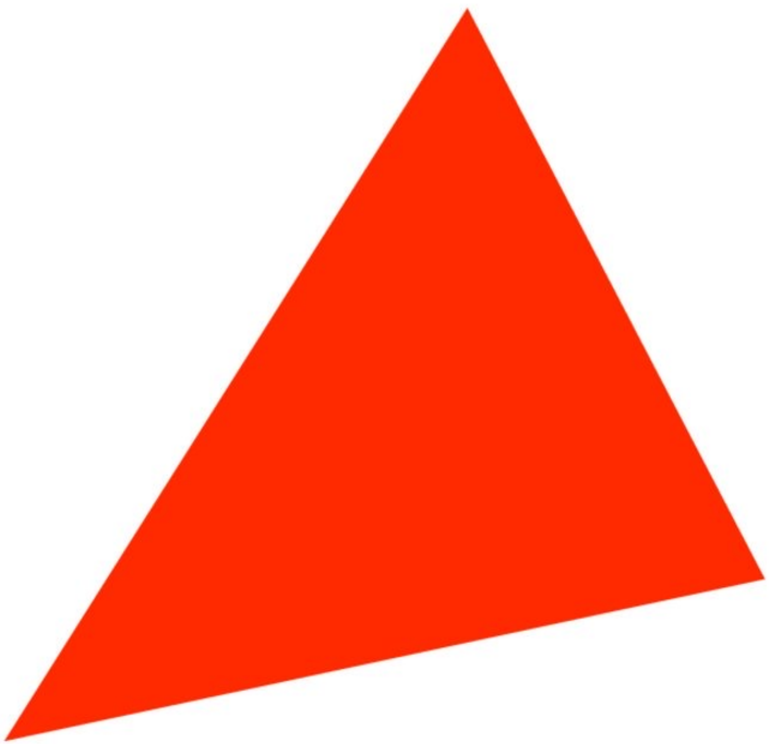


The display physically emits this signal



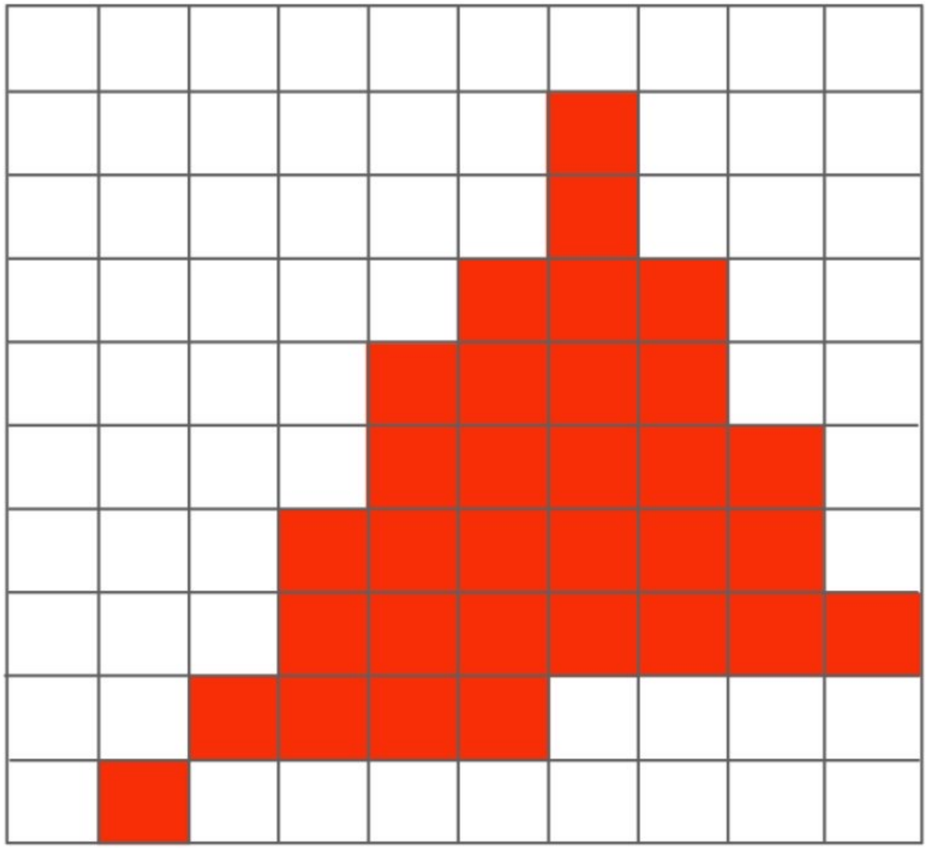
Given our simplified "square pixel" display assumption, we've effectively performed a piecewise constant reconstruction

Compare: the continuous triangle function



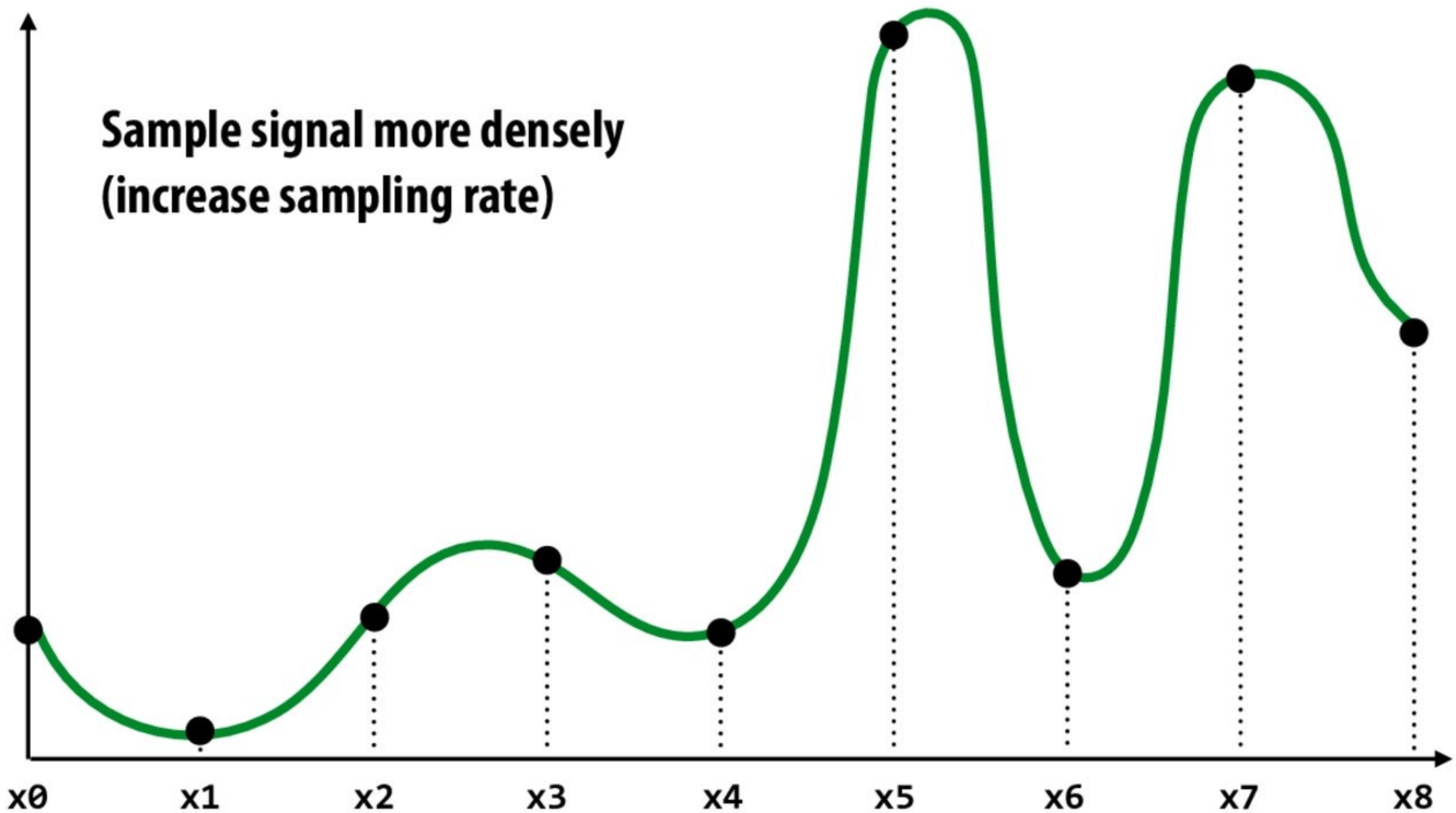
Super-sampling

What's wrong with this picture?

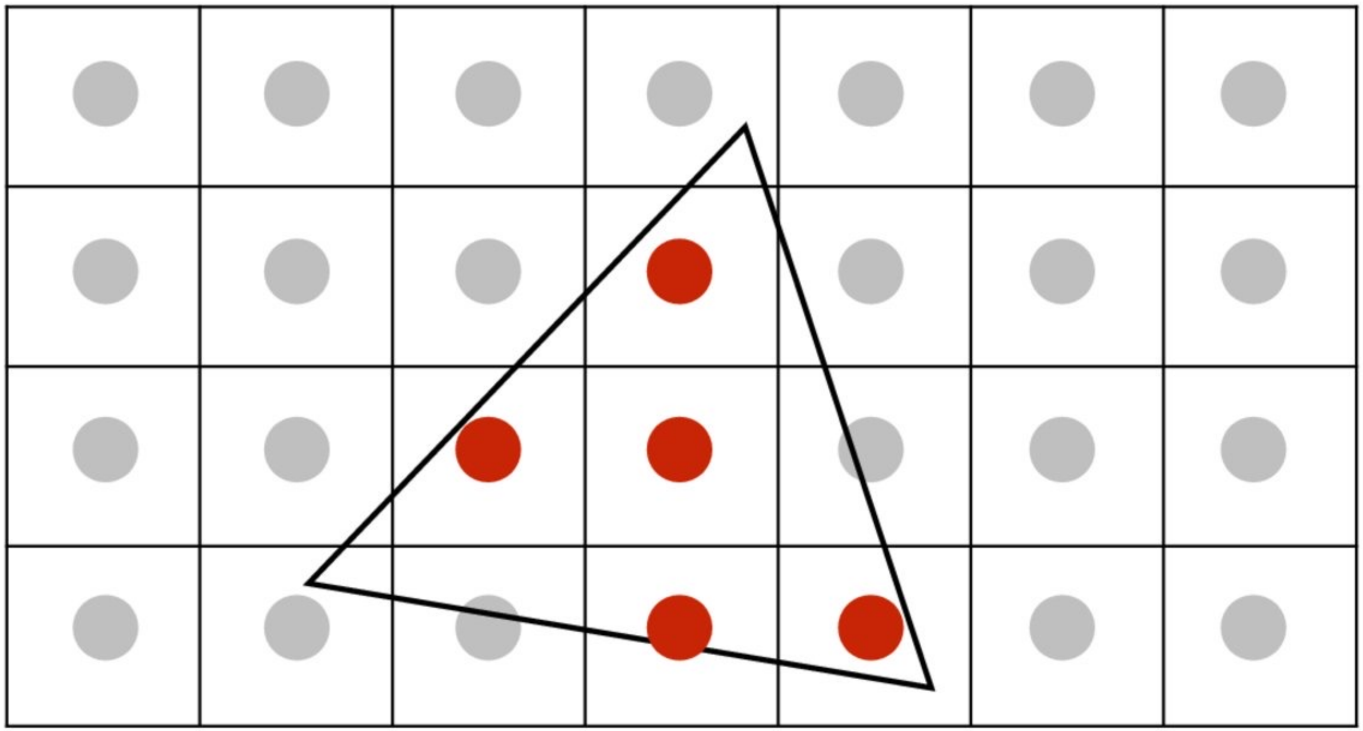


Jaggies!

Reminder: how can we represent a sampled signal more accurately?



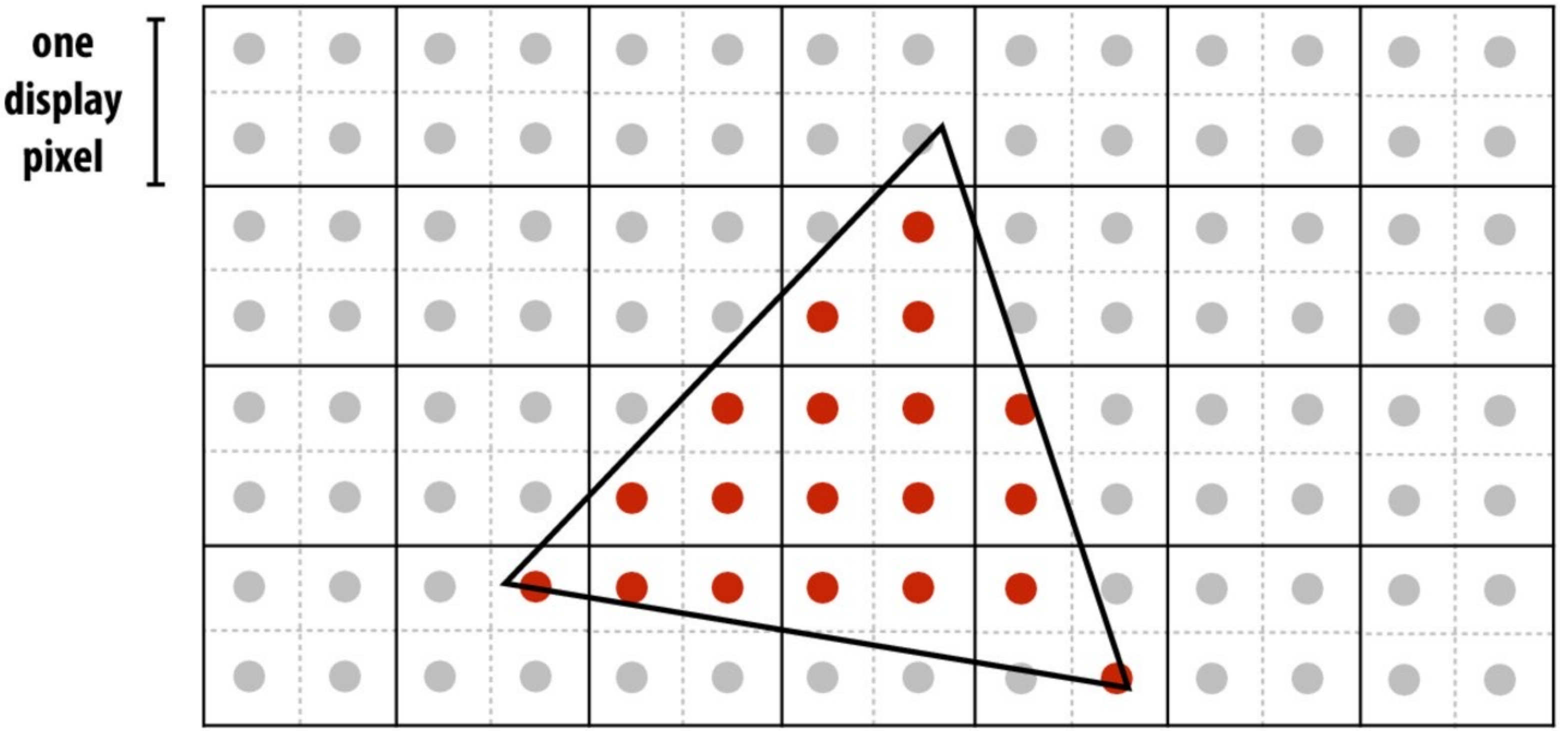
Point sampling: one sample per pixel



Supersampling: step 1

Take NxN samples in each pixel

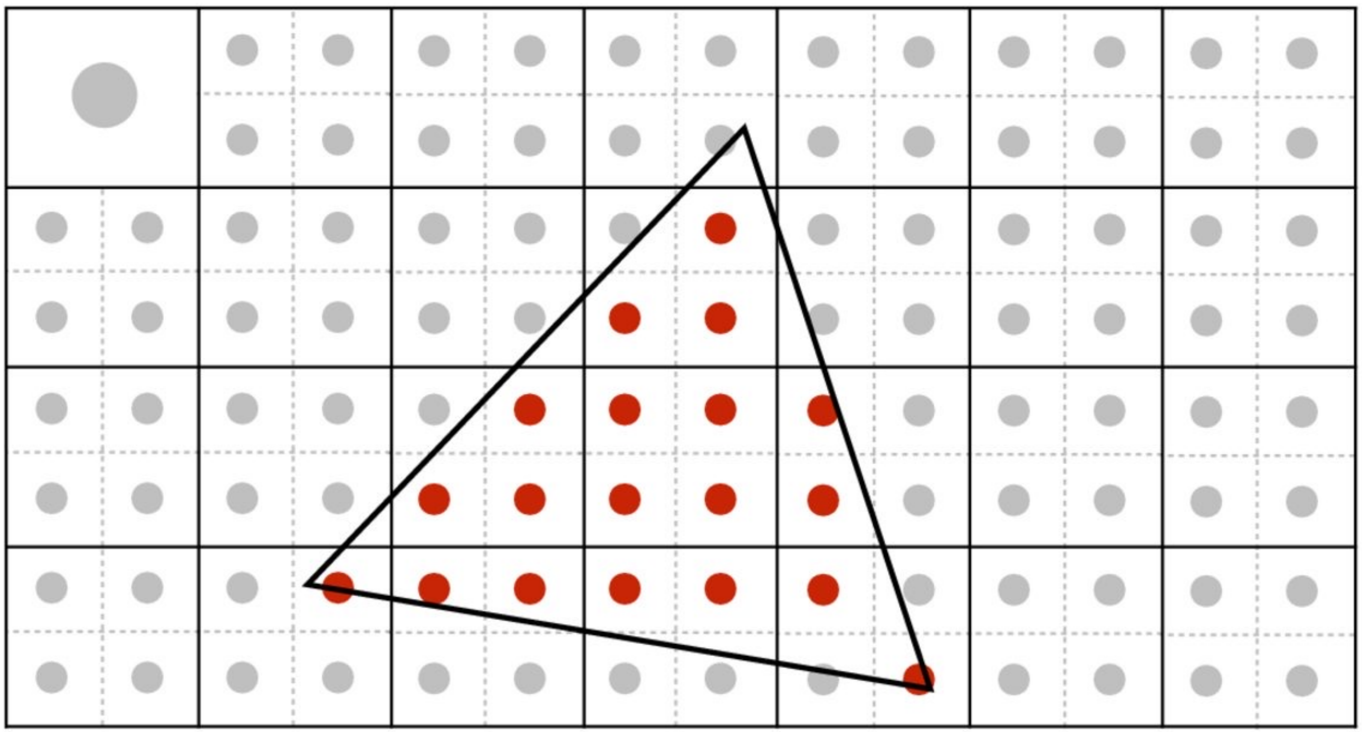
(but… how do we use these samples to drive a display, since there are four times more samples than display pixels!)



2x2 supersampling

Supersampling: step 2

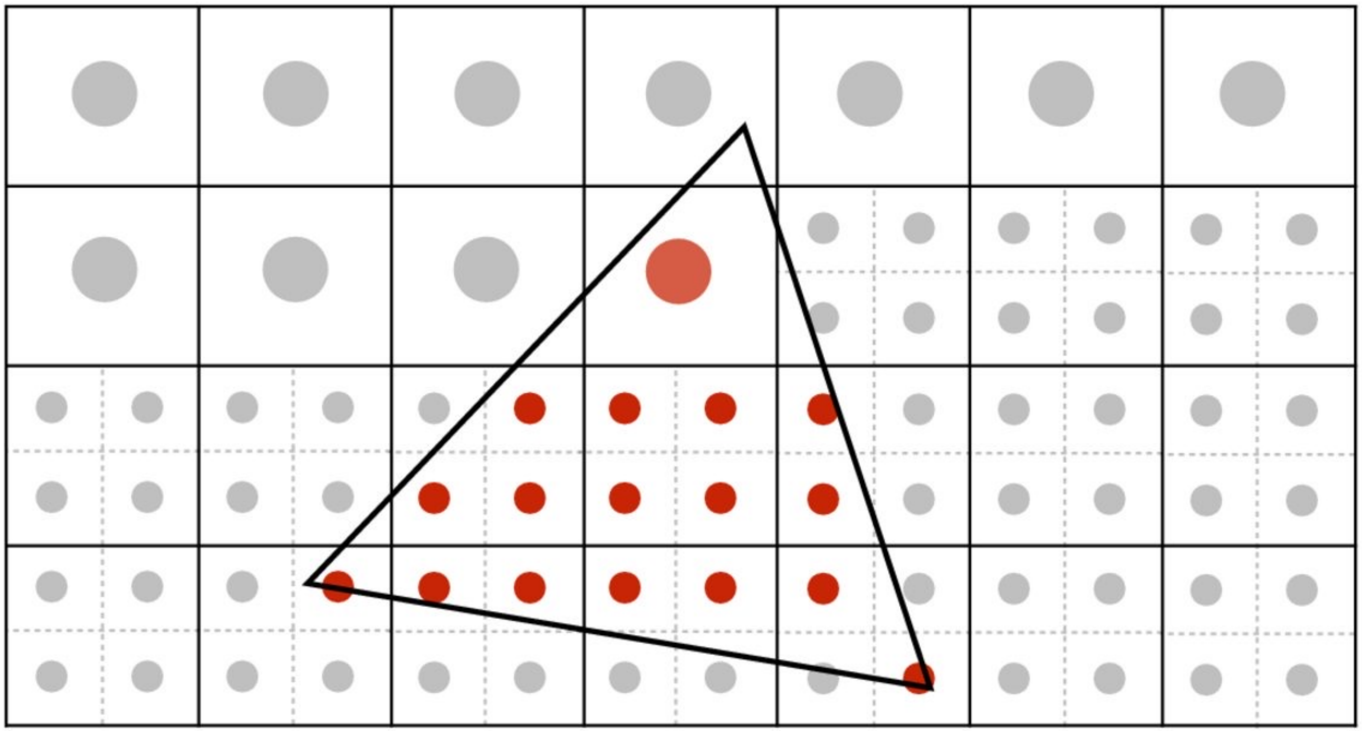
Average the NxN samples "inside" each pixel



Averaging down

Supersampling: step 2

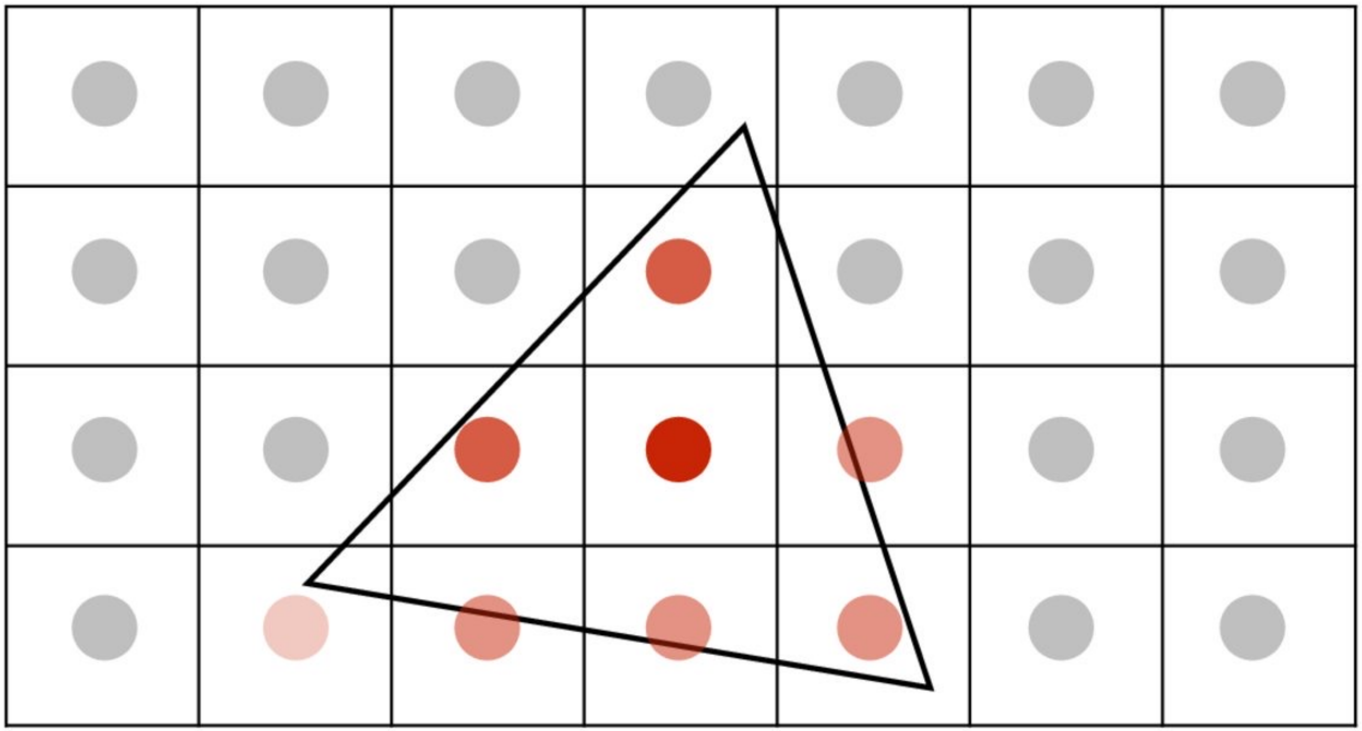
Average the NxN samples "inside" each pixel



Averaging down

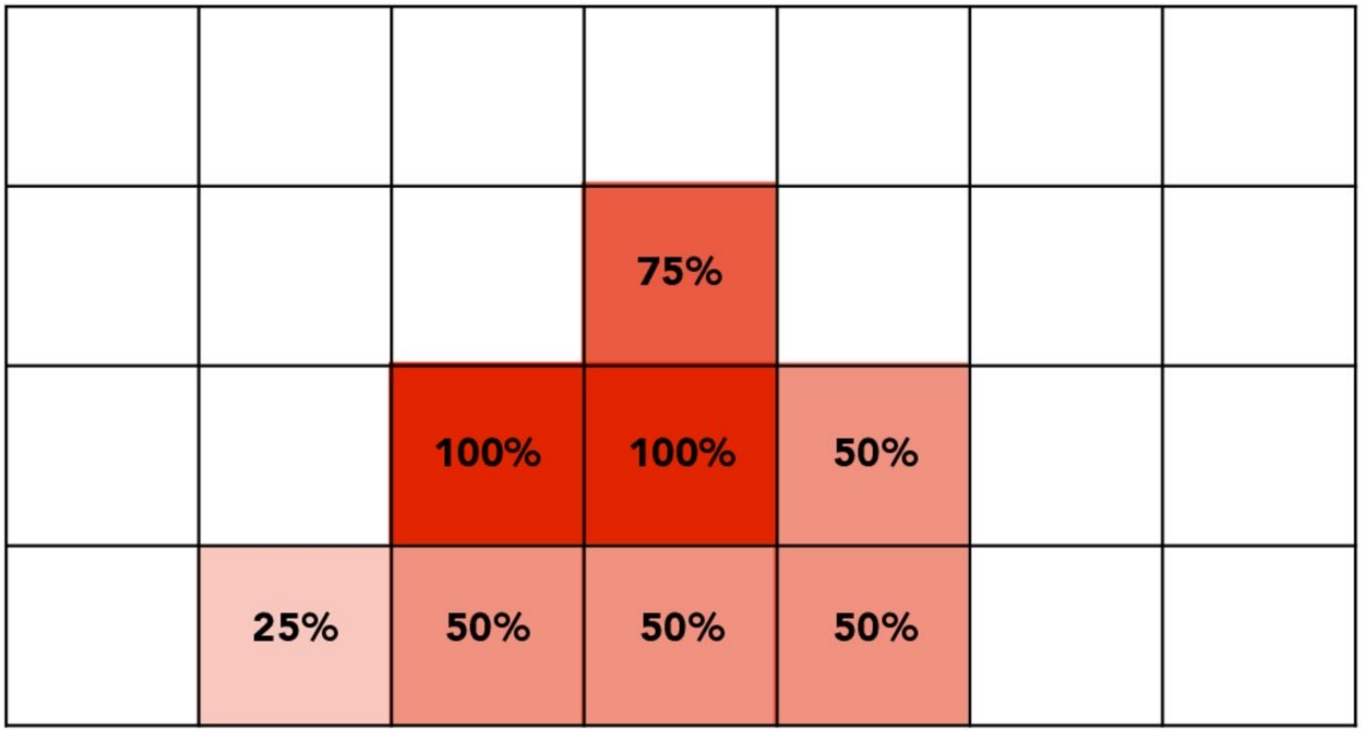
Supersampling: step 2

Average the NxN samples "inside" each pixel

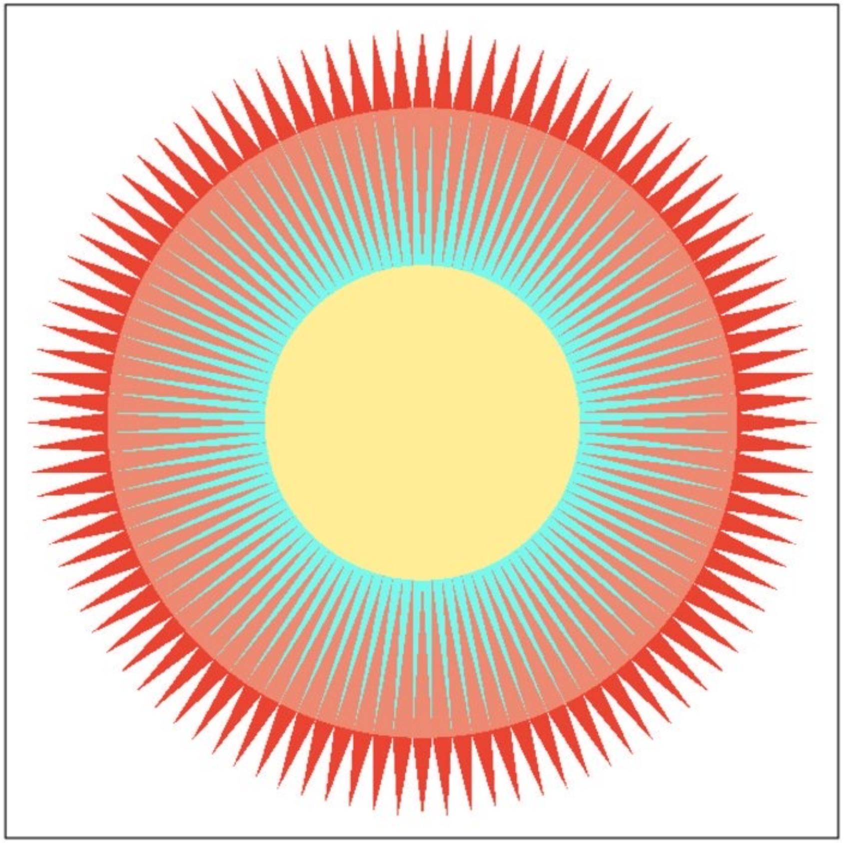


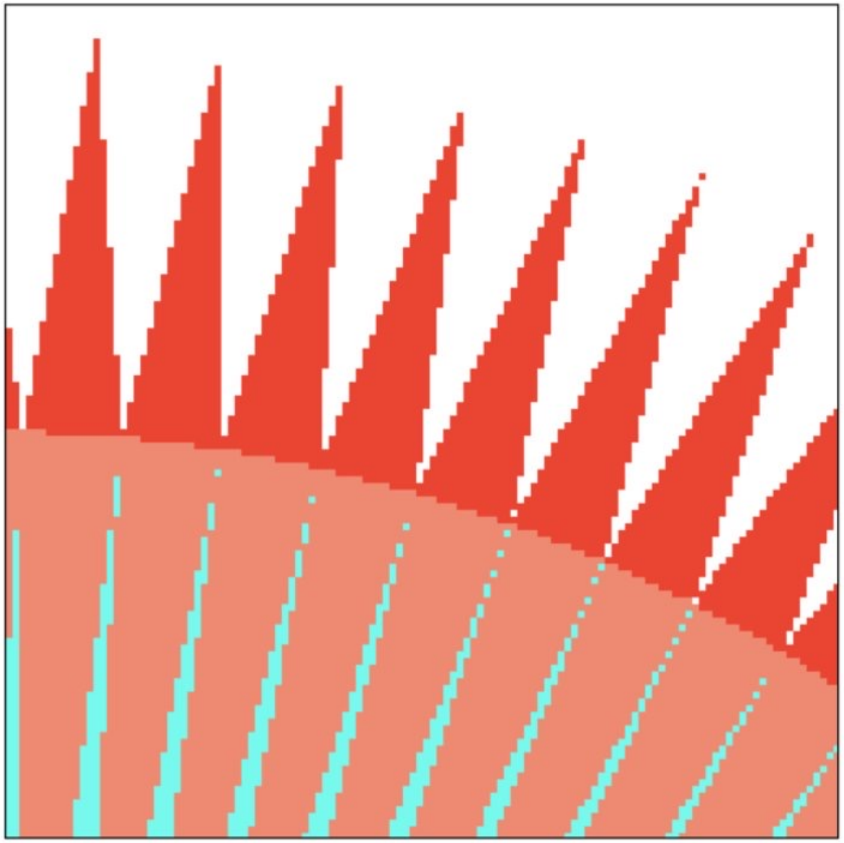
Supersampling: result

This is the corresponding signal emitted by the display



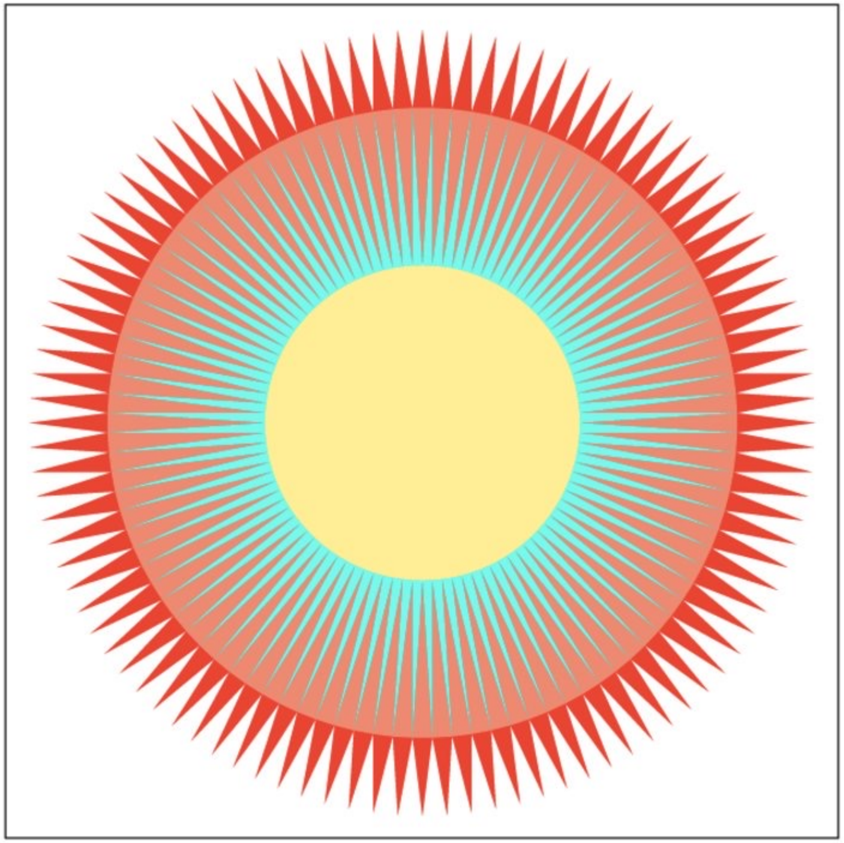
Point sampling

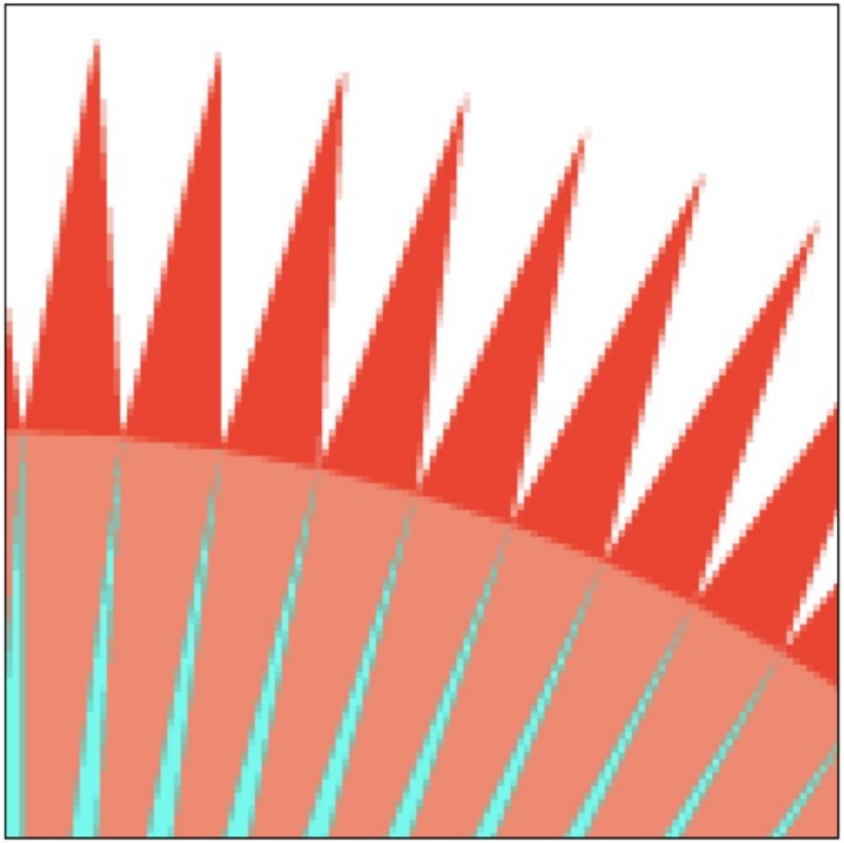




One sample per pixel

4x4 supersampling + downsampling

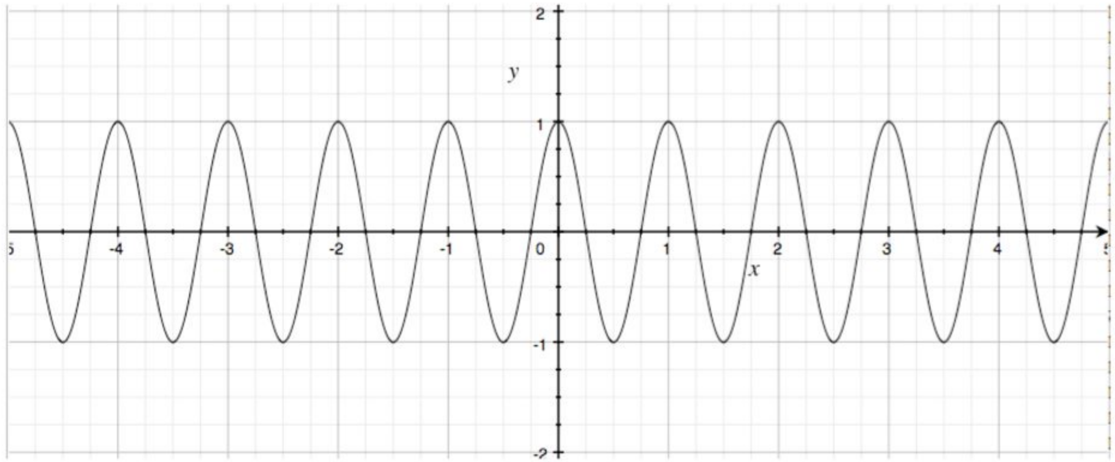




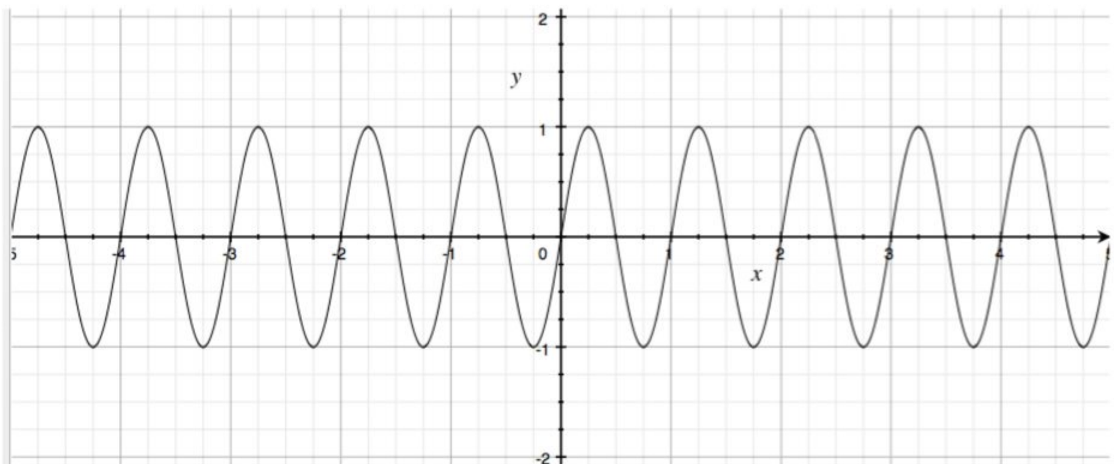
Pixel value is average of 4x4 samples per pixel

Representing functions as sines and cosines

Sines and cosines



$$\cos 2\pi x$$

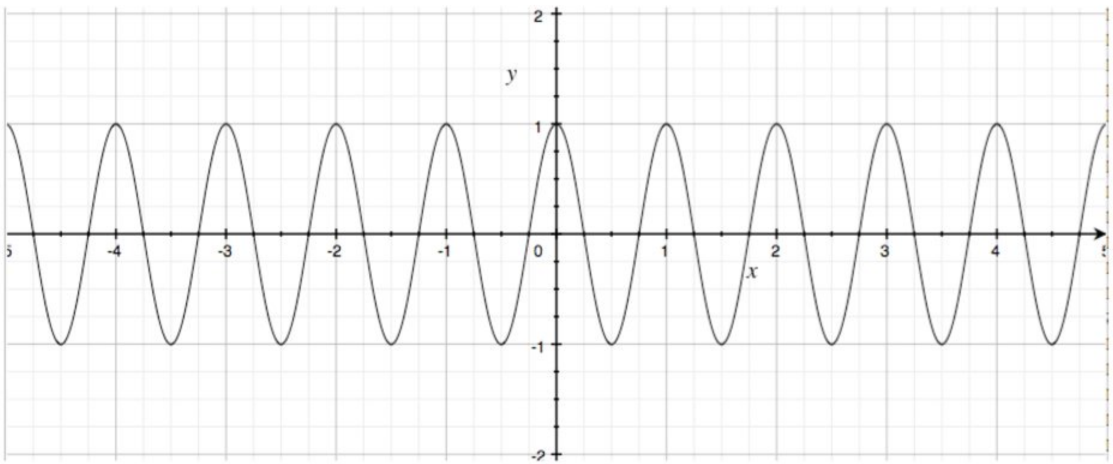


$$\sin 2\pi x$$

Frequencies

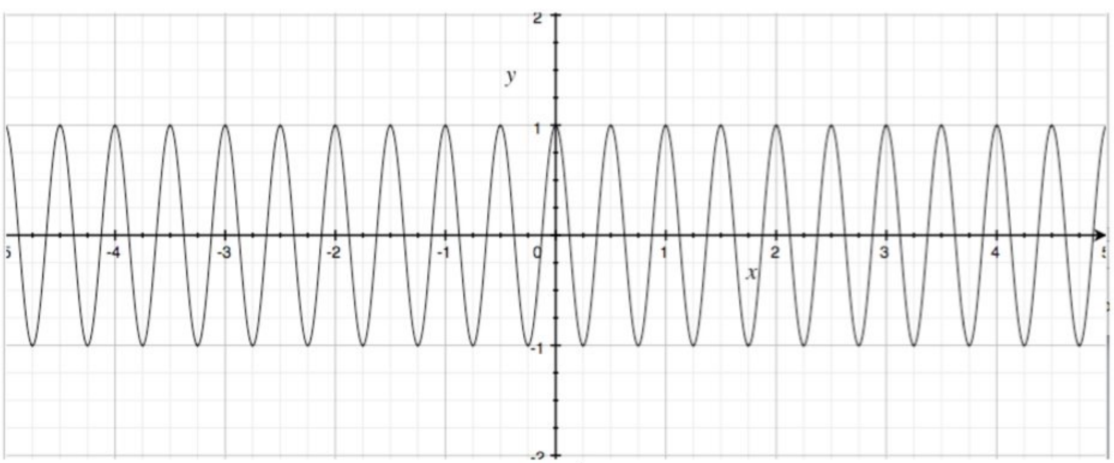
$$\cos 2\pi f x$$

$$f = \frac{1}{T}$$



$$f = 1$$

$$\cos 2\pi x$$

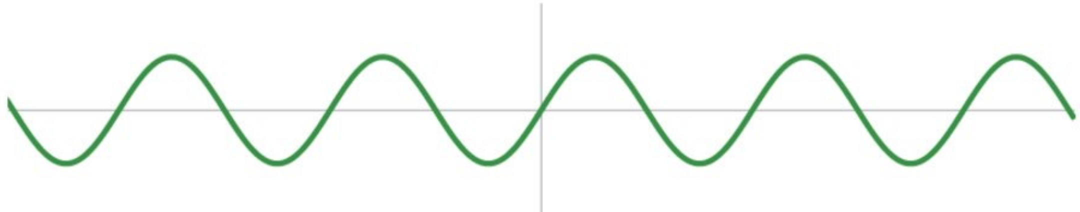


$$f = 2$$

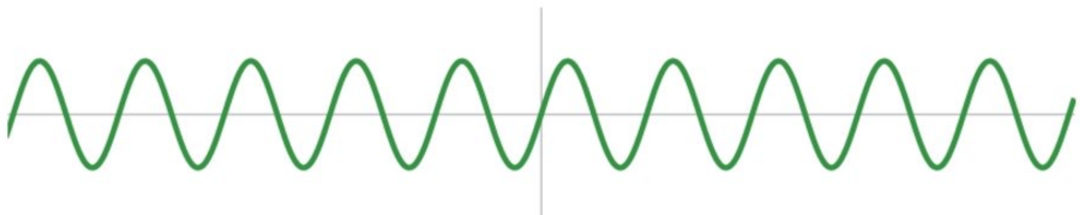
$$\cos 4\pi x$$

Representing sound wave as a superposition of frequencies

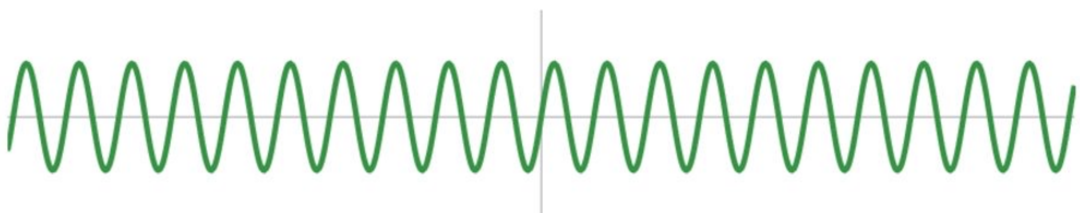
$f_1(x) = sin(\pi x)$



$f_2(x) = sin(2\pi x)$

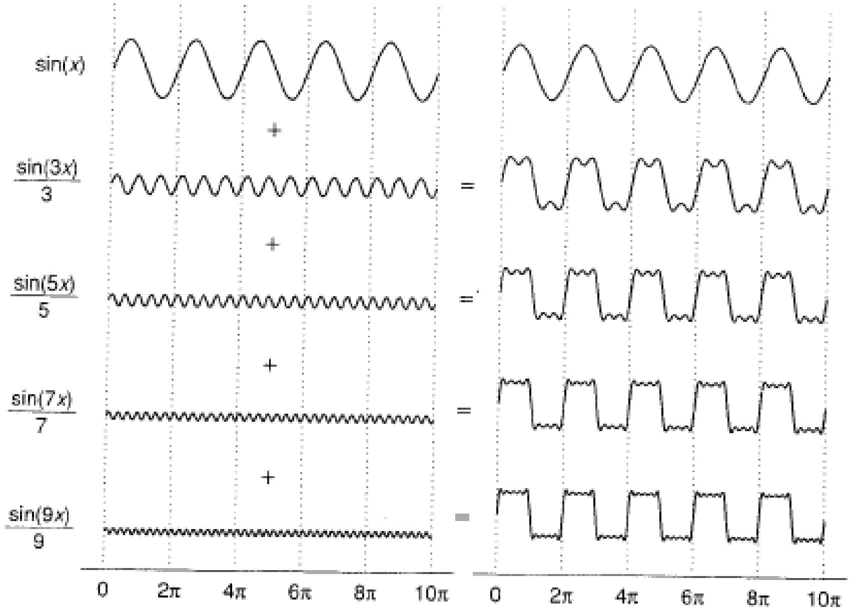


$f_4(x) = sin(4\pi x)$



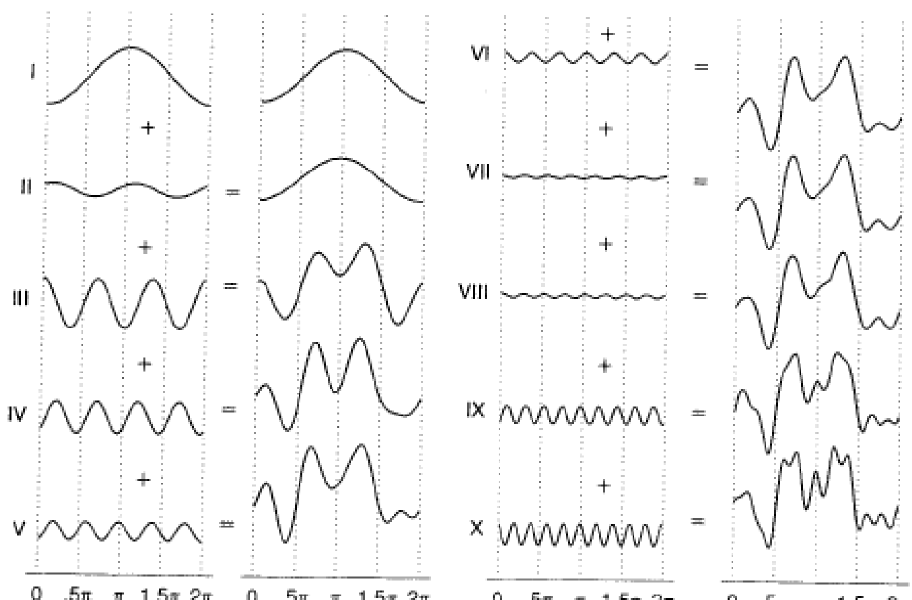
Keep these weights, instead of the f(x) samples

$f(x) = 1.0\, f_1(x) + 0.75\, f_2(x) + 0.5\, f_4(x)$



sin(x)
+
sin(3x)/3
=
+
sin(5x)/5
=
+
sin(7x)/7
=
+
sin(9x)/9
=
0
2π
4π
6π
8π
10π

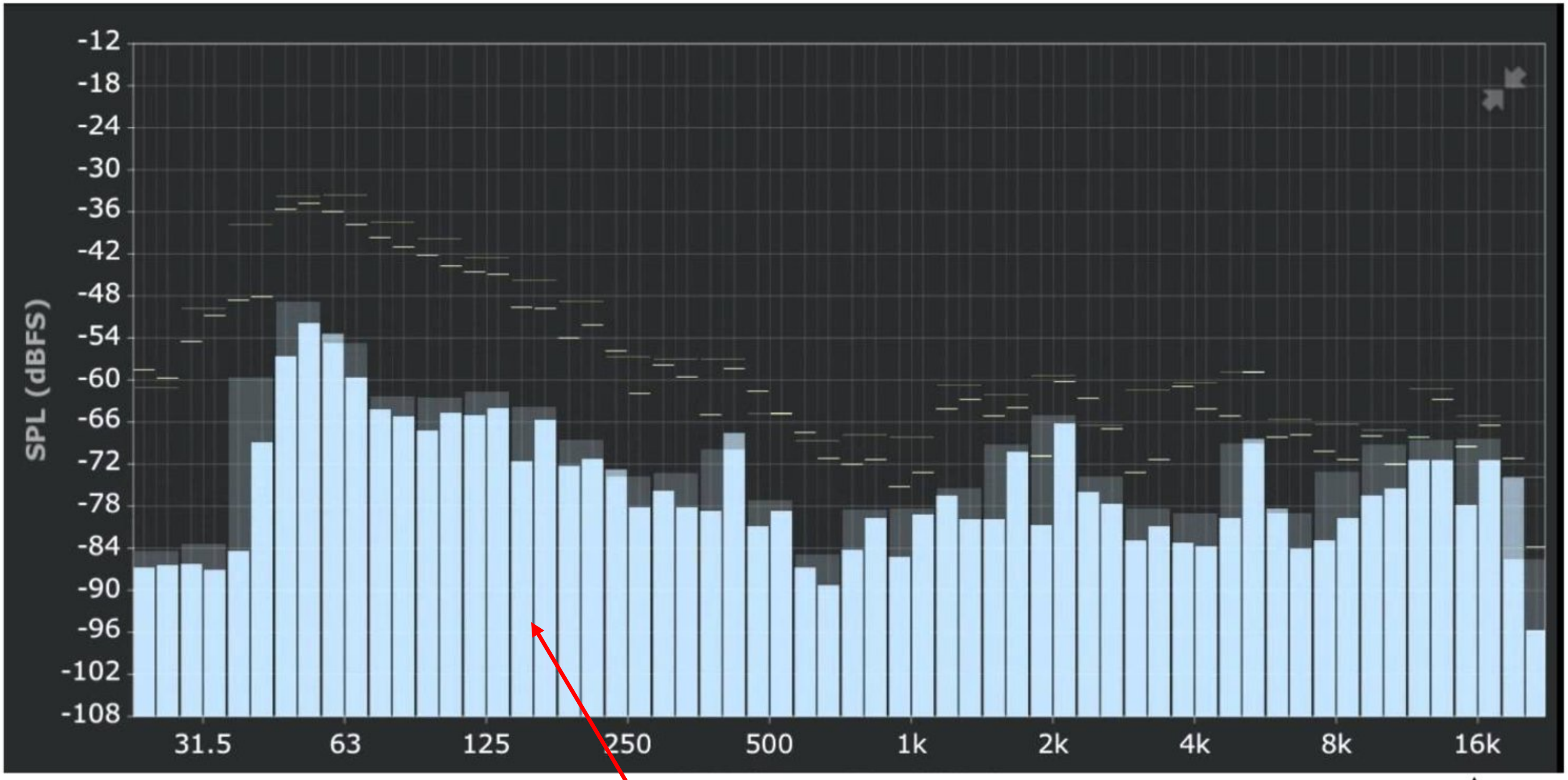
(a)



I
II
III
IV
V
VI
VII
VIII
IX
X
+
=
0
.5π
π
1.5π
2π

(b)

Audio spectrum analyzer: representing sound as a sum of its constituent frequencies



Intensity of low-frequencies (bass)

The weights are here

Intensity of high frequencies

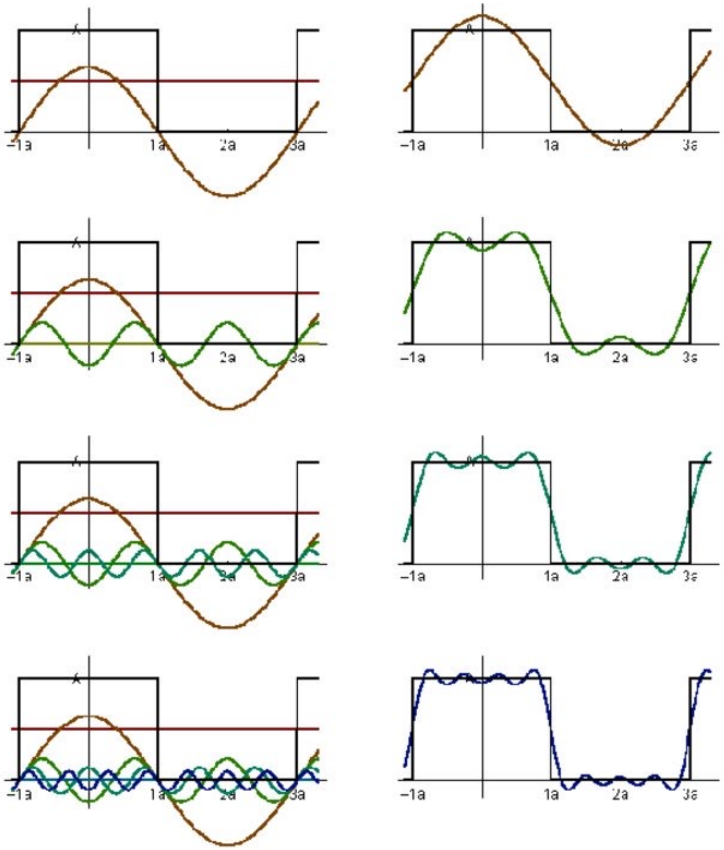
Image credit: ONYX Apps

How to compute frequency-domain representation of a signal?

Fourier transform

Represent a function as a weighted sum of sines and cosines

Joseph Fourier 1768 - 1830



$$f(x) = \frac{A}{2} + \frac{2A\cos(t\omega)}{\pi} - \frac{2A\cos(3t\omega)}{3\pi} + \frac{2A\cos(5t\omega)}{5\pi} - \frac{2A\cos(7t\omega)}{7\pi} + \cdots$$

Fourier transform

- **Convert representation of signal from primal domain (spatial/temporal) to frequency domain by projecting signal into its component frequencies**

Recall! I don't think I took that class...

Recall:

$$e^{ix} = \cos x + i \sin x$$

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

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

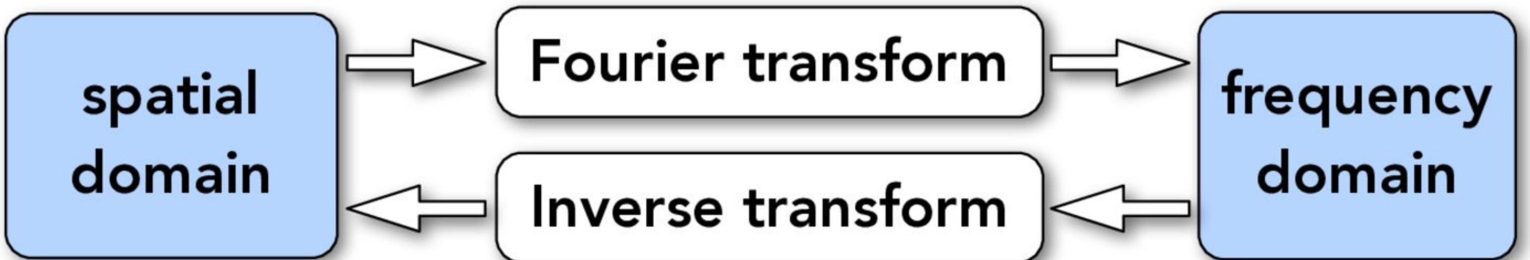
Ow! My head hurts!

- **2D form:**

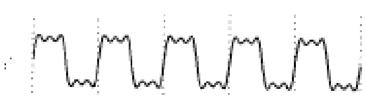
$$F(u,v) = \int\int f(x,y) e^{-2\pi i(ux+vy)} dx dy$$

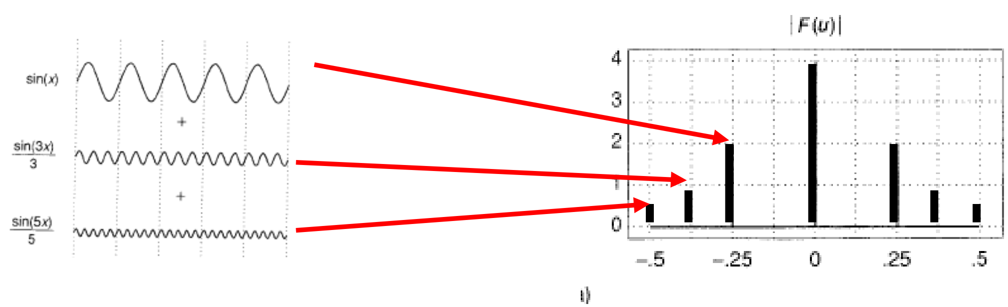
Fourier transform decomposes a signal into its constituent frequencies

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



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





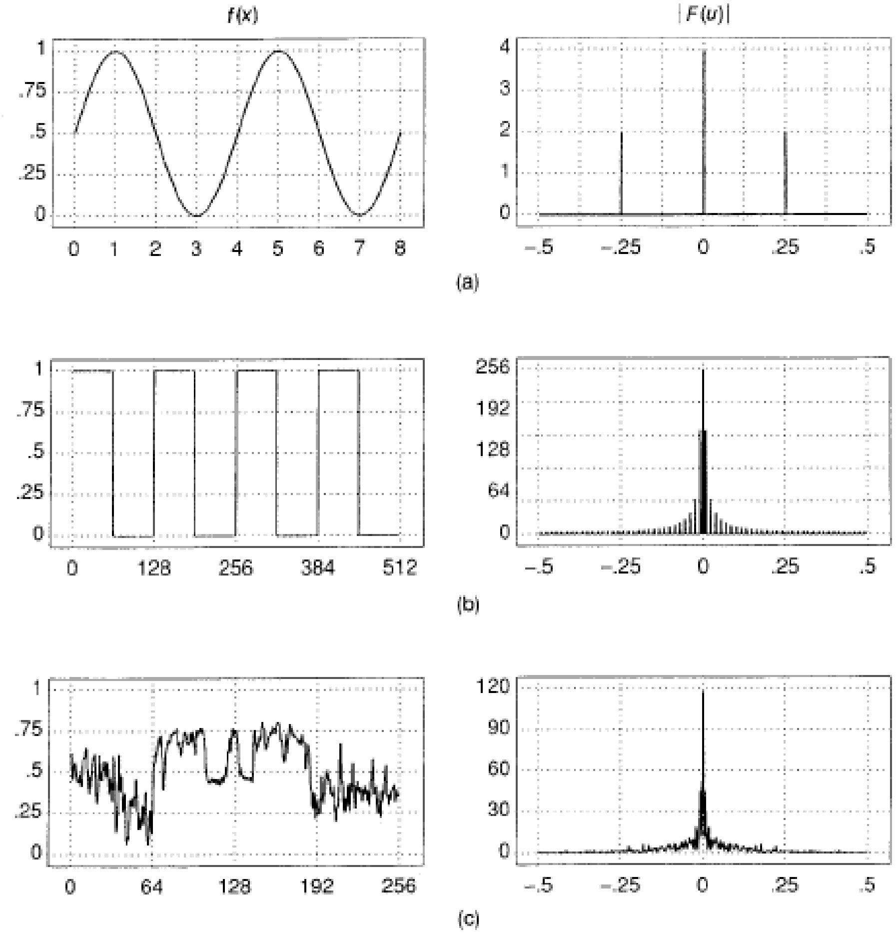


Fig. 14.15 Signals in the spatial and frequency domains. (a) Sine. (b) Square Wave. (c) Mandrill. (Courtesy of George Wolberg, Columbia University.)

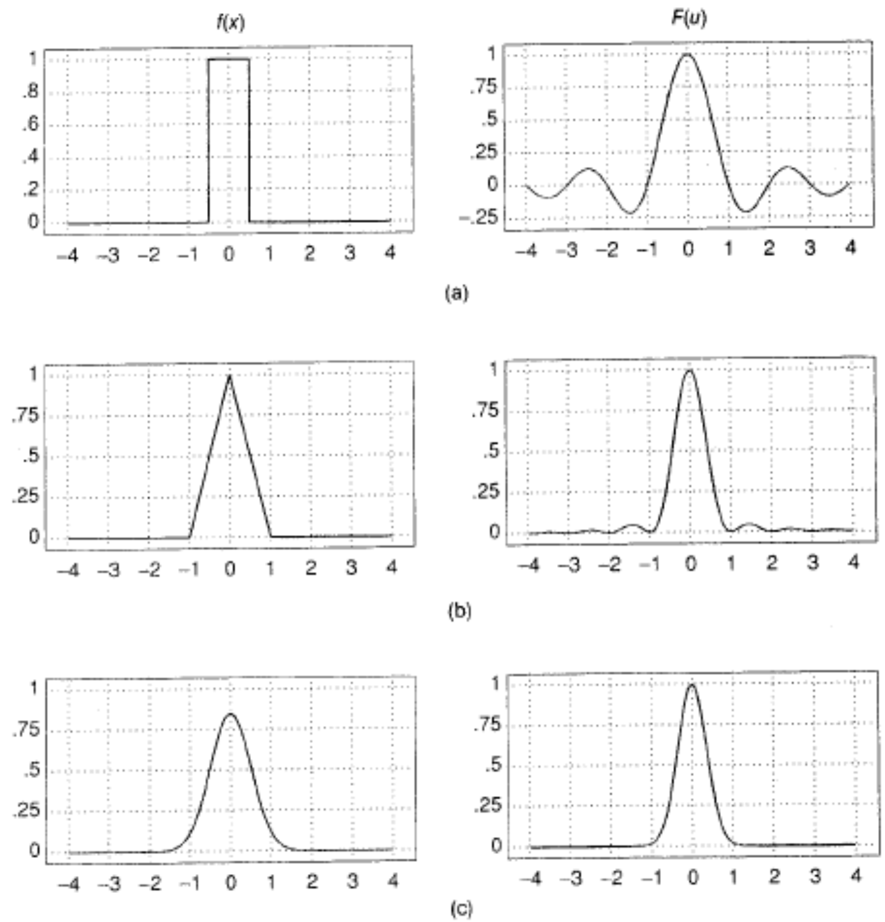
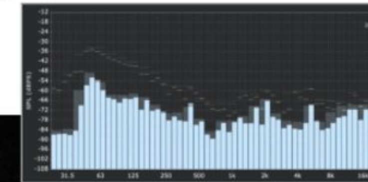


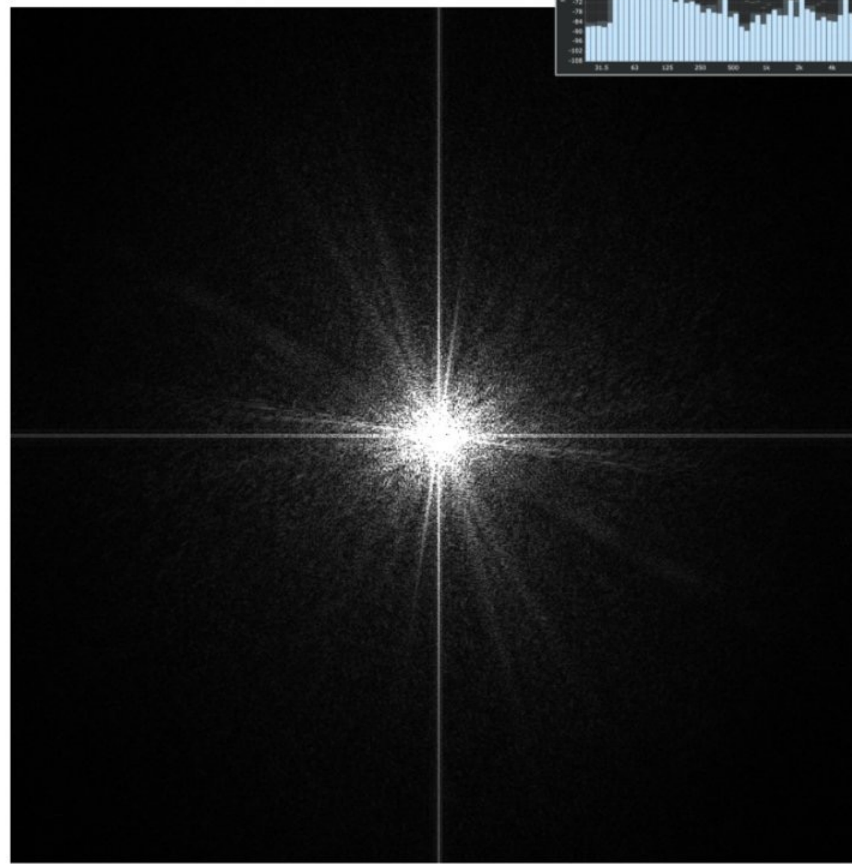
Fig. 14.25 Filters in spatial and frequency domains. (a) Pulse—sinc. (b) Triangle—sinc^2. (c) Gaussian—Gaussian. (Courtesy of George Wolberg, Columbia University.)

Visualizing the frequency content of images

Visualization below is the 2D frequency domain equivalent of the 1D audio spectrum I showed you earlier *

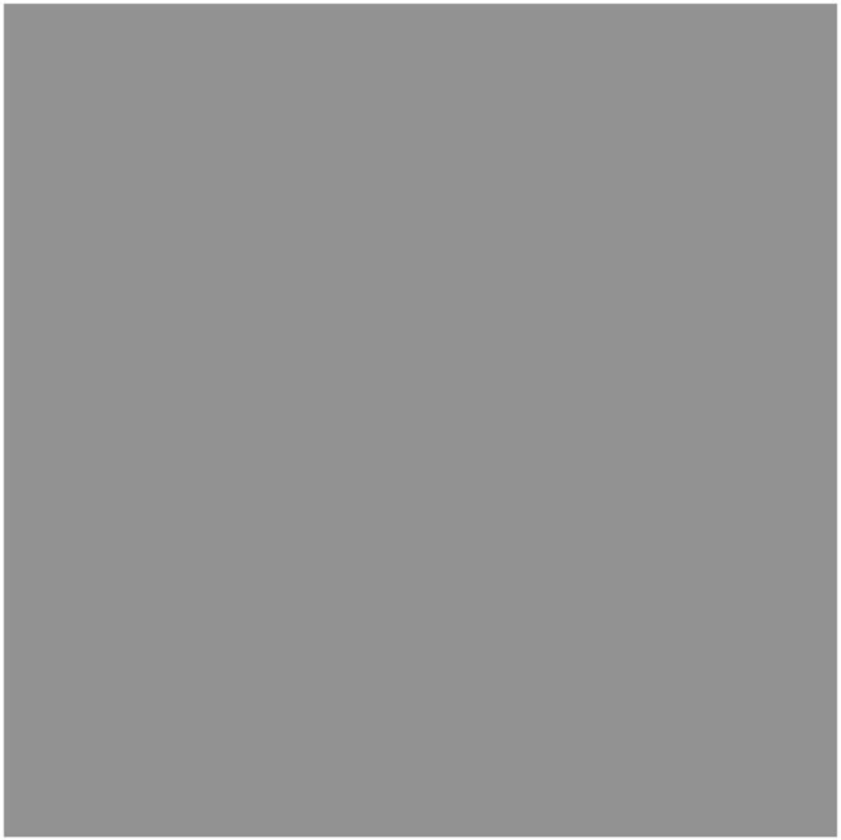


Spatial domain result

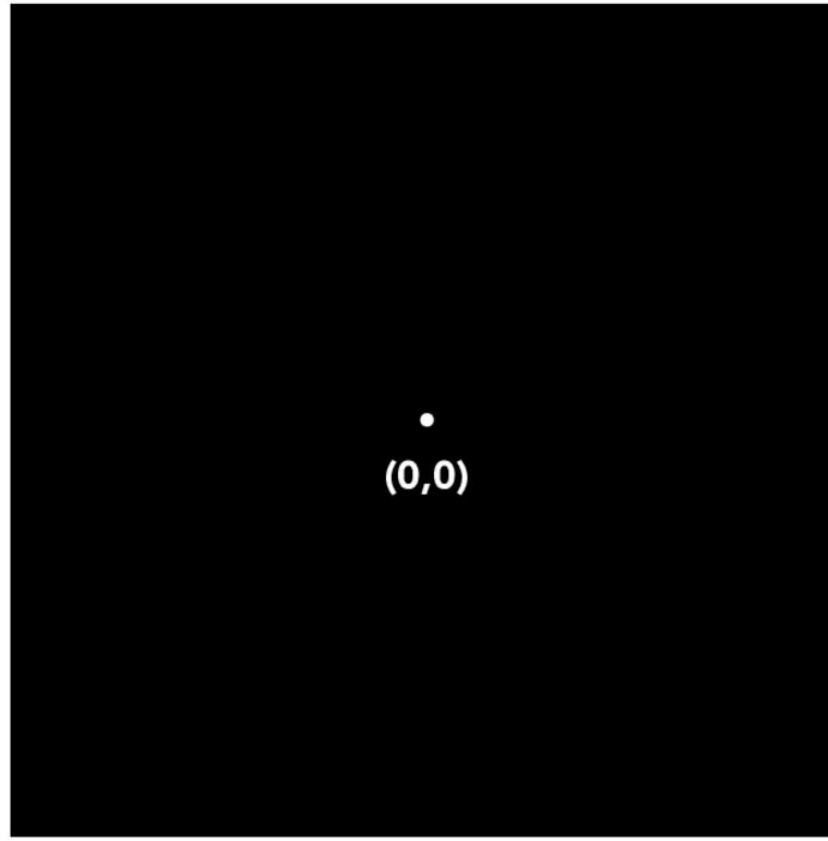


Spectrum

Constant signal (in primal domain)

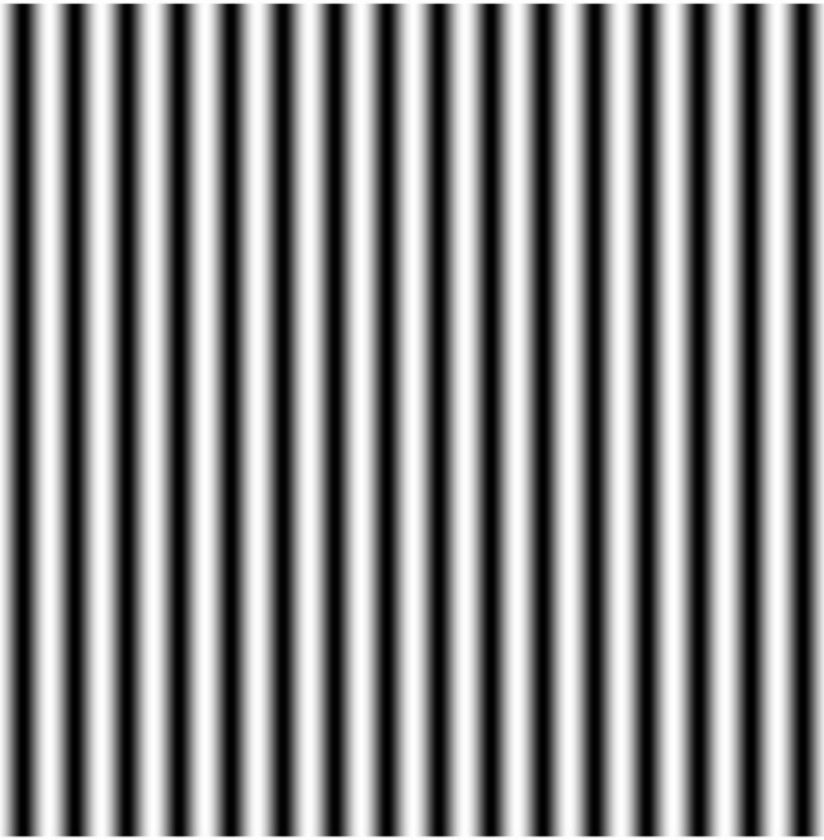


Spatial domain

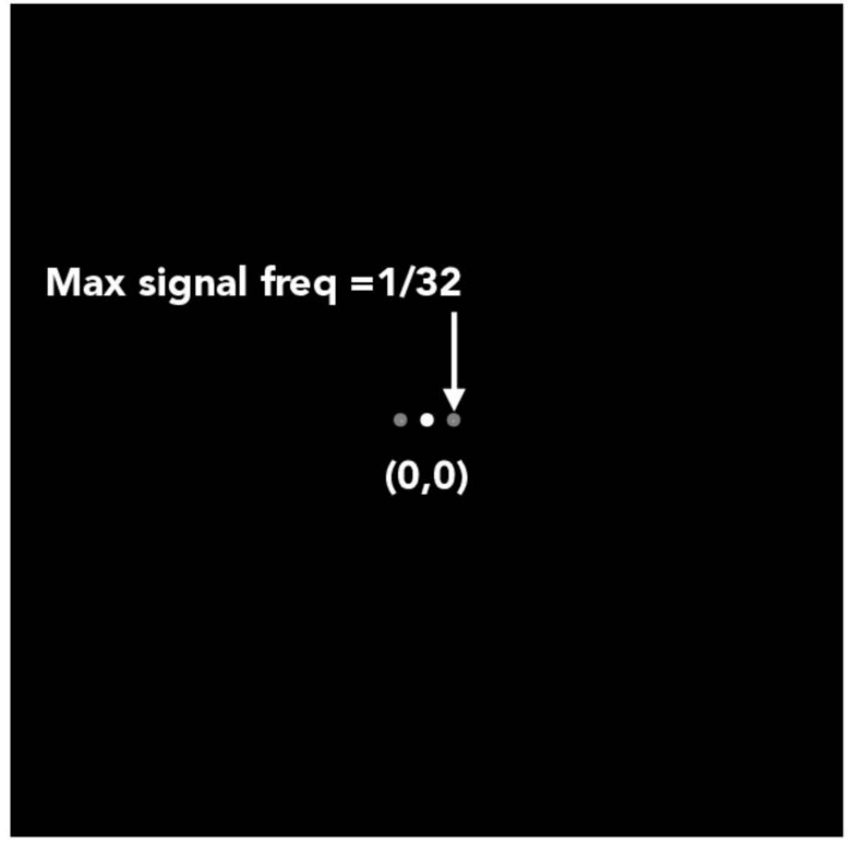


Frequency domain

$\sin(2\pi/32)x$ — frequency 1/32; 32 pixels per cycle

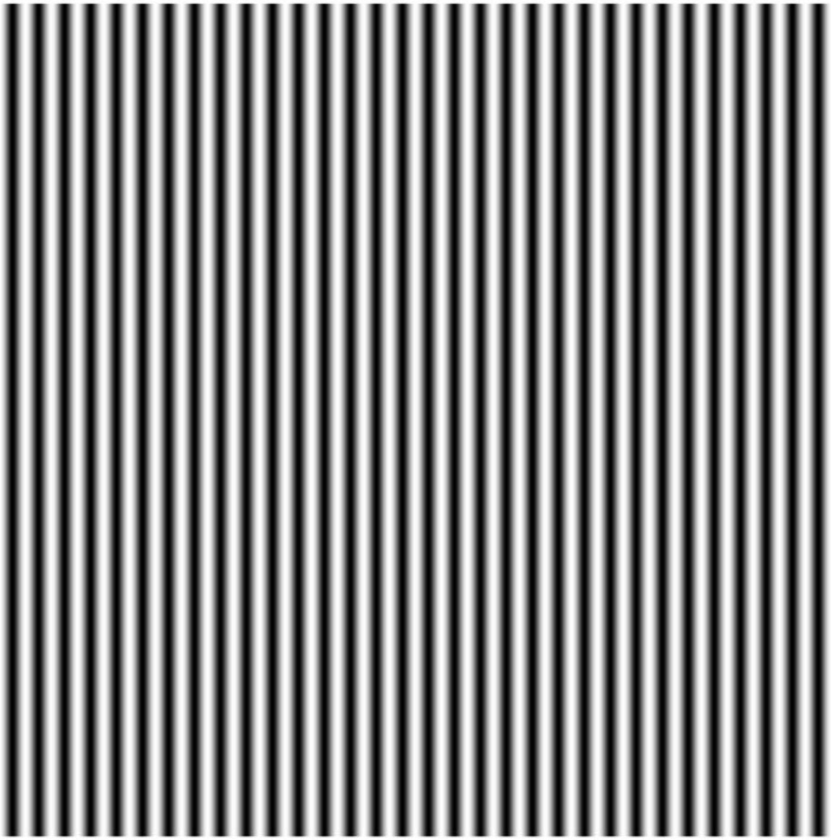


Spatial domain

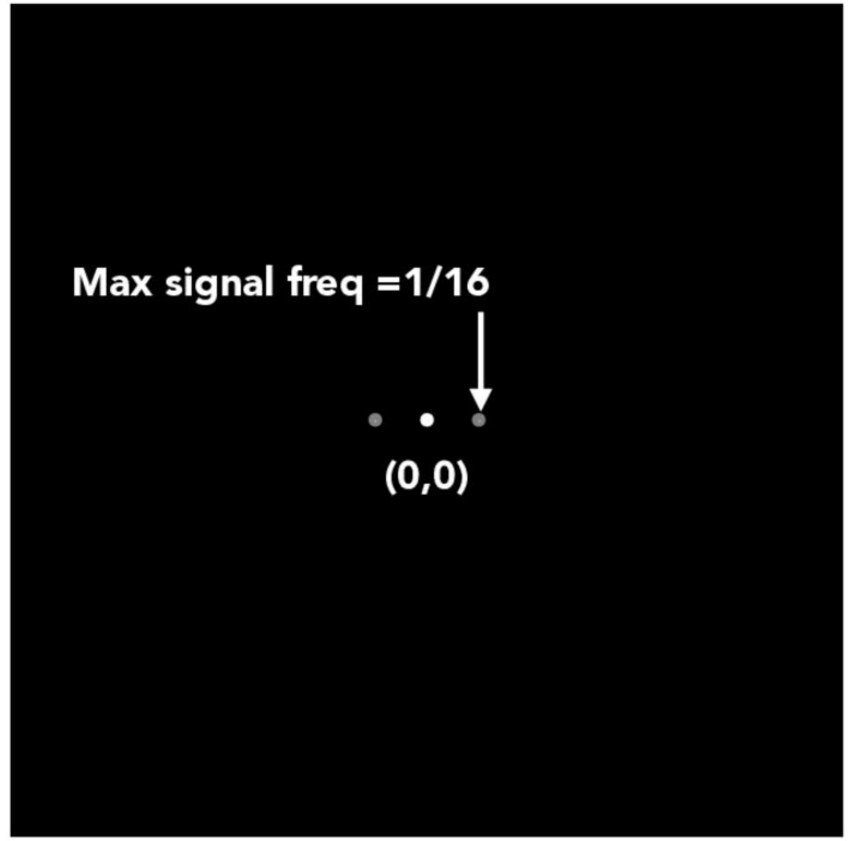


Frequency domain

$\sin(2\pi/16)x$ — frequency 1/16; 16 pixels per cycle

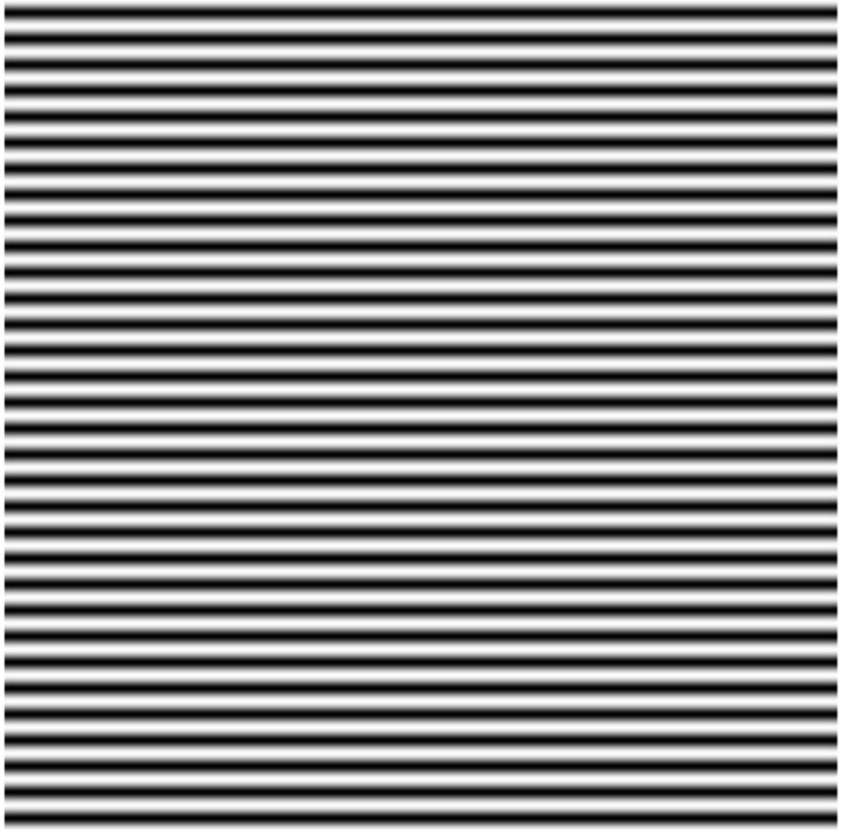


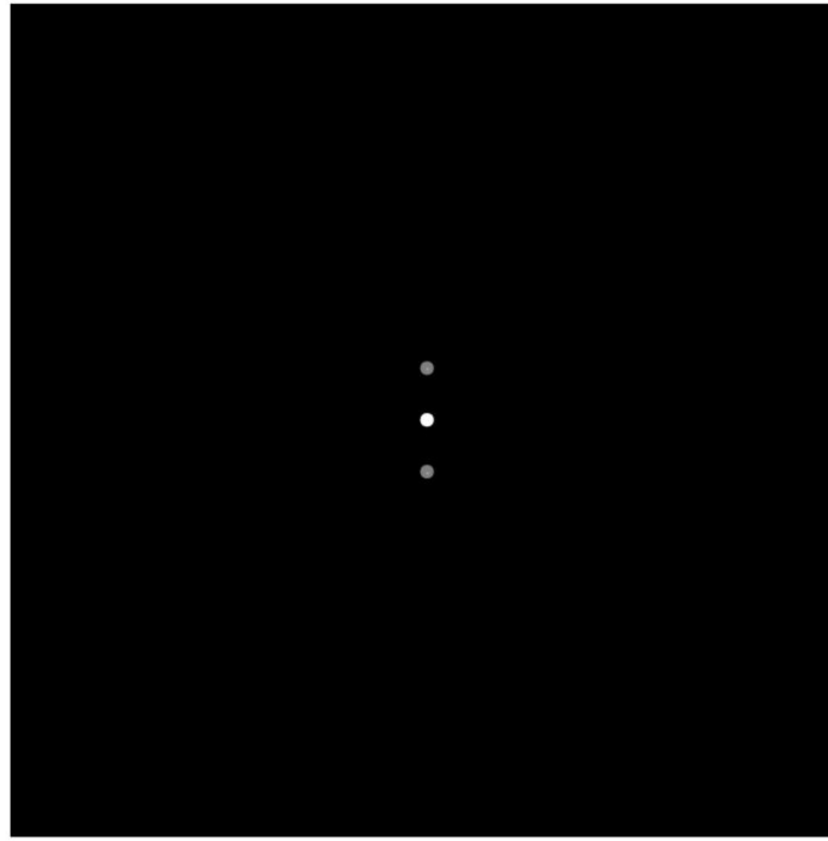
Spatial domain



Frequency domain

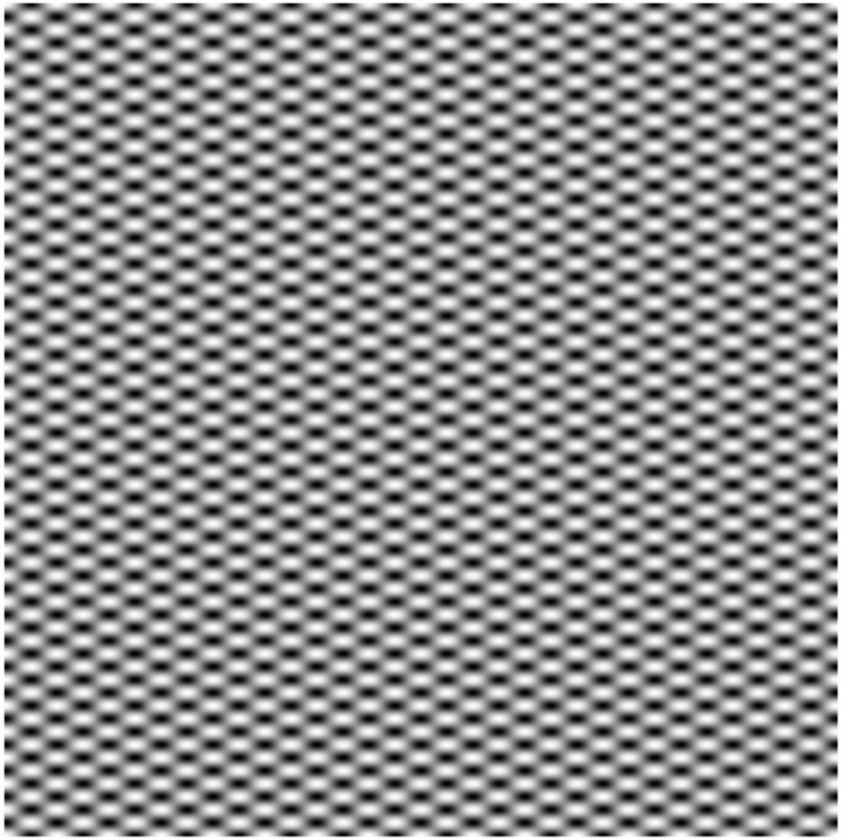
$$\sin(2\pi/16)y$$



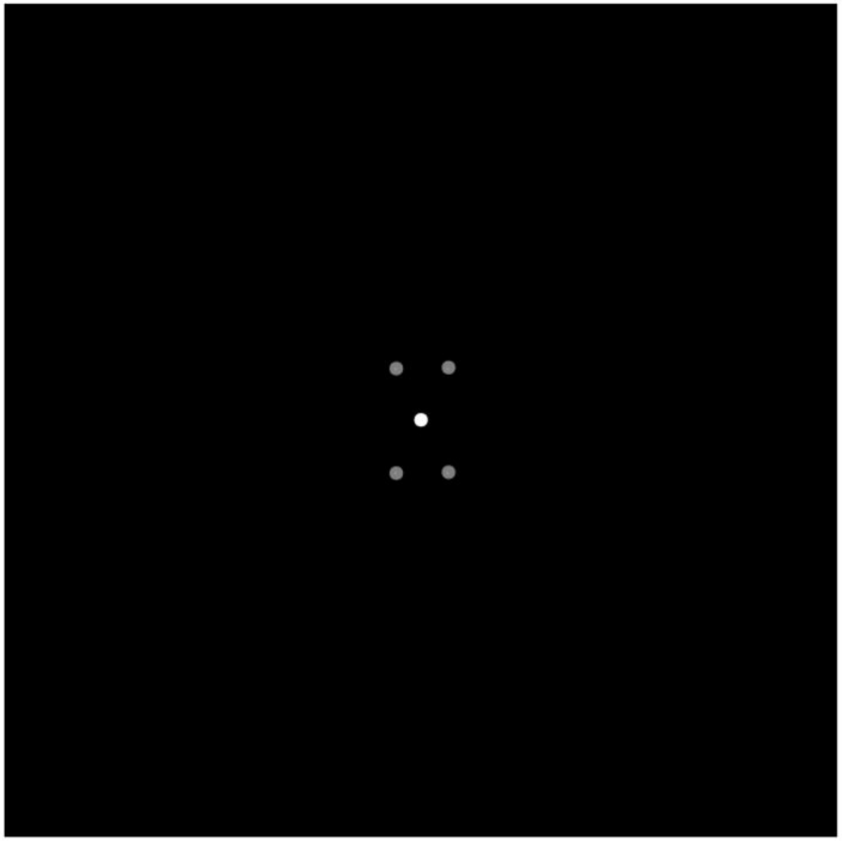


Spatial domain **Frequency domain**

$$\sin(2\pi/32)x \times \sin(2\pi/16)y$$

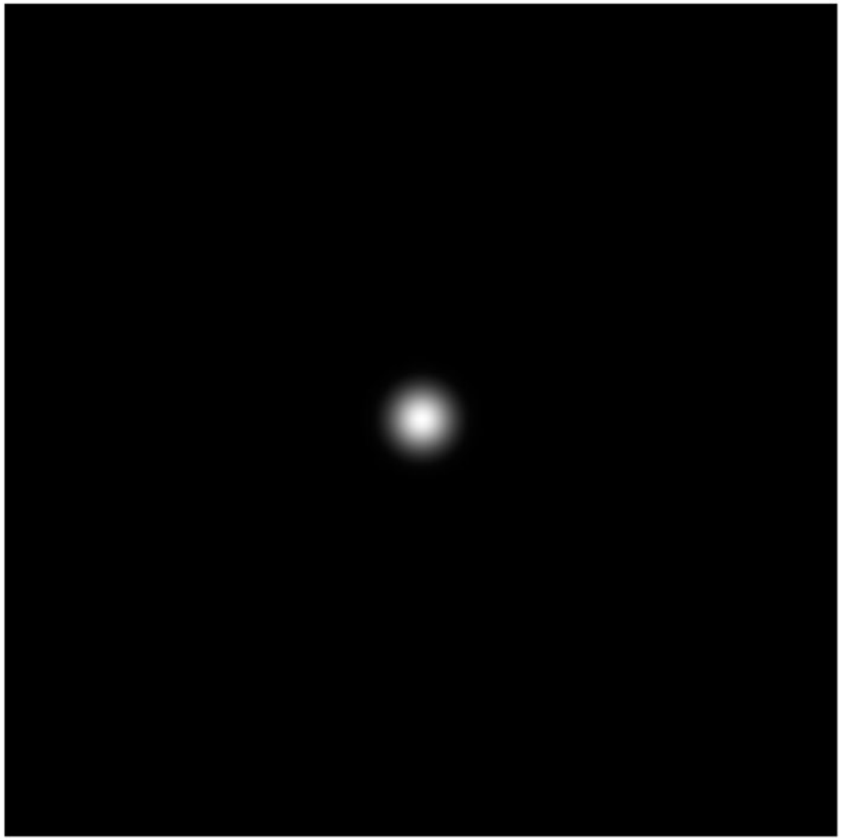


Spatial domain

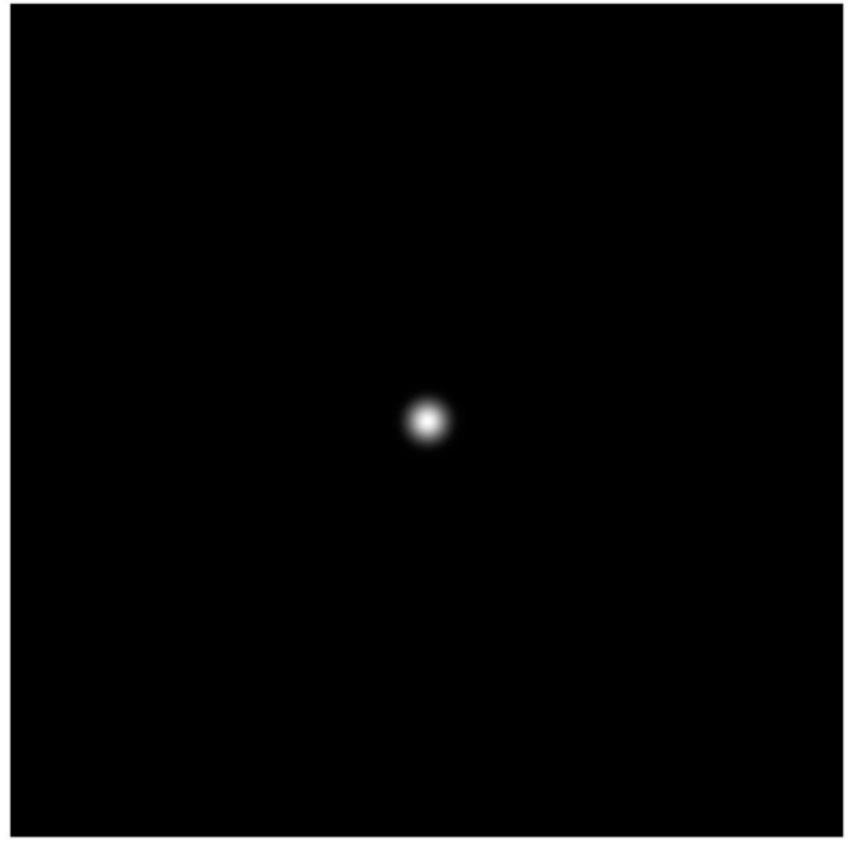


Frequency domain

$$\exp(-r^2/16^2)$$

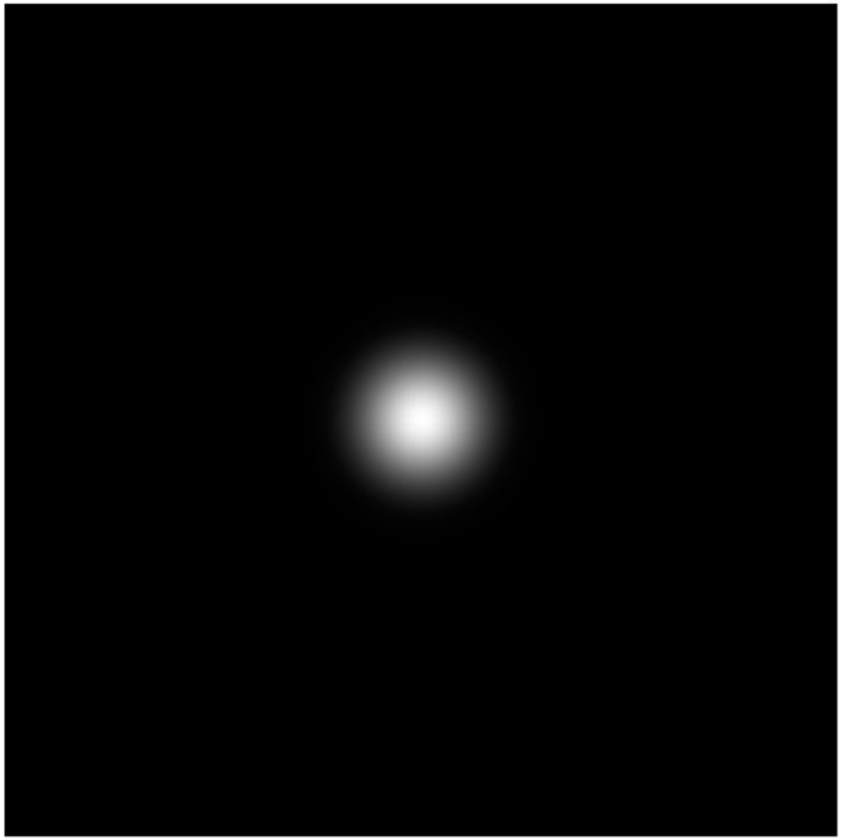


Spatial domain

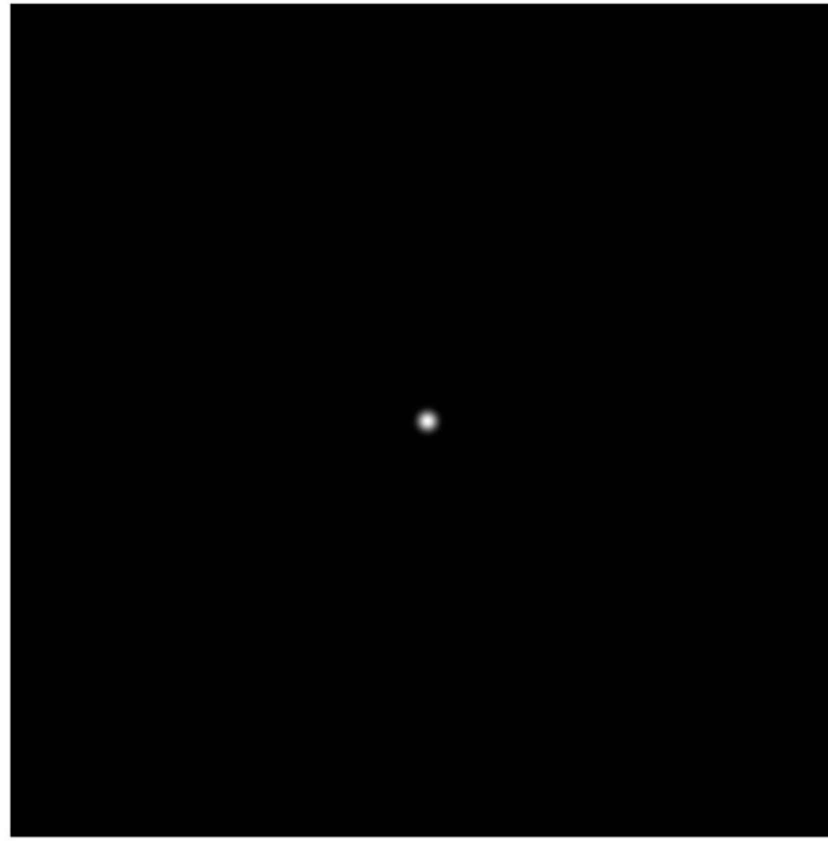


Frequency domain

$$\exp(-r^2/32^2)$$

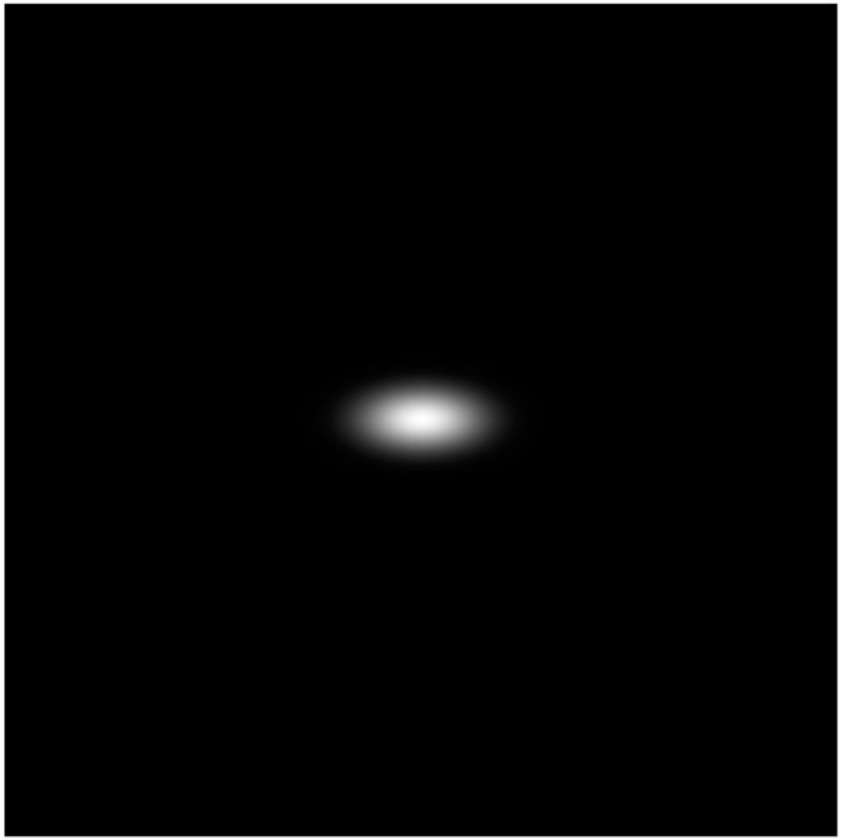


Spatial domain

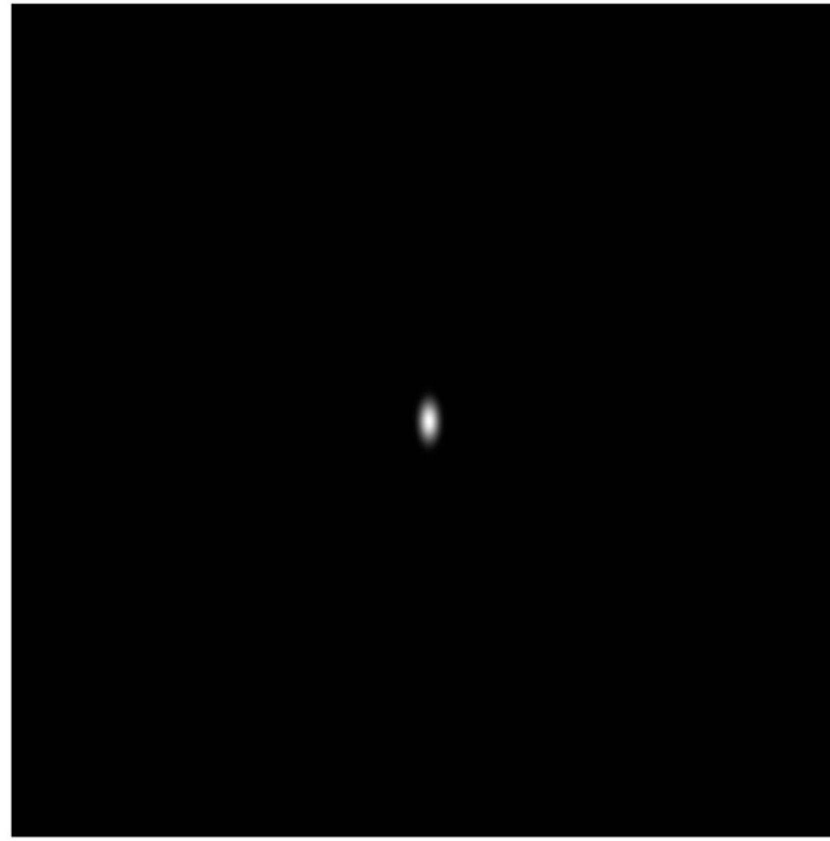


Frequency domain

$$\exp(-x^2/32^2) \times \exp(-y^2/16^2)$$

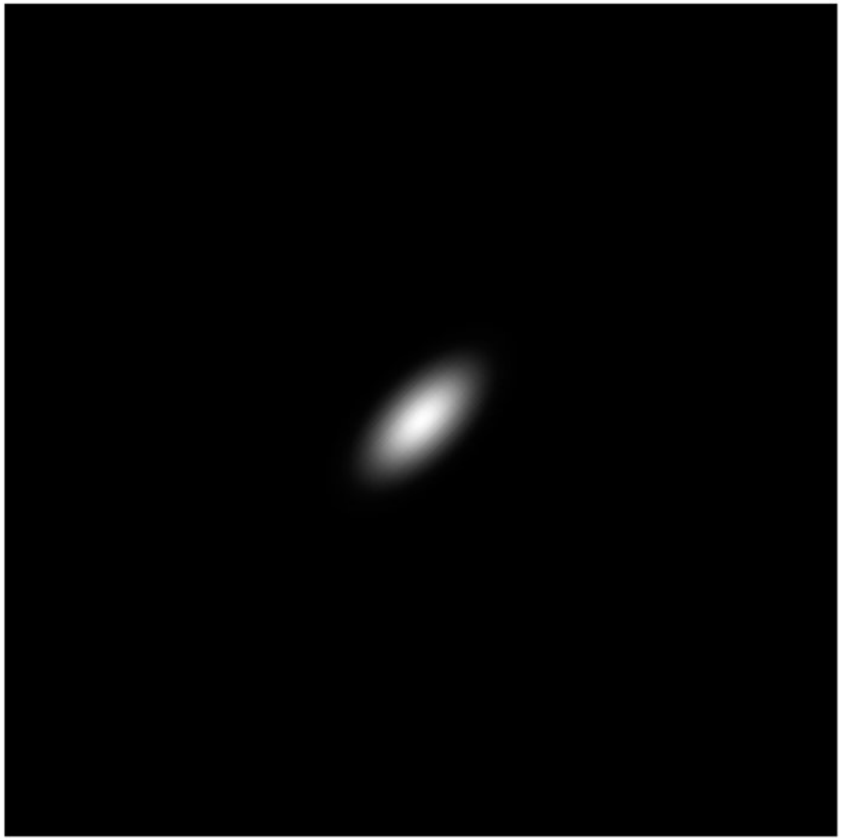


Spatial domain

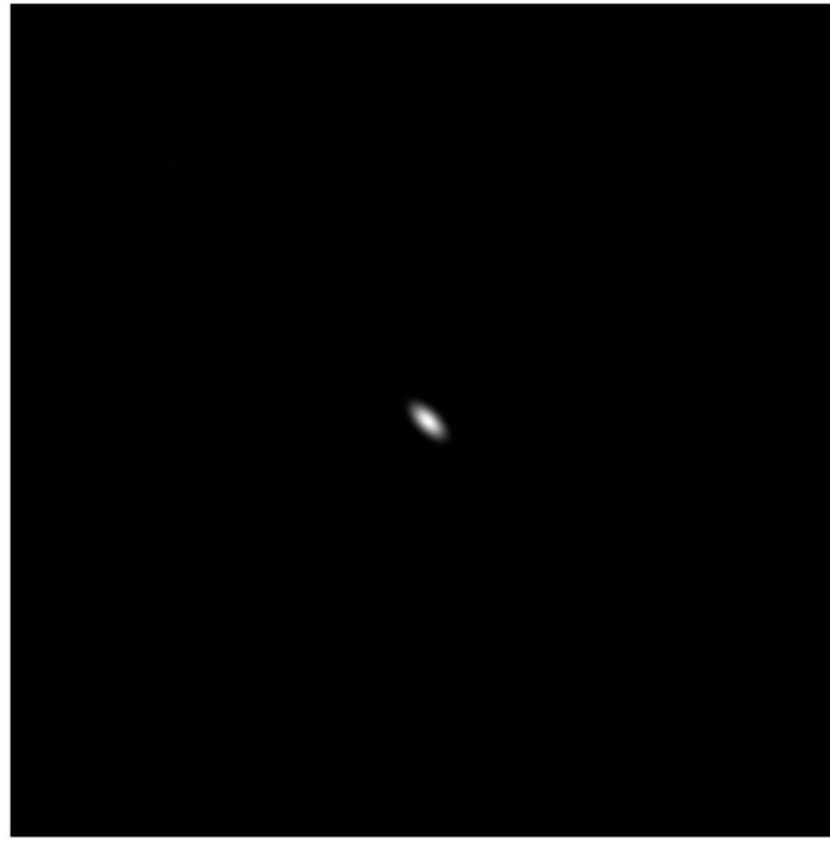


Frequency domain

Rotate 45 $\exp(-x^2/32^2) \times \exp(-y^2/16^2)$



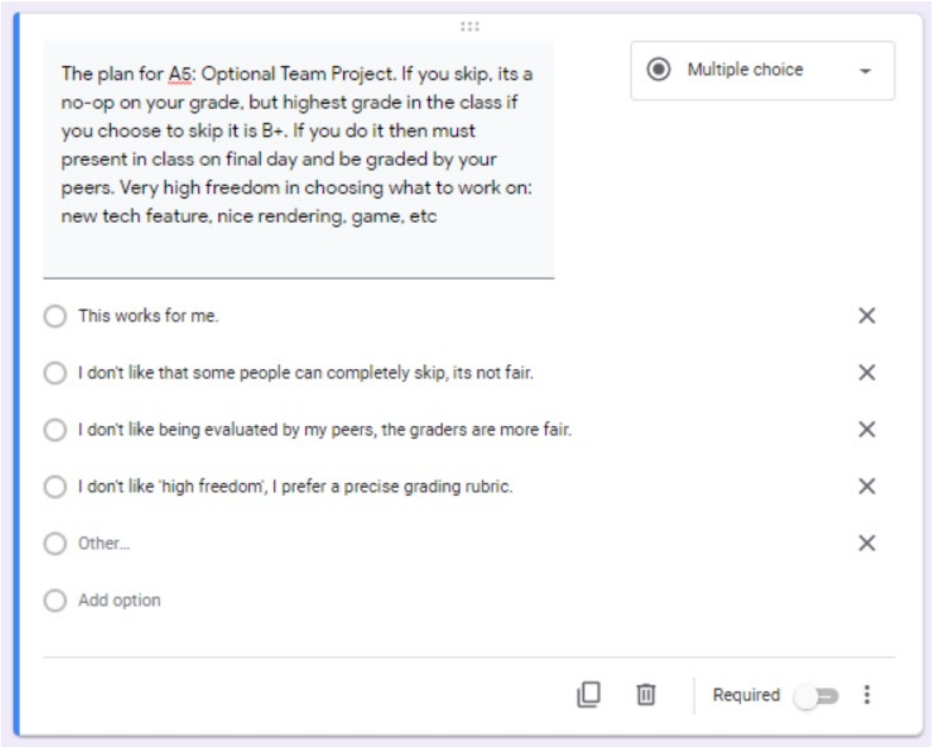
Spatial domain



Frequency domain

Participation Survey

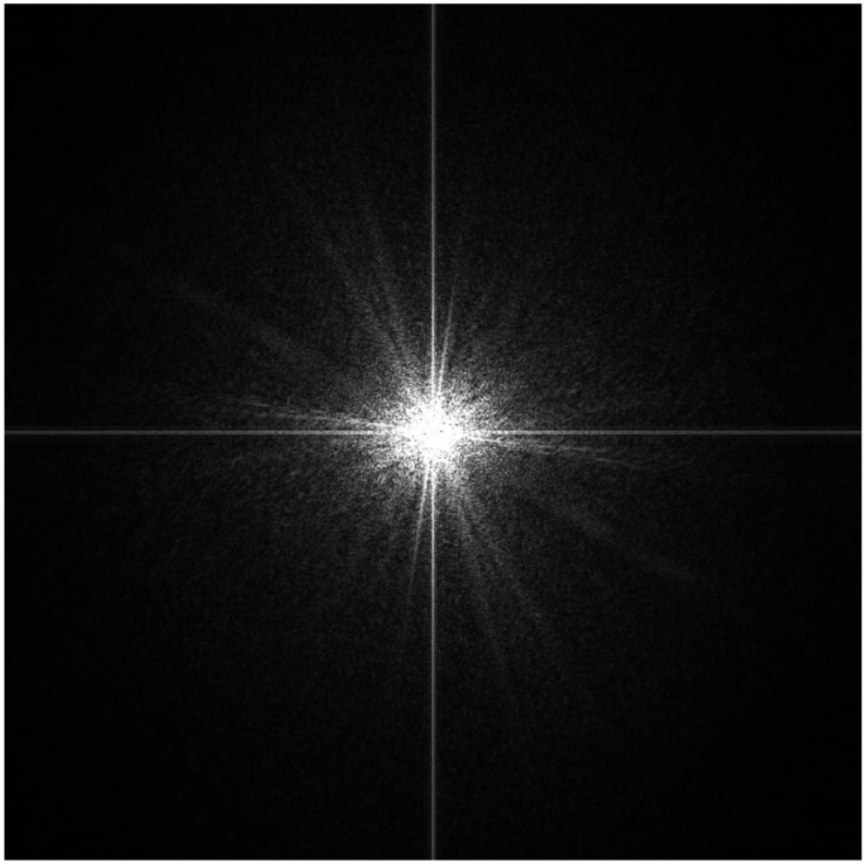
- About the project plan



Filtering (frequency domain)

Manipulating the frequency content of images

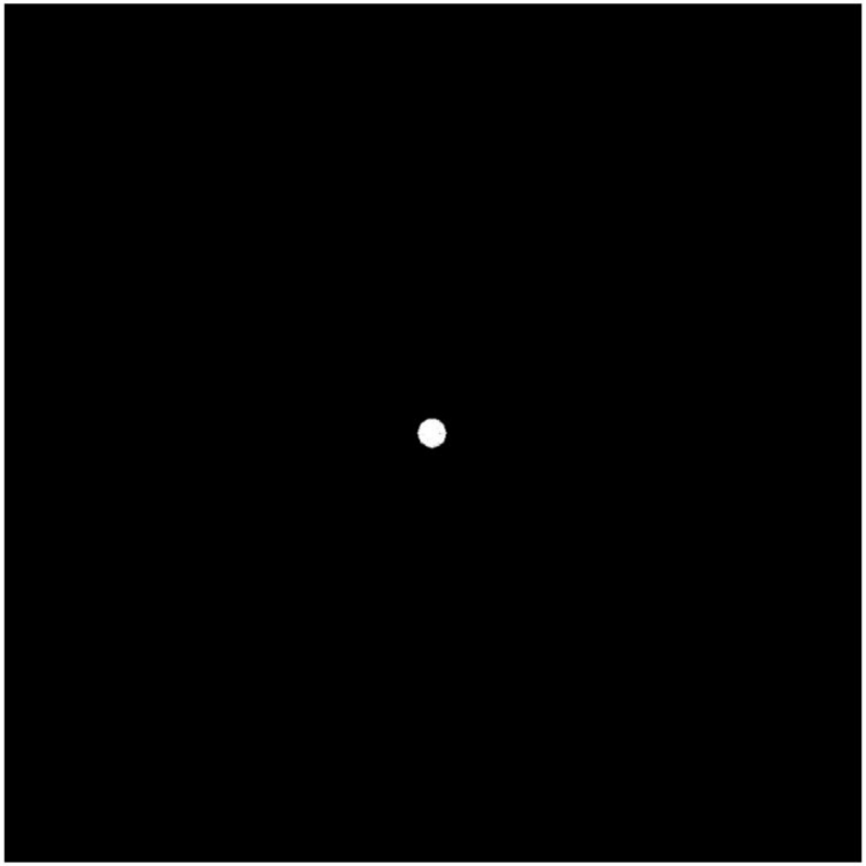
Spatial domain



Frequency domain

Low frequencies only (smooth gradients)

Spatial domain



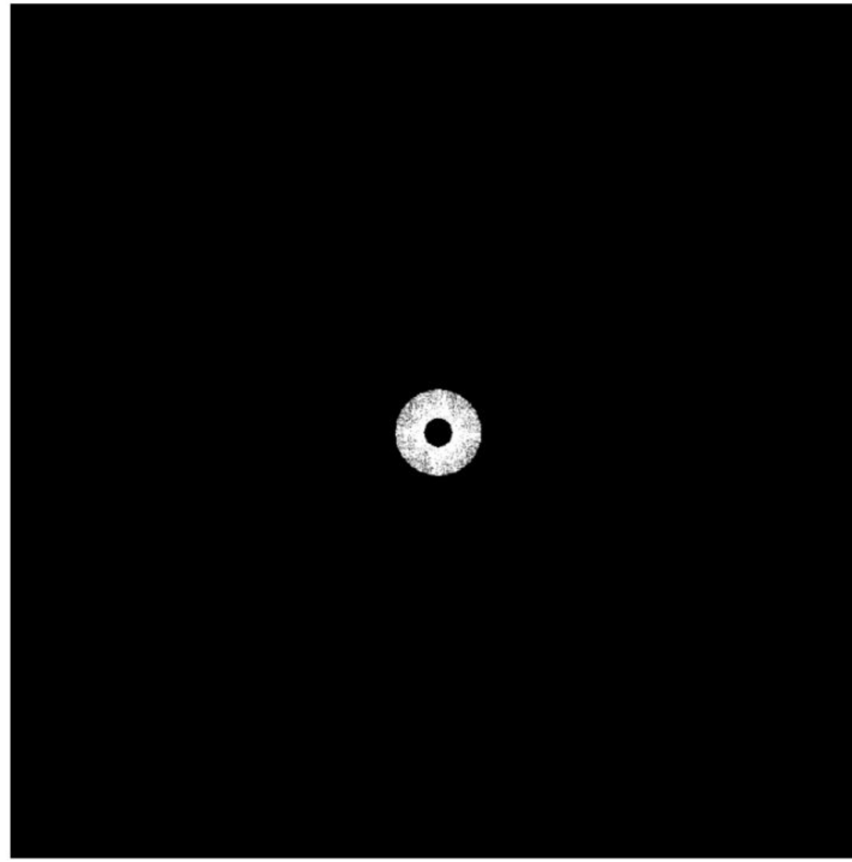
Frequency domain

(after low-pass filter)

All frequencies above cutoff have 0 magnitude

Mid-range frequencies

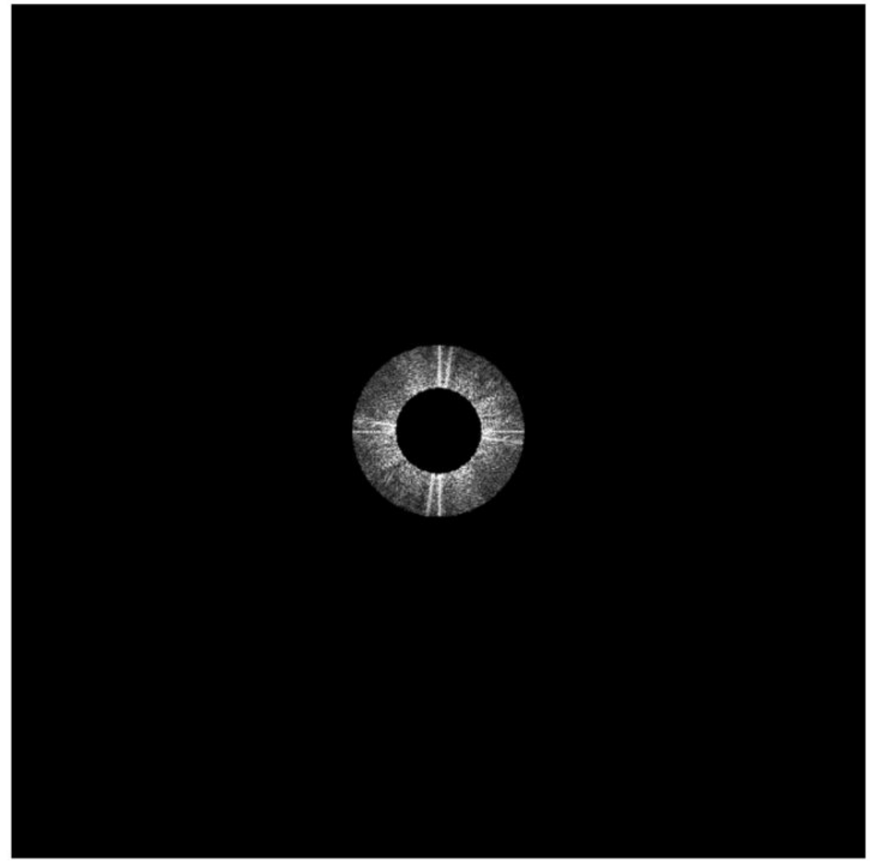
Spatial domain



Frequency domain
(after band-pass filter)

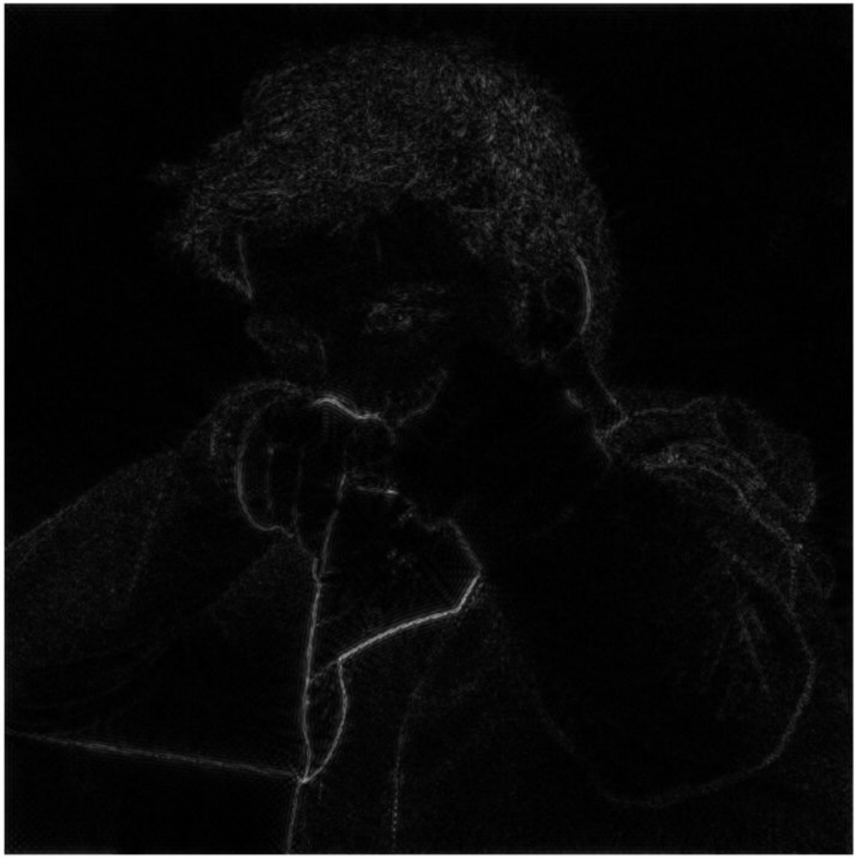
Mid-range frequencies

Spatial domain

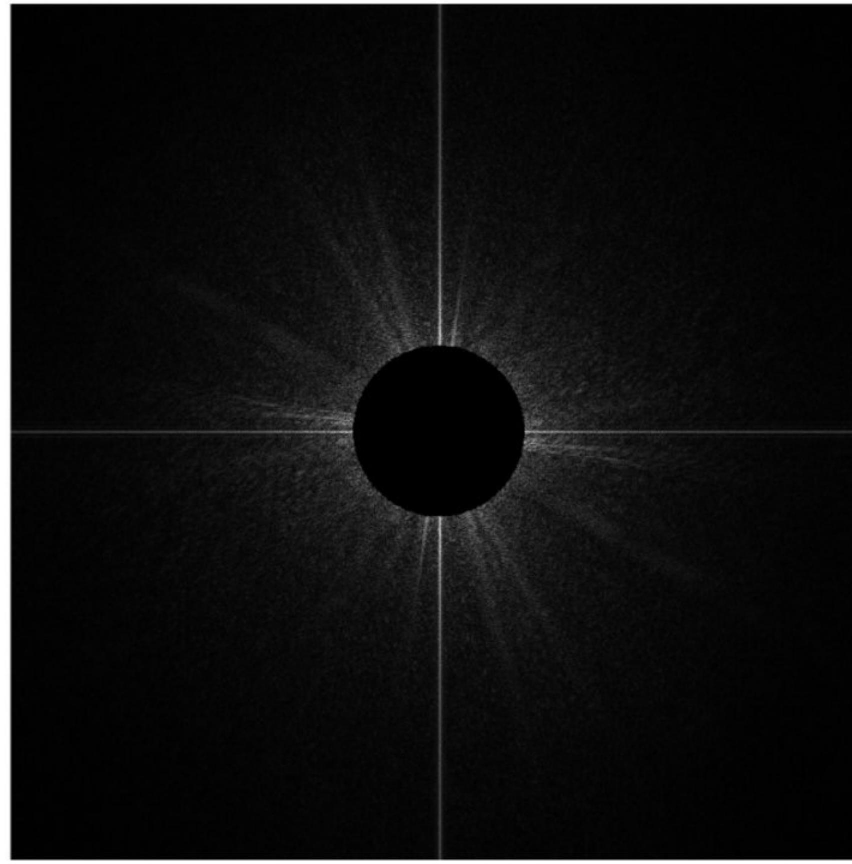


Frequency domain
(after band-pass filter)

High frequencies (edges)

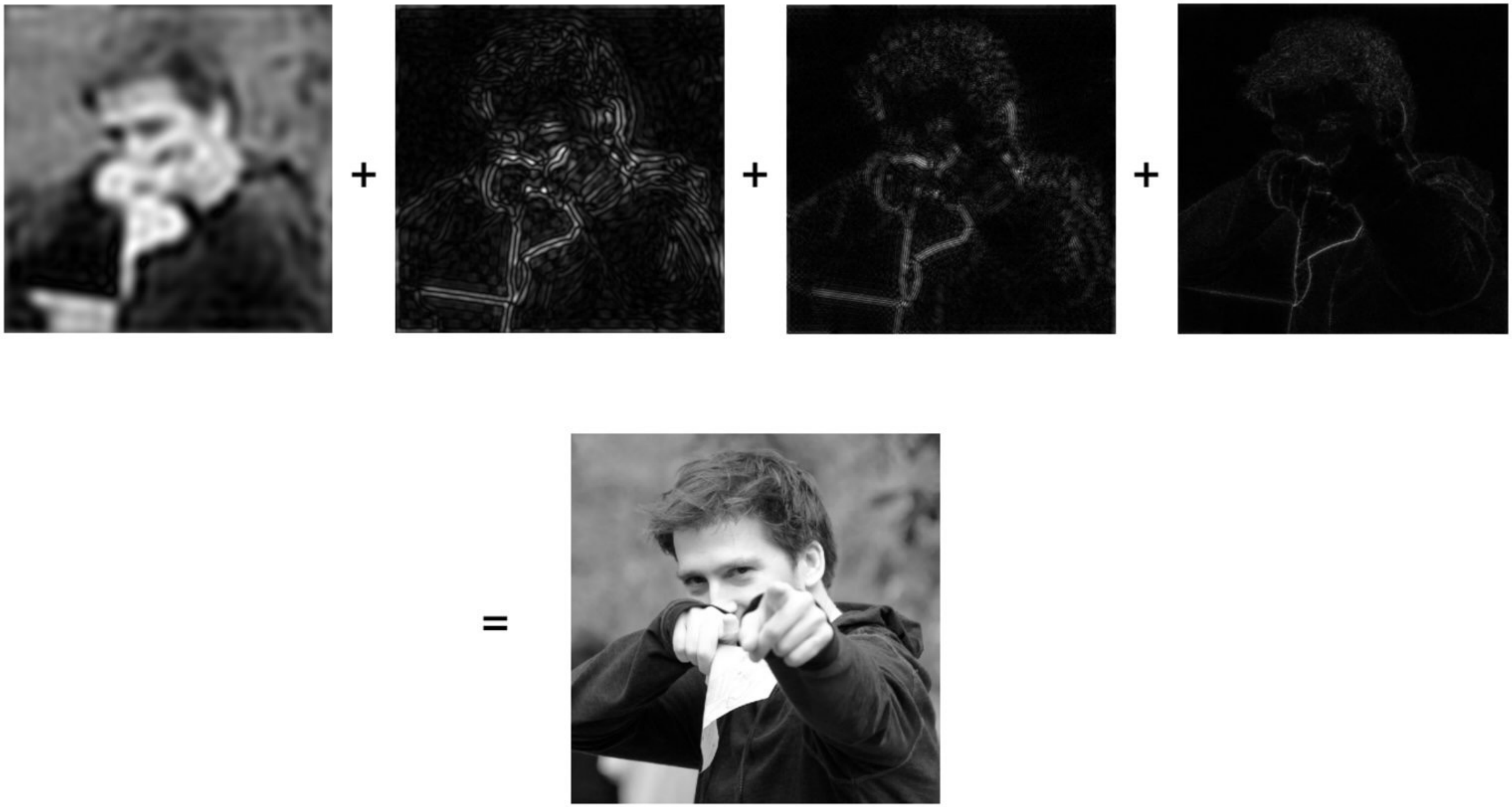


Spatial domain
(strongest edges)



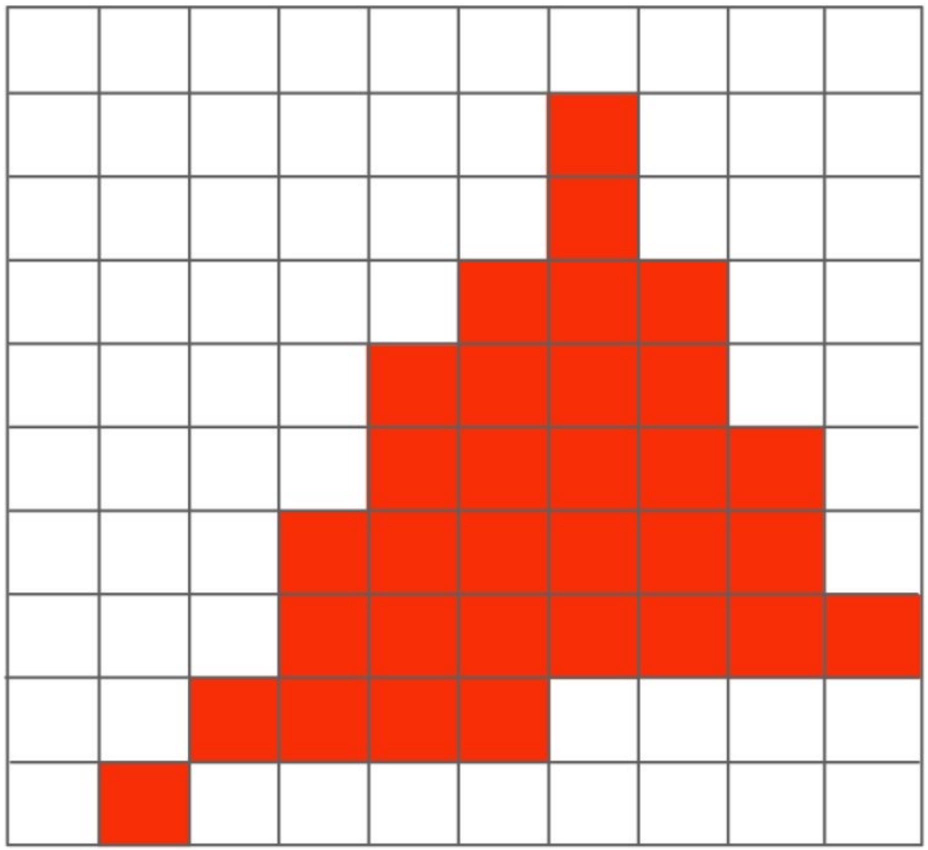
Frequency domain
(after high-pass filter)
All frequencies below threshold have 0 magnitude

An image as a sum of its frequency components



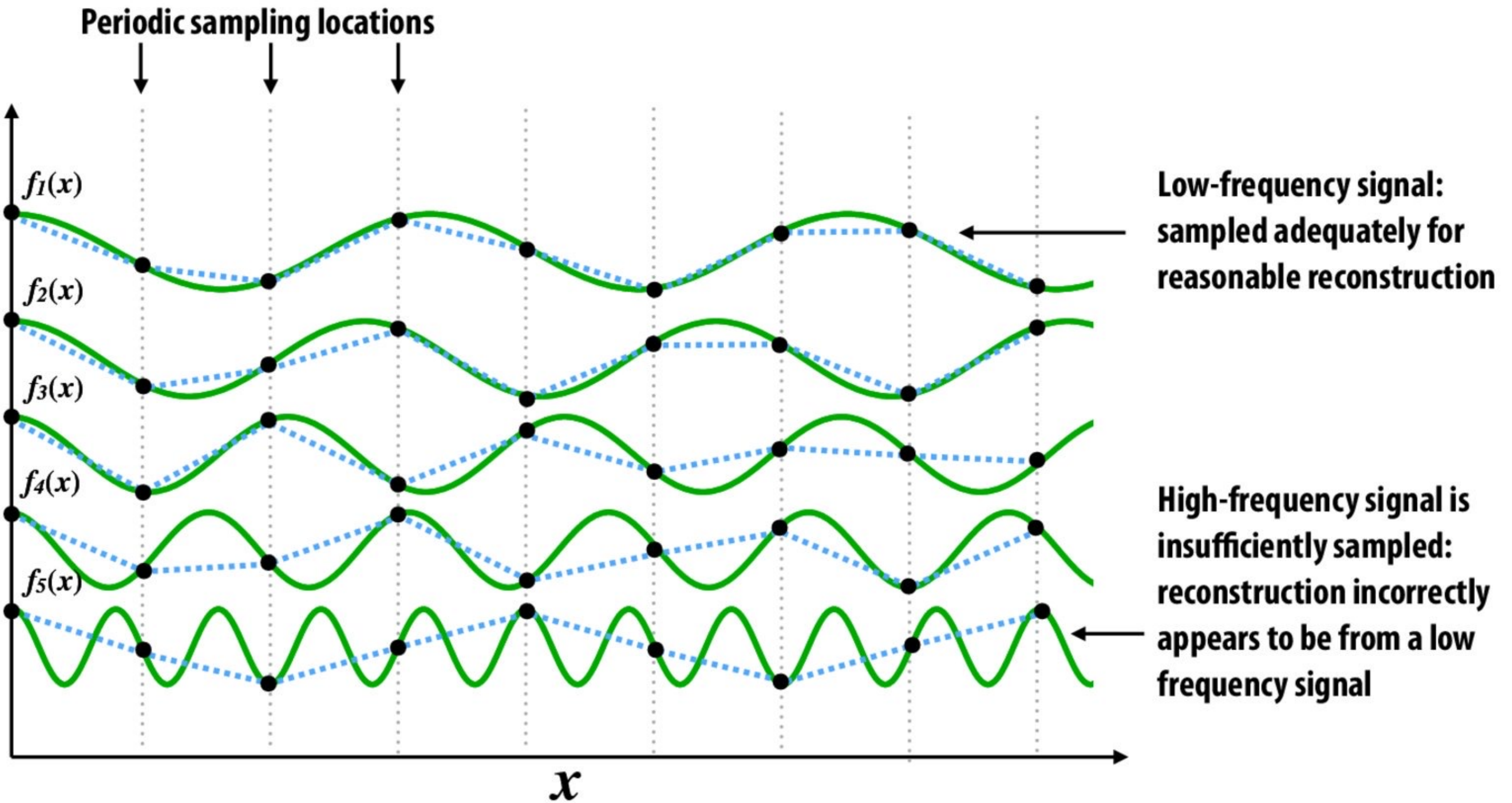
Pre-filtering for anti-aliasing

Back to our problem of artifacts in images

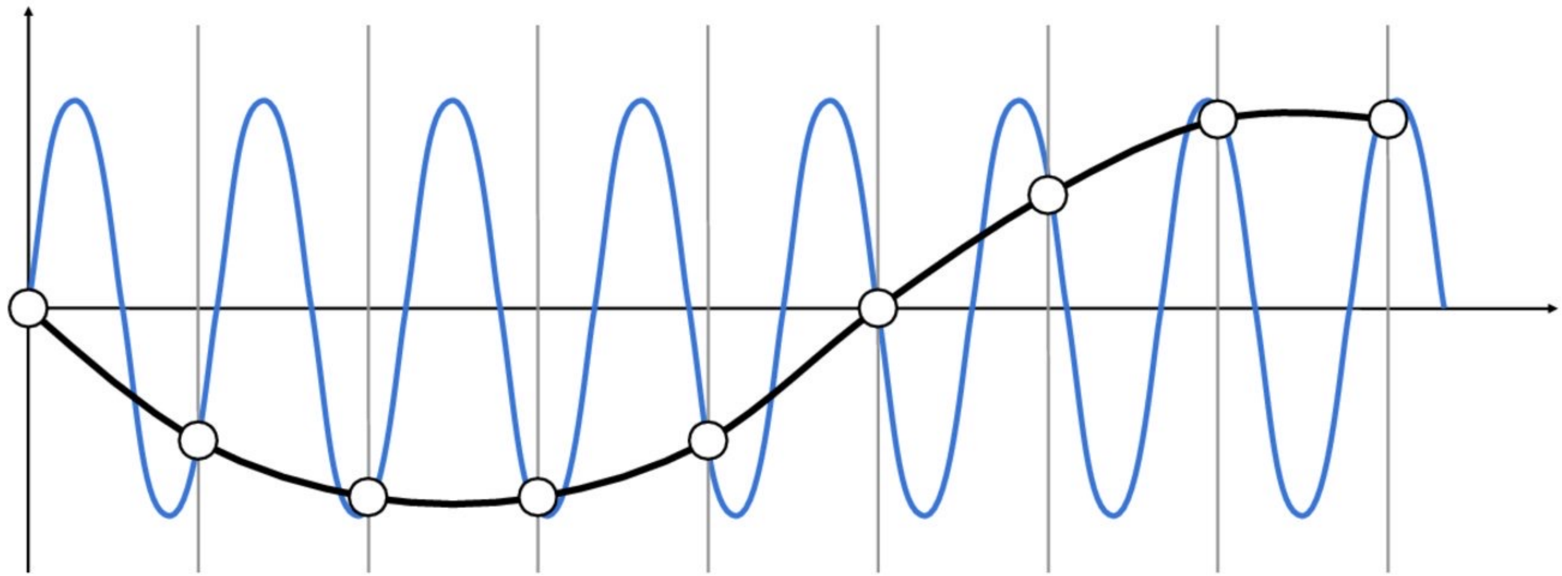


Jaggies!

Higher frequencies need denser sampling



Undersampling creates frequency "aliases"

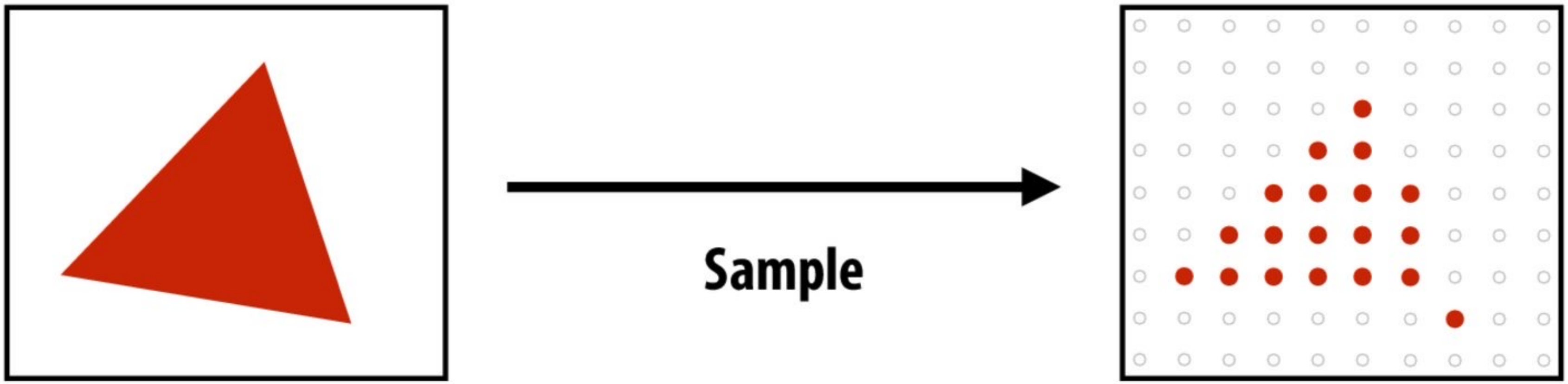


High-frequency signal is insufficiently sampled: samples erroneously appear to be from a low-frequency signal

Two frequencies that are indistinguishable at a given sampling rate are called "aliases"

Anti-aliasing idea: filter out high frequencies before sampling

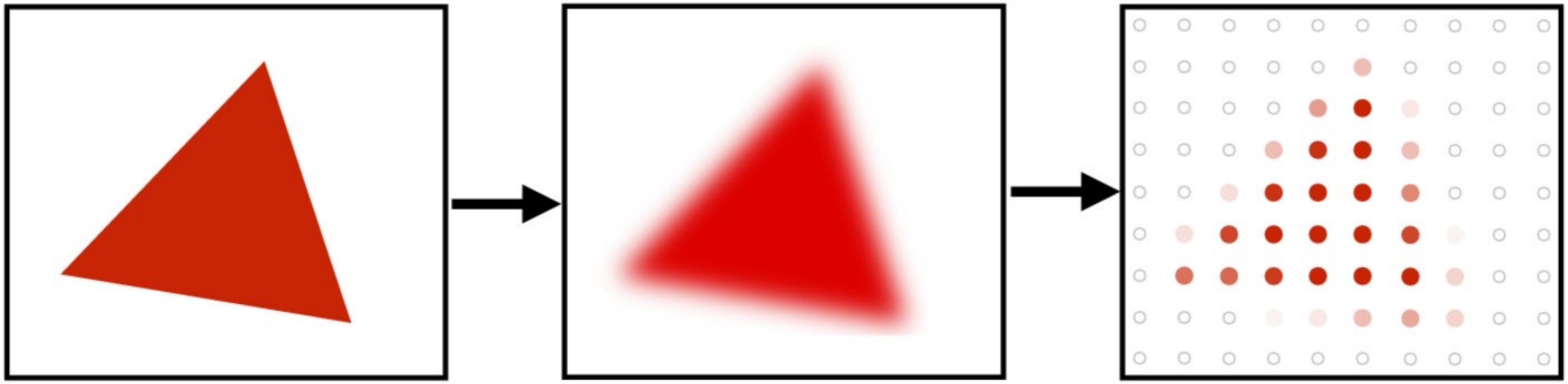
Rasterization: point sampling in 2D space



Note jaggies in rasterized triangle

(pixel values are either red or white: sample is in or out of triangle)

Rasterization: anti-aliased sampling



Pre-filter
(remove frequencies above Nyquist limit)

Sample

Note anti-aliased edges of rasterized triangle: where pixel values take intermediate values

How much pre-filtering do we need to avoid aliasing?

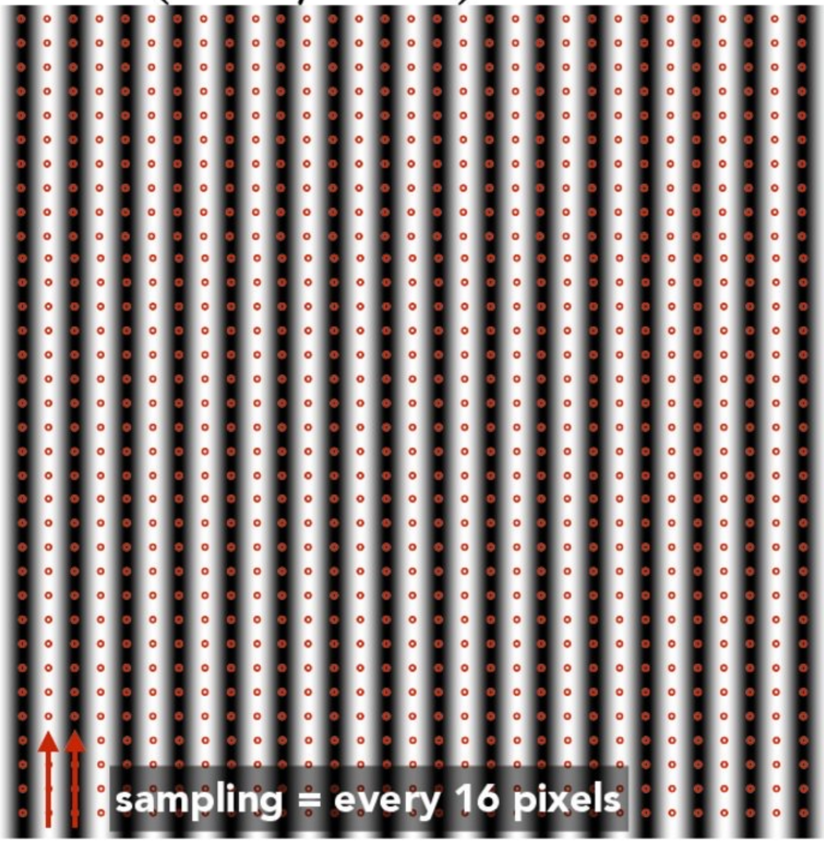
Nyquist theorem

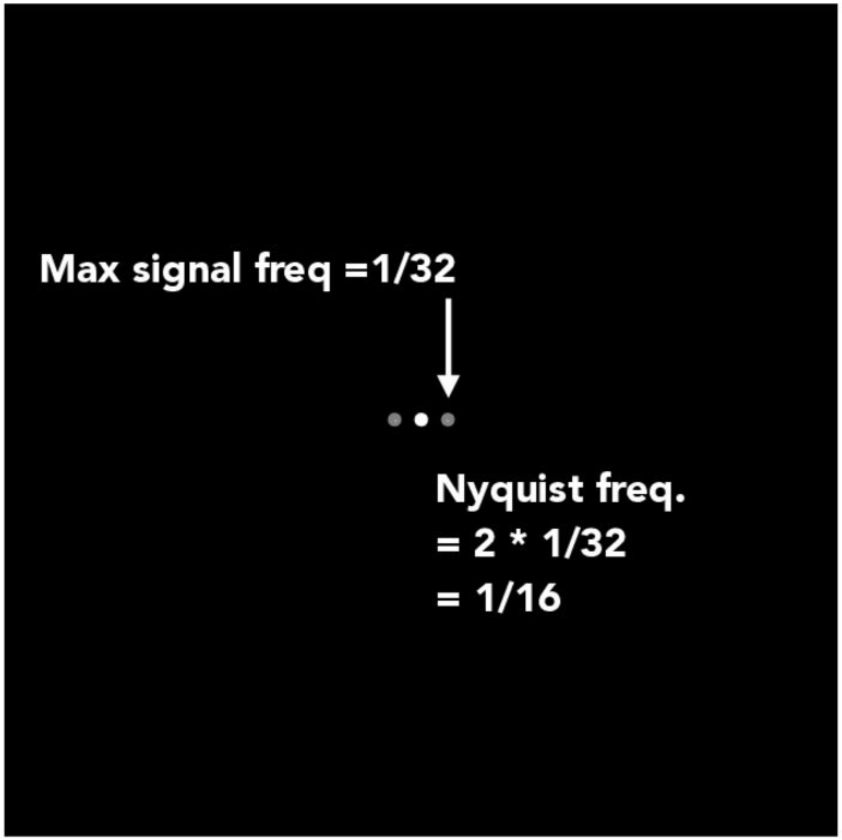
Theorem: We get no aliasing from frequencies in the signal that are less than the Nyquist frequency (which is defined as half the sampling frequency)

Consequence: sampling at twice the highest frequency in the signal will eliminate aliasing

Signal vs Nyquist frequency: example

$\sin(2\pi/32)x$ — **frequency 1/32; 32 pixels per cycle**



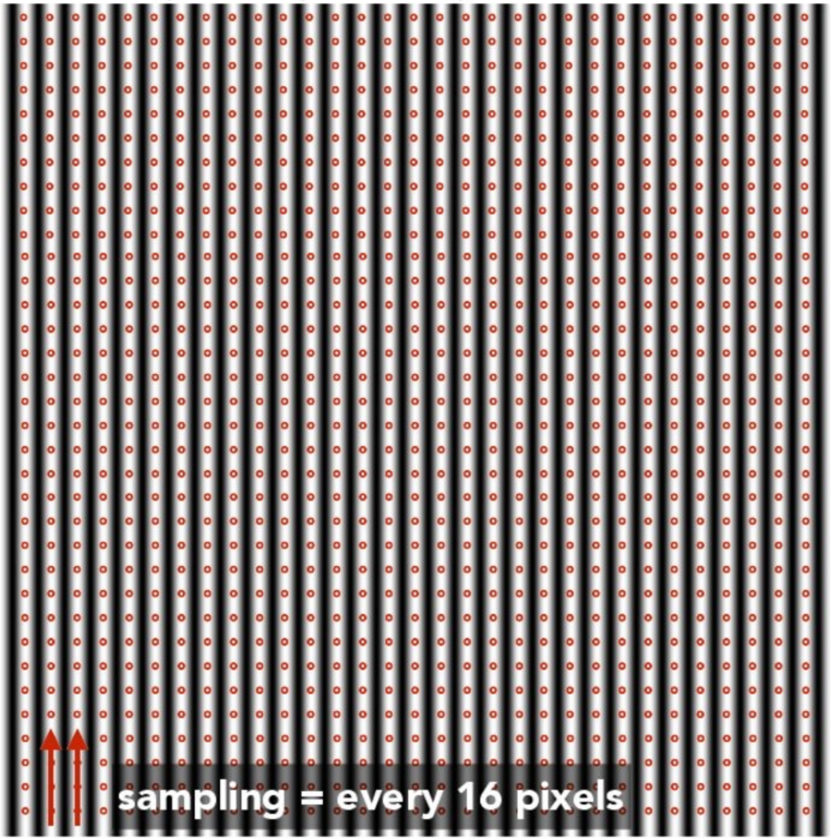


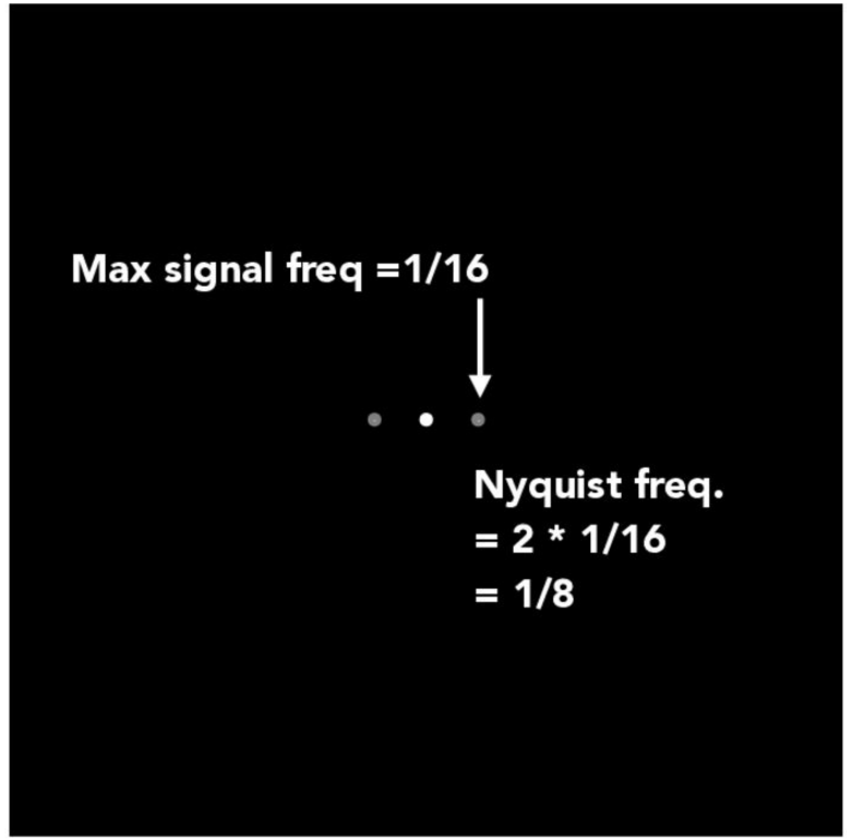
Spatial domain **Frequency domain**

No Aliasing!

Signal vs Nyquist frequency: example

$\sin(2\pi/16)x$ — **frequency 1/16; 16 pixels per cycle**



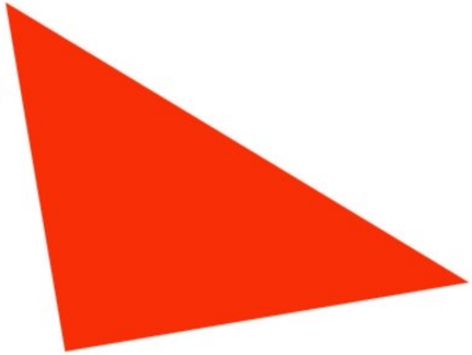


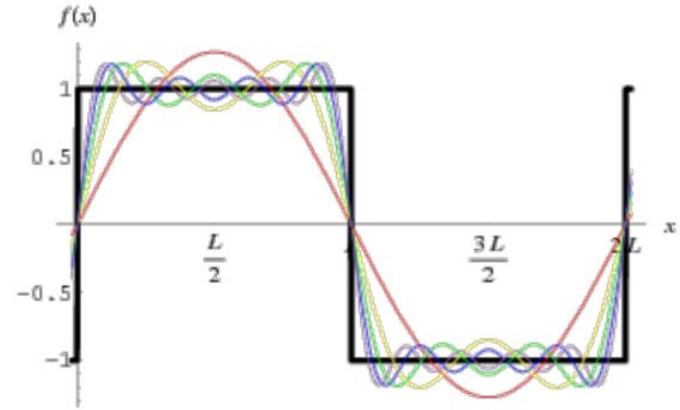
Aliasing! (due to undersampling)

Challenges of sampling-based approaches in graphics

- **Our signals are not always band-limited in computer graphics. Why?**

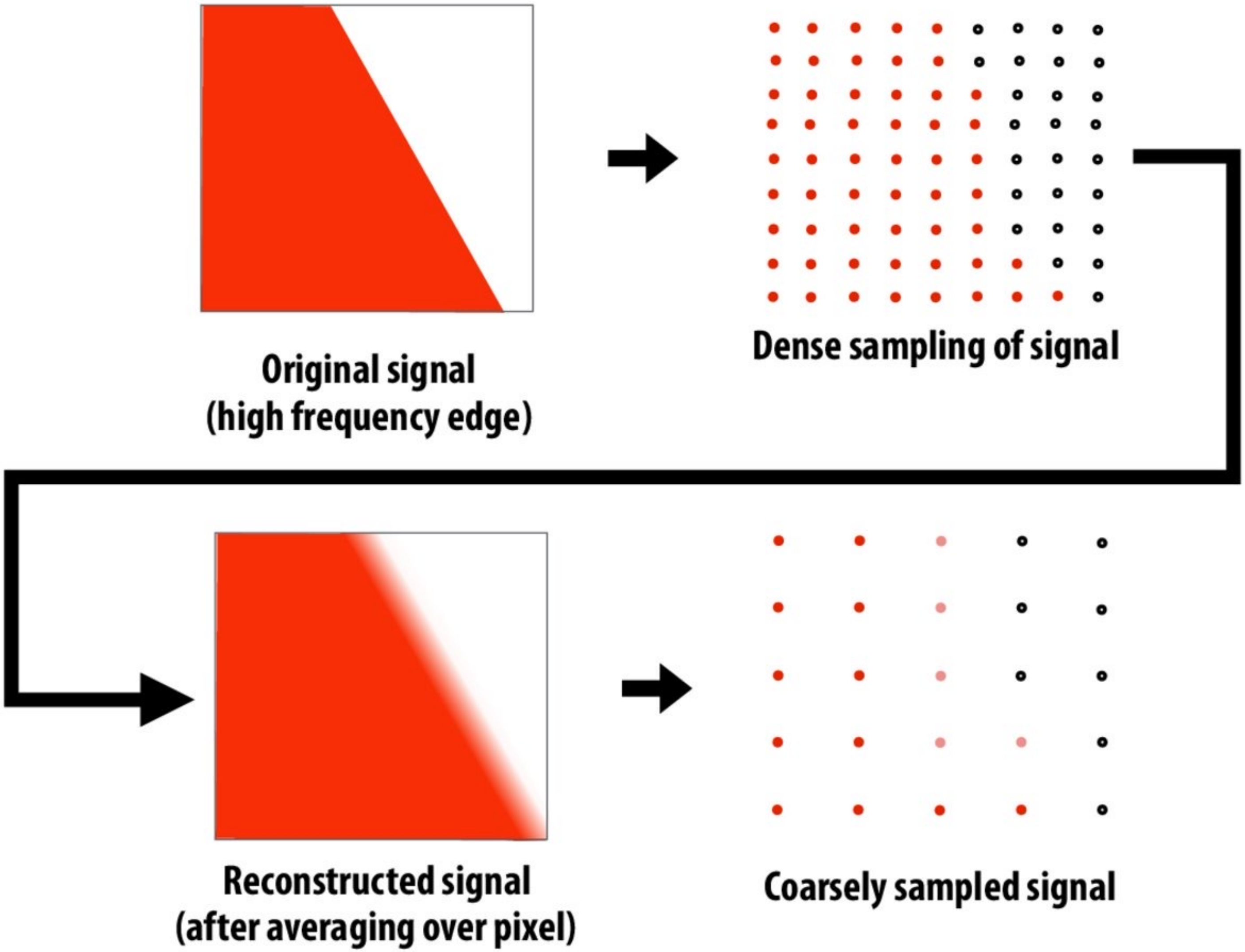
Hint:





Convolution

Recall our anti-aliasing technique in the first half of lecture



Convolution

Signal

1	3	5	3	7	1	3	8	6	4

Filter

1	2	1

Convolution

Signal	1	3	5	3	7	1	3	8	6	4

Filter	1	2	1

1x1 + 3x2 + 5x1 = 12

Result	12									

Convolution

Signal	1	3	5	3	7	1	3	8	6	4
Filter		1	2	1						

3x1 + 5x2 + 3x1 = 16

Result	12	16								

Convolution

Signal	1	3	5	3	7	1	3	8	6	4
Filter			1	2	1					

5x1 + 3x2 + 7x1 = 18

Result	12	16	18							

Discrete 2D convolution

$$\underbrace{(f * g)(x, y)}_{\text{output image}} = \sum_{i,j=-\infty}^{\infty} \underbrace{f(i, j)}_{\text{filter}}\underbrace{I(x - i, y - j)}_{\text{input image}}$$

Consider $f(i, j)$ **that is nonzero only when:** $-1 \leq i, j \leq 1$

Then:

$$(f * g)(x, y) = \sum_{i,j=-1}^{1} f(i, j)I(x - i, y - j)$$

And we can represent f(i,j) as a 3x3 matrix of values where:

$$f(i, j) = \mathbf{F}_{i,j}$$

(often called: "filter weights", "filter kernel")

Box filter (used in a 2D convolution)

$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Example: 3x3 box filter

2D convolution with box filter blurs the image

Original image

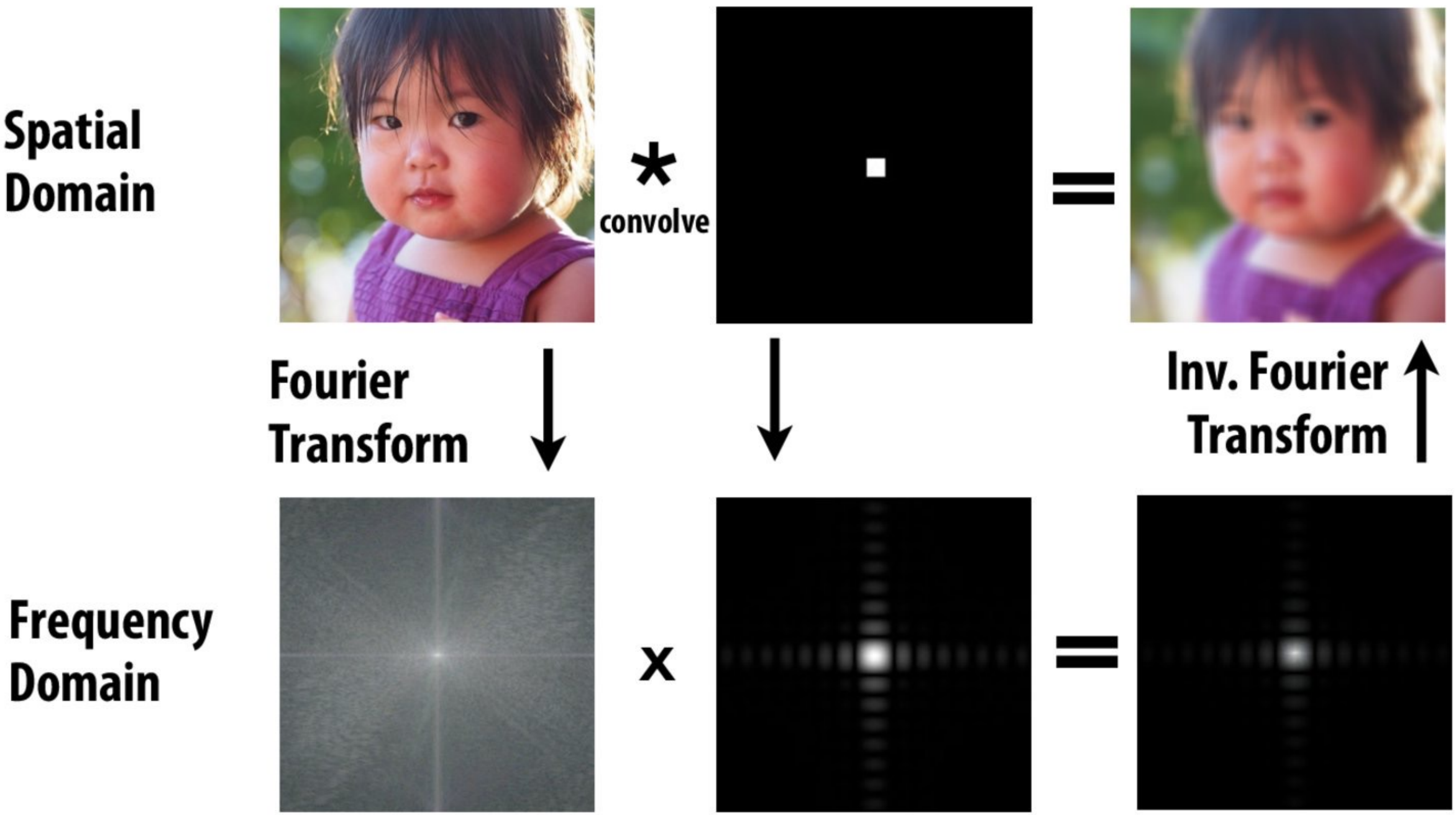
Blurred
(convolve with box filter)

Hmm… this reminds me of a low-pass filter…

Convolution Theorem

Convolution theorem

Convolution in the spatial domain is equal to multiplication in the frequency domain, and vice versa



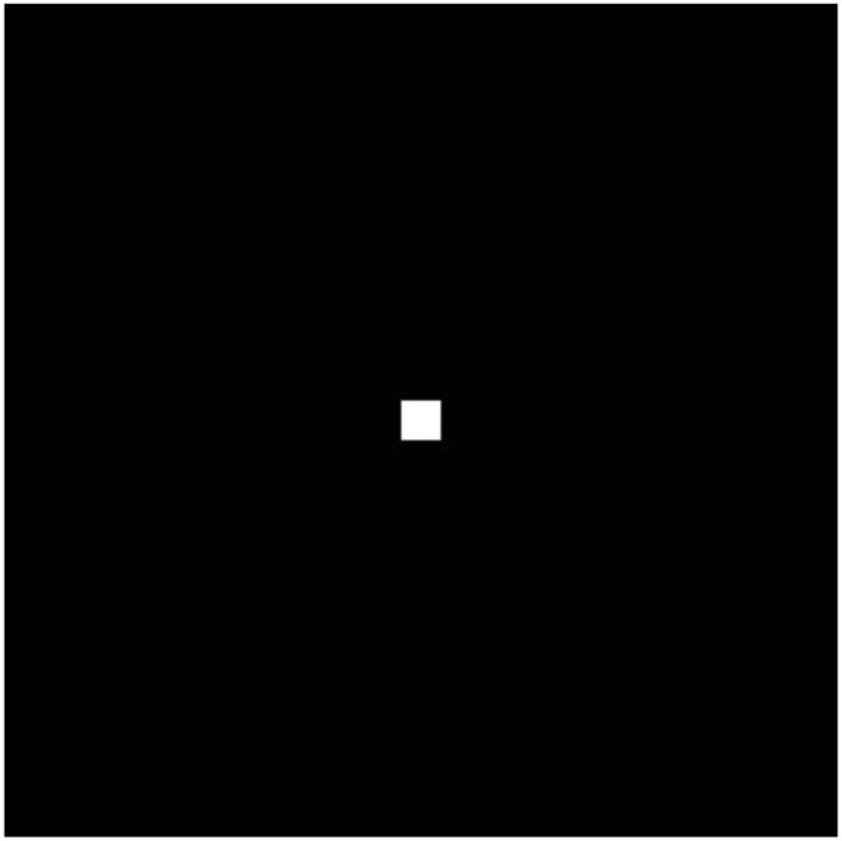
Convolution theorem

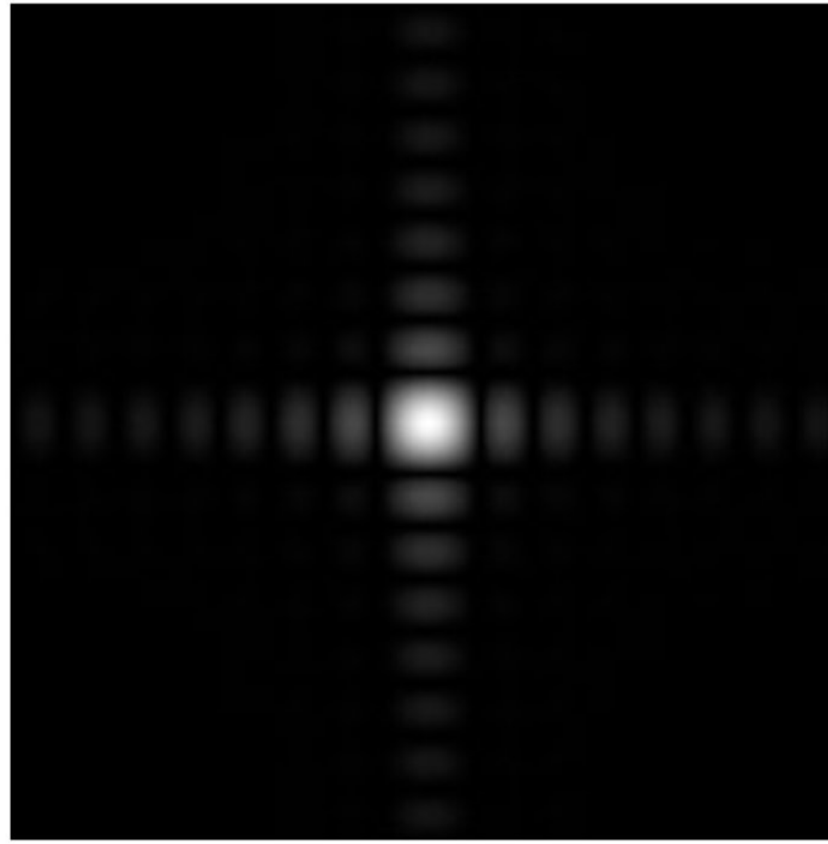
- Convolution in the spatial domain is equal to multiplication in the frequency domain, and vice versa

- Pre-filtering option 1:
  - Filter by convolution in the spatial domain

- Pre-filtering option 2:
  - Transform to frequency domain (Fourier transform)
  - Multiply by Fourier transform of convolution kernel
  - Transform back to spatial domain (inverse Fourier)

Box function = "low pass" filter

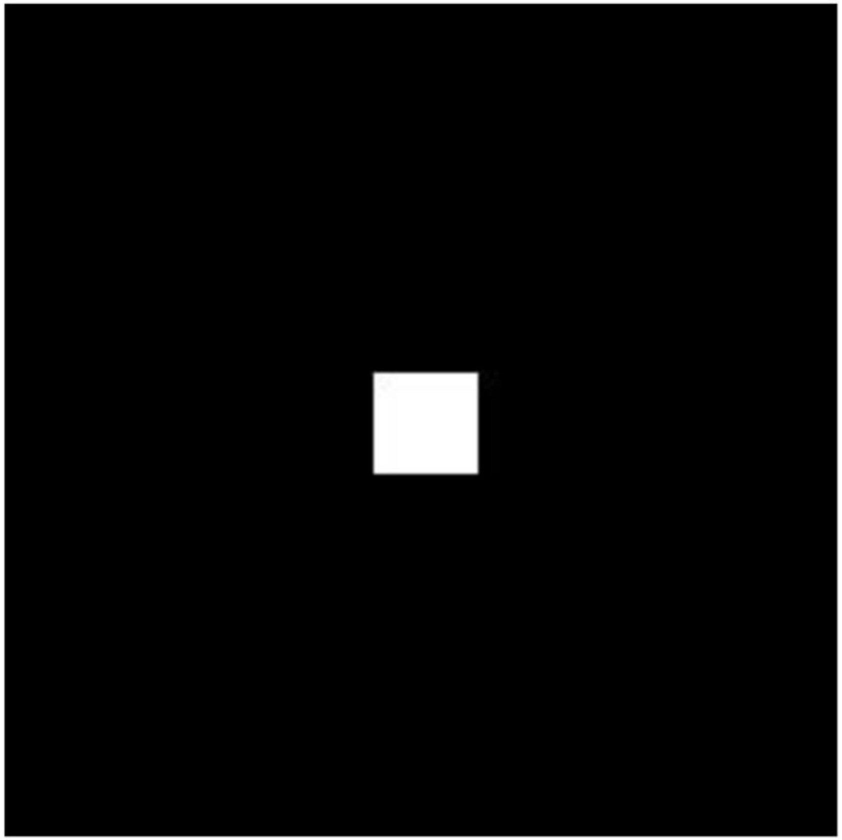




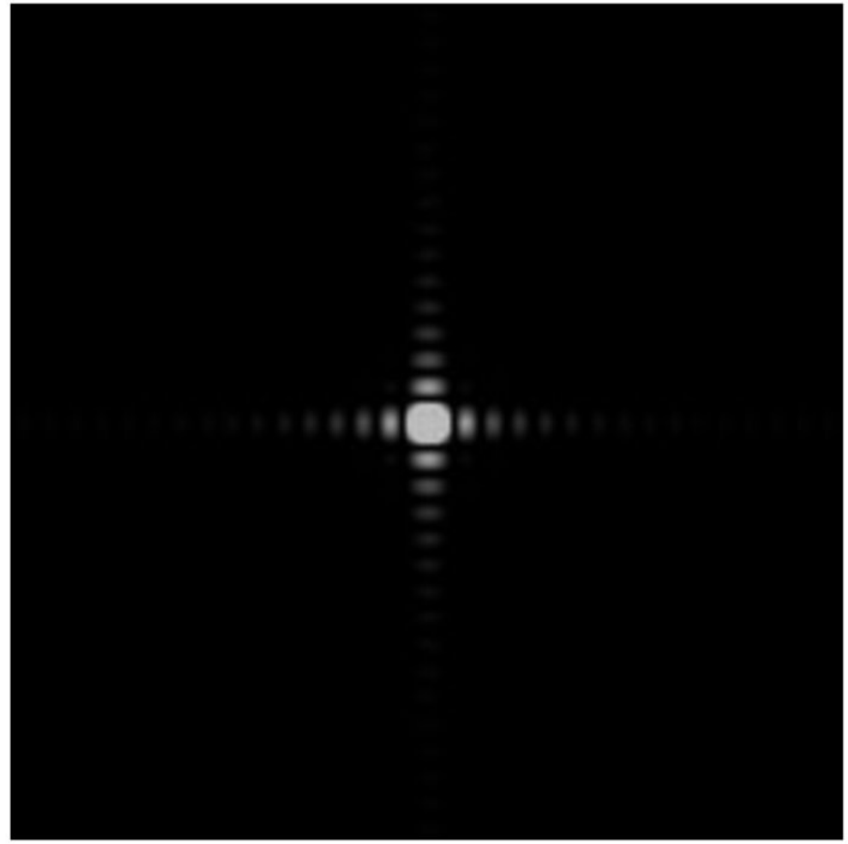
Spatial domain

Frequency domain

Wider filter kernel = lower frequencies



Spatial domain

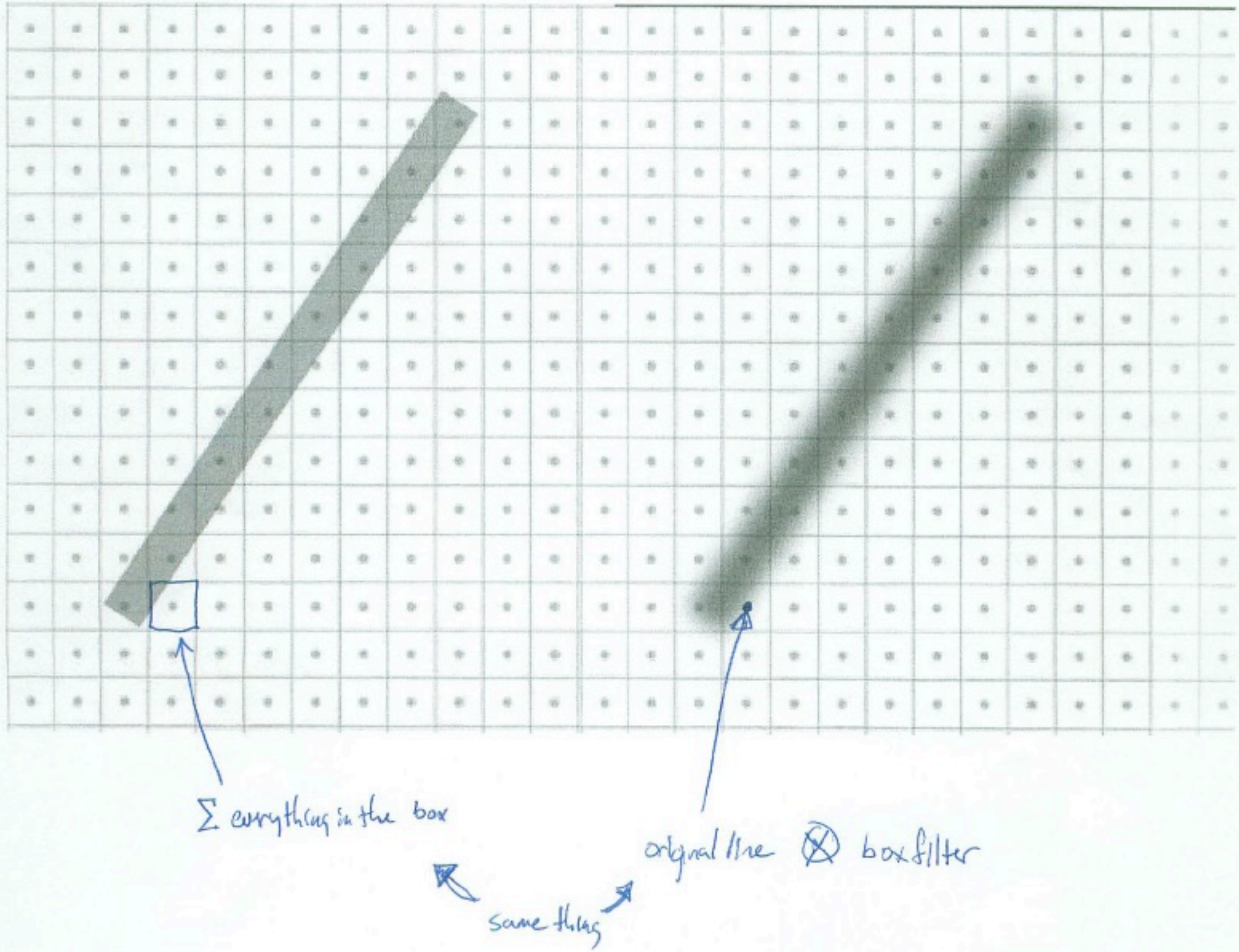


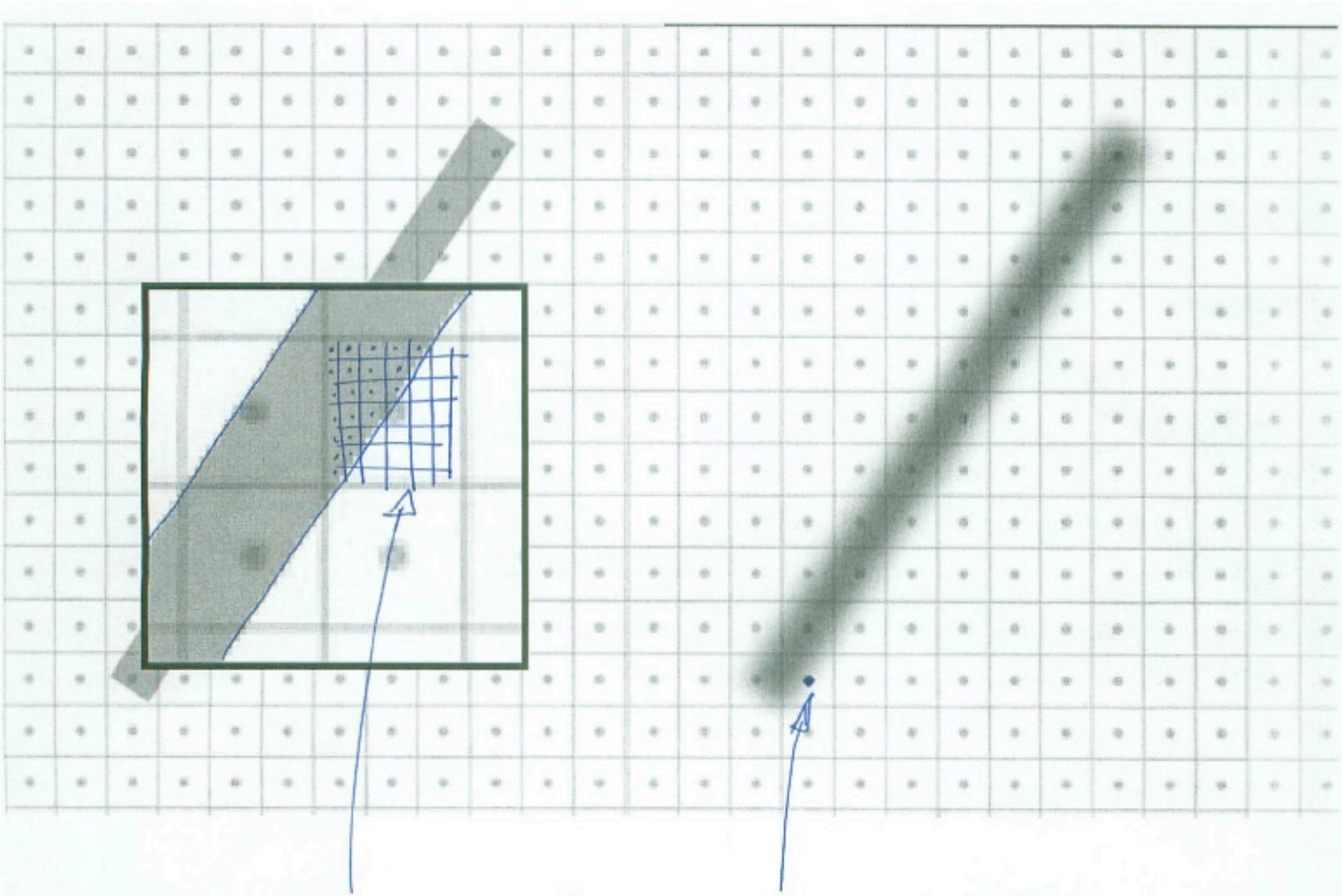
Frequency domain

How can we reduce aliasing error?

- Increase sampling rate (increase Nyquist frequency)
  - Higher resolution displays, sensors, framebuffers…
  - But: costly and may need very high resolution to sufficiently reduce aliasing

- Anti-aliasing
  - Simple idea: remove (or reduce) signal frequencies above the Nyquist frequency before sampling
  - How to filter out high frequencies before sampling?


Σ everything in the box
original line ⊗ box filter
same thing

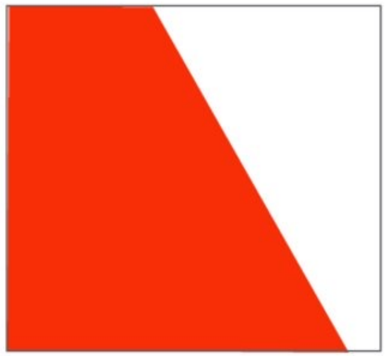


discrete approximation
to integral

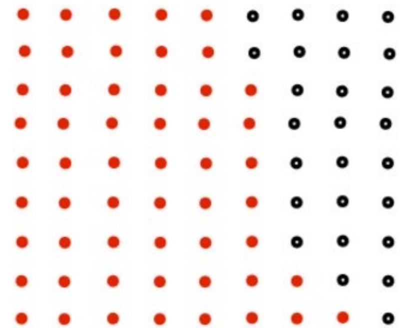
"supersample" at "subpixel"
locations and sum to
get value of normal sample

continuous case is analytic
convolution of box filter

Putting it all together: anti-aliasing via supersampling

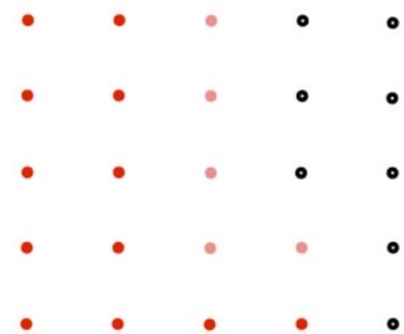


Original signal
(with high frequency edge)



Dense sampling of signal
(supersampling)

Reconstructed signal
(averaging over pixel (via convolution) yields new signal with high frequencies removed)

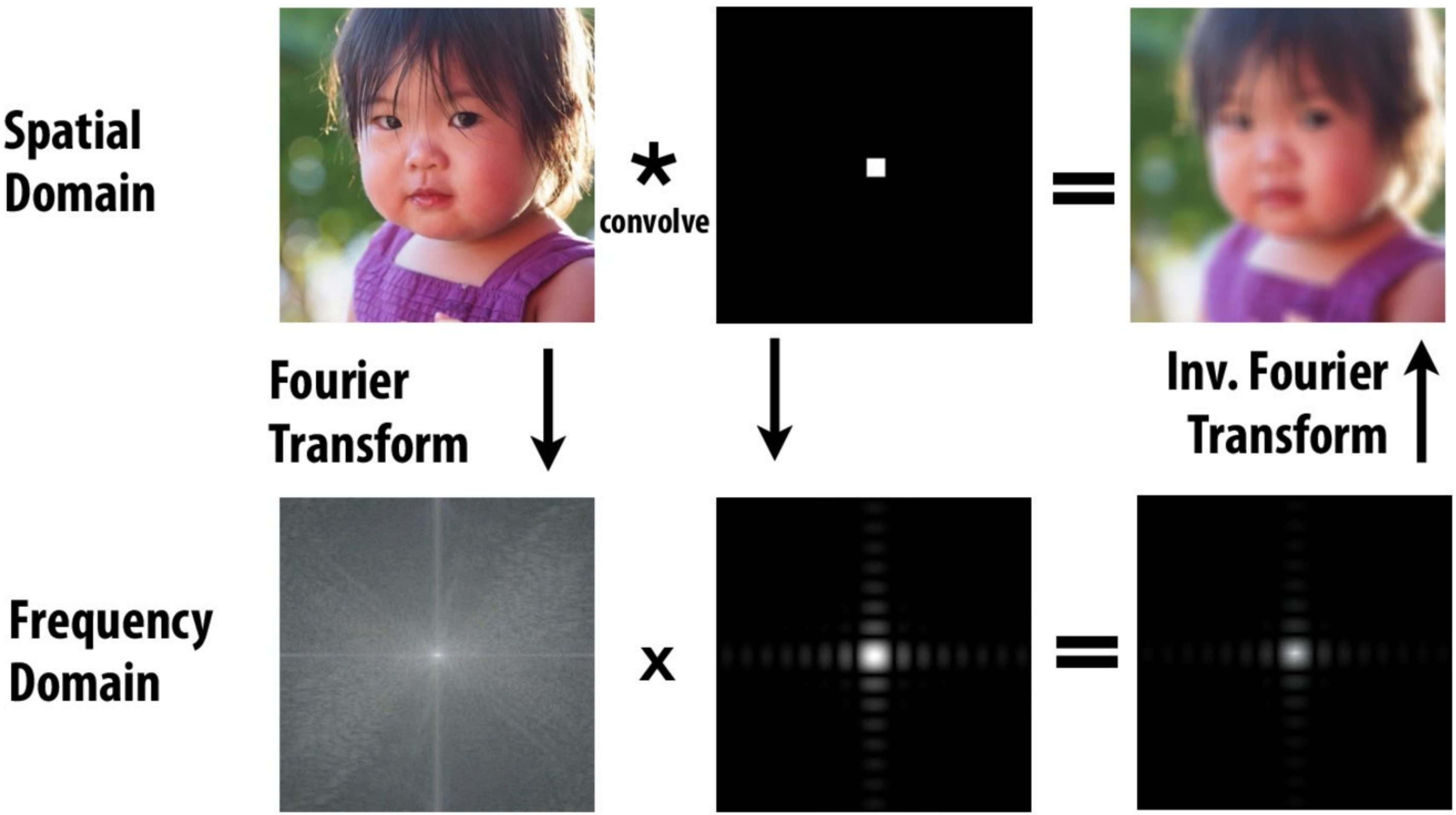


Coarse sampling of reconstructed signal exhibits less aliasing

Why so complicated?

Convolution theorem

Convolution in the spatial domain is equal to multiplication in the frequency domain, and vice versa



Convolution theorem

Convolution in the spatial domain is equal to multiplication in the frequency domain, and vice versa

Spatial Domain

* convolve

Box filter (used in a 2D convolution)

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

Example: 3x3 box filter

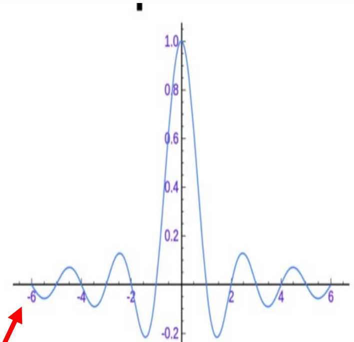
=

Fourier Transform

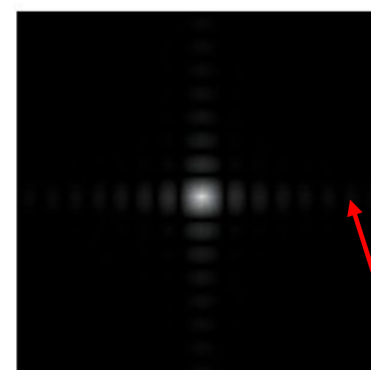
Inv. Fourier Transform

Frequency Domain

x



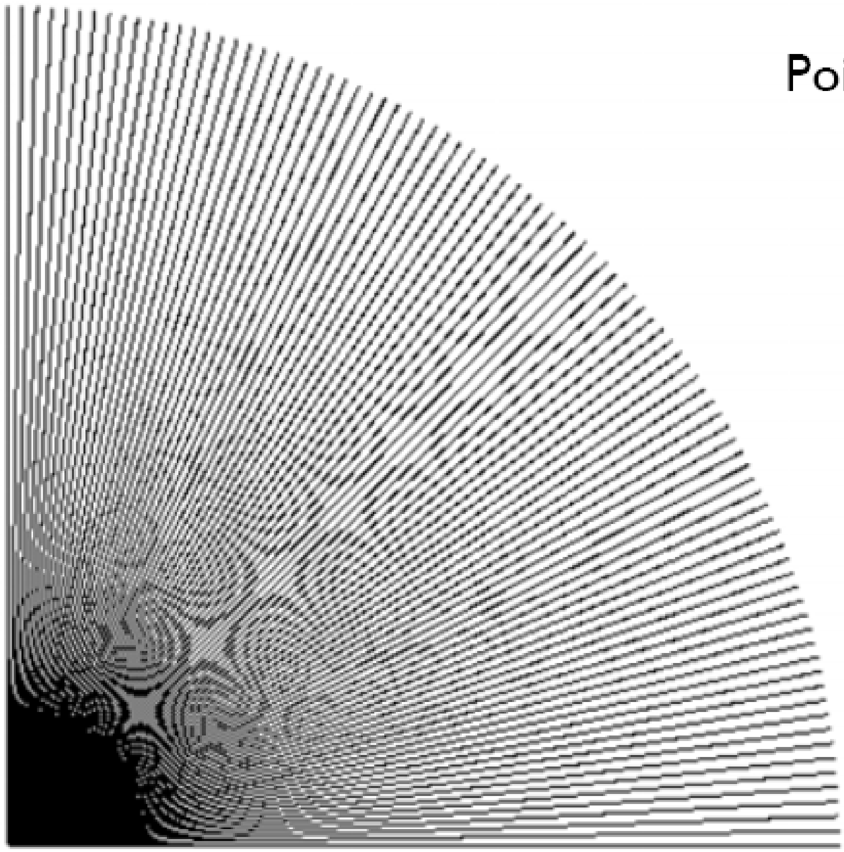
=



Infinite extent

Sinc filter

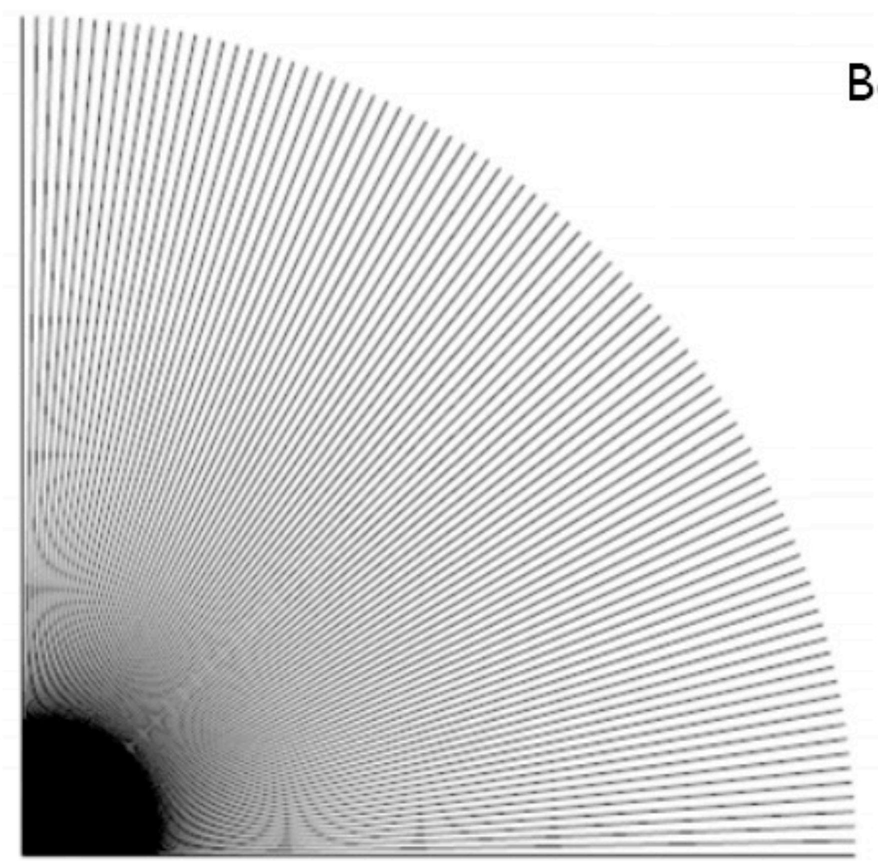
High frequency not really gone



Point sampling in action

How do I get rid of the rest of these artifacts?

Go learn more about signal processing. It’s a major tool in your mental toolbox.



Box filtering in action

Administrative

Due Dates

- Due Tomorrow
  - Quiz4
- Due next Monday
  - A4 (Lighting)

Q&A

End

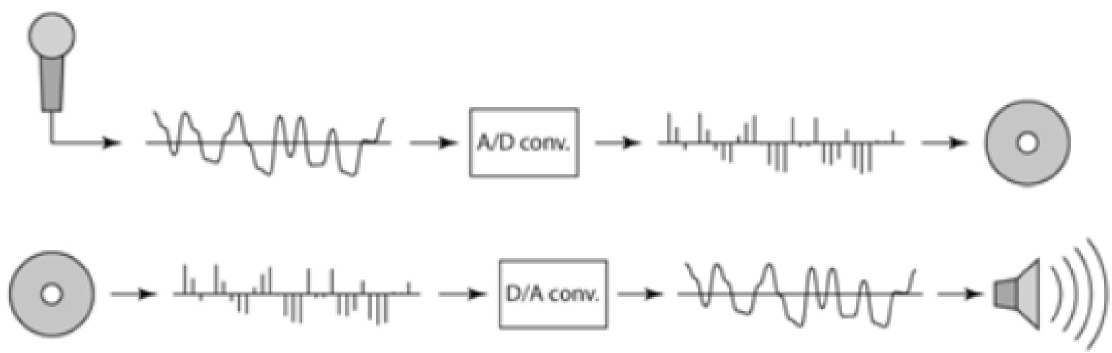
Not using slides below this year

Roots of sampling

- Nyquist 1928; Shannon 1949
  - famous results in information theory
- 1940s: first practical uses in telecommunications
- 1960s: first digital audio systems
- 1970s: commercialization of digital audio
- 1982: introduction of the Compact Disc
  - the first high-profile consumer application
- This is why all the terminology has a communications or audio "flavor"
  - early applications are 1D; for us 2D (images) is important

Sampling in digital audio

- Recording: sound to analog to samples to disc
- Playback: disc to samples to analog to sound again
  – how can we be sure we are filling in the gaps correctly?



What if our samples missed something important?

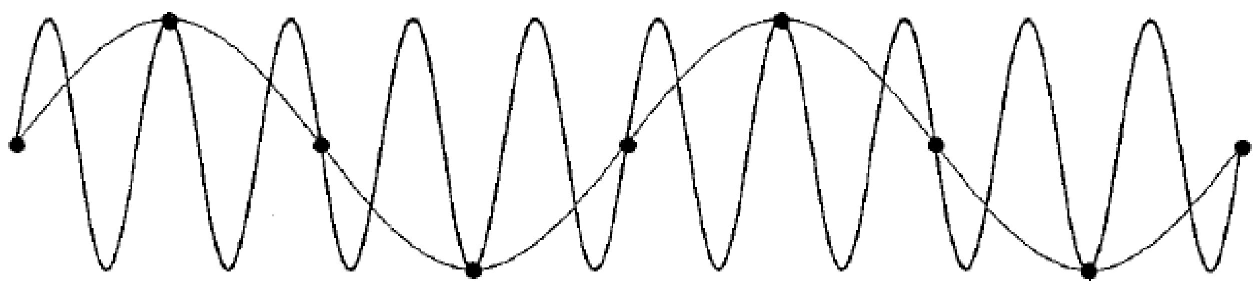
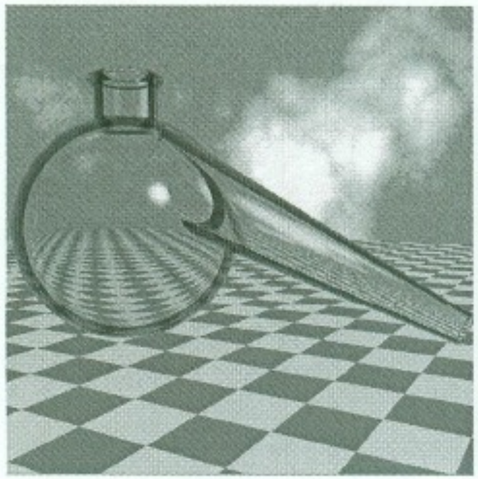
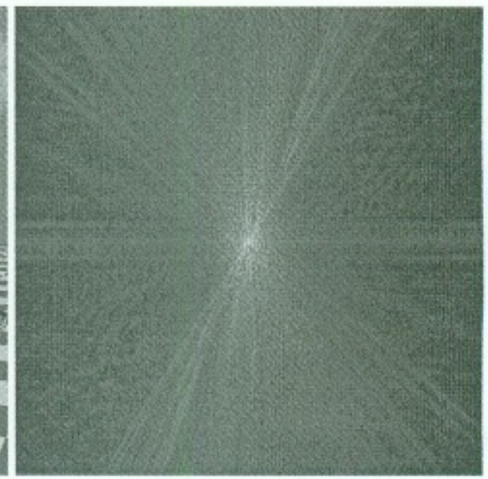


Fig. 14.17 Sampling below the Nyquist rate. (Courtesy of George Wolberg, Columbia University.)

Spatial and Frequency Domain



Spatial Domain



Frequency Domain

Convolution and multiplication

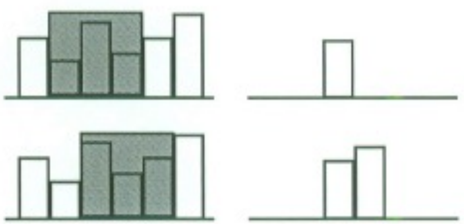
- They are dual to one another under F.T.

$$\mathcal{F}\{f * g\}(u) = F(u)G(u)$$
$$\mathcal{F}\{fg\}(u) = (F * G)(u)$$

- Lowpass filters
  - Most of our "blurring" filters have most of their F.T. at low frequencies
  - Therefore they attenuate higher frequencies

Filtering: Spatial Domain

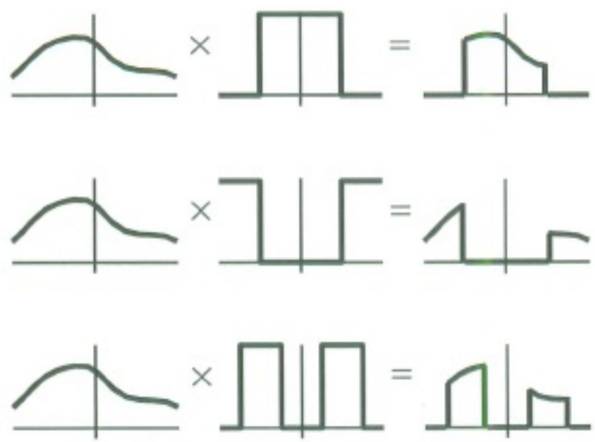
Filtering



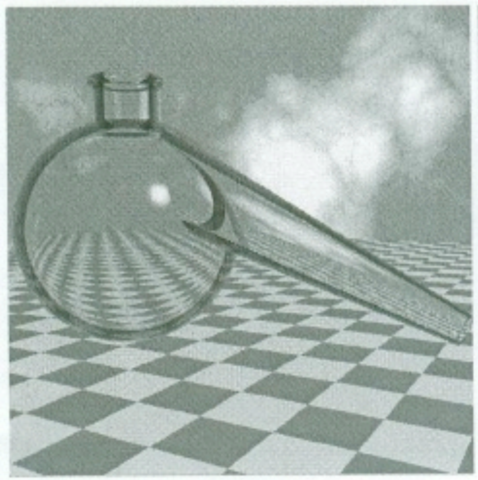
Convolution of two functions

$$h(x) = f \otimes g = \int f(y)g(x-y)\,dy$$

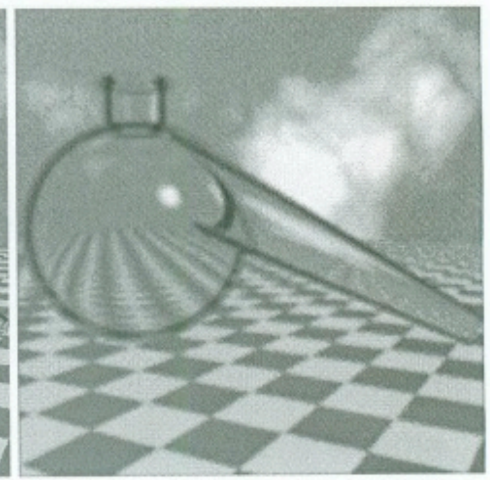
Filtering: Frequency Domain



Low-Pass Filter

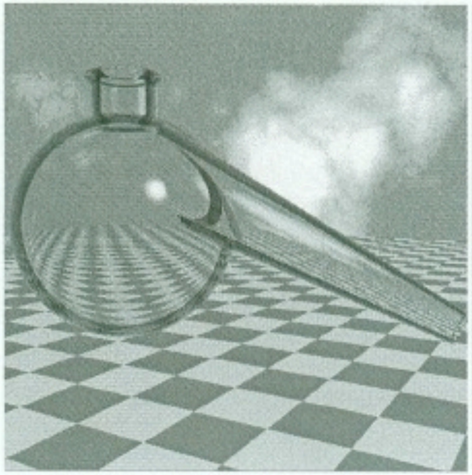


Original

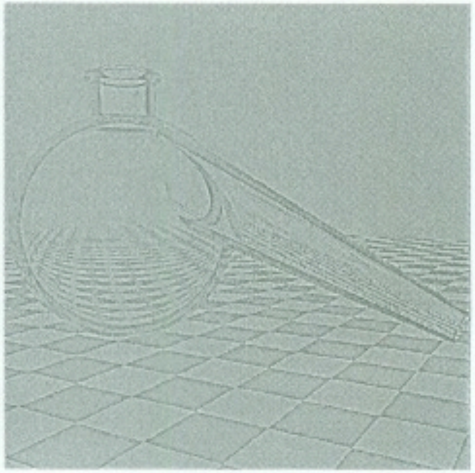


Blurred

High-Pass Filter

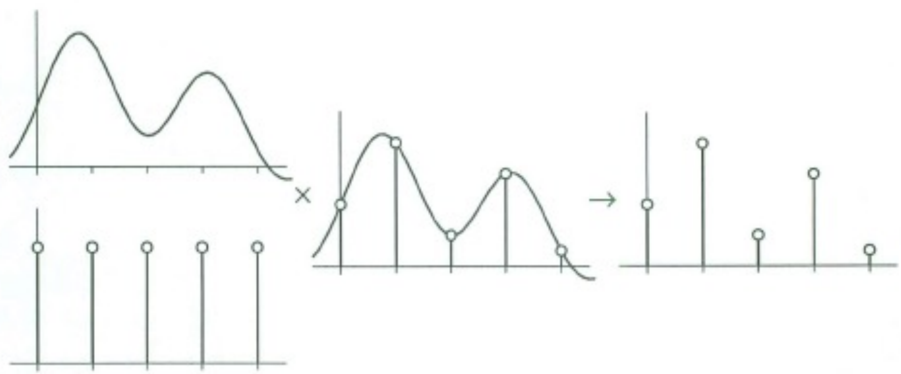


Original

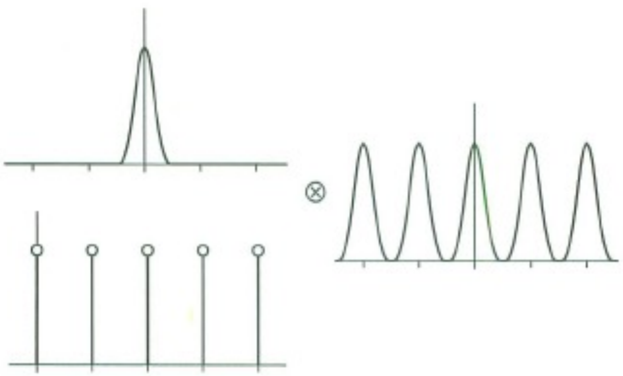


Edge enhancement

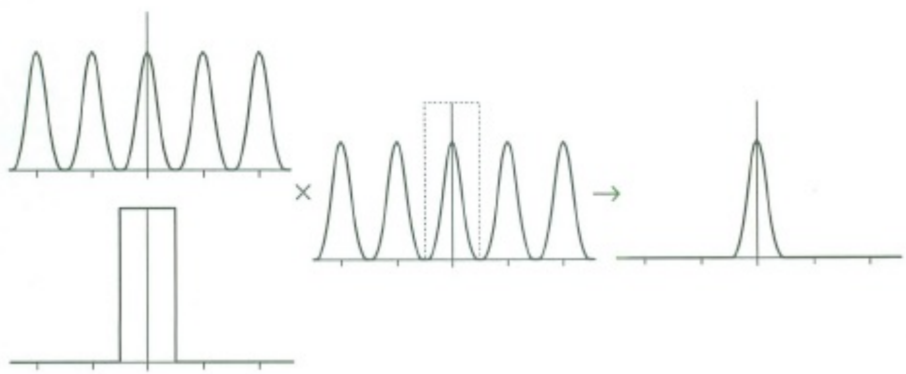
Sampling: Spatial Domain



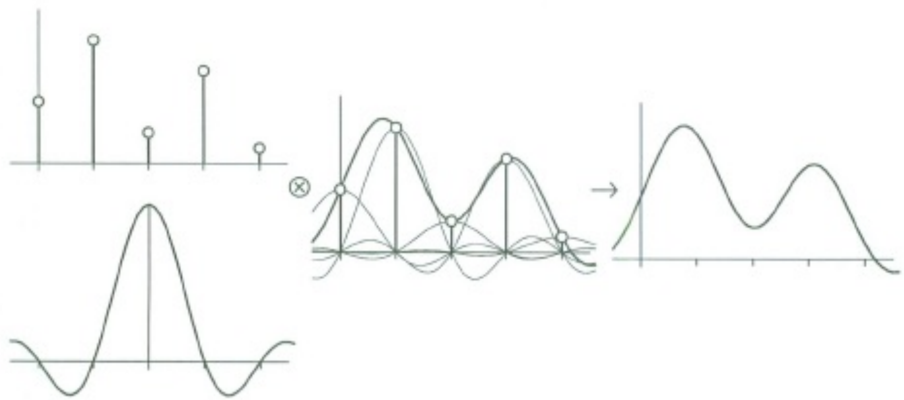
Sampling: Frequency Domain



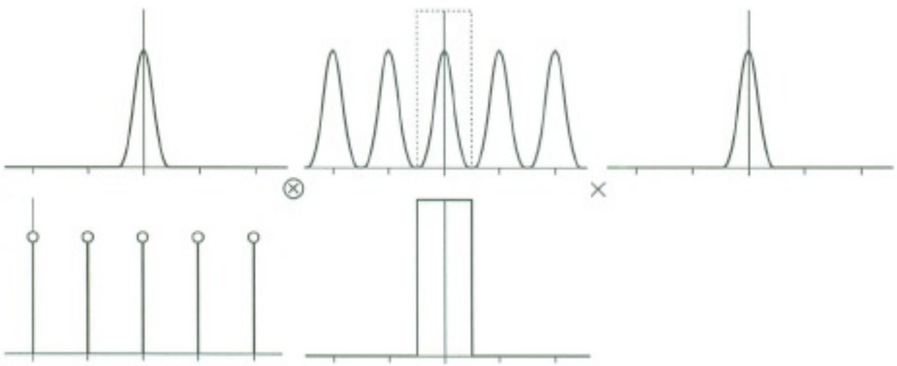
Reconstruction: Frequency Domain

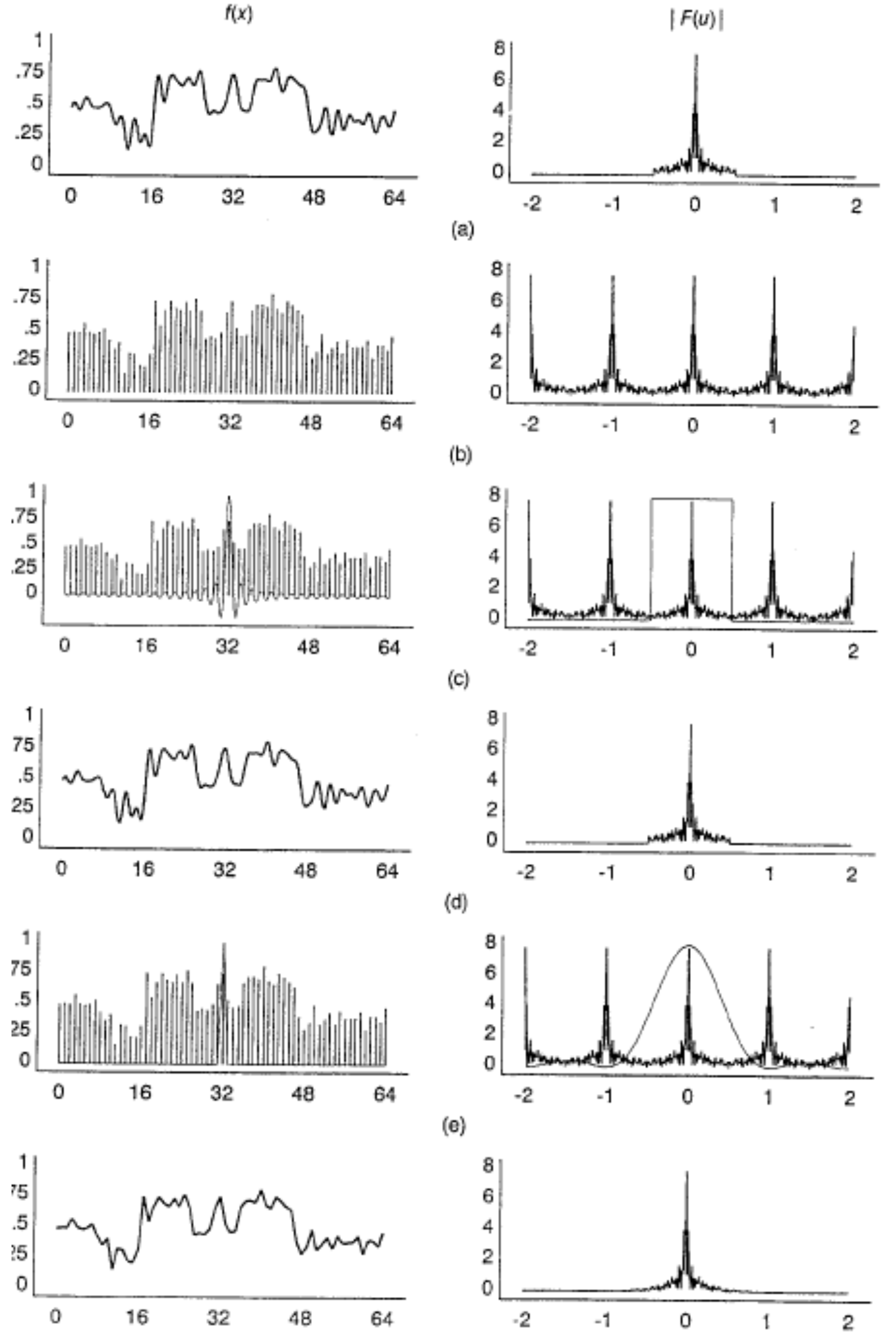


Reconstruction: Spatial Domain

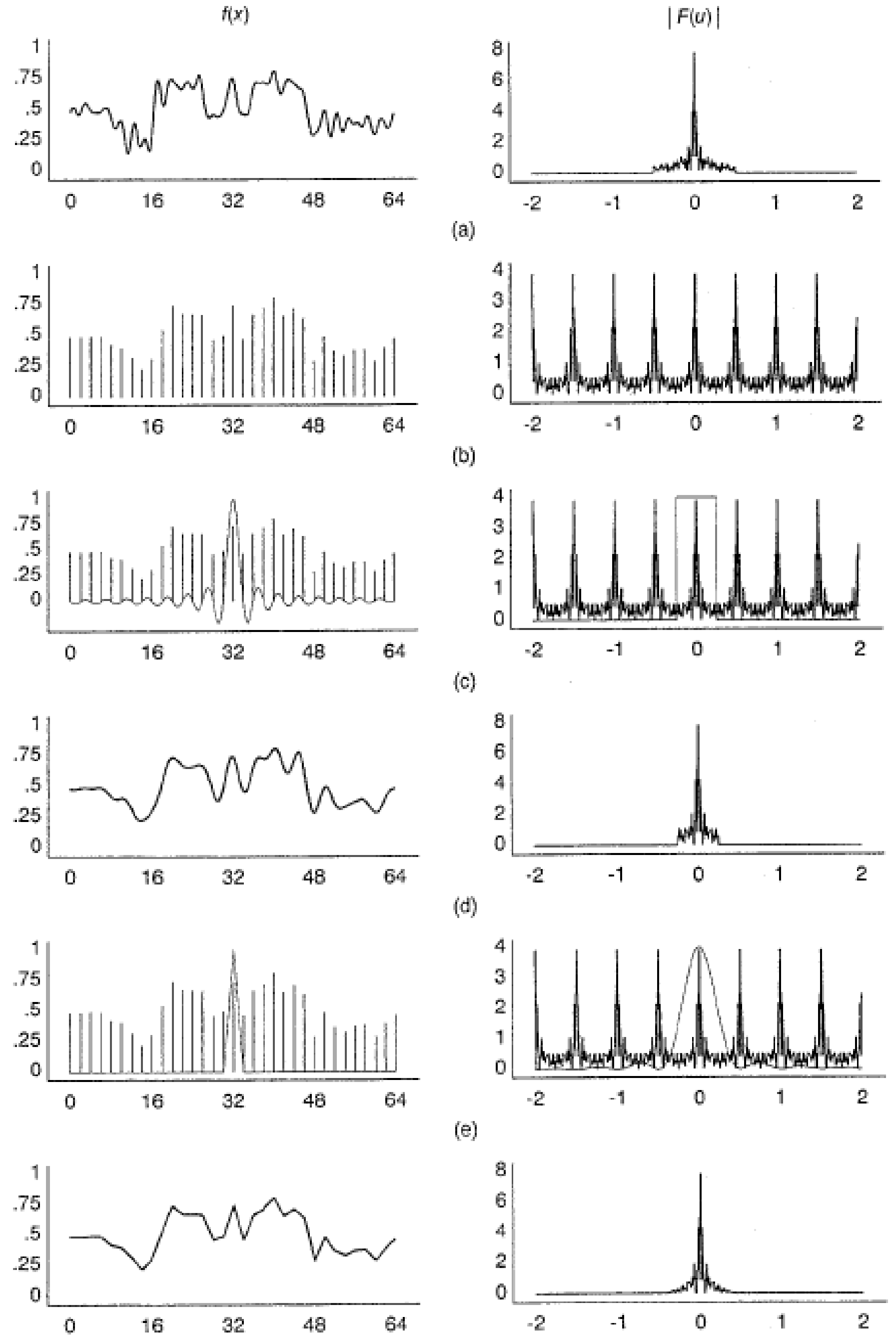


Sampling and Reconstruction





f(x)
| F(u) |
(a)
(b)
(c)
(d)
(e)
(f)



f(x)
|F(u)|
(a)
(b)
(c)
(d)
(e)
(f)

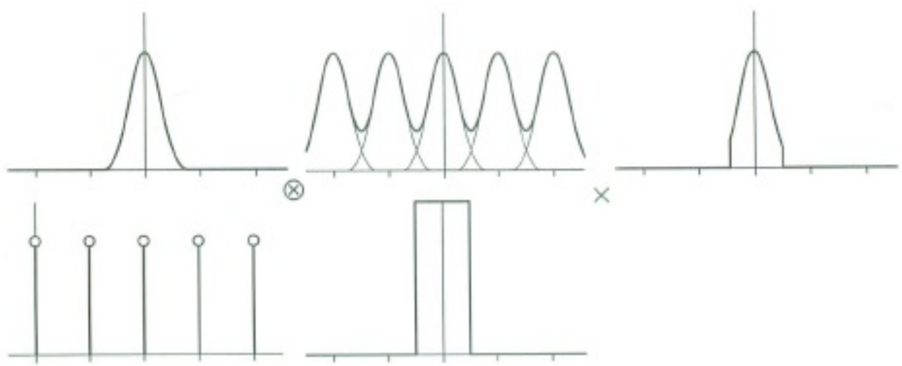
Sampling Theorem

This result if known as the *Sampling Theorem* and is due to Claude Shannon who first discovered it in 1949

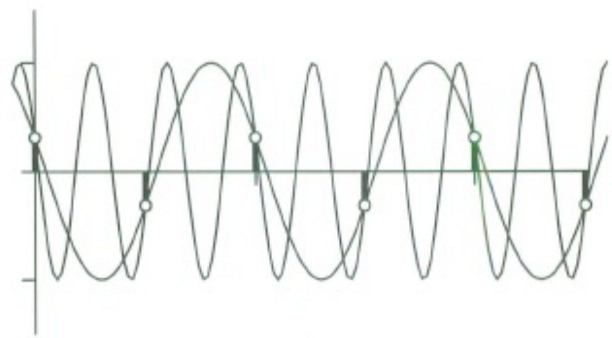
> ***A signal can be reconstructed from its samples without loss of information, if the original signal has no frequencies above 1/2 the sampling frequency***

For a given bandlimited function, the rate at which it must be sampled is called the *Nyquist Frequency*

Undersampling: Aliasing



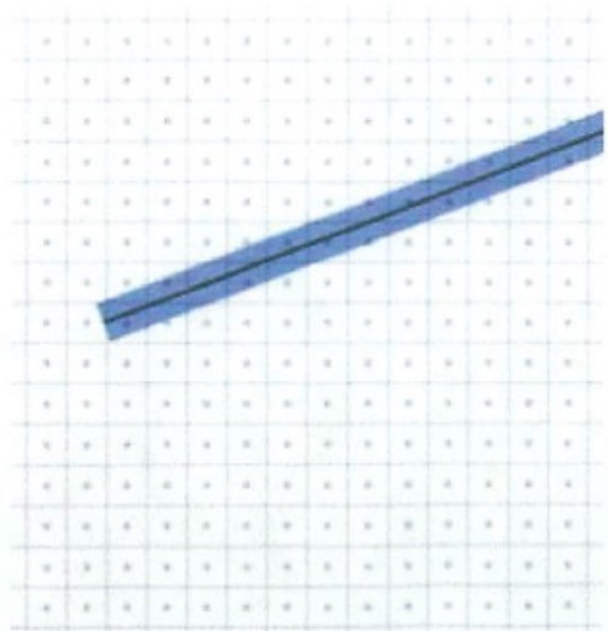
"Aliases"



①

Rasterizing lines

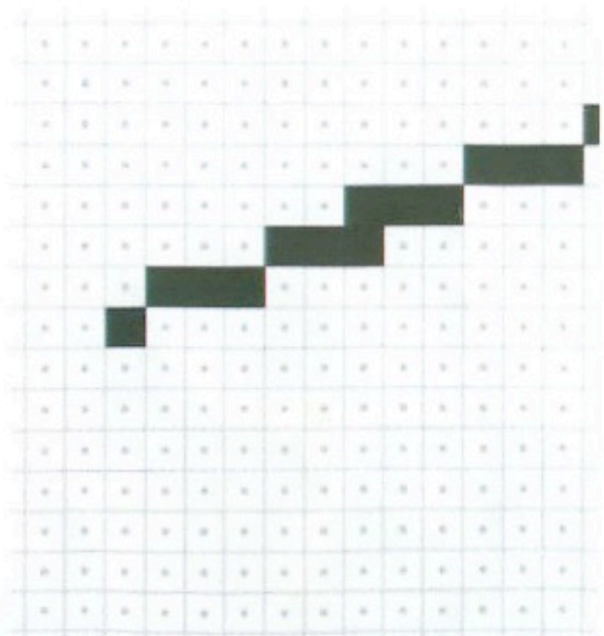
- Define line as a rectangle
- Specify by two endpoints
- Ideal image: black inside, white outside



2

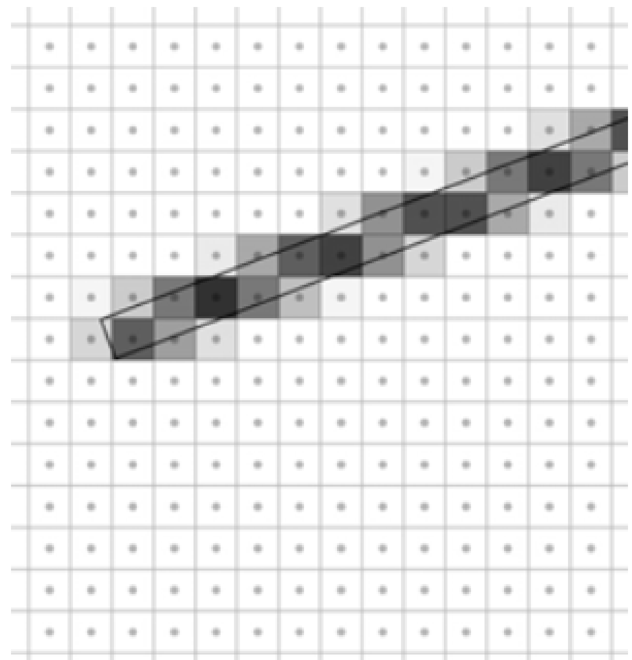
Point sampling

- Approximate rectangle by drawing all pixels whose centers fall within the line
- Problem: all-or-nothing leads to jaggies
  - this is sampling with no filter (aka. point sampling)



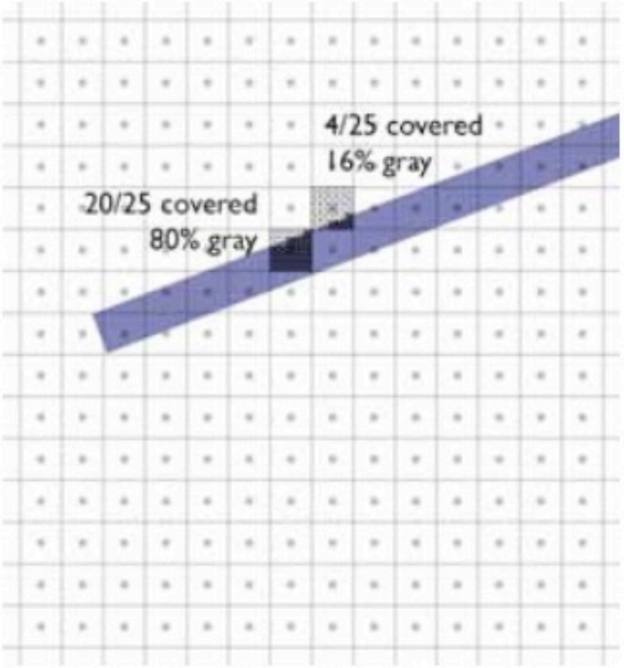
Antialiasing

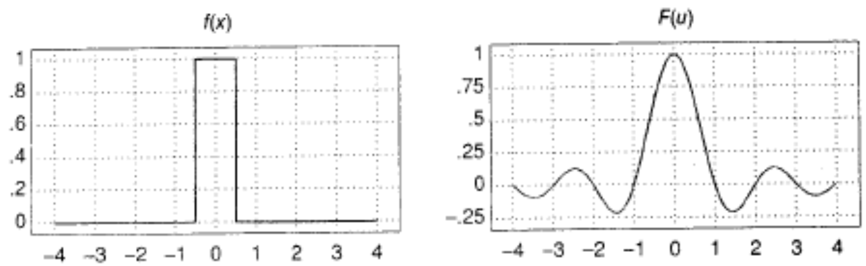
- Basic idea: replace "is the image black at the pixel center?" with "how much is pixel covered by black?"
- Replace yes/no question with quantitative question.



Box filtering by supersampling

- Compute coverage fraction by counting subpixels
- Simple, accurate
- But slow

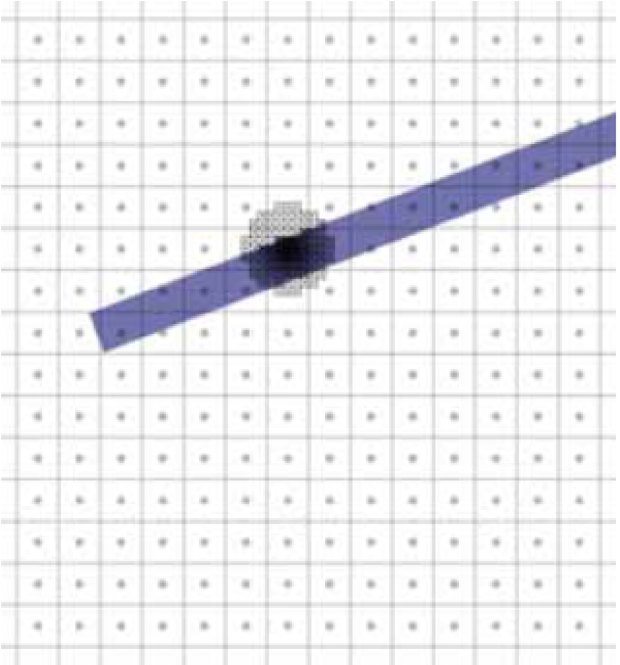




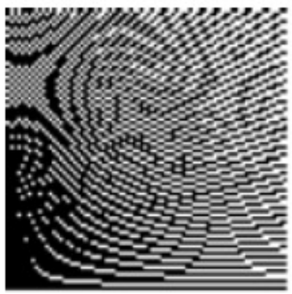
(a)

Weighted filtering by supersampling

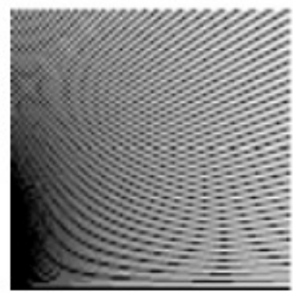
- Compute filtering integral by summing filter values for covered subpixels
- Simple, accurate
- But really slow



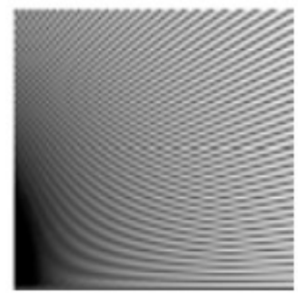
Filter comparison



Point sampling

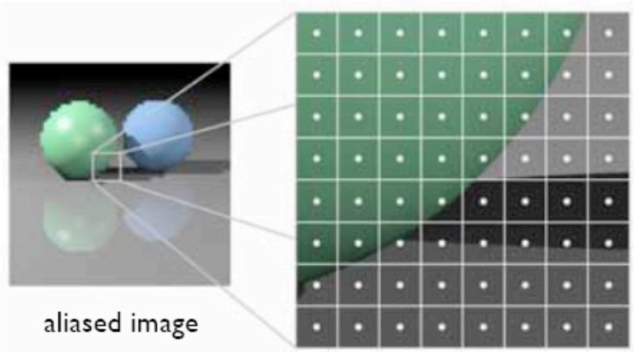


Box filtering



Gaussian filtering

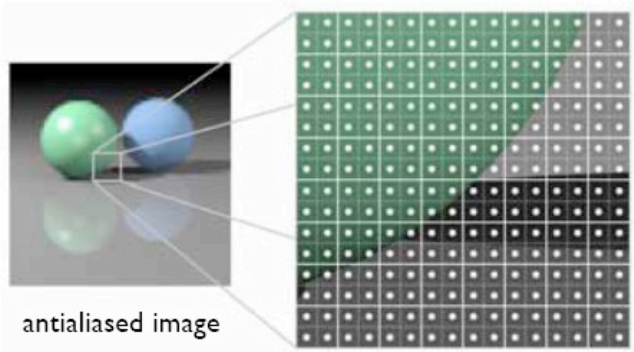
Antialiasing in ray tracing



aliased image

one sample per pixel

Antialiasing in ray tracing



antialiased image

four samples per pixel

