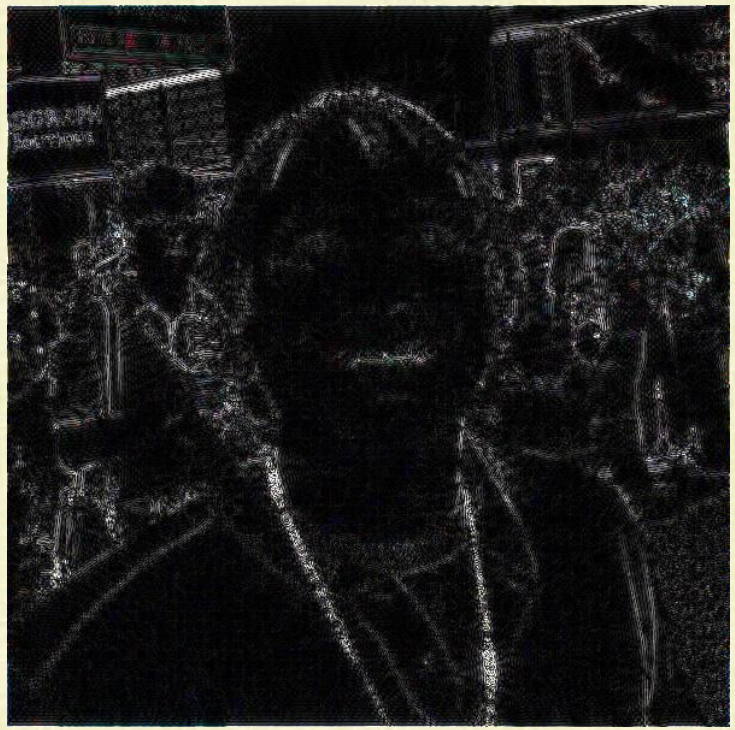
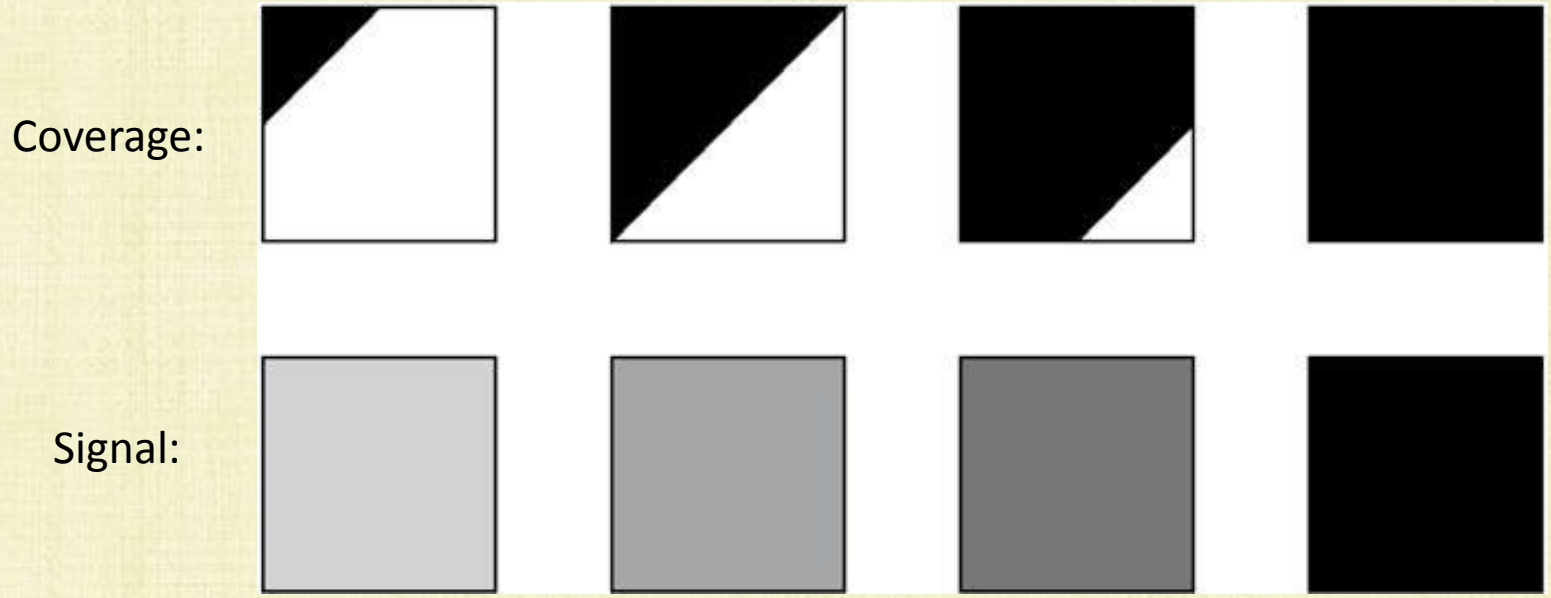
Sampling



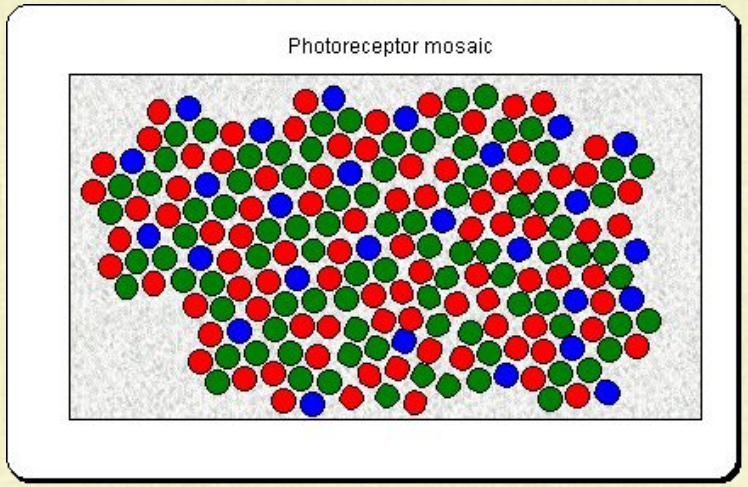
Sampling

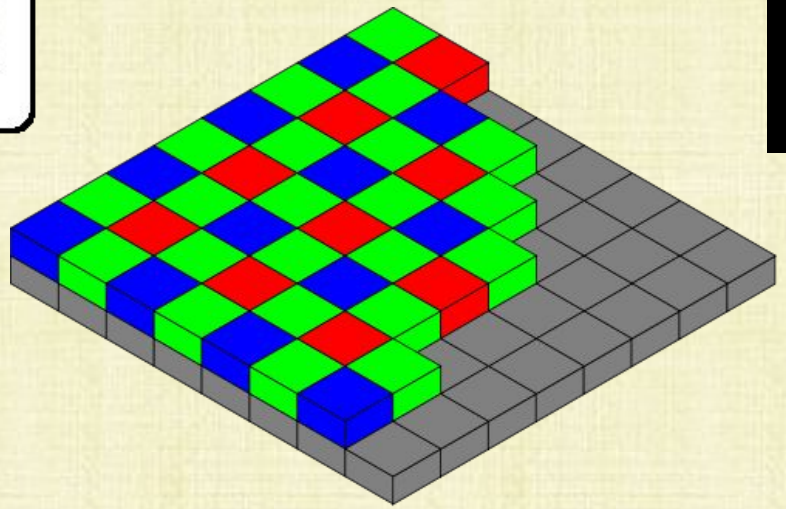
- The ray tracer samples geometry, only gathering information from a finite number of ray intersections (points)
- In contrast, a scanline renderer can project every single triangle onto the image plane
  - However, the simple approach of testing which pixel centers are inside which triangles is also sampling
  - Although, one can calculate the exact overlap between projected triangles and finite size (square) pixels in order to avoid sampling
  - This allows one to mimic a sensor, where the signal corresponds to the fraction of the sensor "covered" by the object (as well as the exposure time)

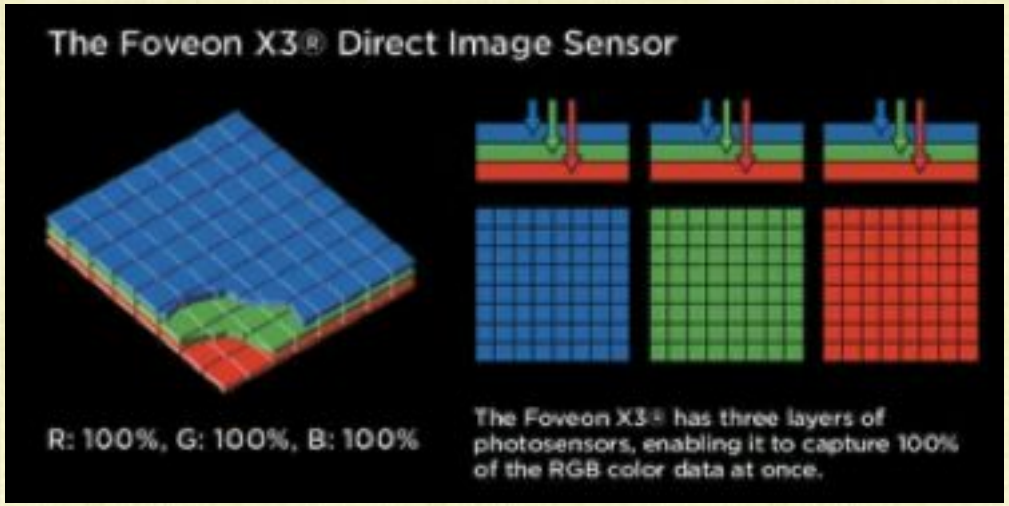


Real World Sensors

- Even the eye and camera do not collect all of the information
- The staggered spatial layout of real-world sensors means that there are large regions where no information about certain wavelengths of light are collected (although a layered approach to “pixels” could help to circumvent this)

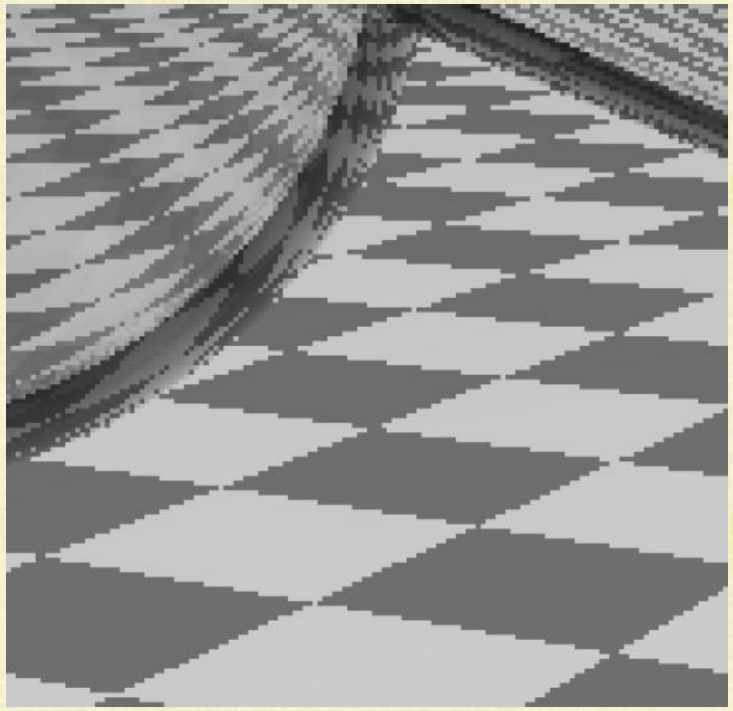


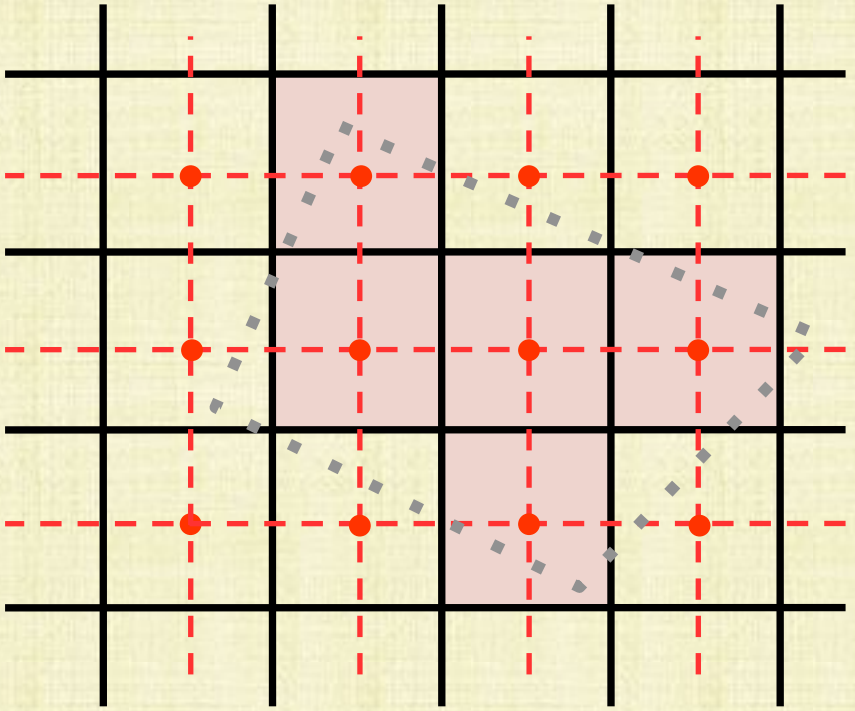




Aliasing

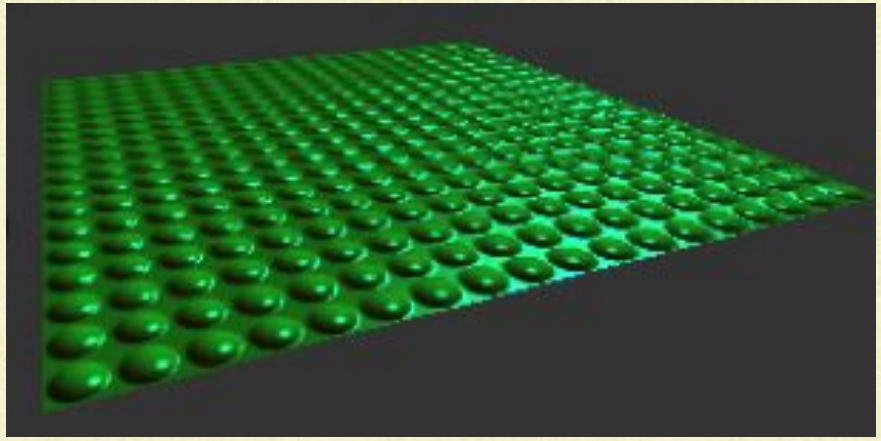
- Coloring an entire pixel based on a ray intersection (or pixel center inside test) leads to jagged edges (“jaggies”), because pixel boundaries do not align with projected object boundaries
- Sampling causes aliasing artifacts (an alias/imposter appears instead of the actual feature)
- Here, a jagged line appears in the place of a straight line
- Anti-aliasing techniques prevent/alleviate aliasing artifacts caused by inadequate sampling

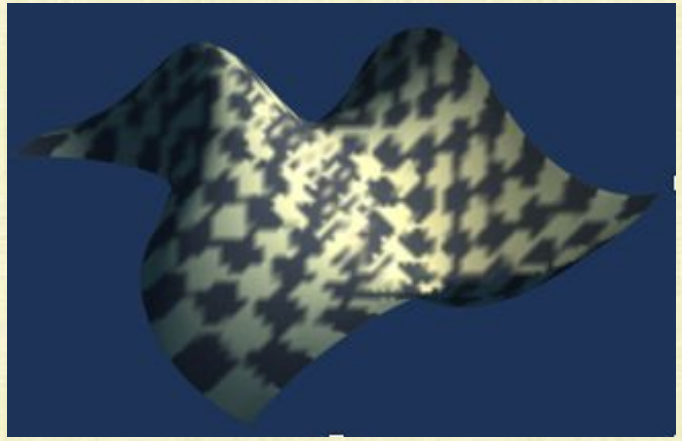


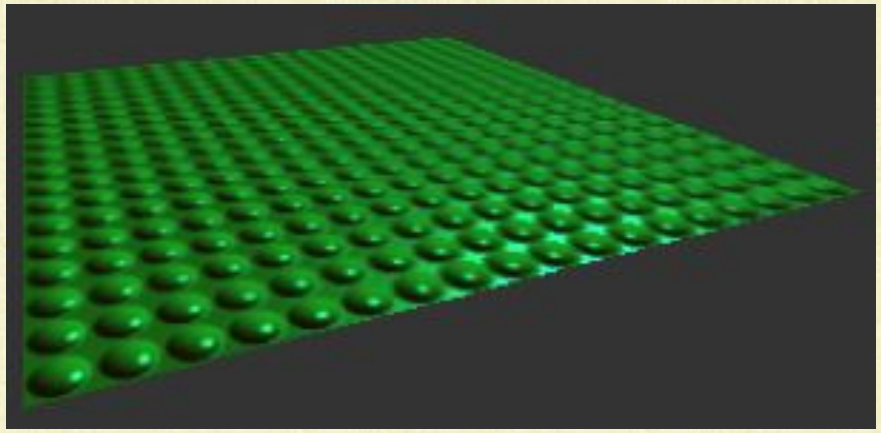


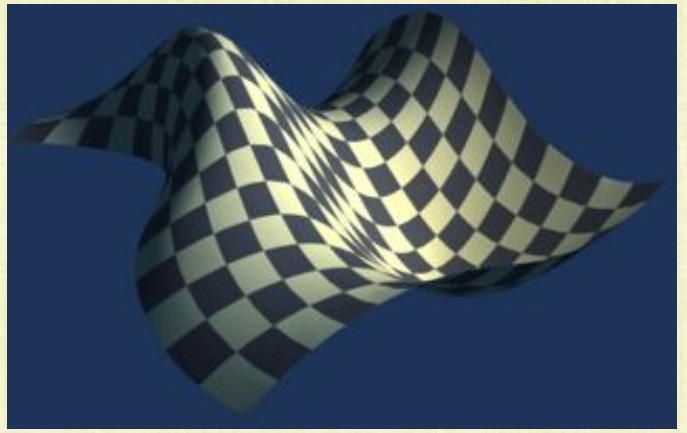
Shaders and Texture Mapping (Aliasing)

- Aliased normal vectors can cause erroneous sparkling highlights (top left)
- Aliasing can occur when mapping textures onto objects too (top right)



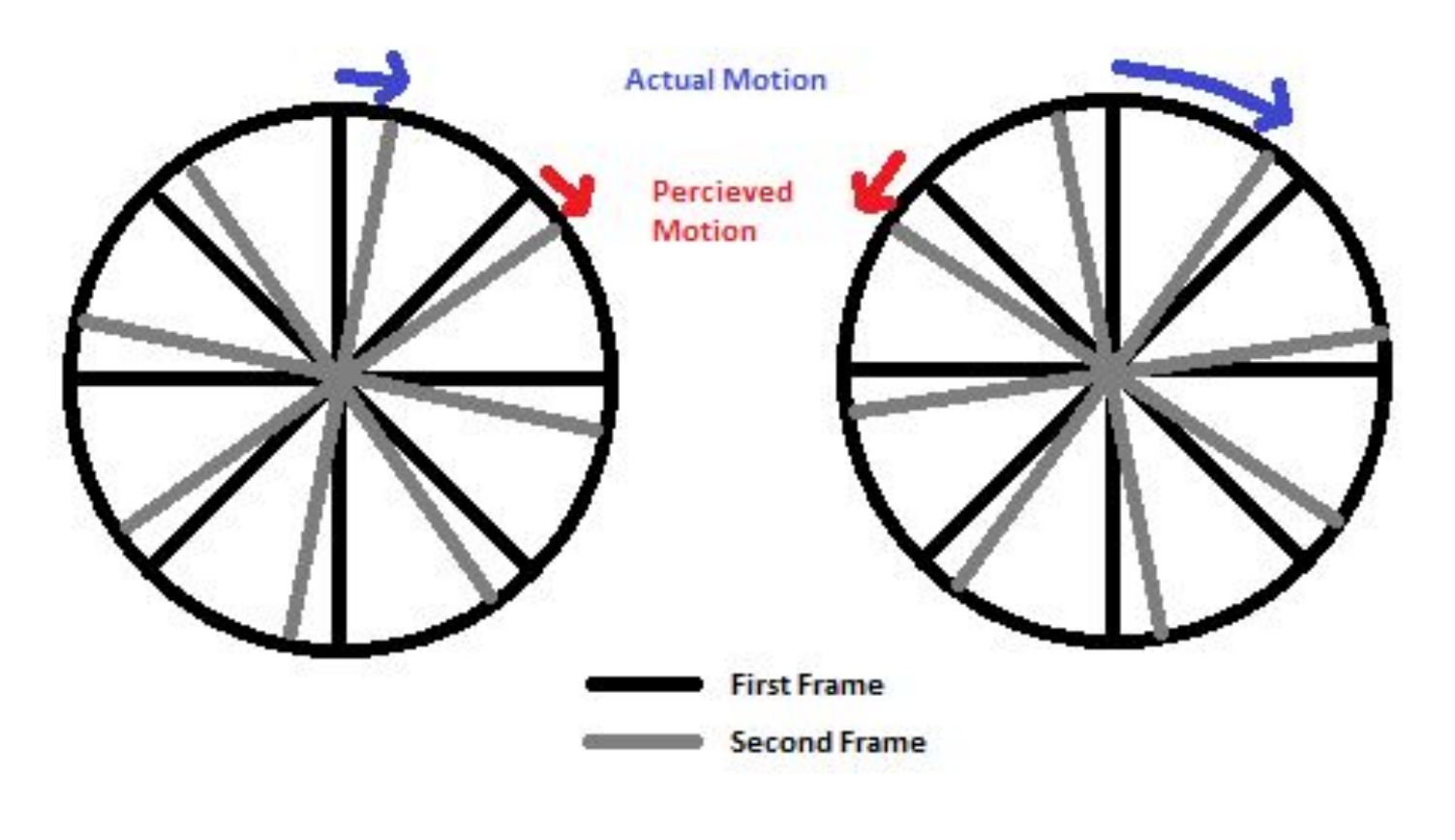






Temporal Aliasing

- A spinning wheel appears to spin backwards when the motion is insufficiently sampled in time ("wagon wheel" effect)

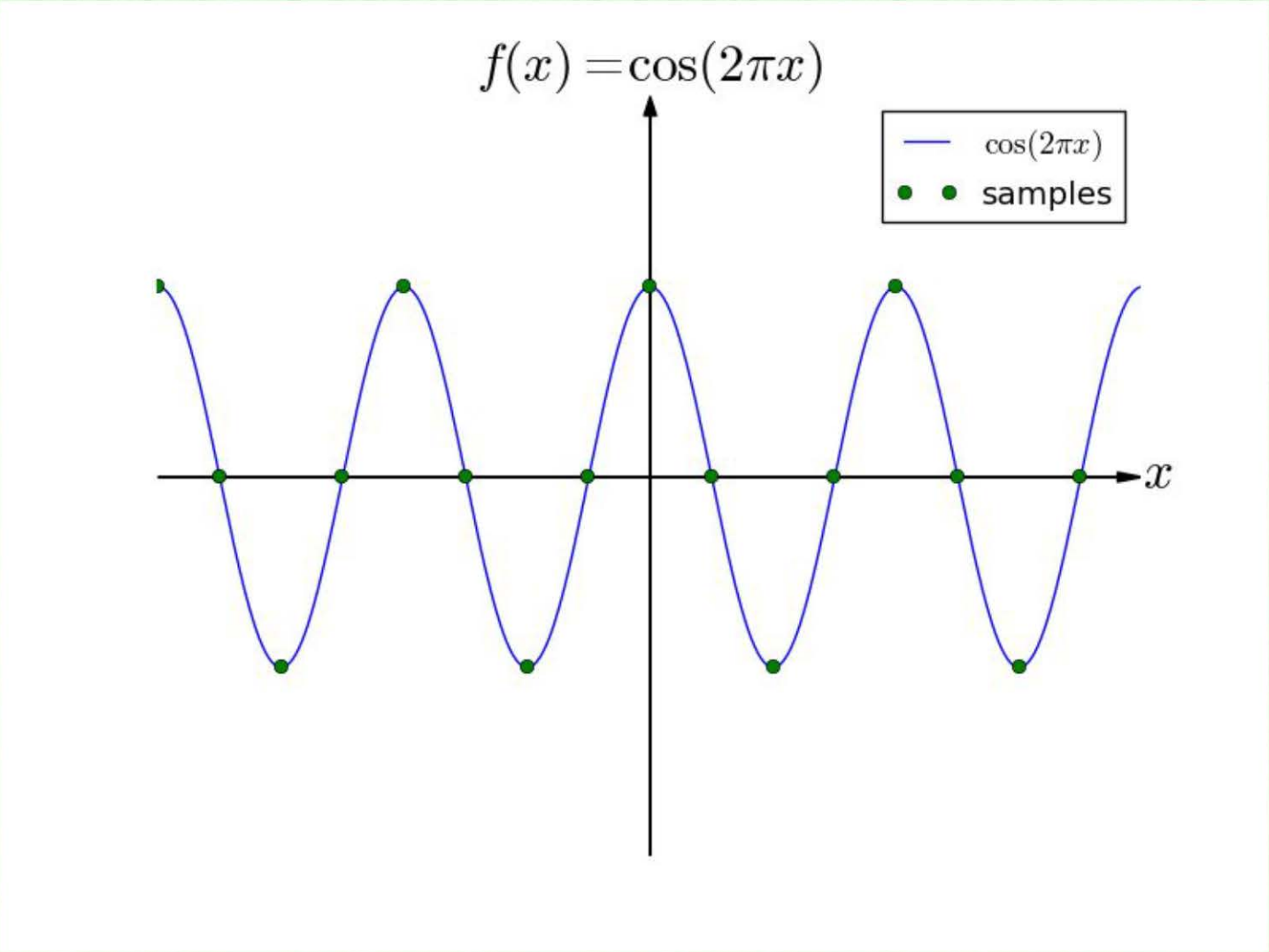


Sampling Rate

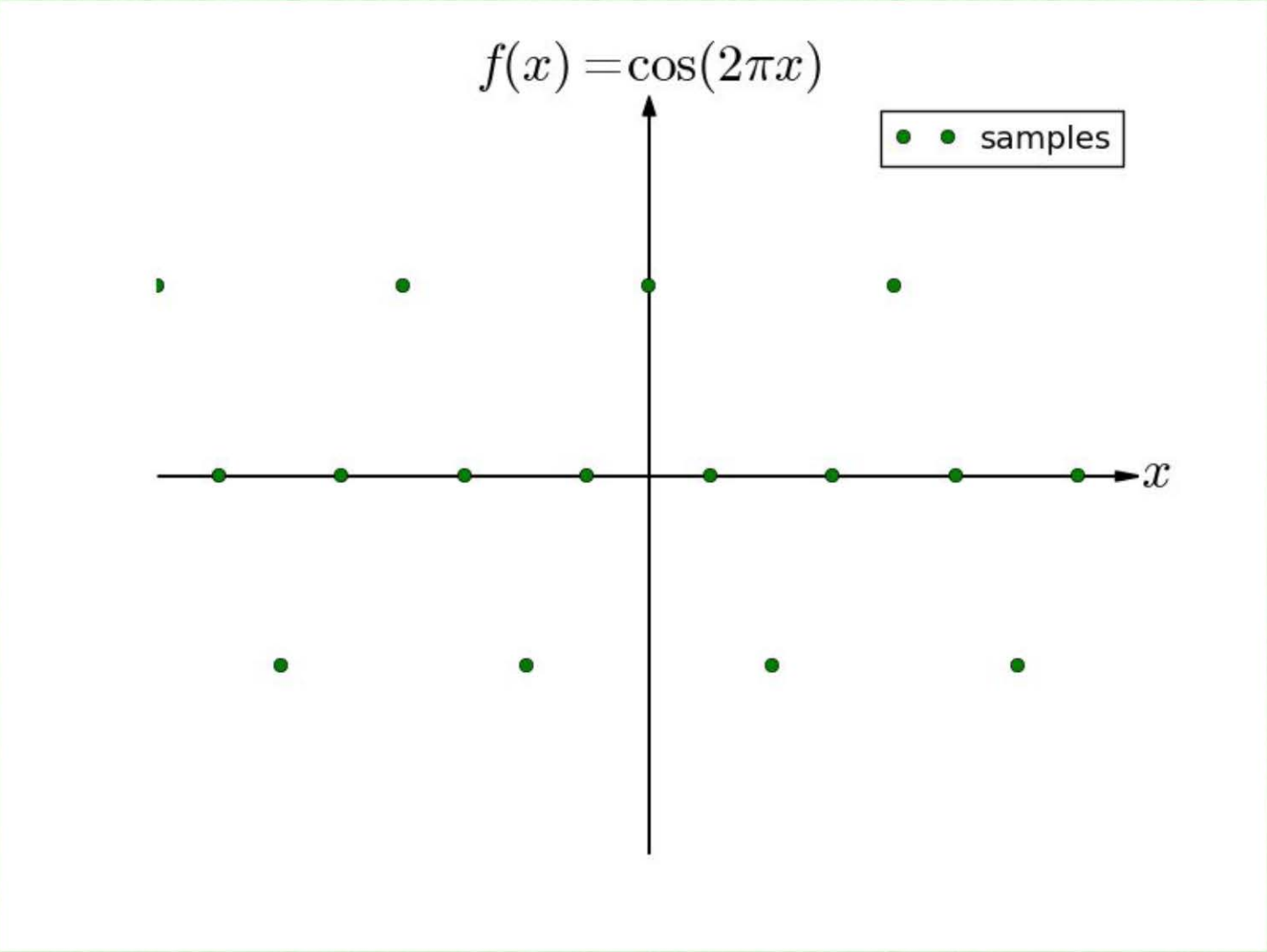
- Sampling artifacts can be reduced by increasing the number of samples
  - i.e. increasing the number of pixels in the image
- However, increasing the resolution of an image is costly:
  - Requires additional time to render because the number of rays increases
  - Displaying the high resolution image (and/or using the image as a texture) requires both additional storage and computation

- Optimize the Sample Rate!
  - Use the lowest possible sampling rate that does not result in noticeable aliasing artifacts
  - What is this ideal sampling rate?

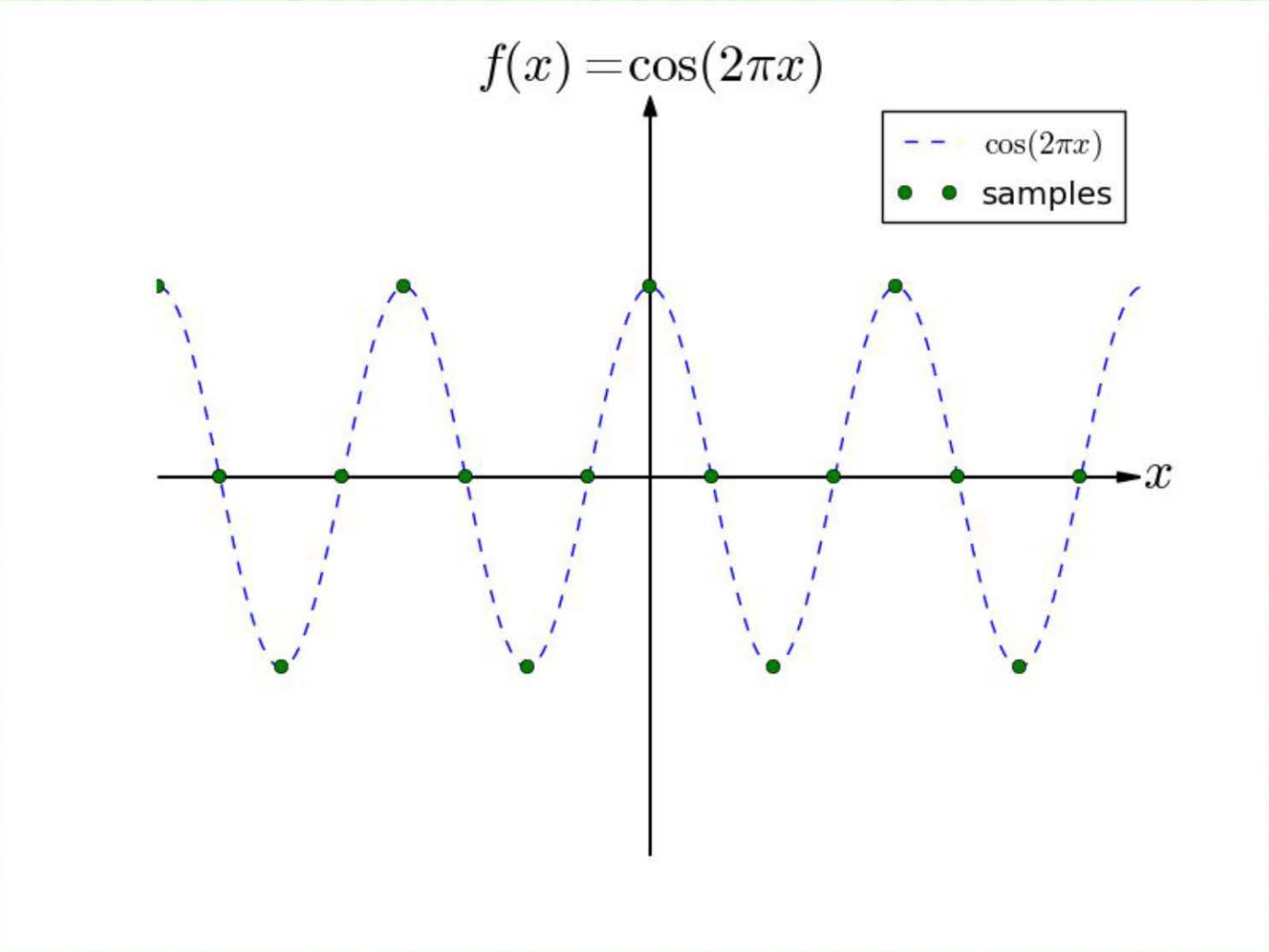
4 samples per period



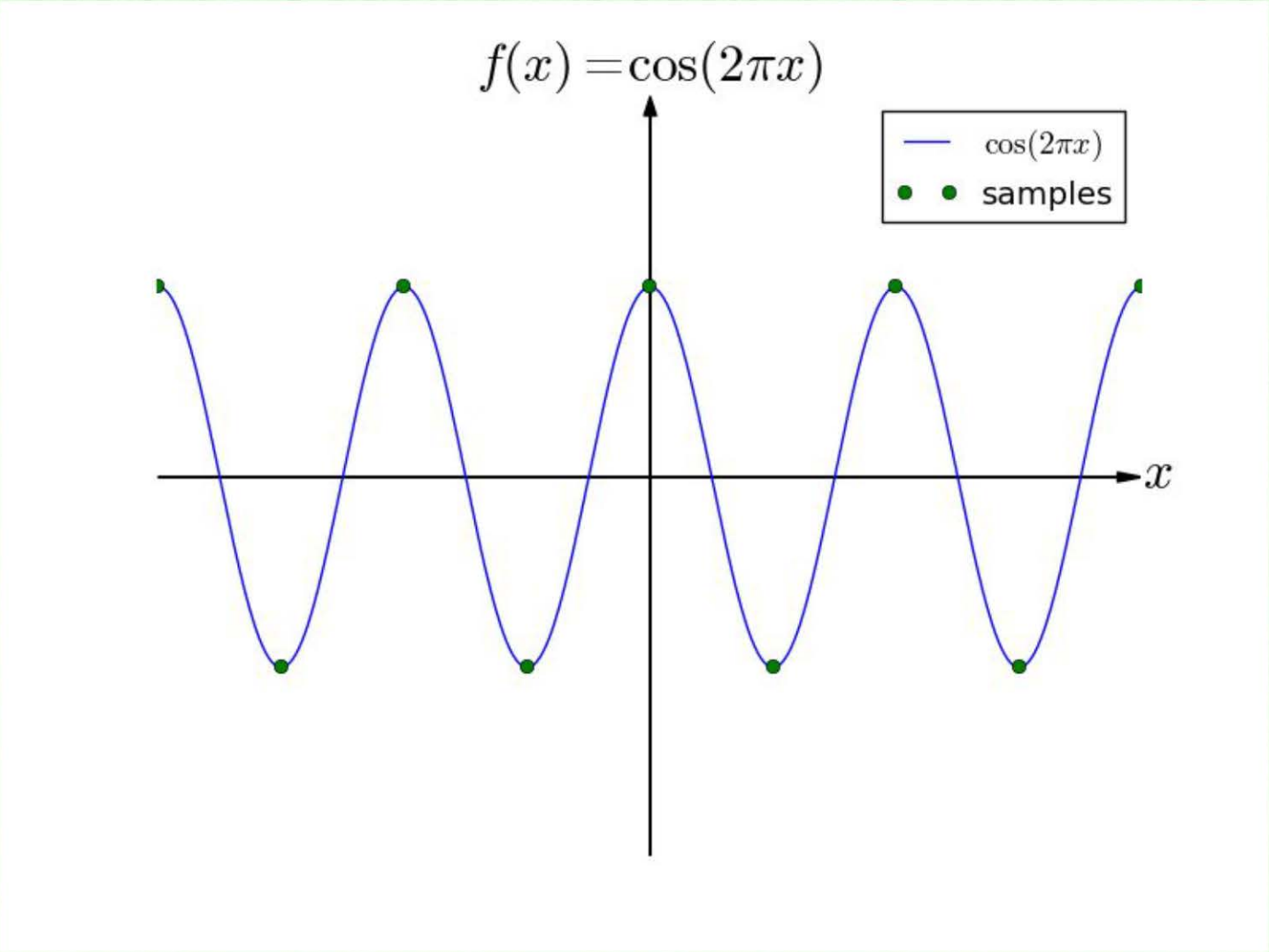
samples



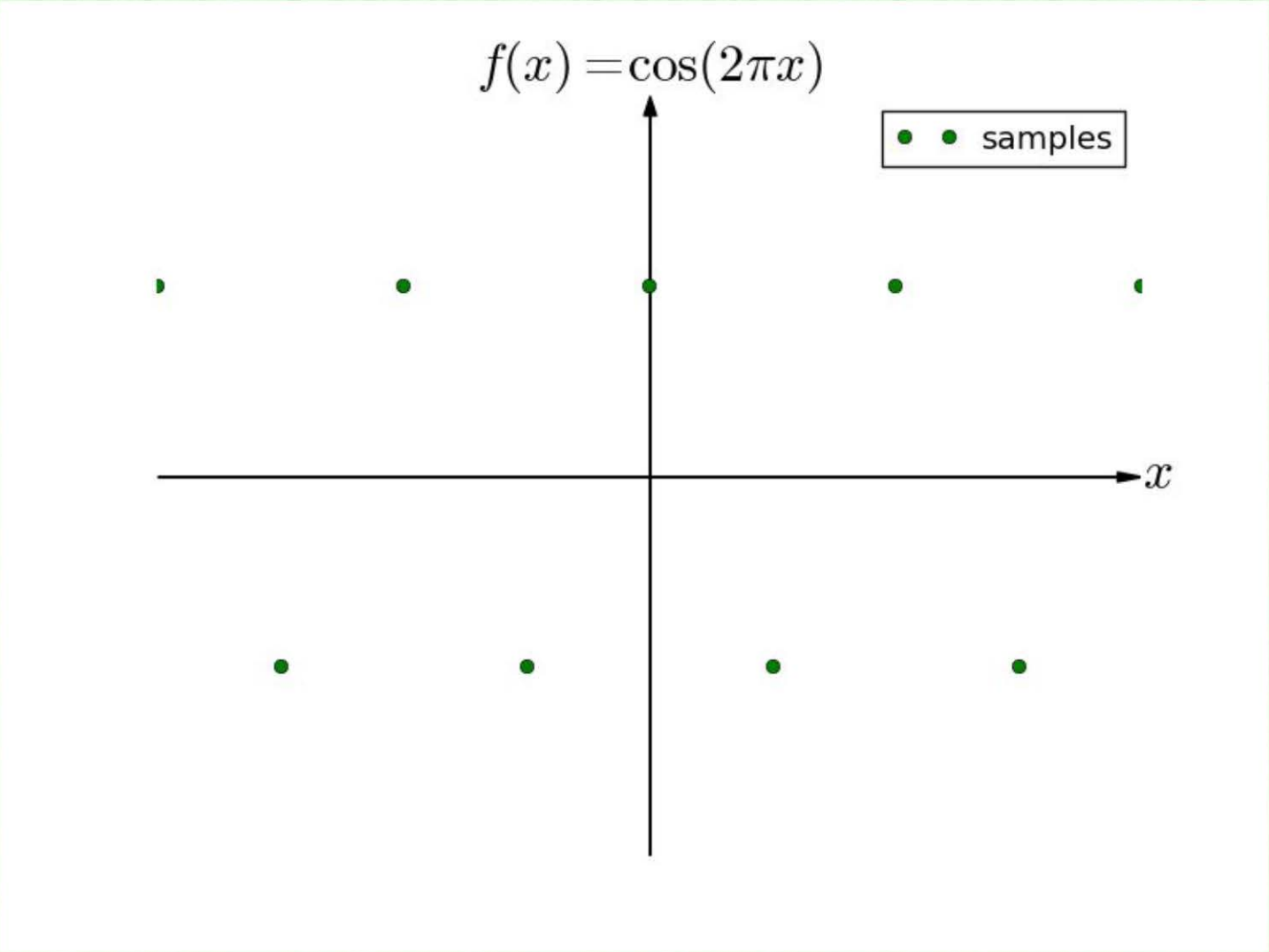
reconstruction



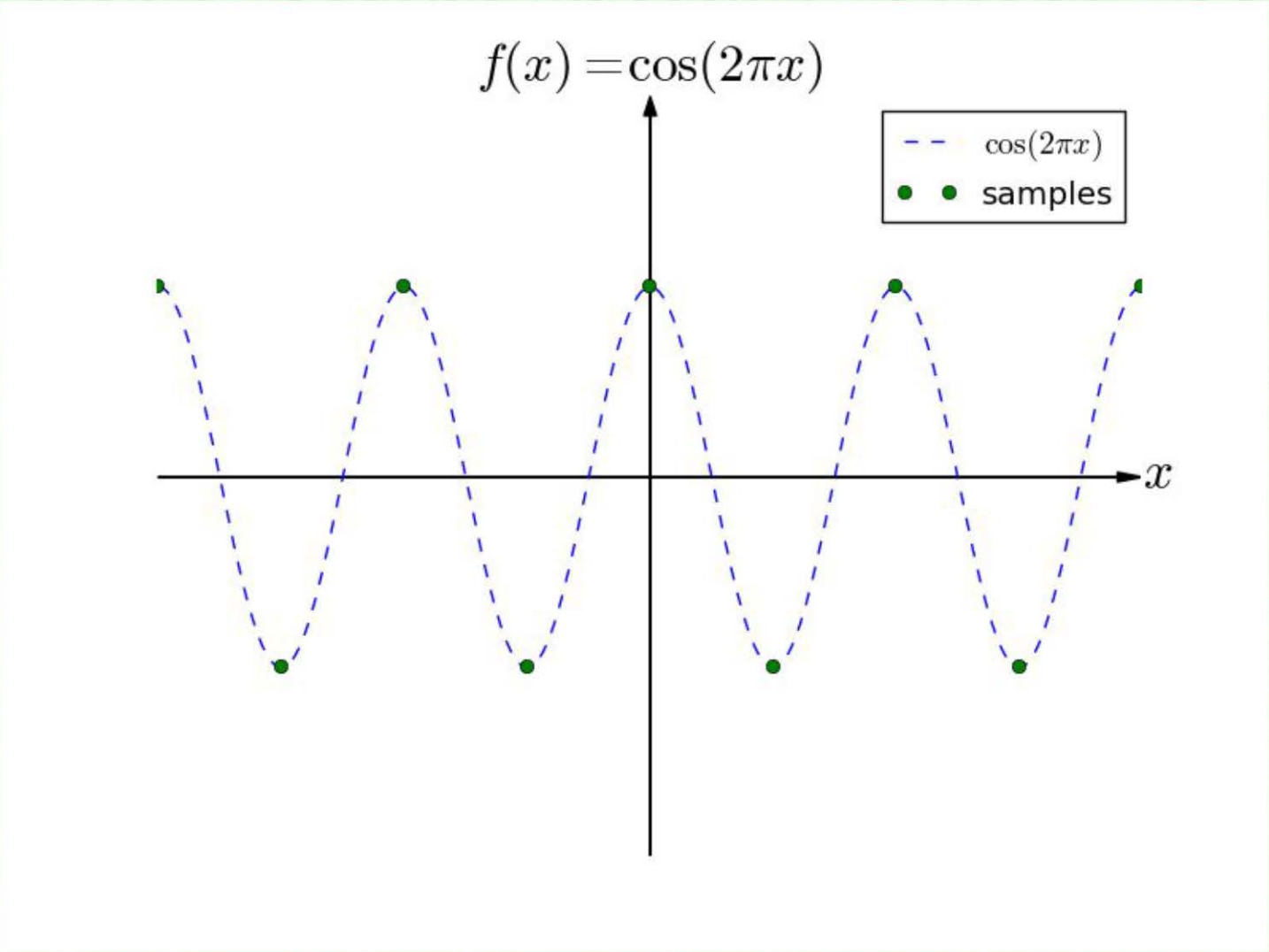
2 samples per period



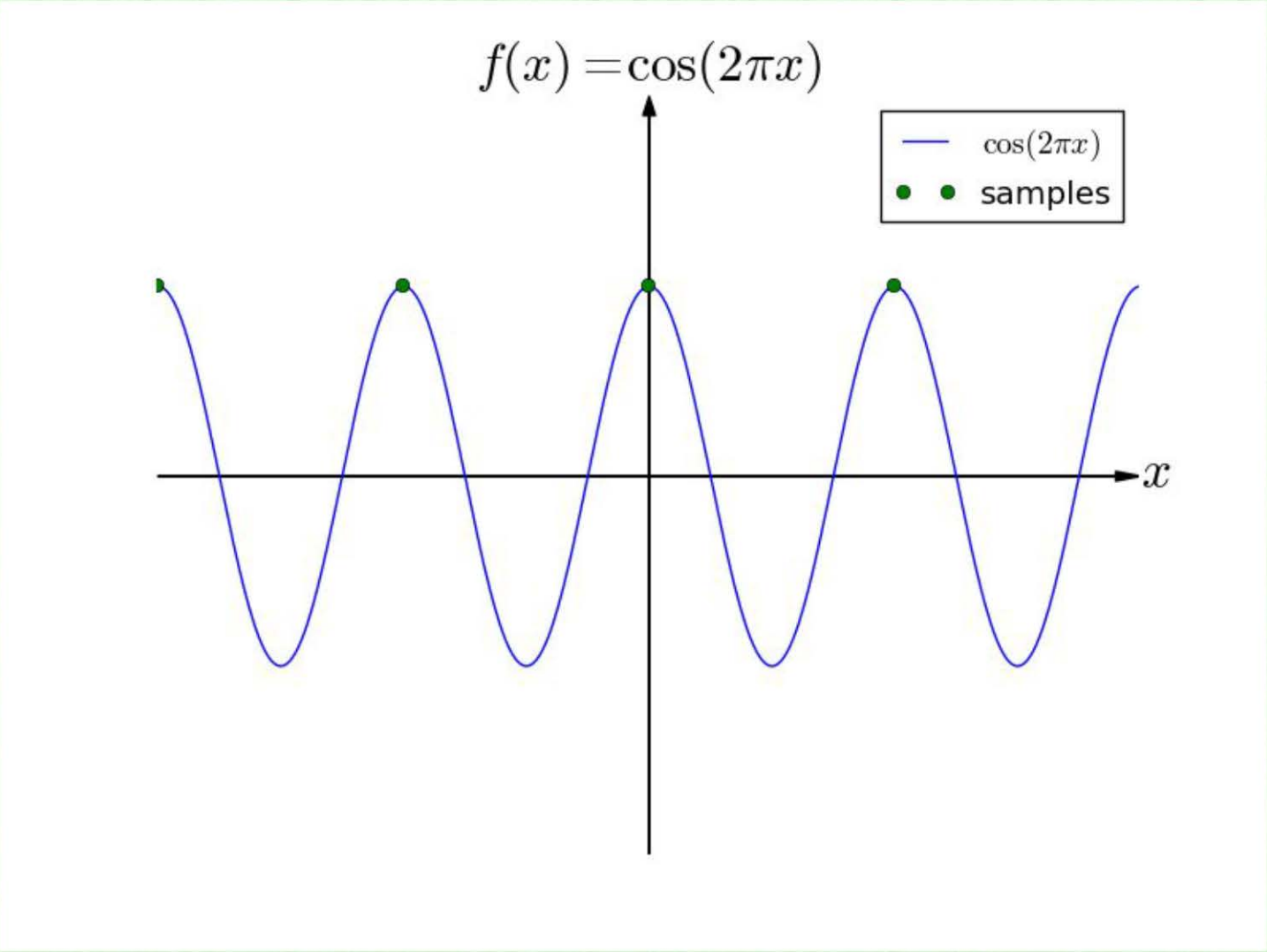
samples



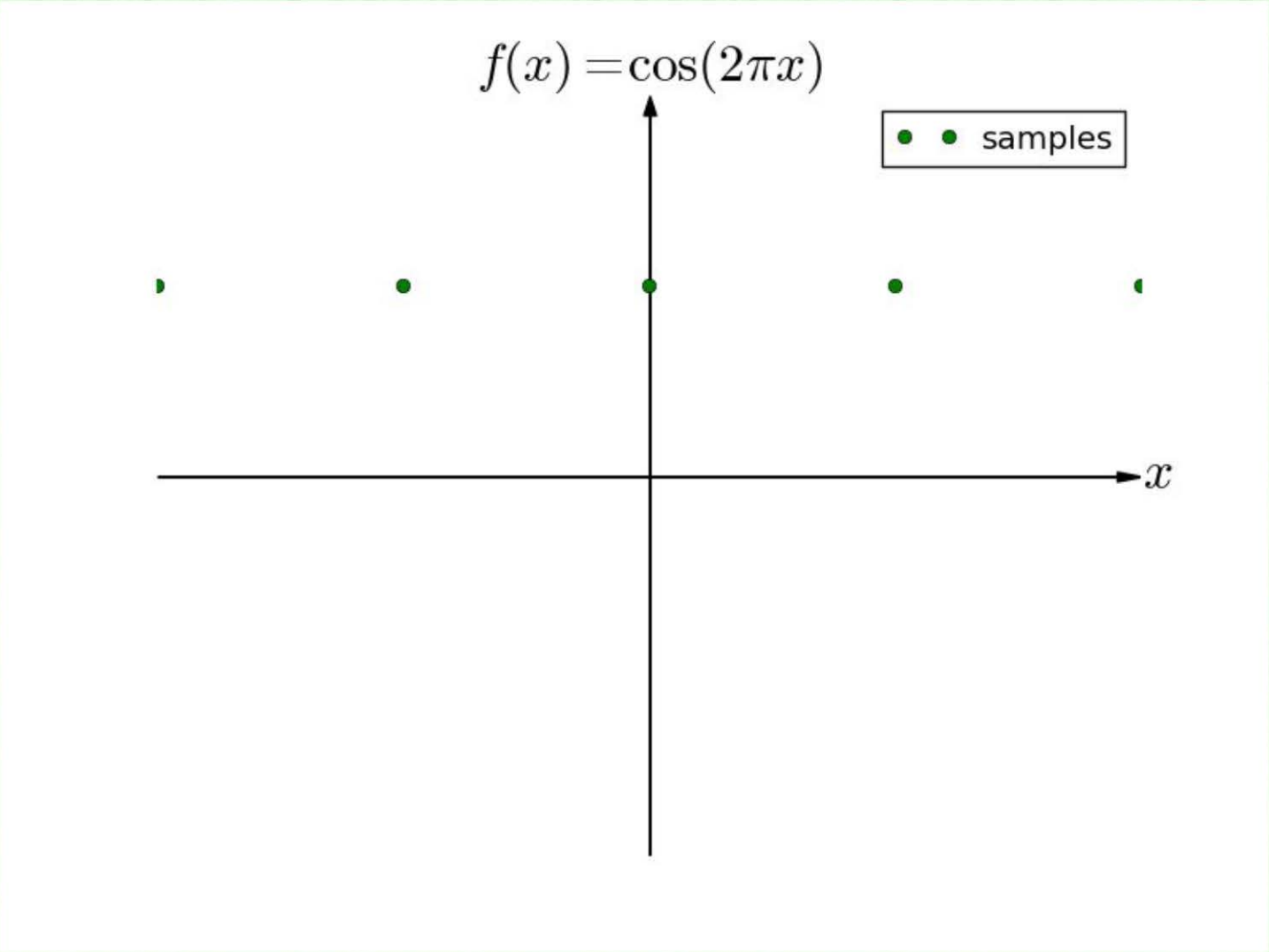
reconstruction



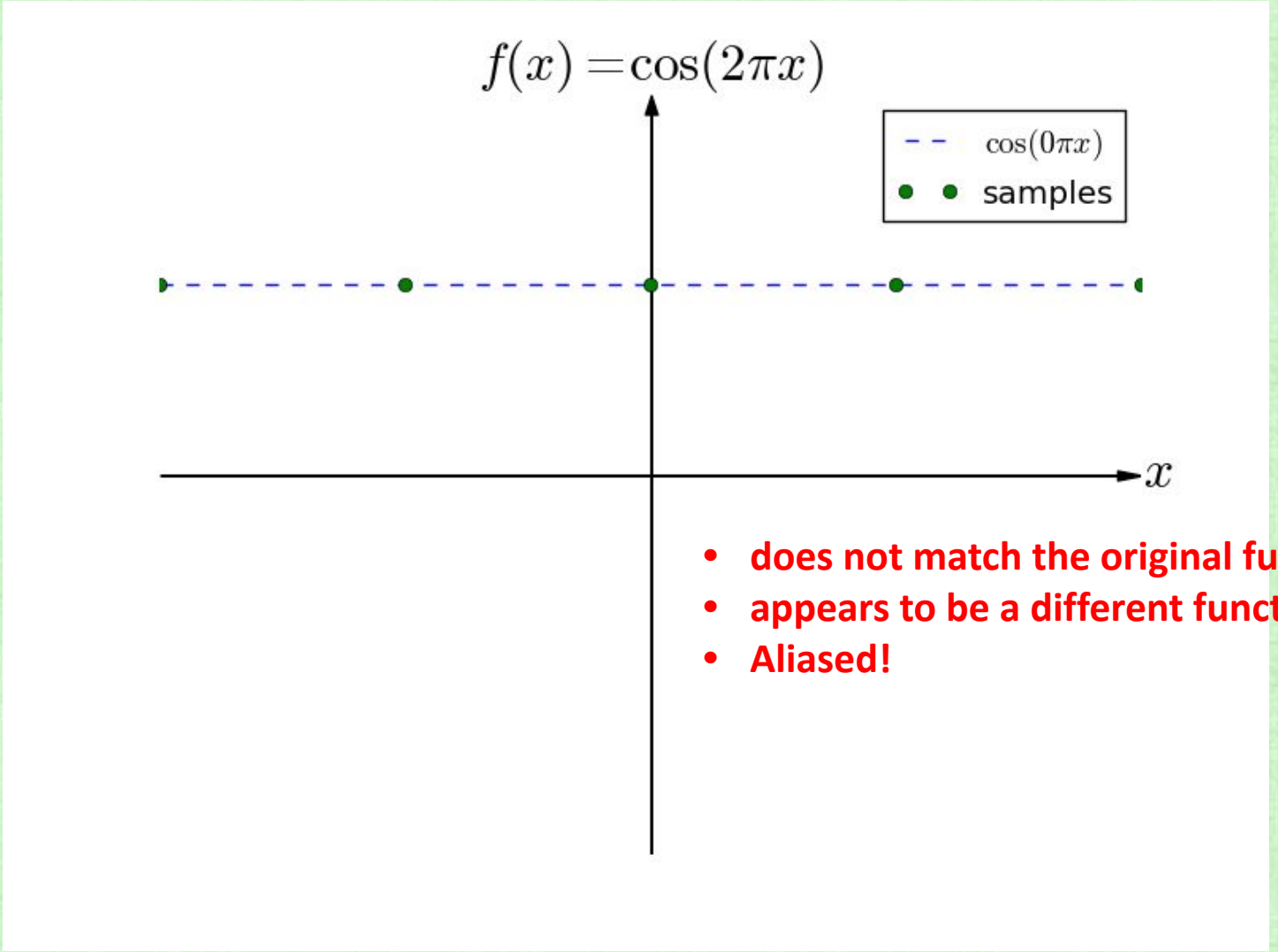
1 sample per period



samples

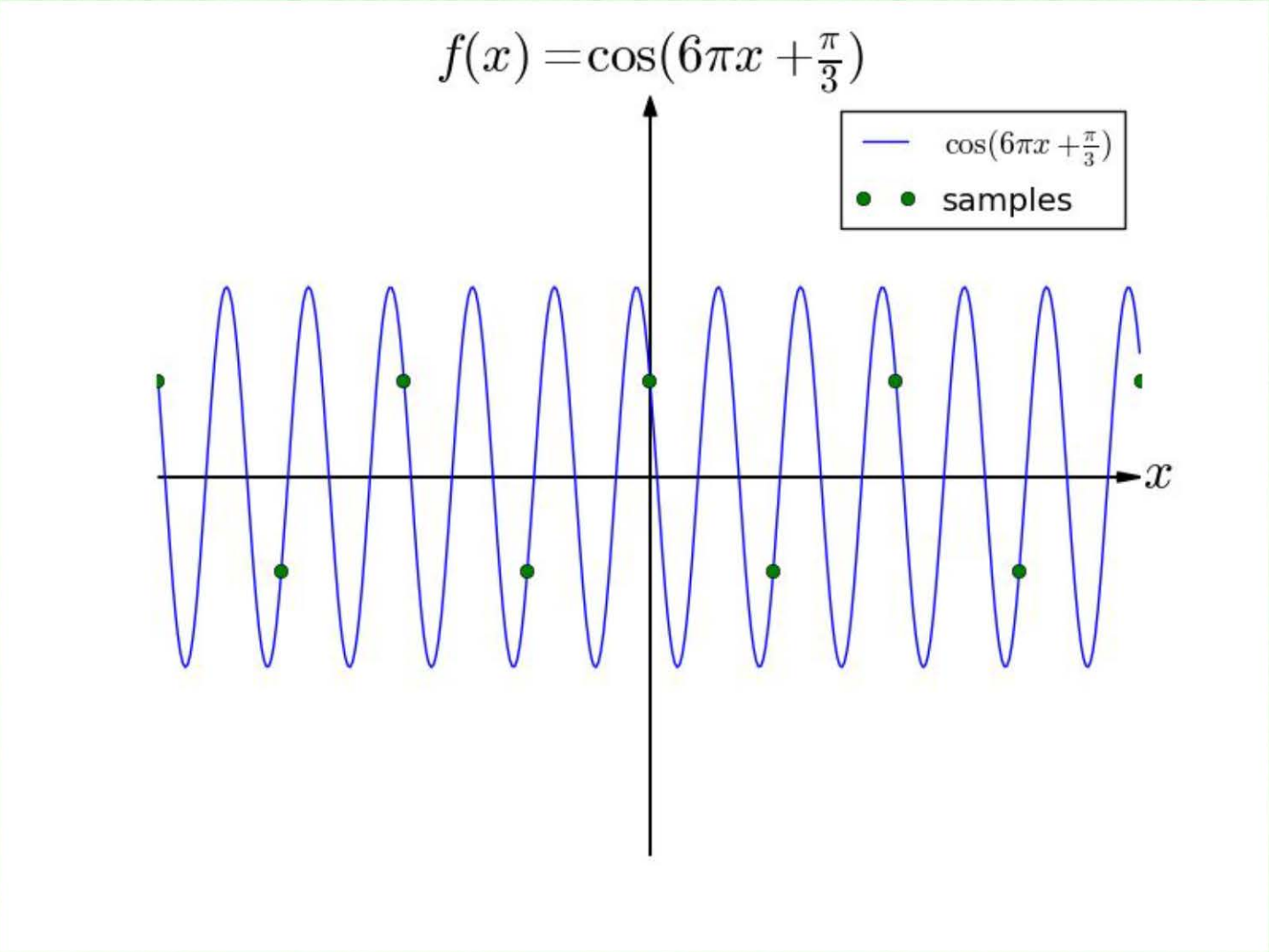


reconstruction

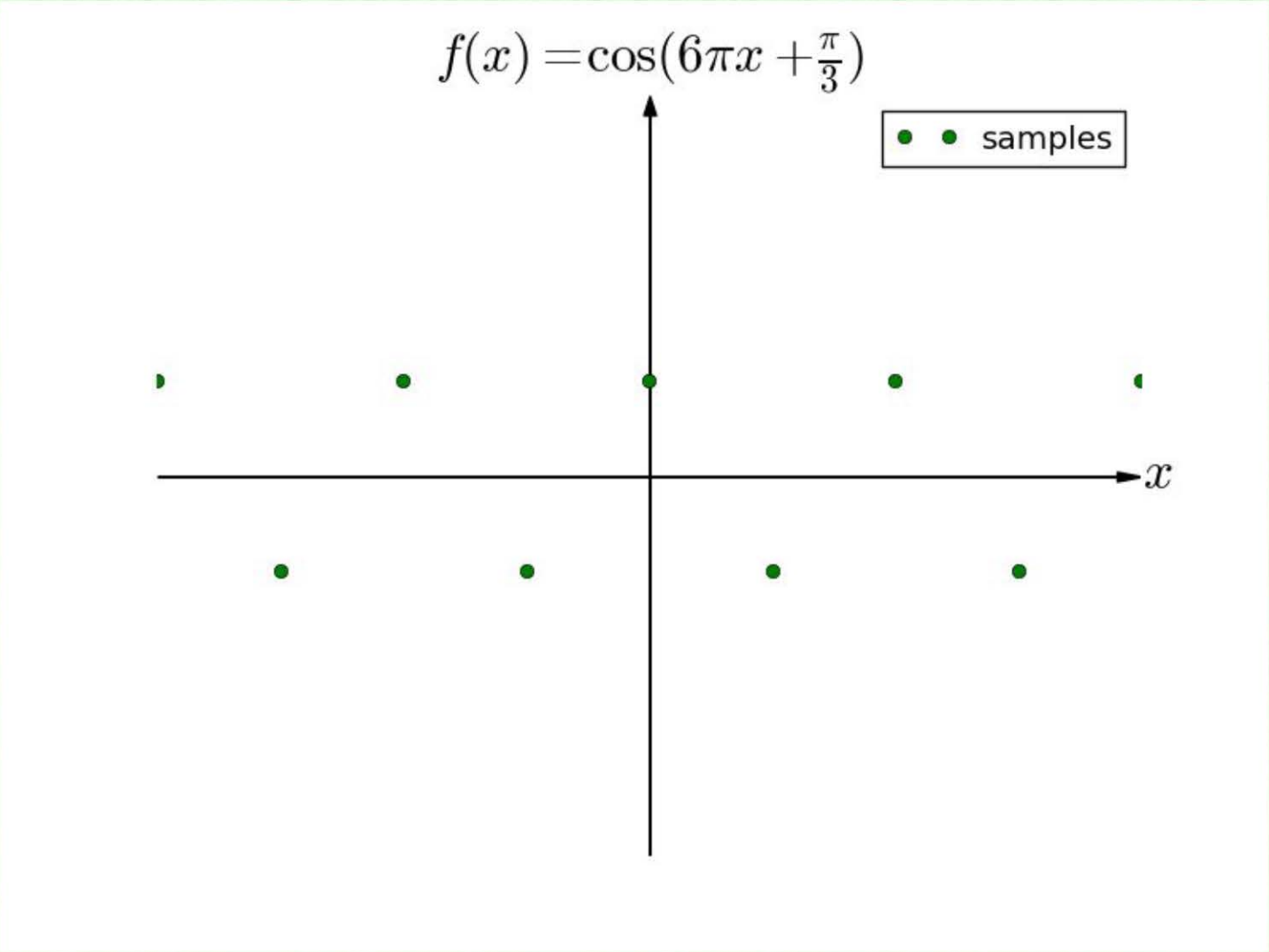


- **does not match the original function**
- **appears to be a different function**
- **Aliased!**

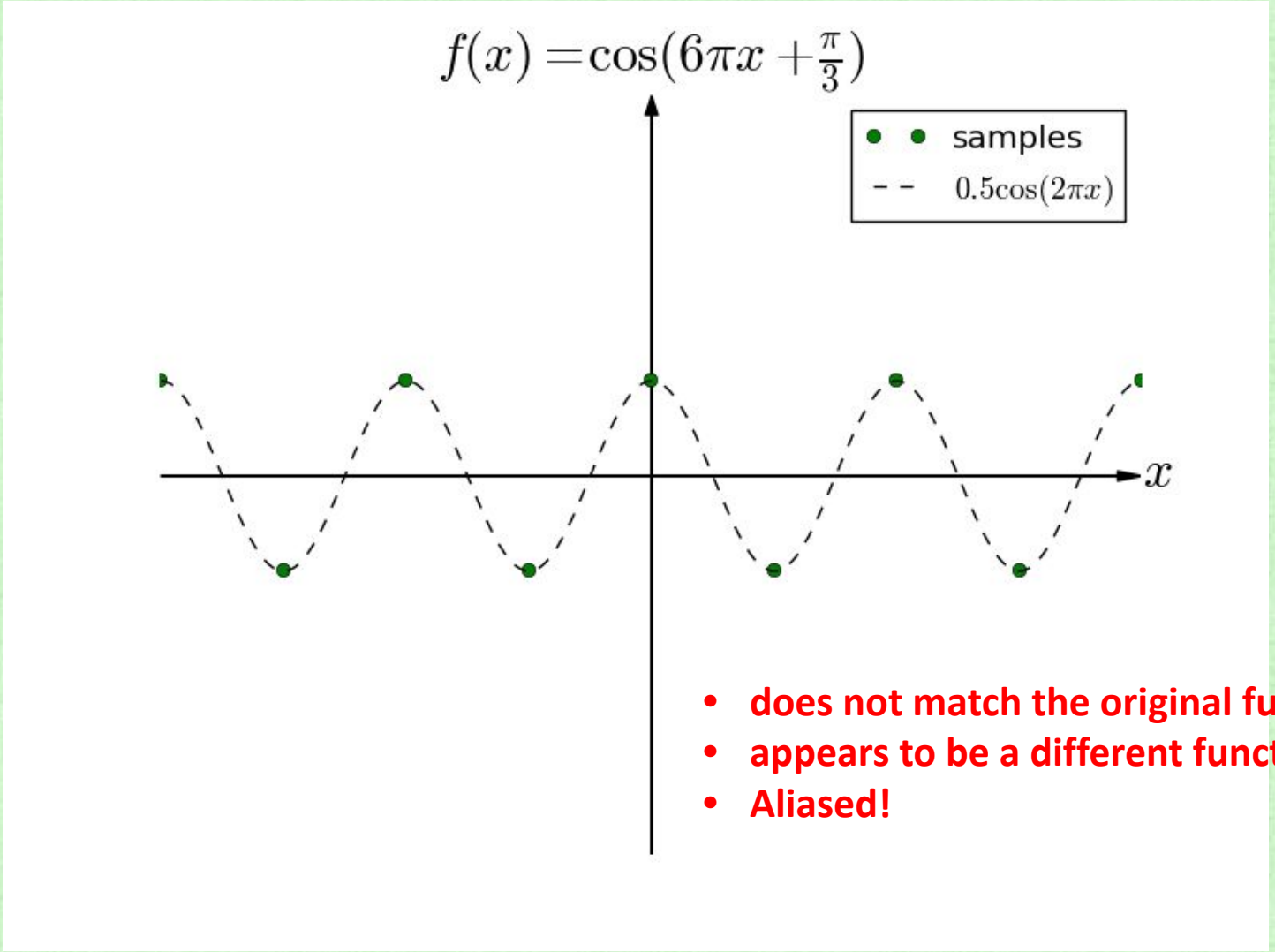
2/3 sample per period



samples

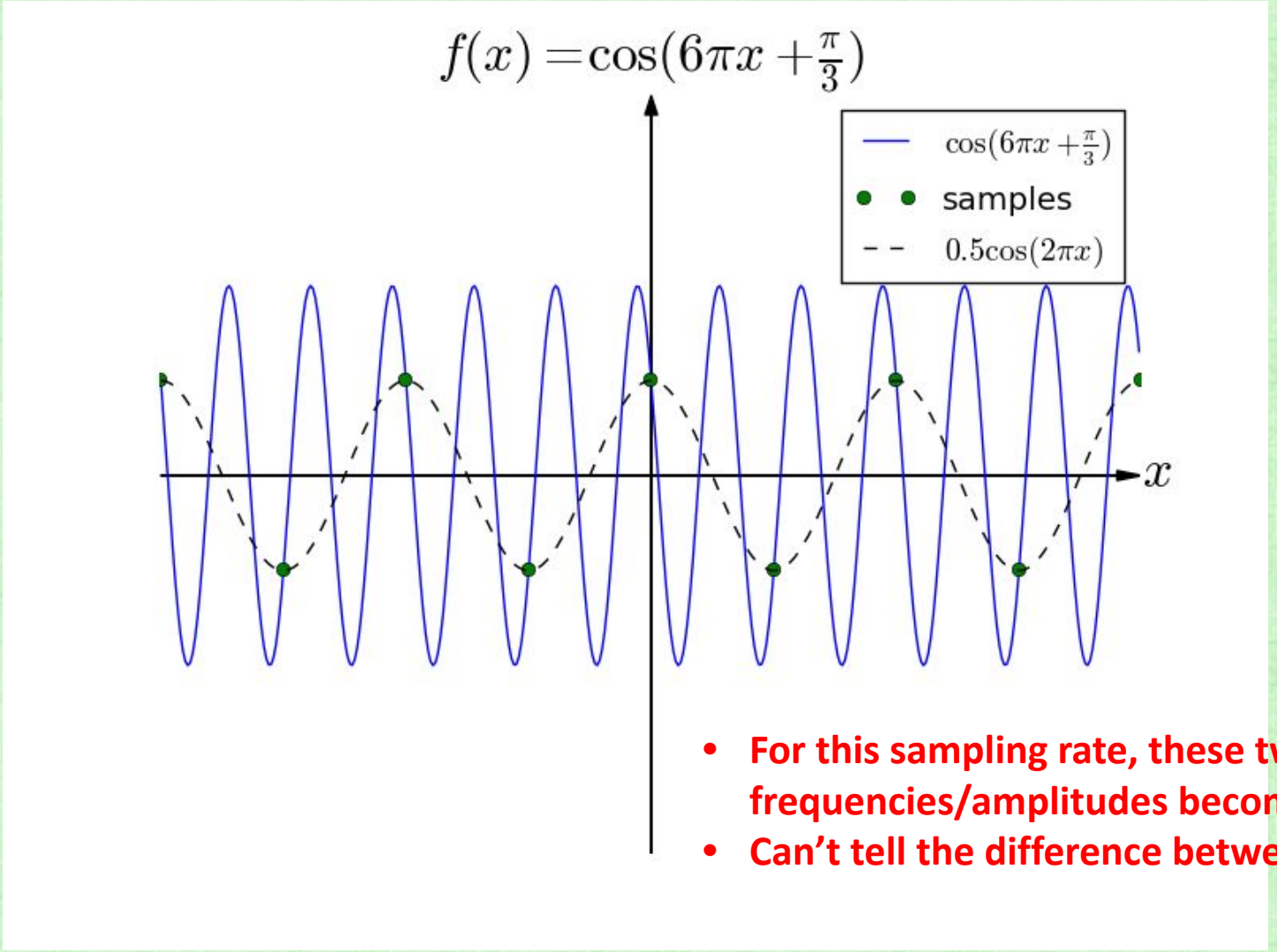


reconstruction



- **does not match the original function**
- **appears to be a different function**
- **Aliased!**

Aliasing



- **For this sampling rate, these two cosine waves of different frequencies/amplitudes become aliased**
- **Can't tell the difference between them**

Sampling Rate

- Sampling at too low a rate results in aliasing
- Two different signals become indistinguishable
  - an alias or imposter

- Nyquist-Shannon Sampling Theorem
  - If $f(t)$ contains no frequencies higher than W hertz, it can be completely determined by samples at a series of points spaced $1/(2W)$ seconds apart
  - That is, a minimum of 2 samples per period are required to prevent aliasing

Anti-Aliasing

- The Nyquist frequency is defined as half the sampling frequency
  - Need 2 samples per period to prevent aliasing
- If the function being sampled has no frequencies above the Nyquist frequency, then no aliasing occurs
- But, ***any real world frequencies above the Nyquist frequency will appear as aliases to the sampler***
- So, **before sampling, remove frequencies higher than the Nyquist frequency** (in order to prevent aliasing)
- This can be accomplished by transforming the function into frequency space

Fourier Transform

- Transform from the spatial domain to the frequency domain (and vice versa)

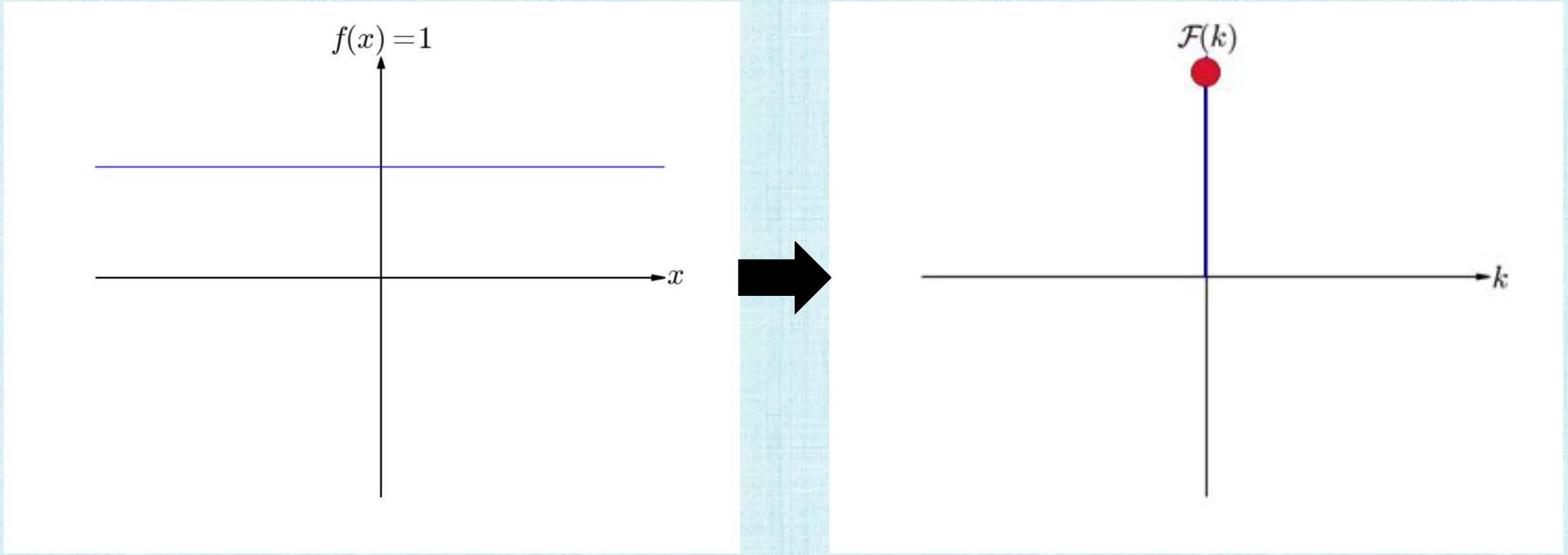
Frequency Domain: $F(k) = \int_{-\infty}^{\infty} f(x) e^{-2\pi i k x} dx$

Spatial Domain: $f(x) = \int_{-\infty}^{\infty} F(k) e^{2\pi i k x} dk$

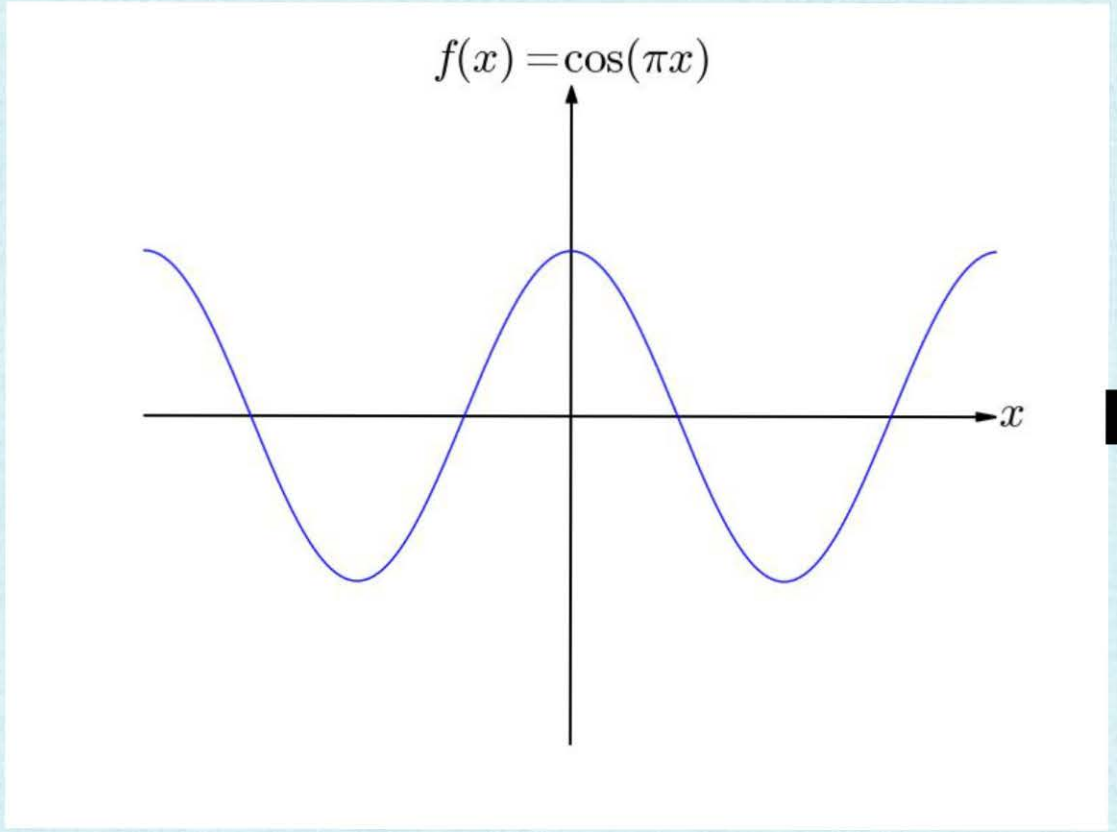
$$e^{i\theta} = \cos\theta + i\sin\theta$$

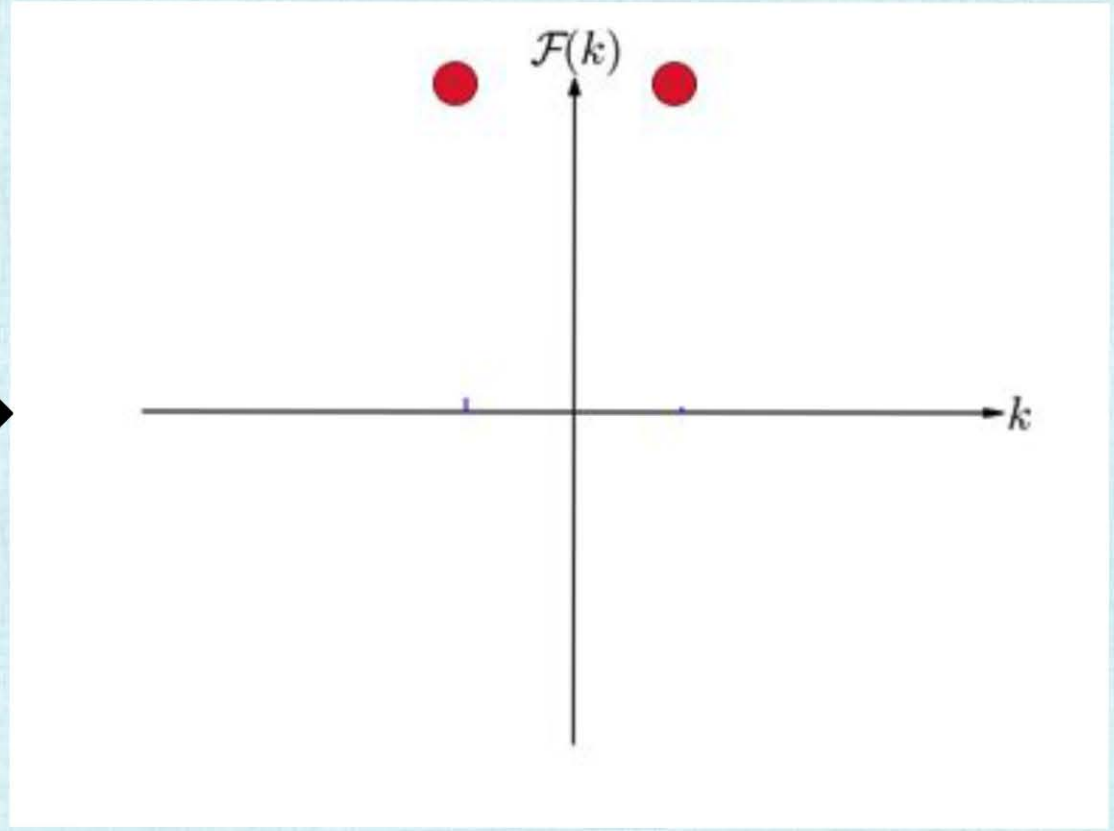
$$\cos\theta = \frac{e^{i\theta} + e^{-i\theta}}{2} \qquad \sin\theta = \frac{e^{i\theta} - e^{-i\theta}}{2i}$$

Constant Function

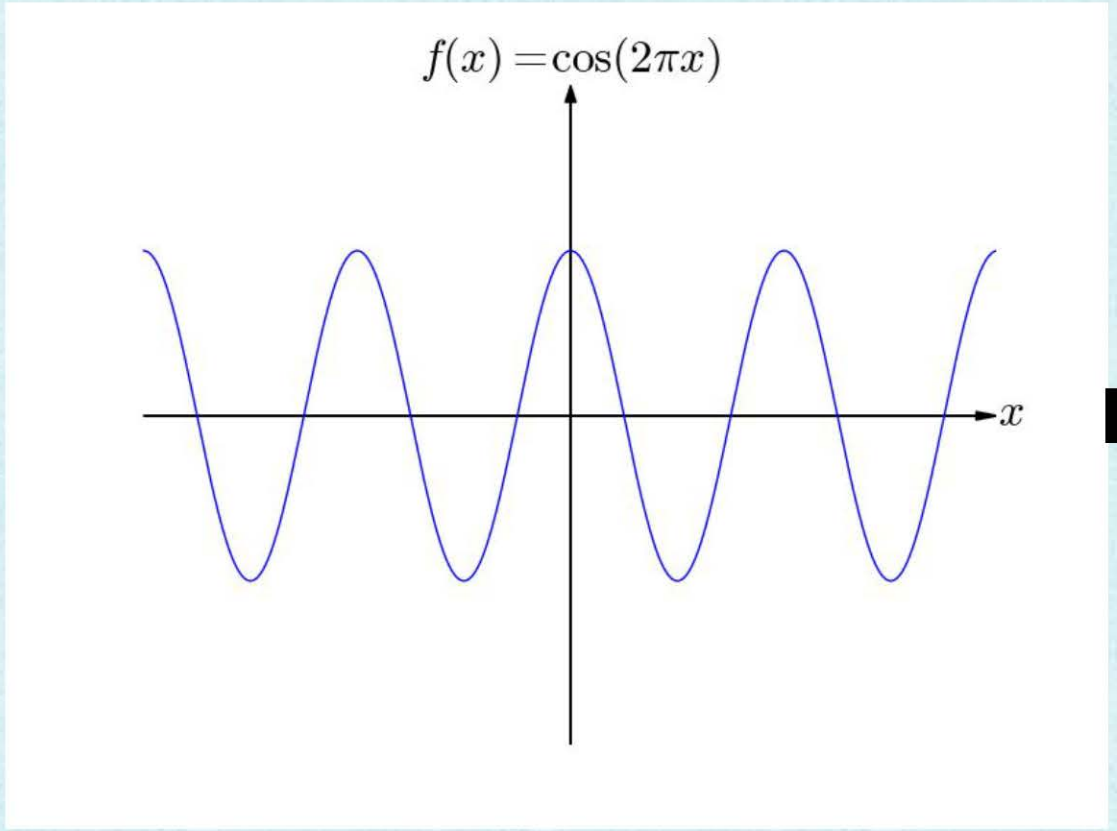


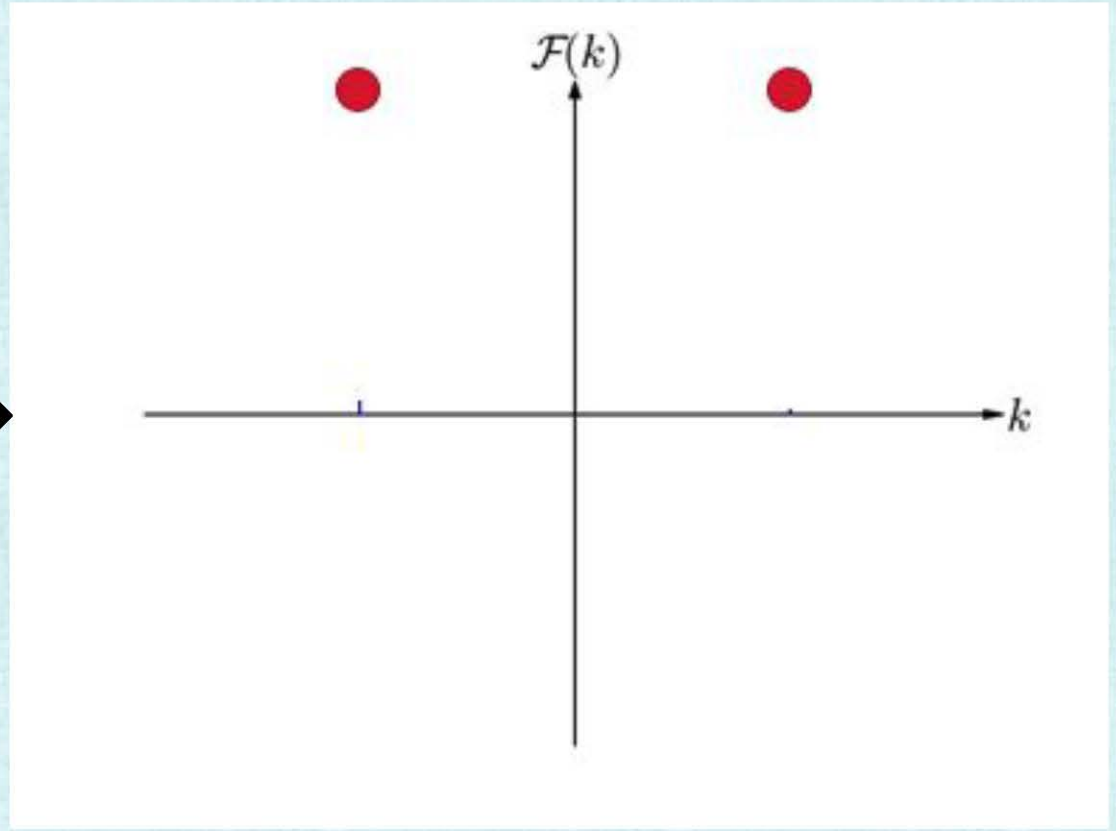
Low Frequency Cosine



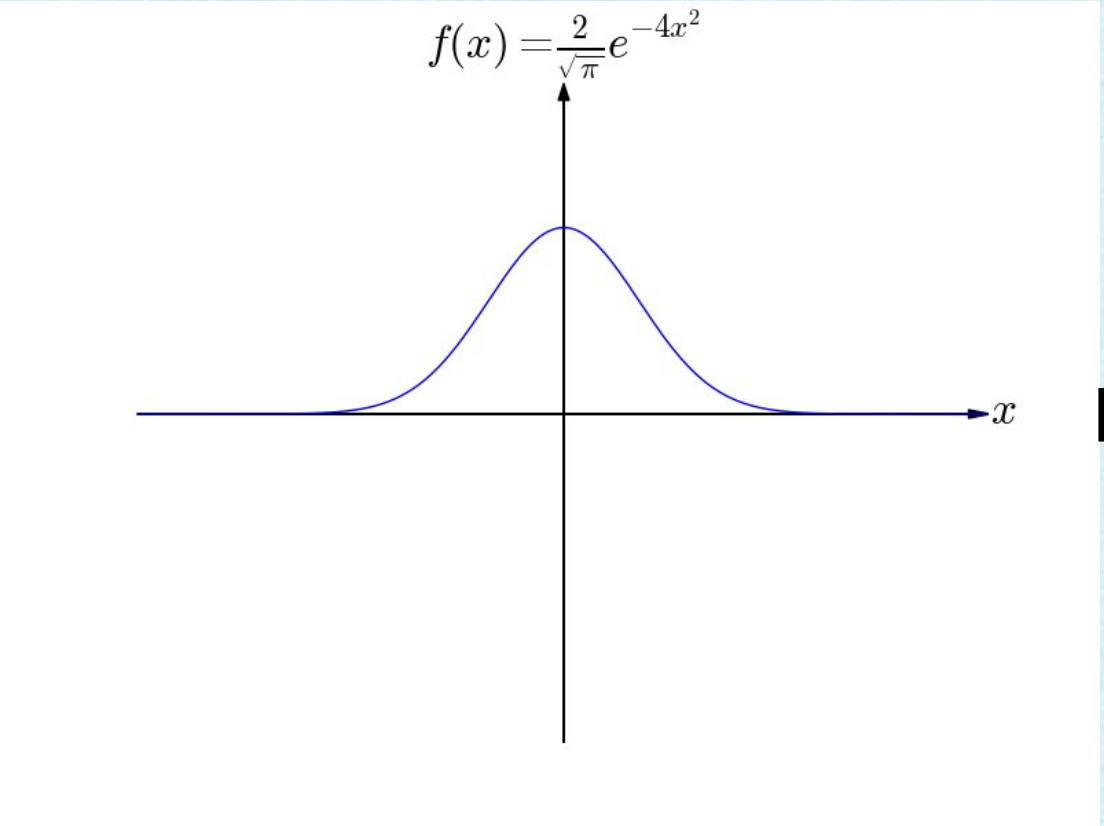


High Frequency Cosine

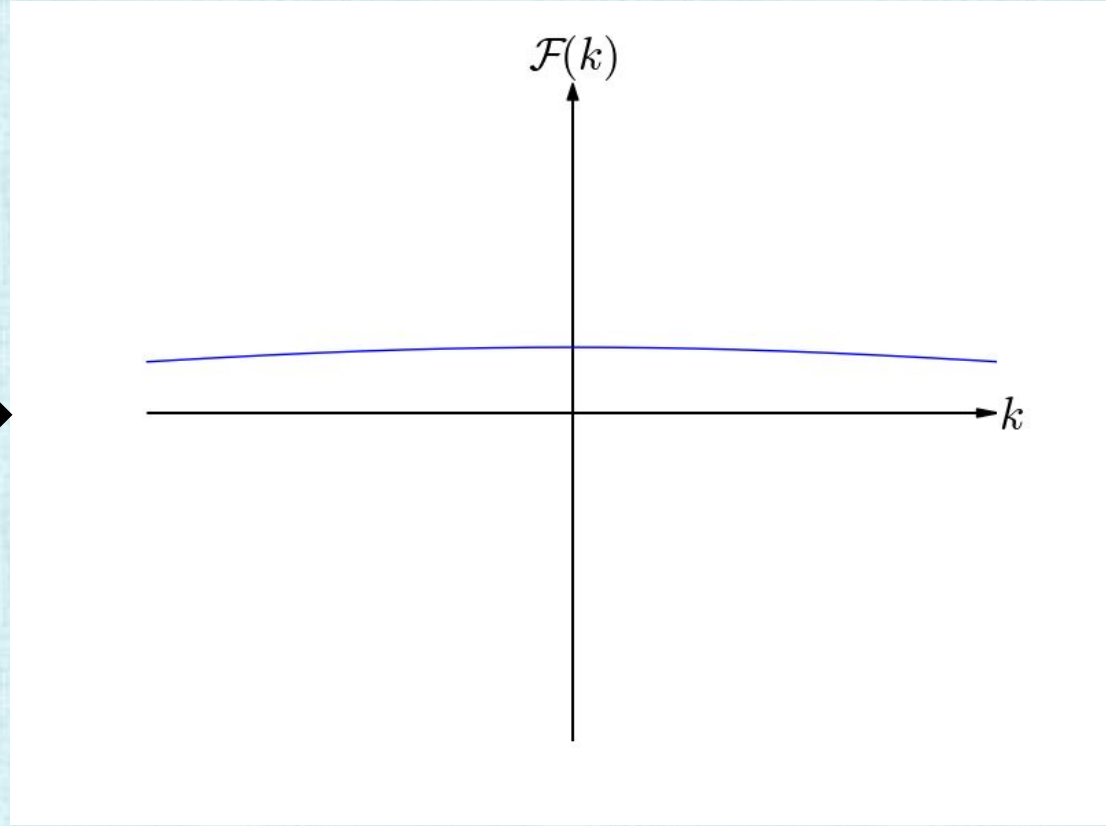




Narrow Gaussian

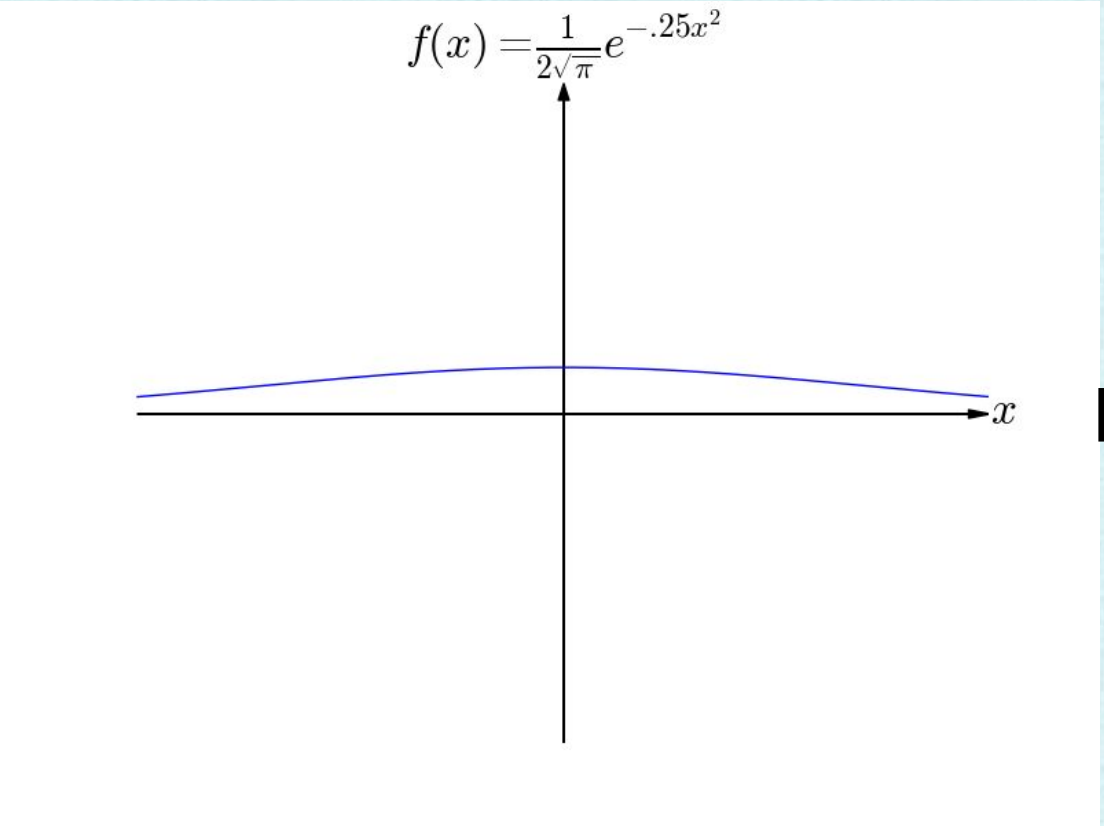


Narrow

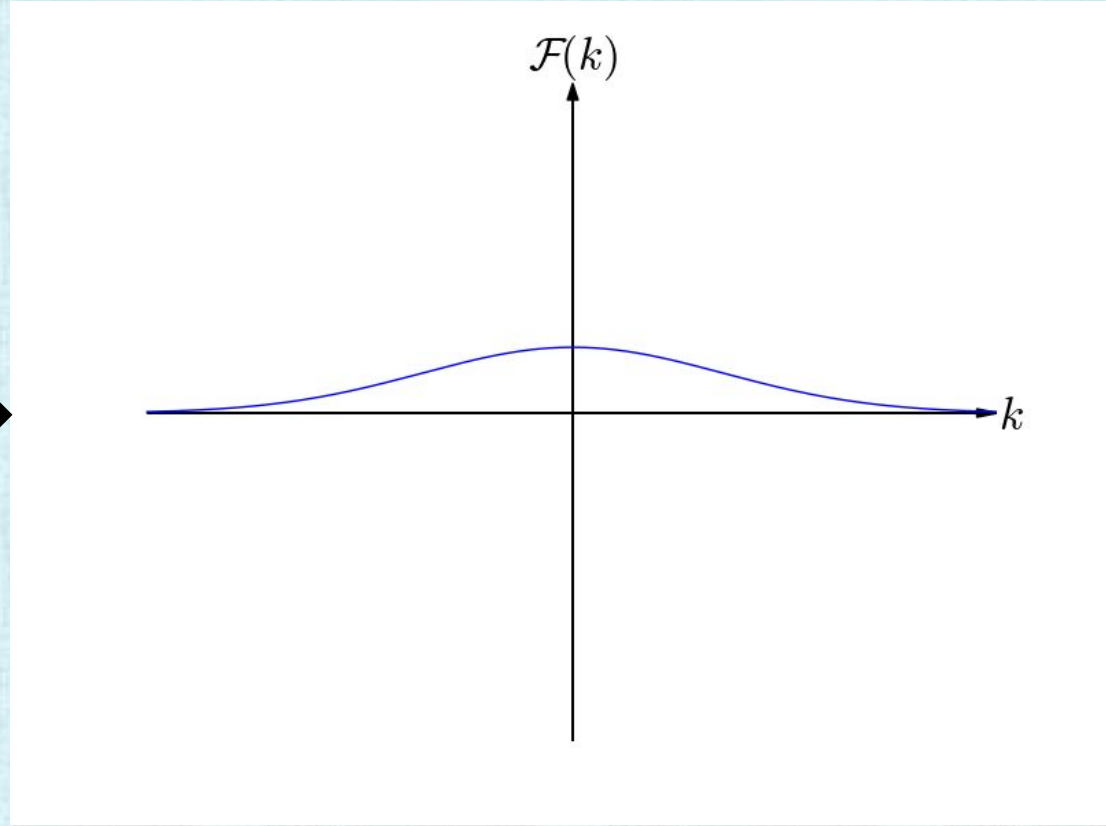


Wide

Wider Gaussian

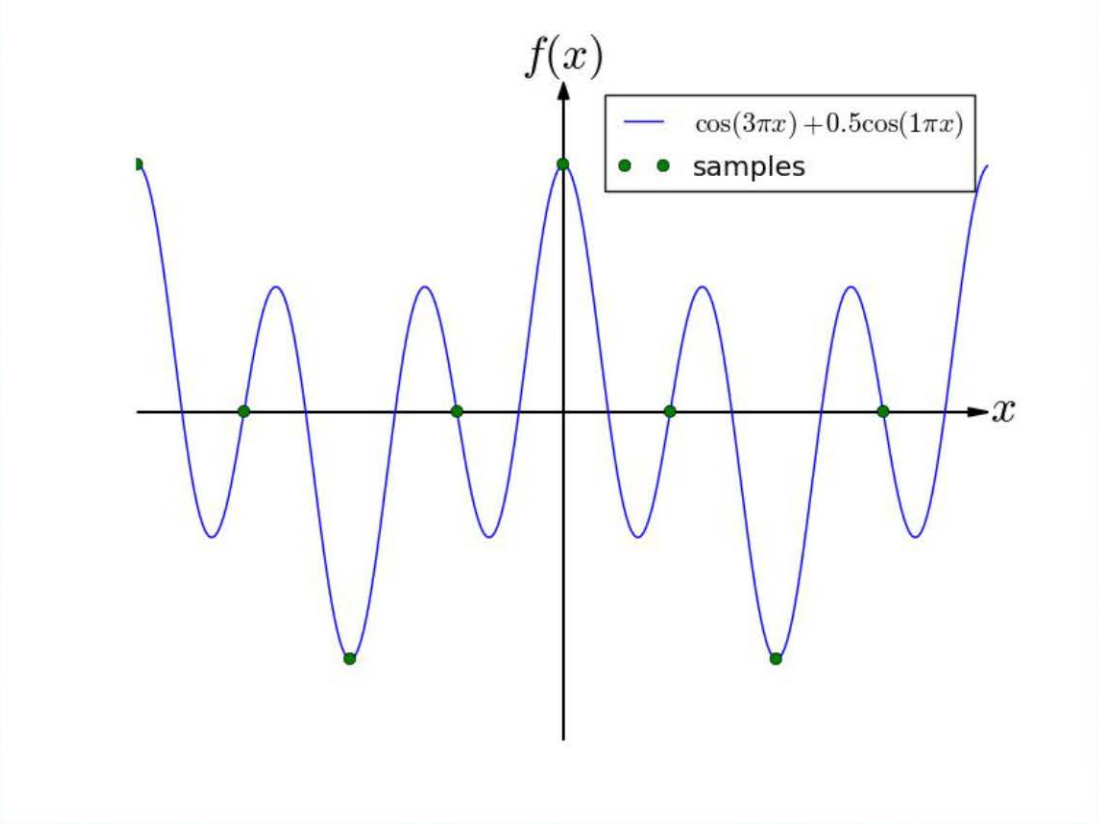


Wider

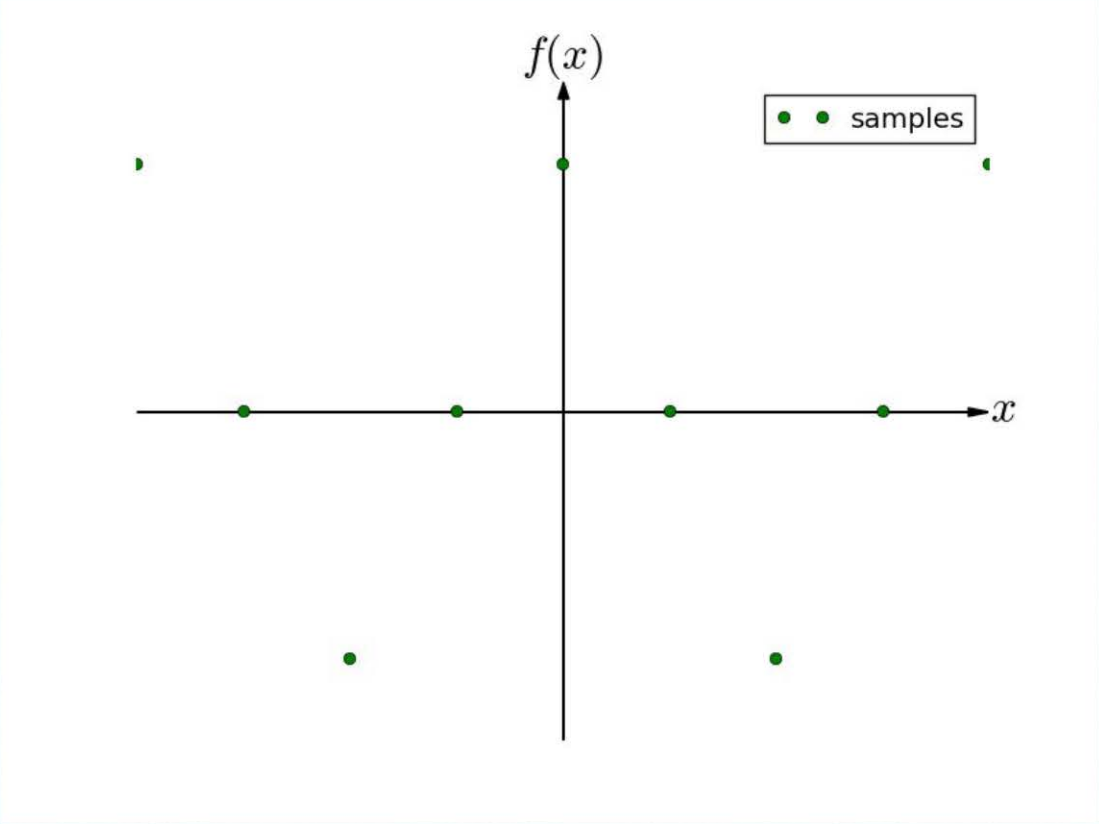


Narrower

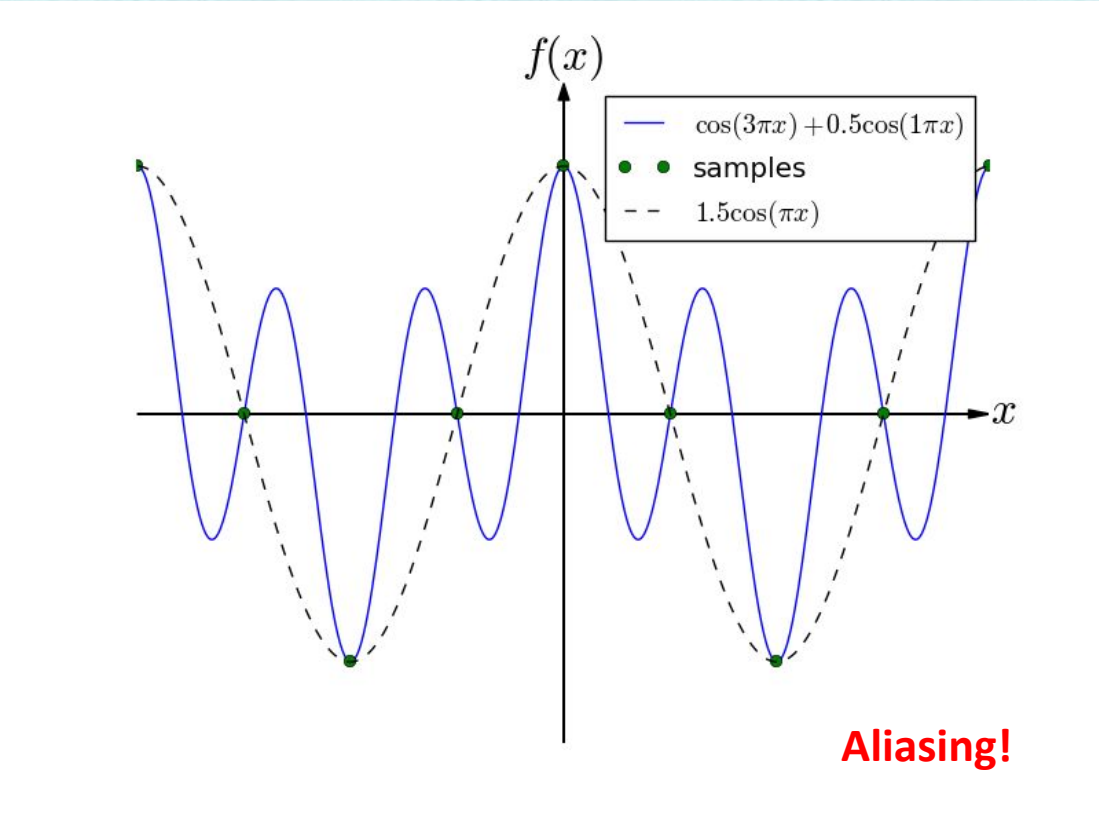
sum of two different cosine functions



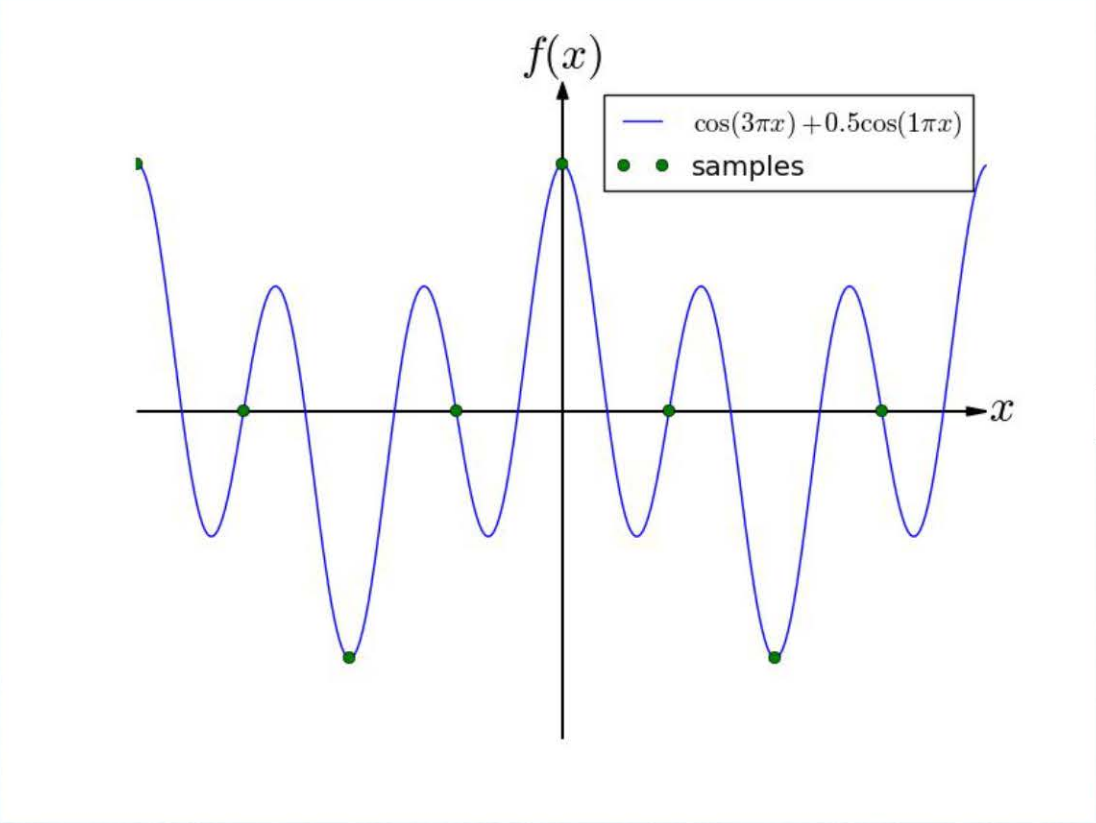
samples

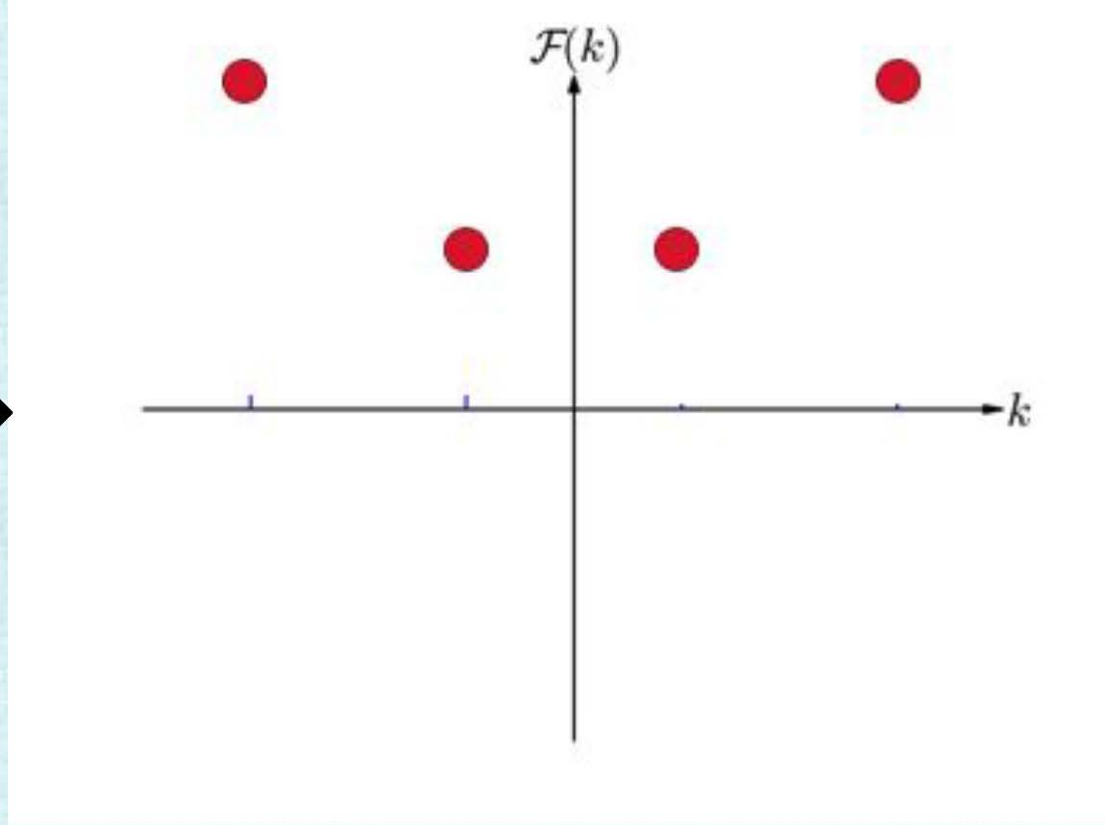


reconstruction

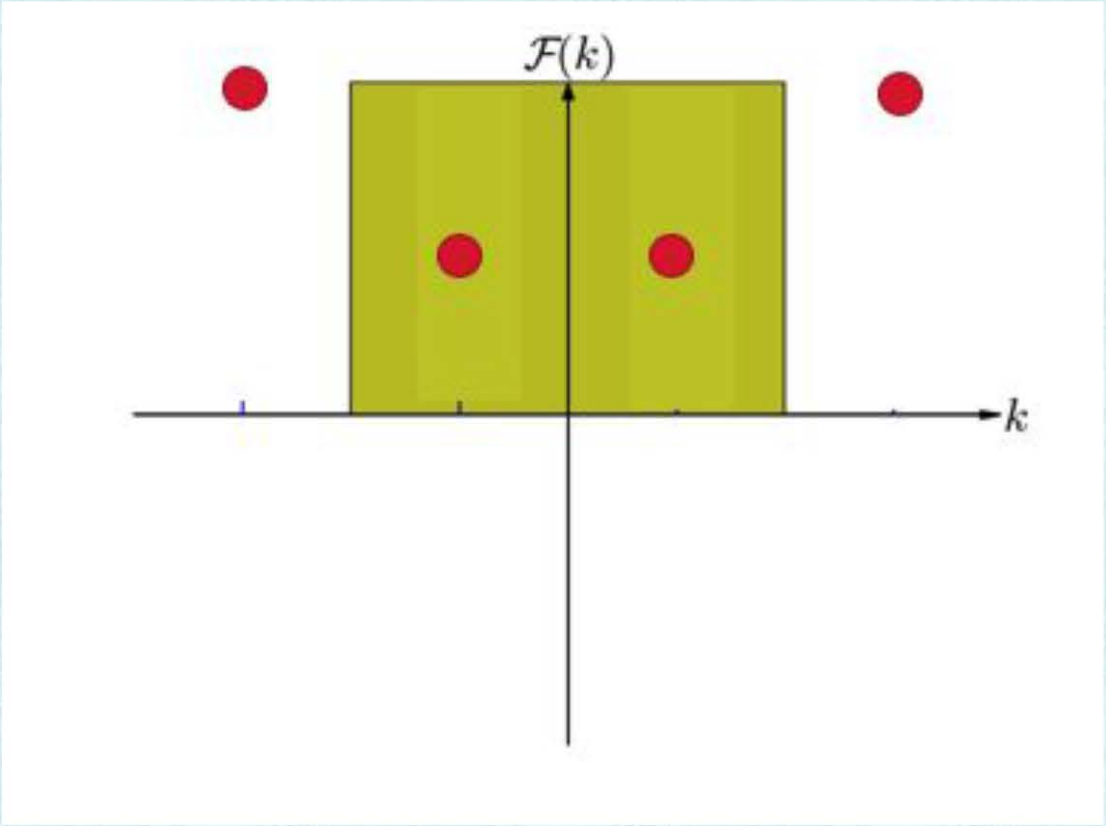


Compute the Fourier Transform

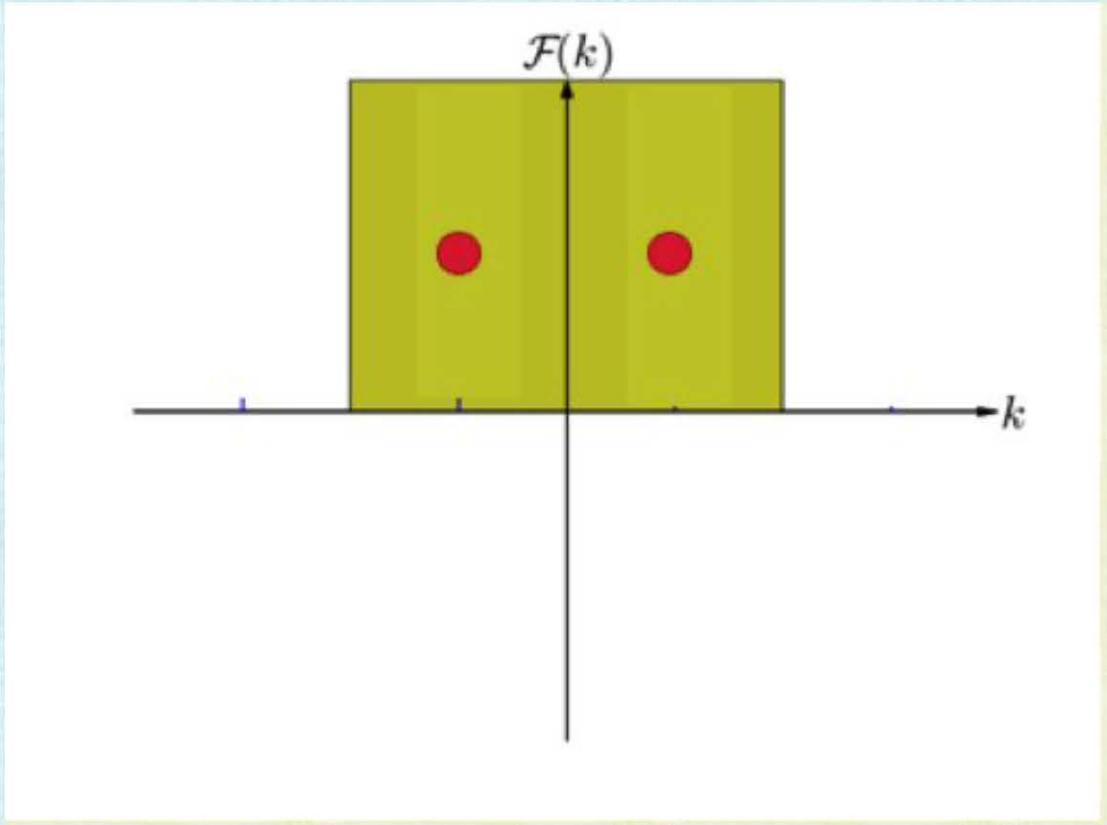




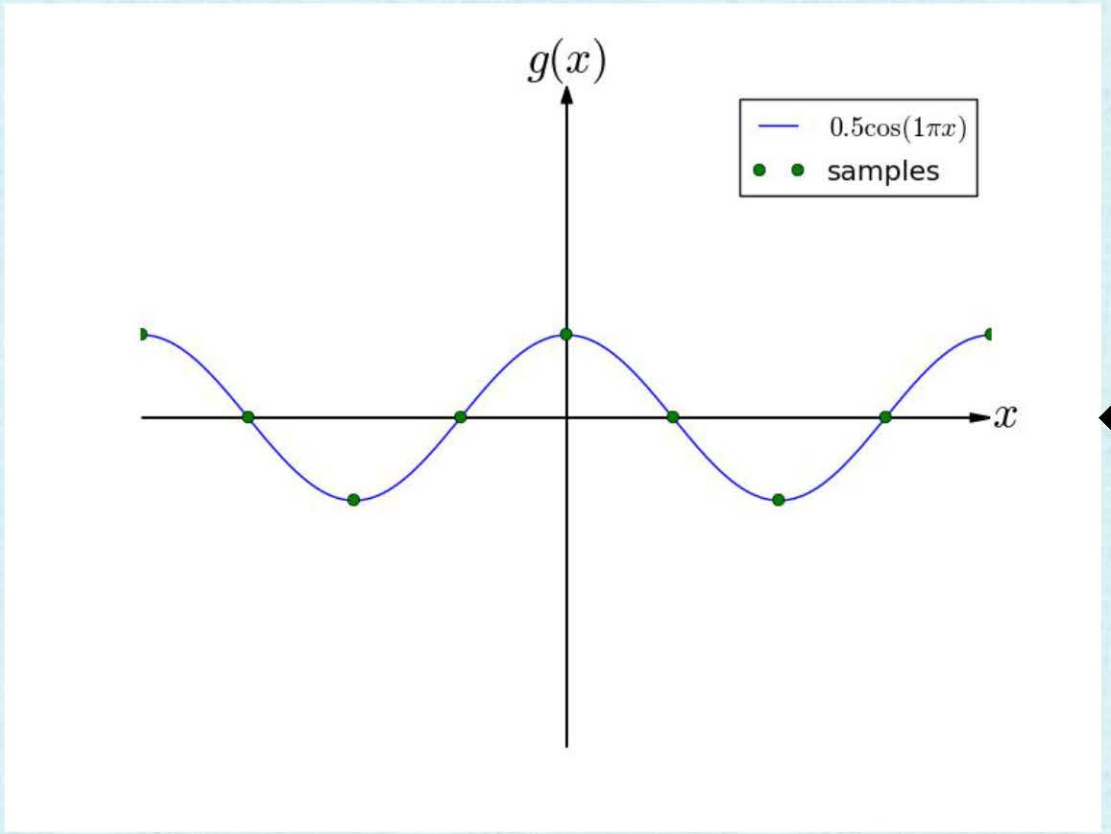
Identify Nyquist Frequency Limits

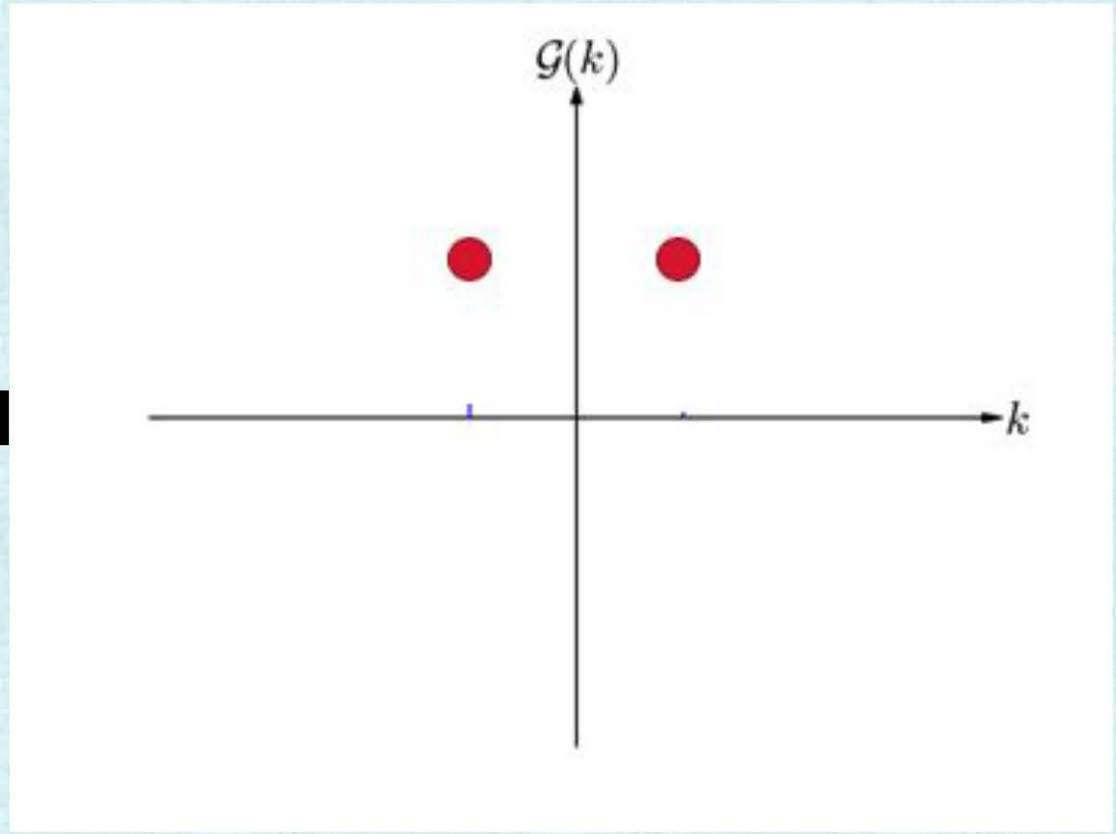


Remove High Frequencies

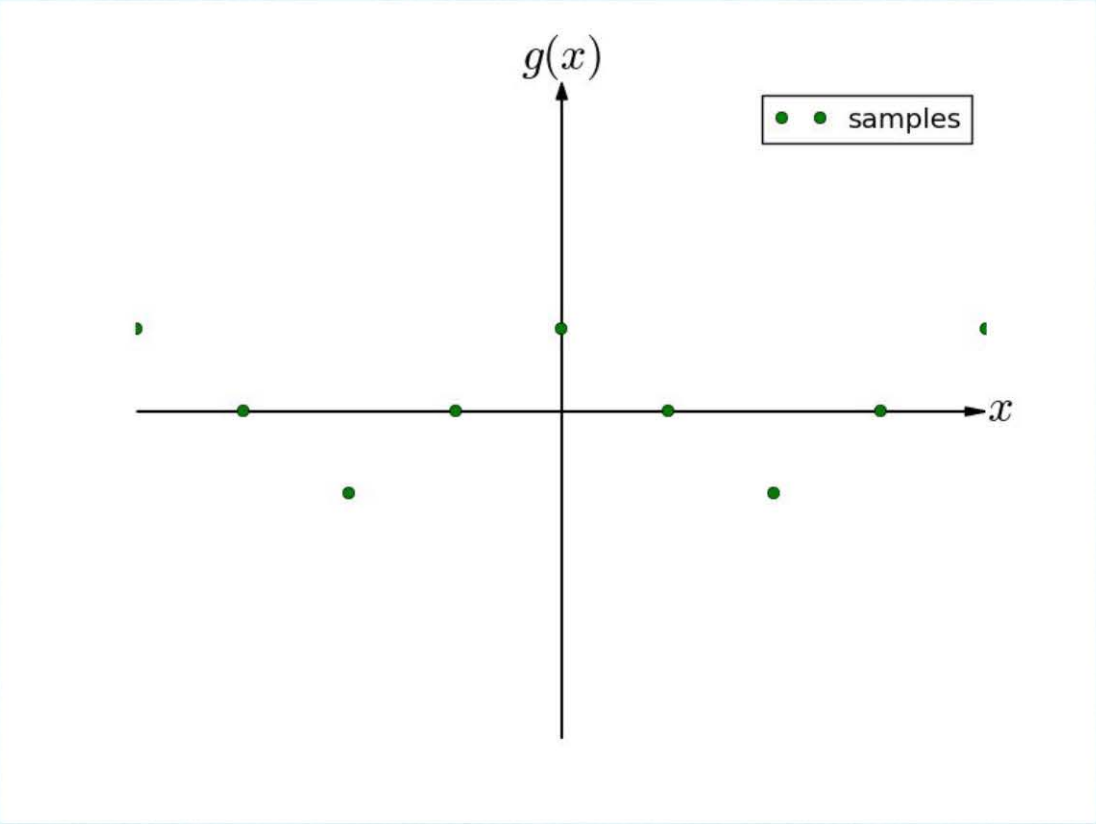


Inverse Fourier Transform

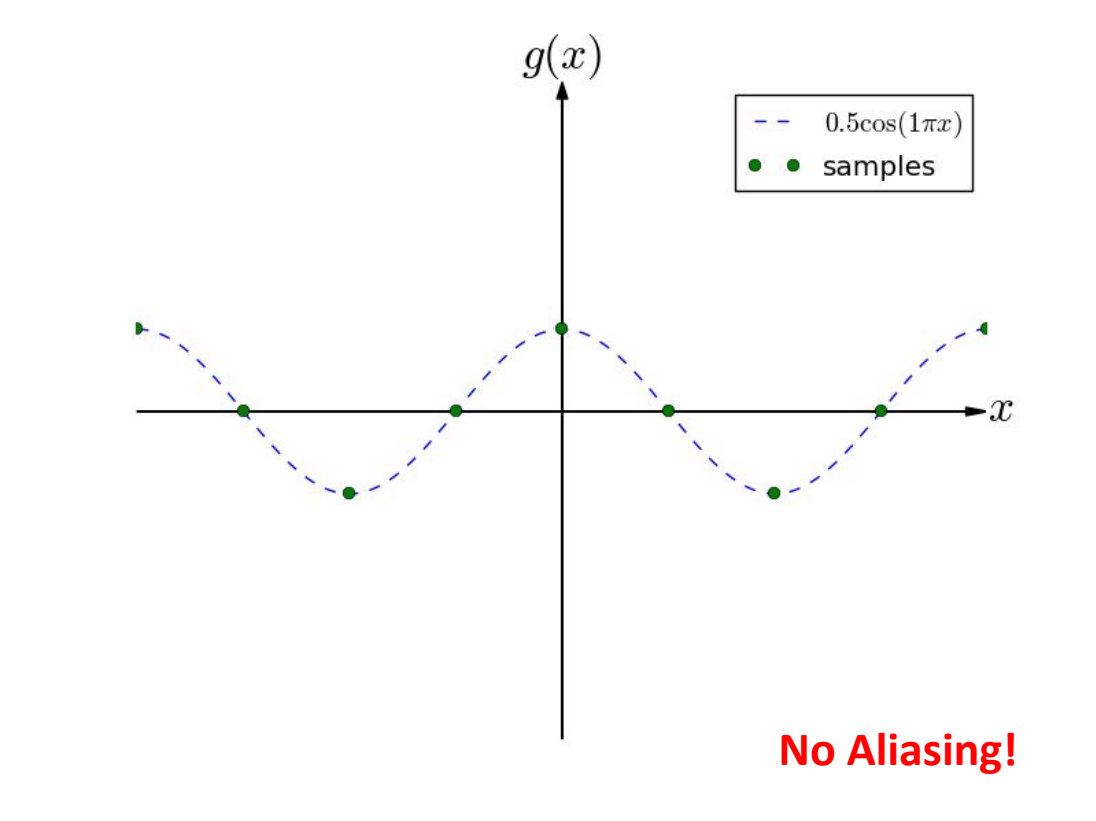




samples



reconstruction

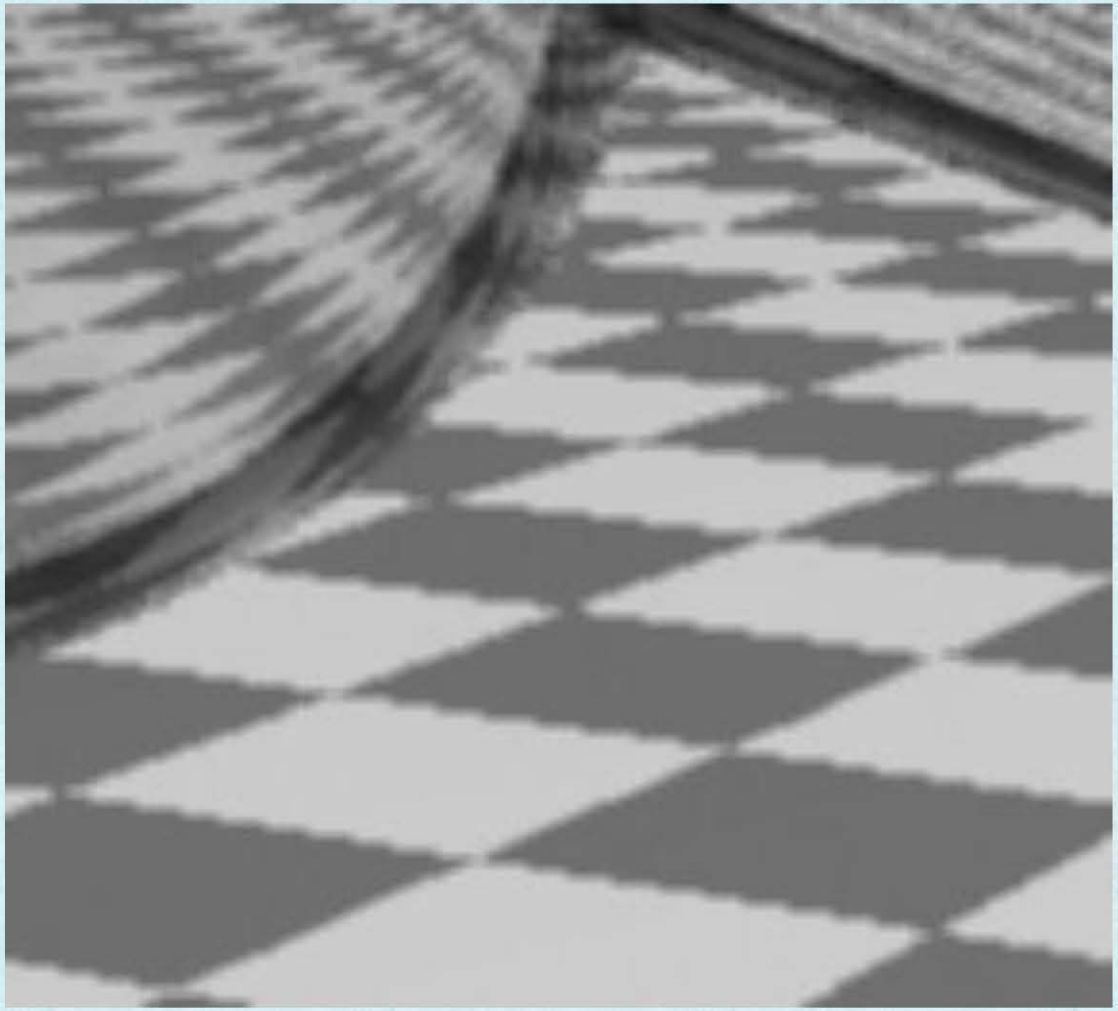


Remove High Frequencies Before Sampling

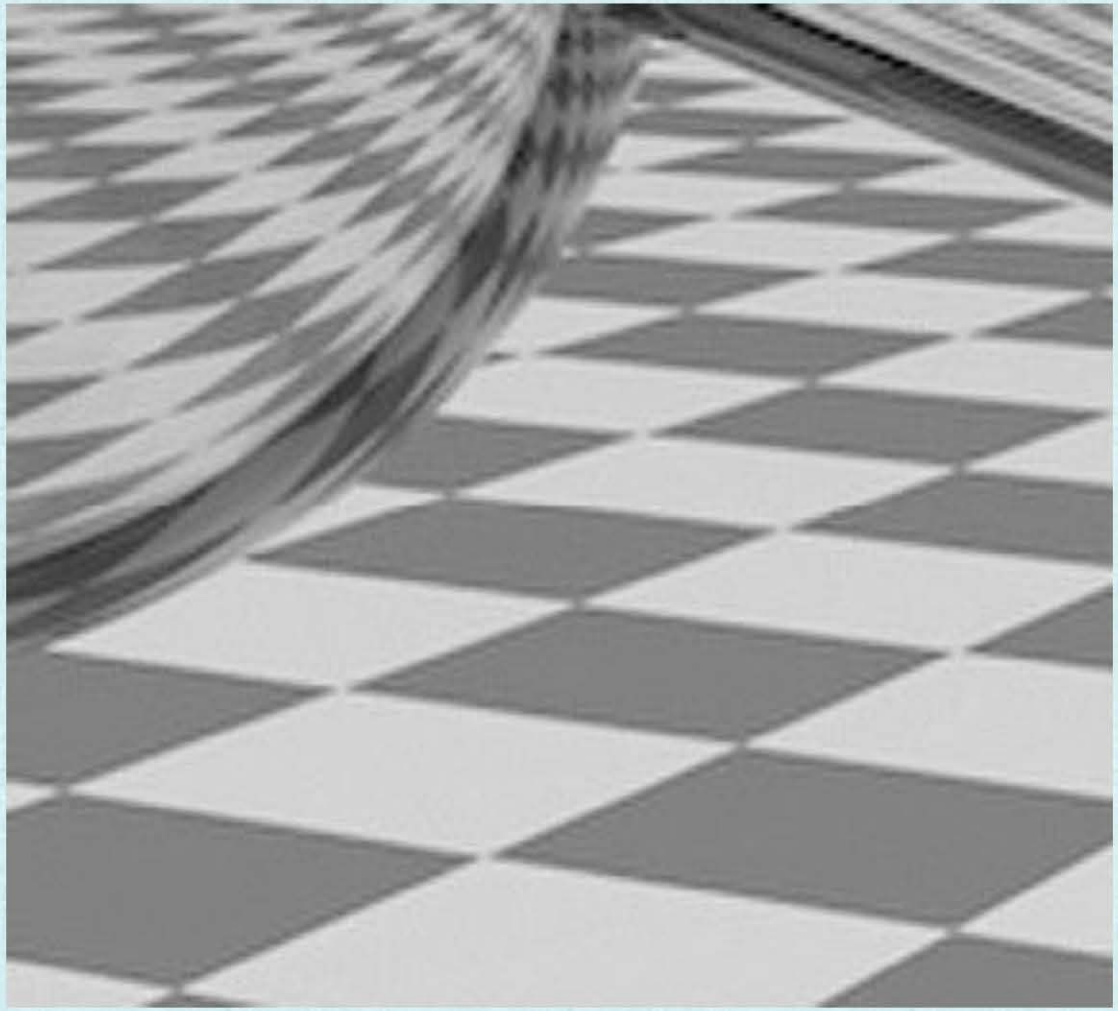
- Sampling causes high frequency components to masquerade as low frequency components
- After sampling, one cannot untangle the high frequencies masquerading as low frequencies from the actual low frequencies
- Must remove the high frequencies before sampling in order to avoid aliasing

- One does lose part of the signal, but this part of the signal was not representable by the sampling!

Blurring vs. Anti-Aliasing



Blurring Jaggies After Sampling



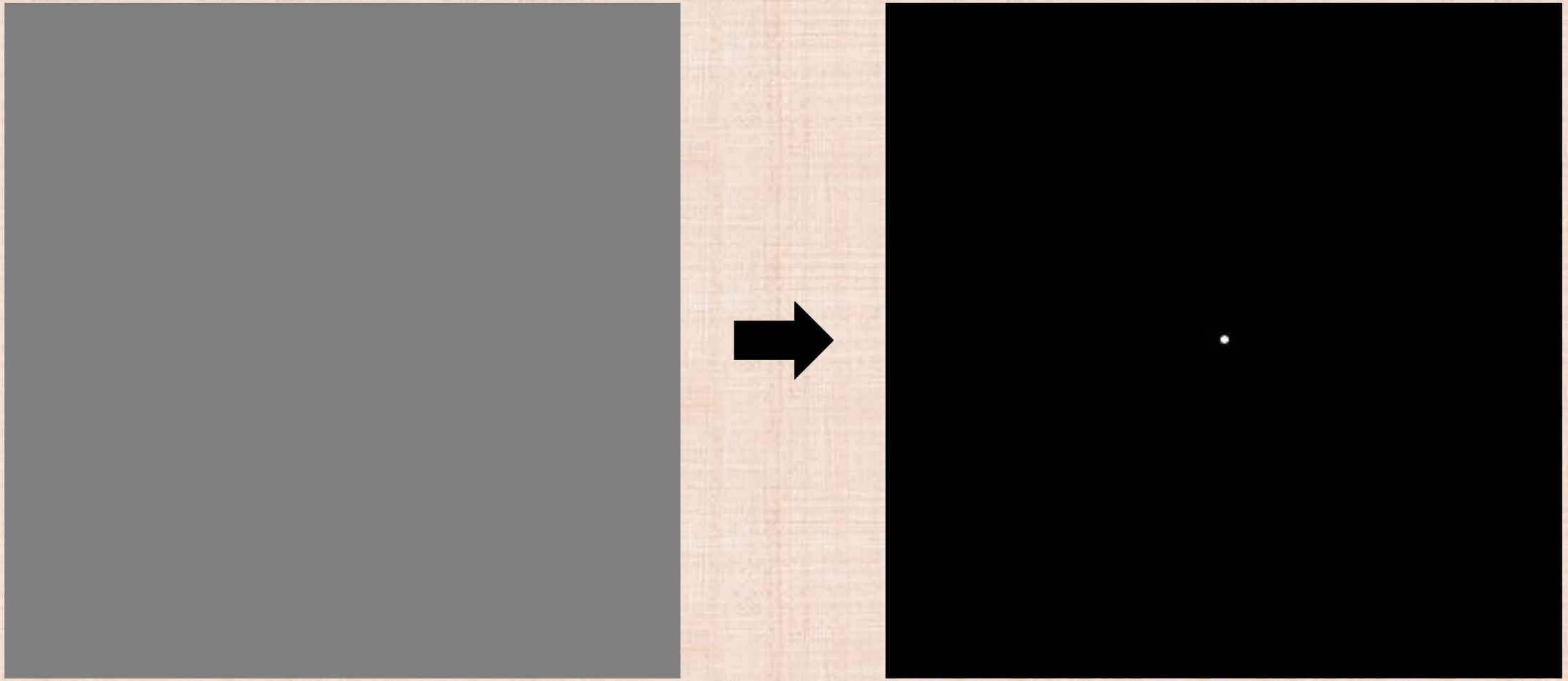
Removing High Frequencies Before Sampling

Images

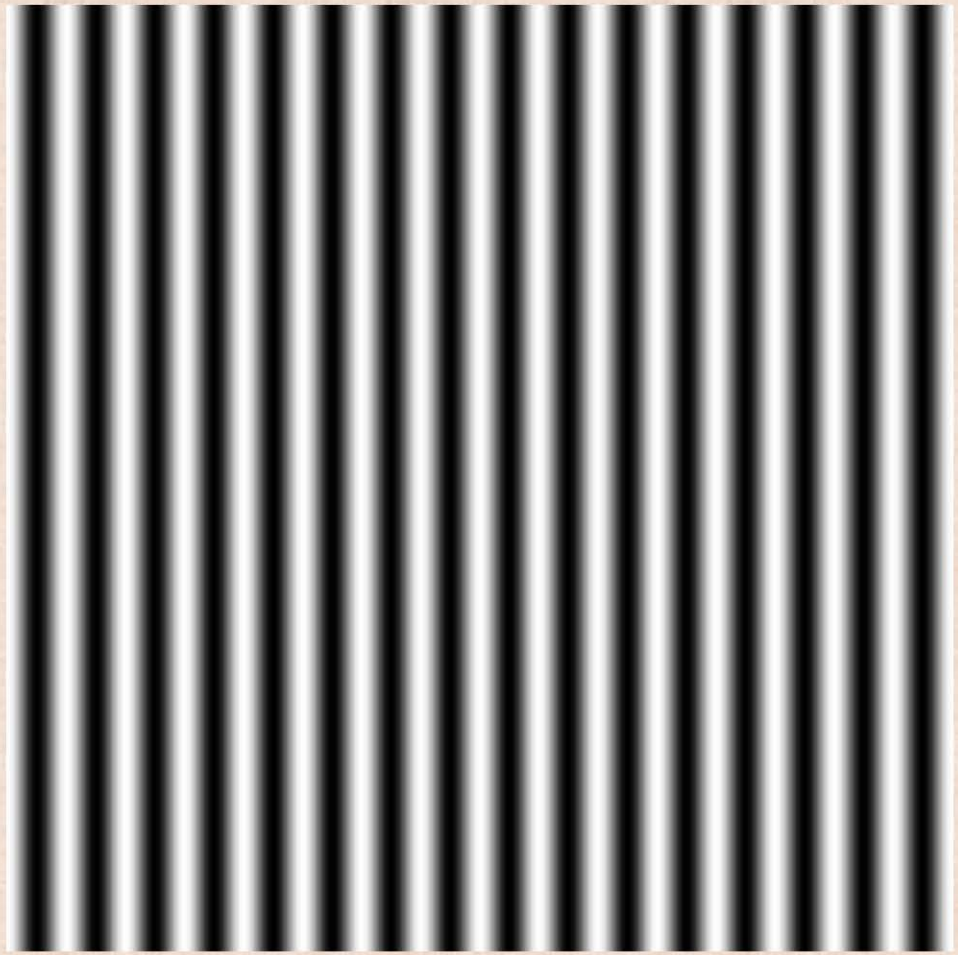
- Images have discrete values (and are not continuous functions, like sin/cos)
  - Use a discrete Fourier transform
  - The Fast Fourier Transform (FFT) algorithm computes the discrete Fourier transform and its inverse in $O(n \log n)$ complexity (where n is the number of samples)
- 2D discrete Fourier transform is computed by applying 1D transforms (i.e. FFTs) along each dimension

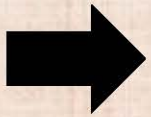
1. Compute the discrete Fourier transform (via FFT)
   - Transforms the discrete image values into another array of discrete values
2. Perform operations (to remove high frequencies)
3. Compute the inverse discrete Fourier transform (via FFT)

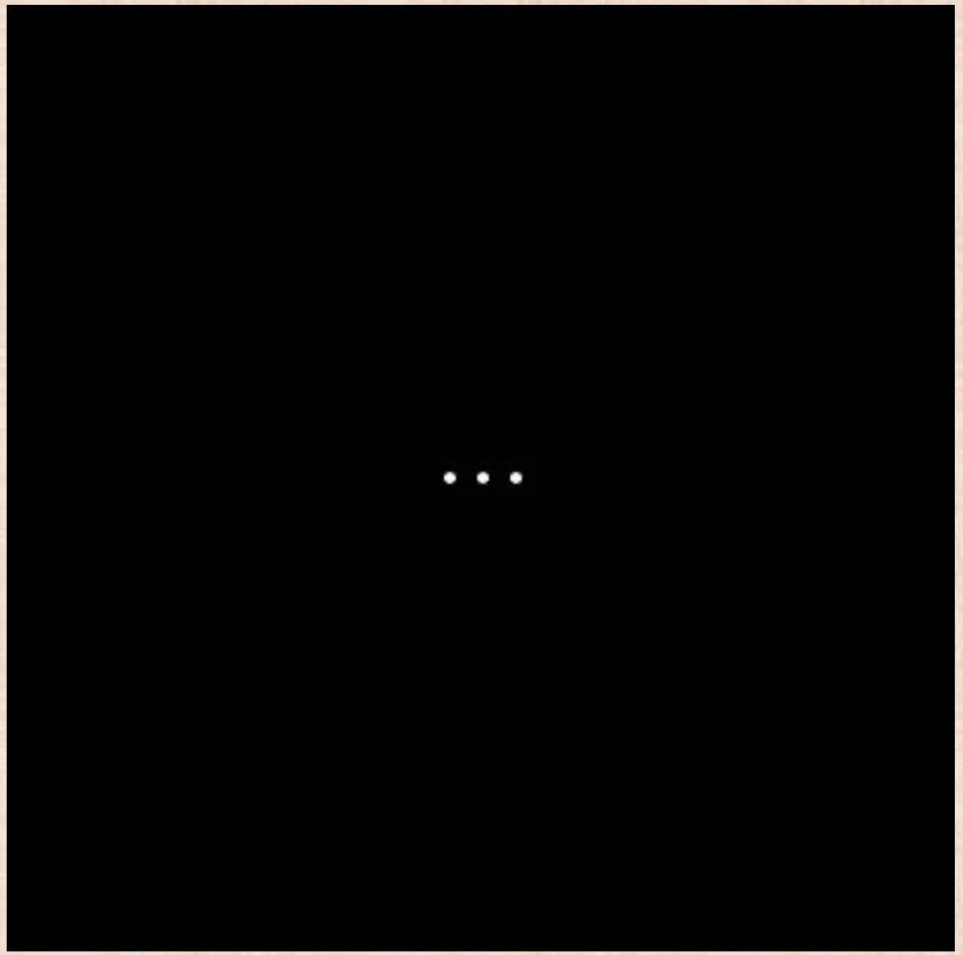
Constant Function



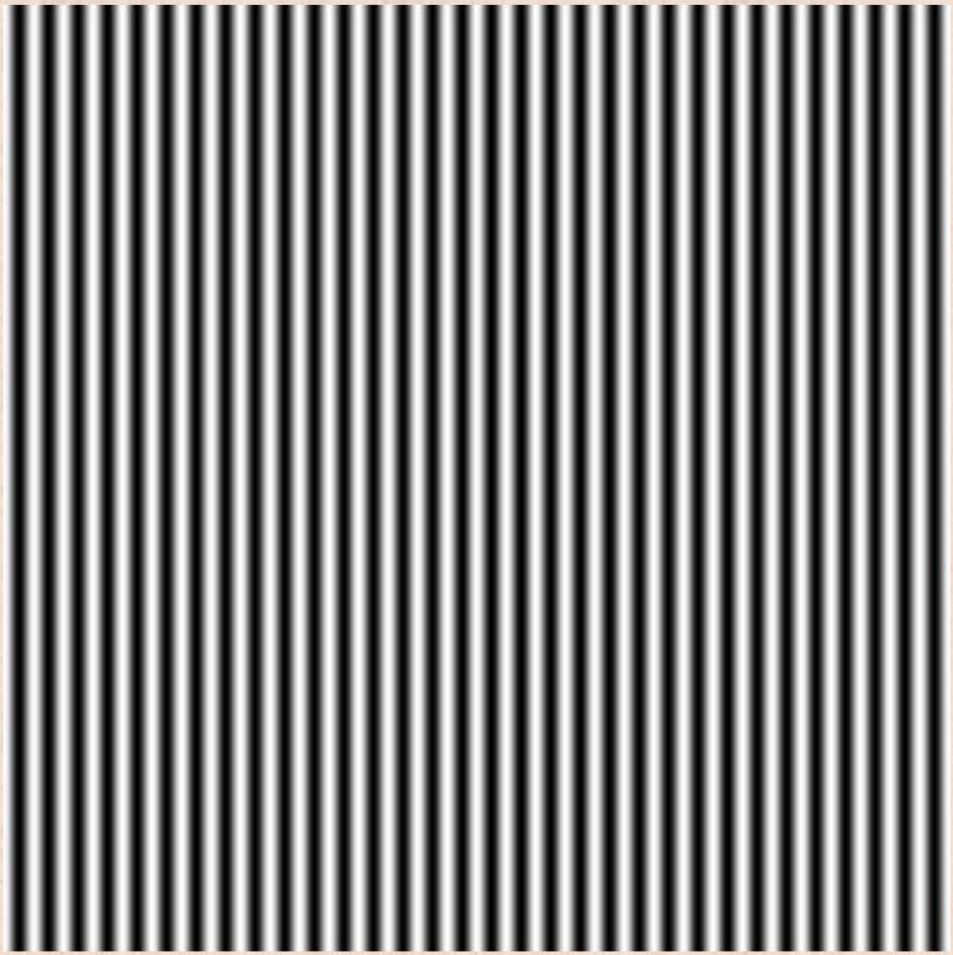
$$\sin(2\pi/32)\,x$$

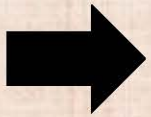


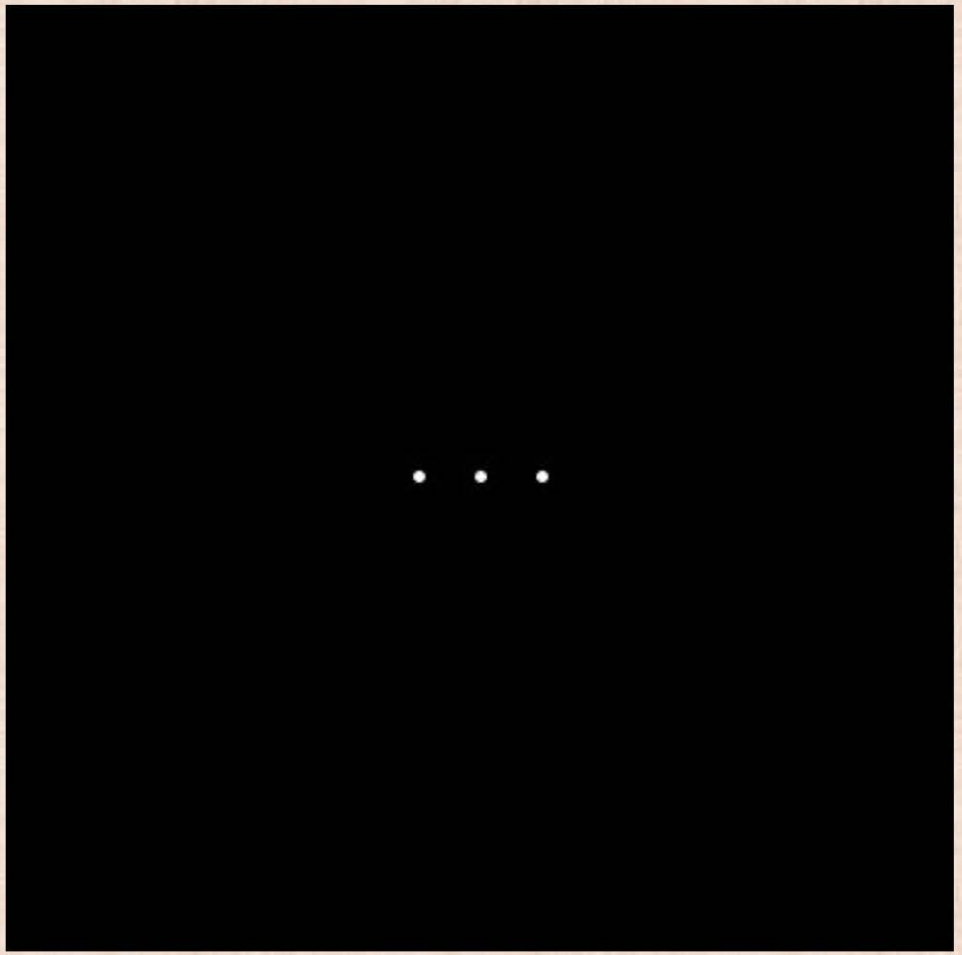




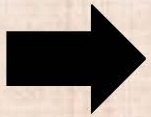
$$\sin(2\pi/16)\,x$$

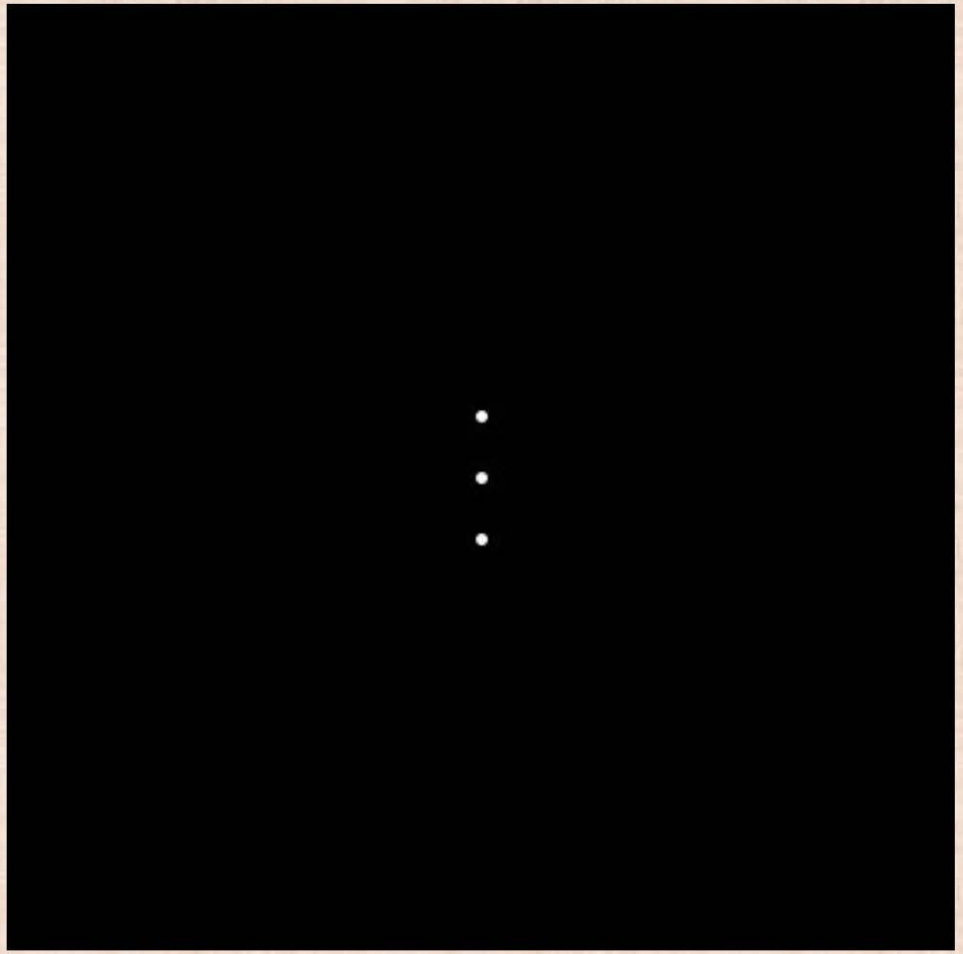




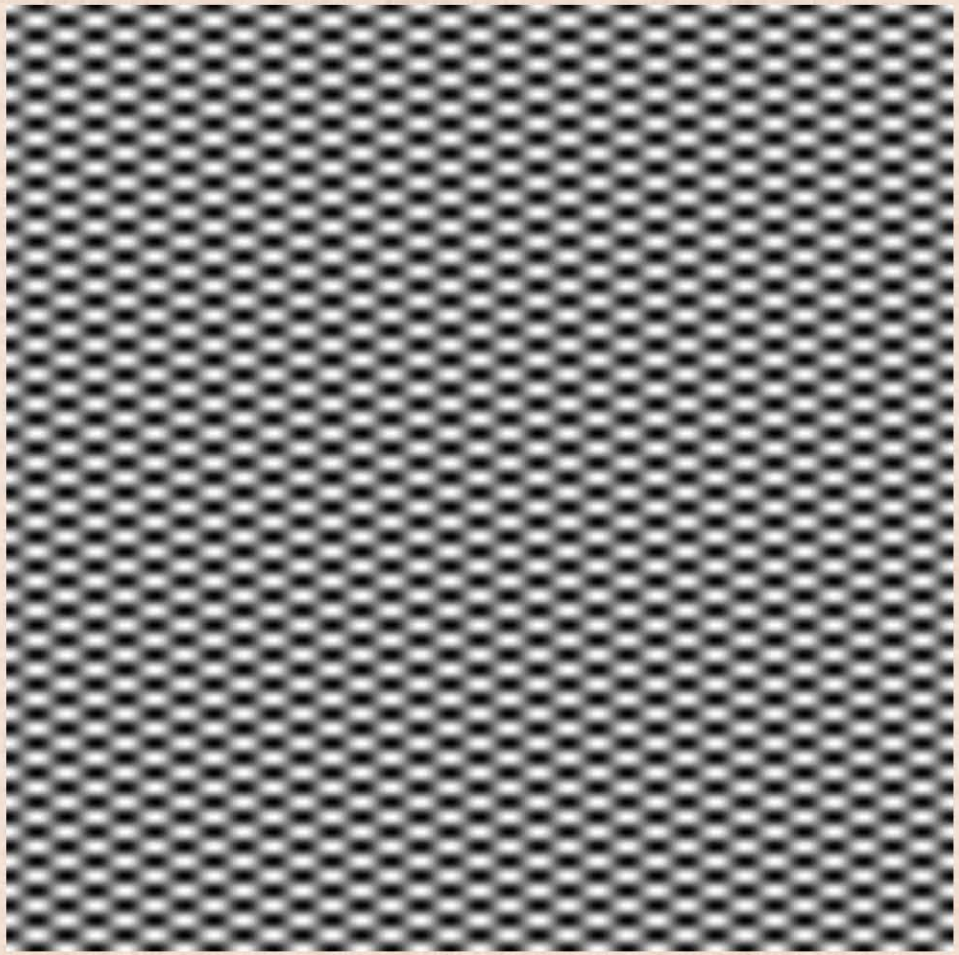


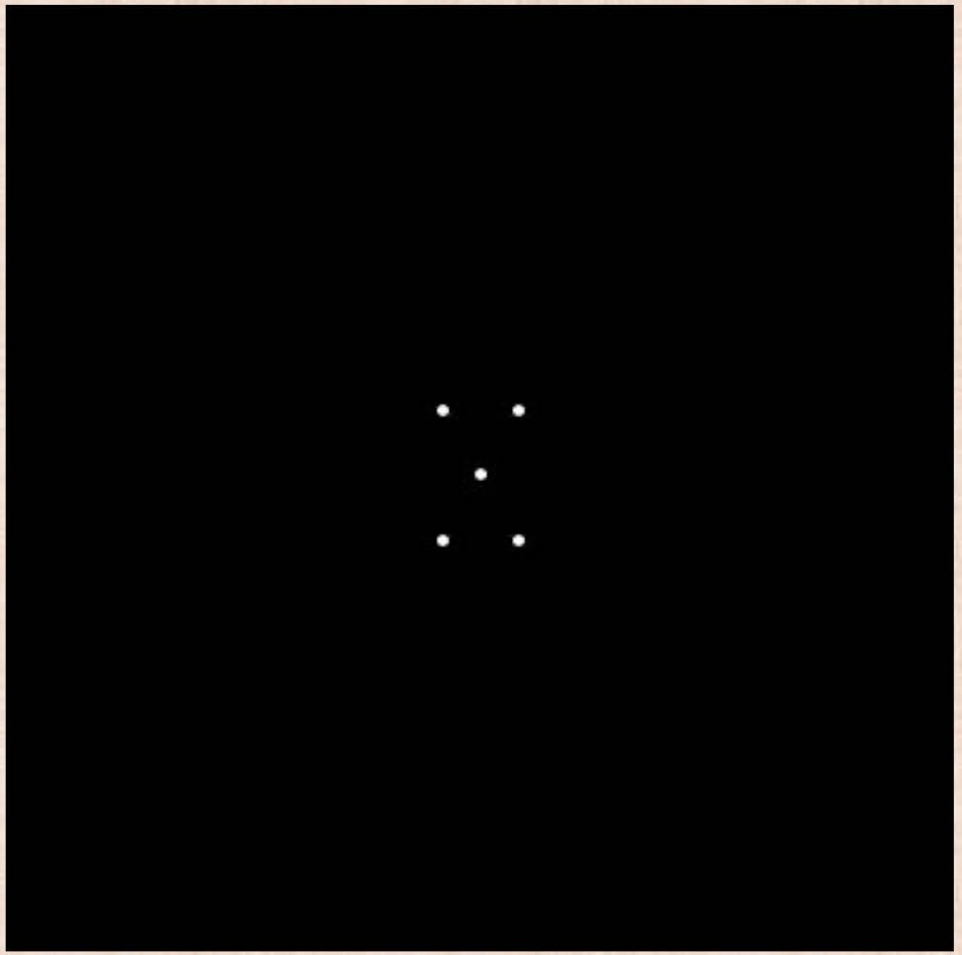
$$\sin(2\pi/16)\,y$$



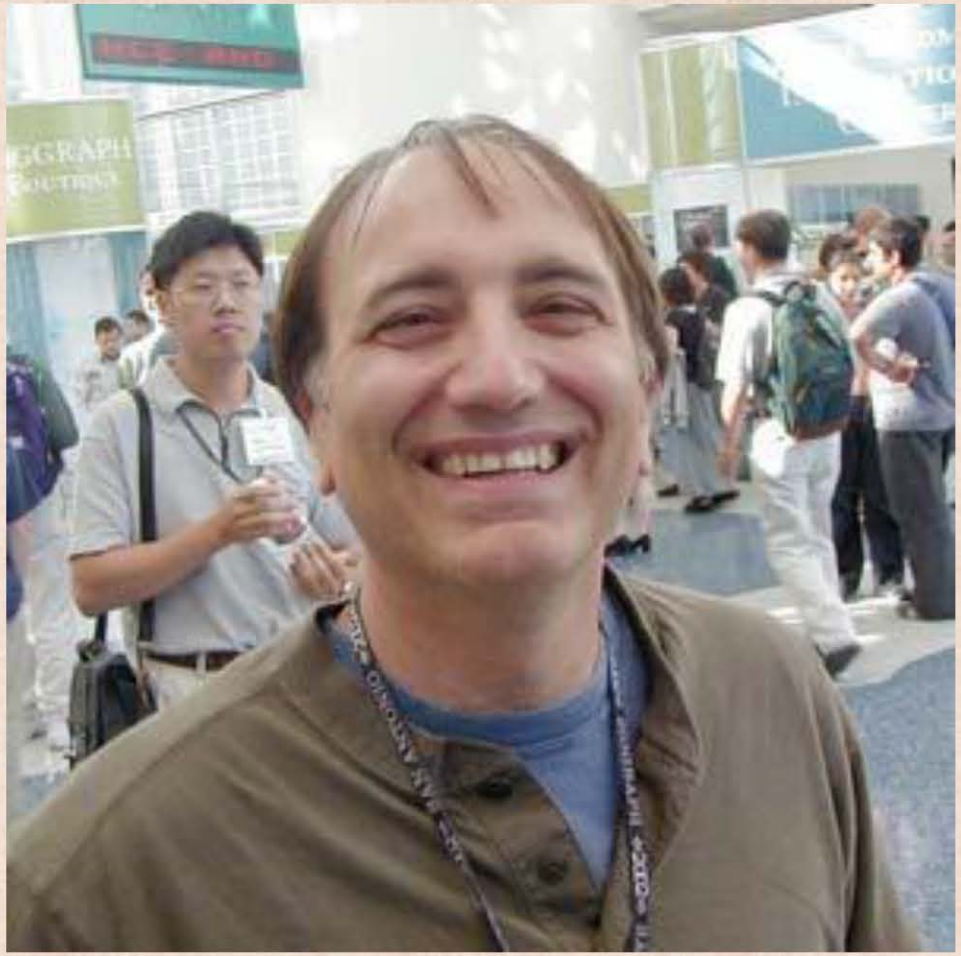


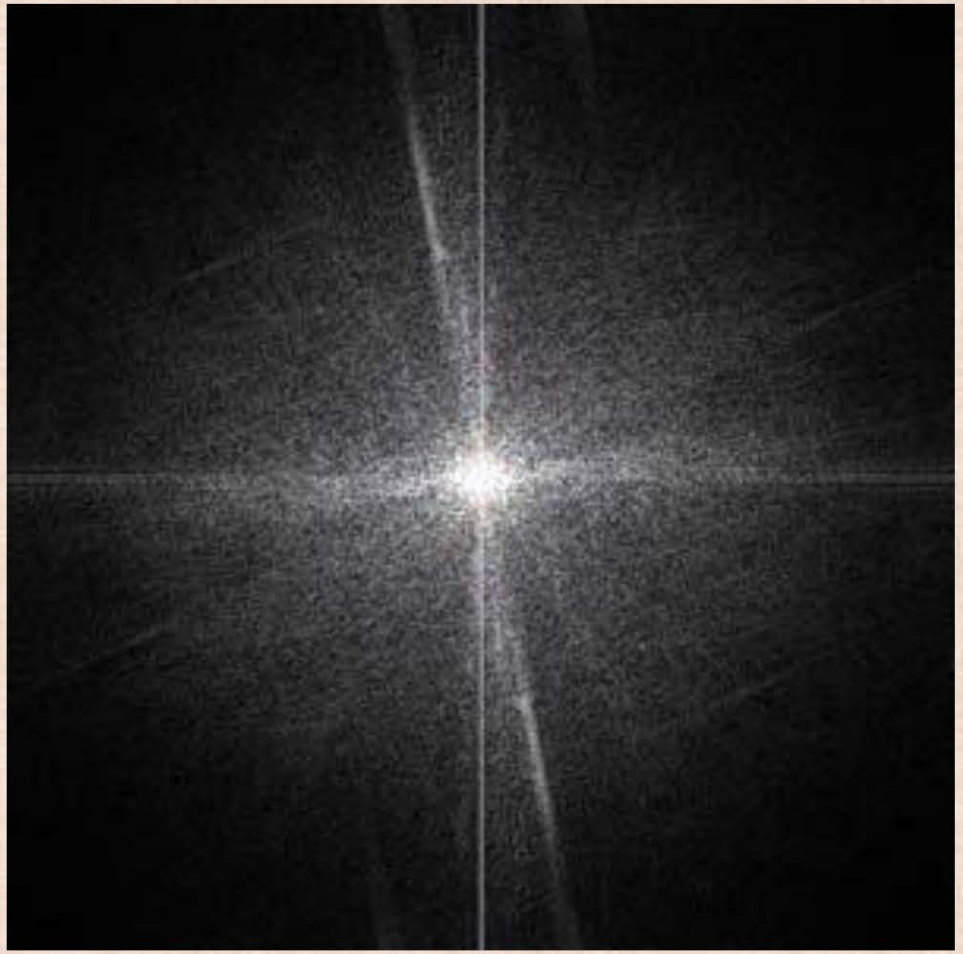
$$\sin(2\pi/32)\,x \;*\; \sin(2\pi/16)\,y$$



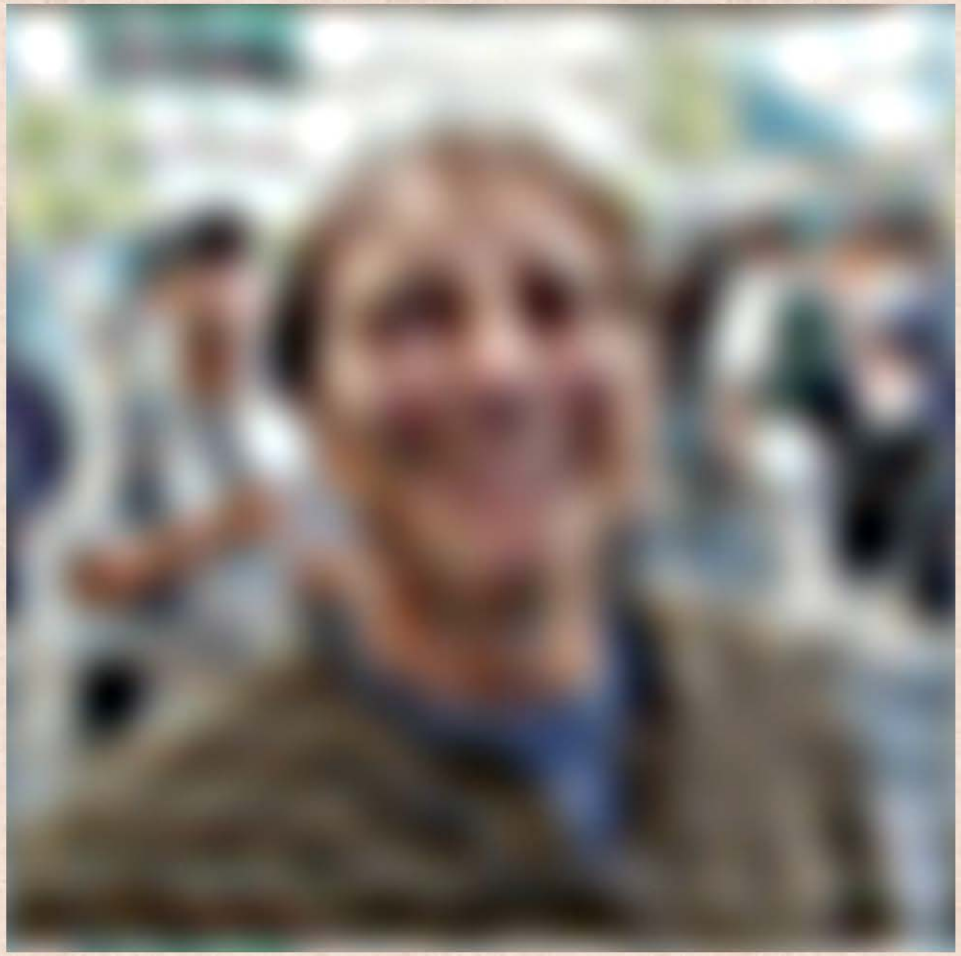


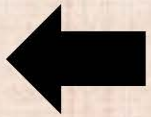
An Obvious Star!

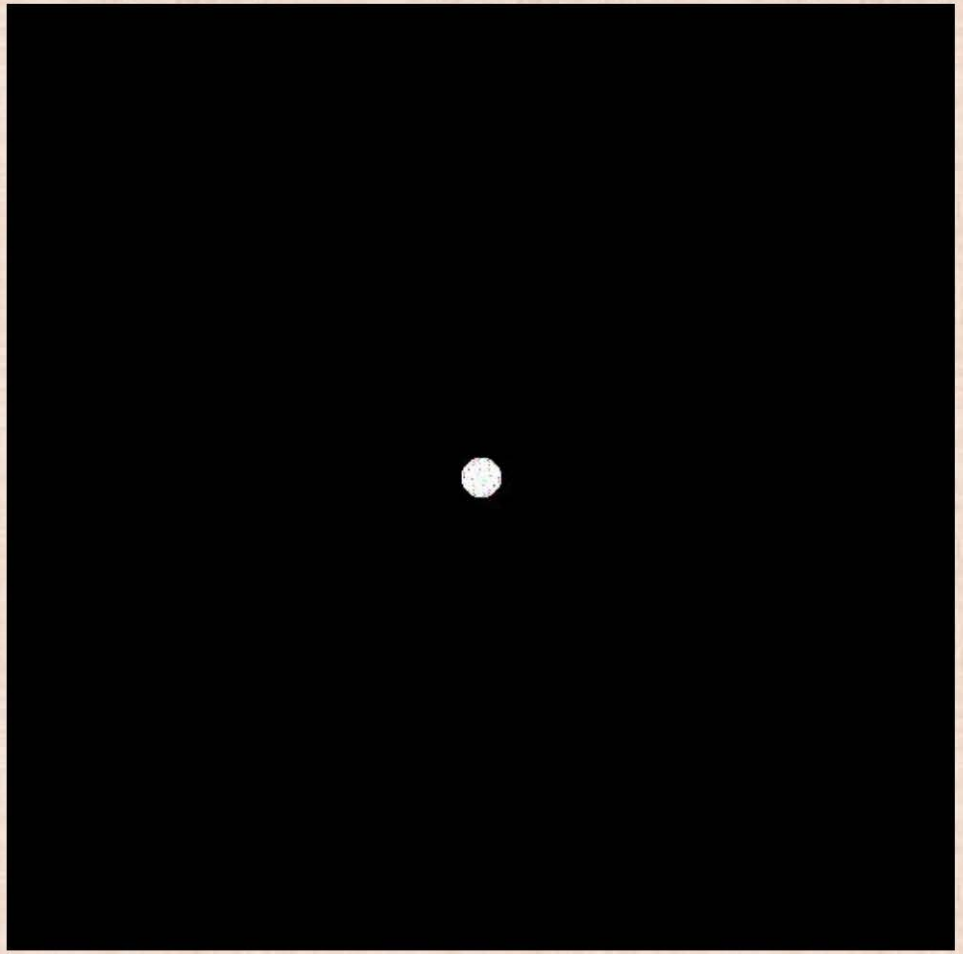




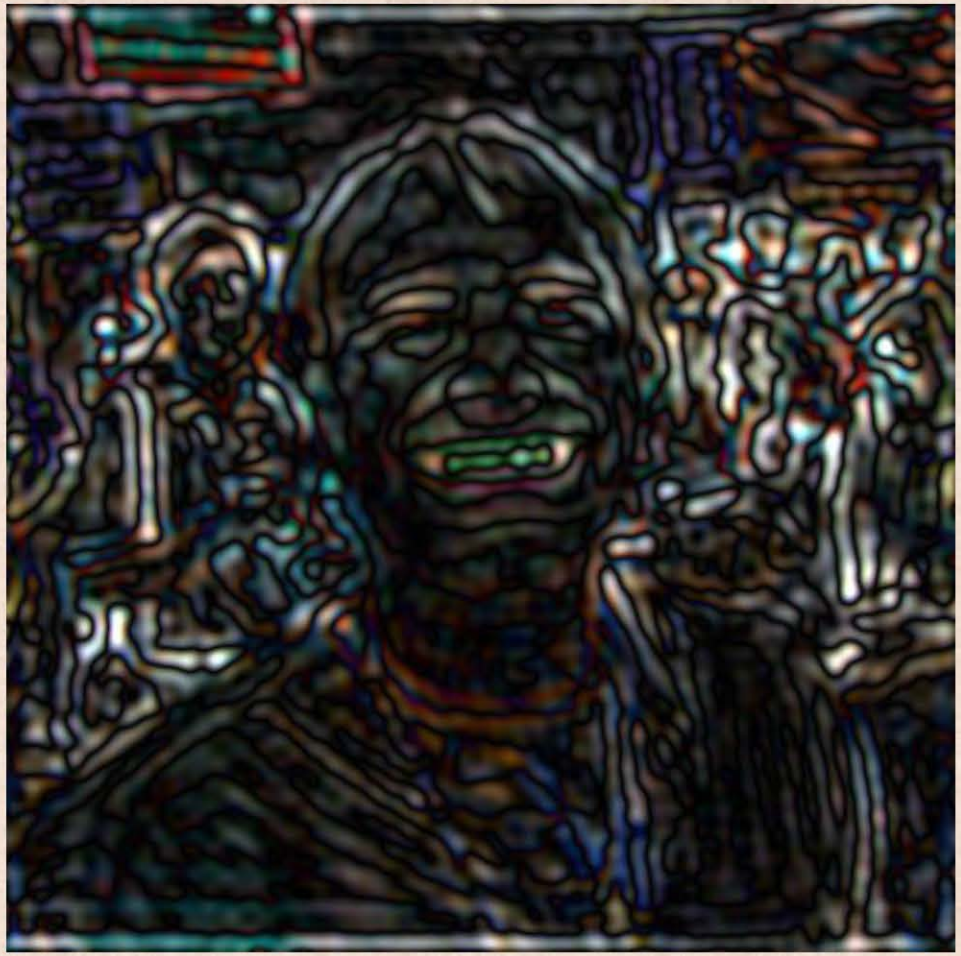
Lowest Frequencies

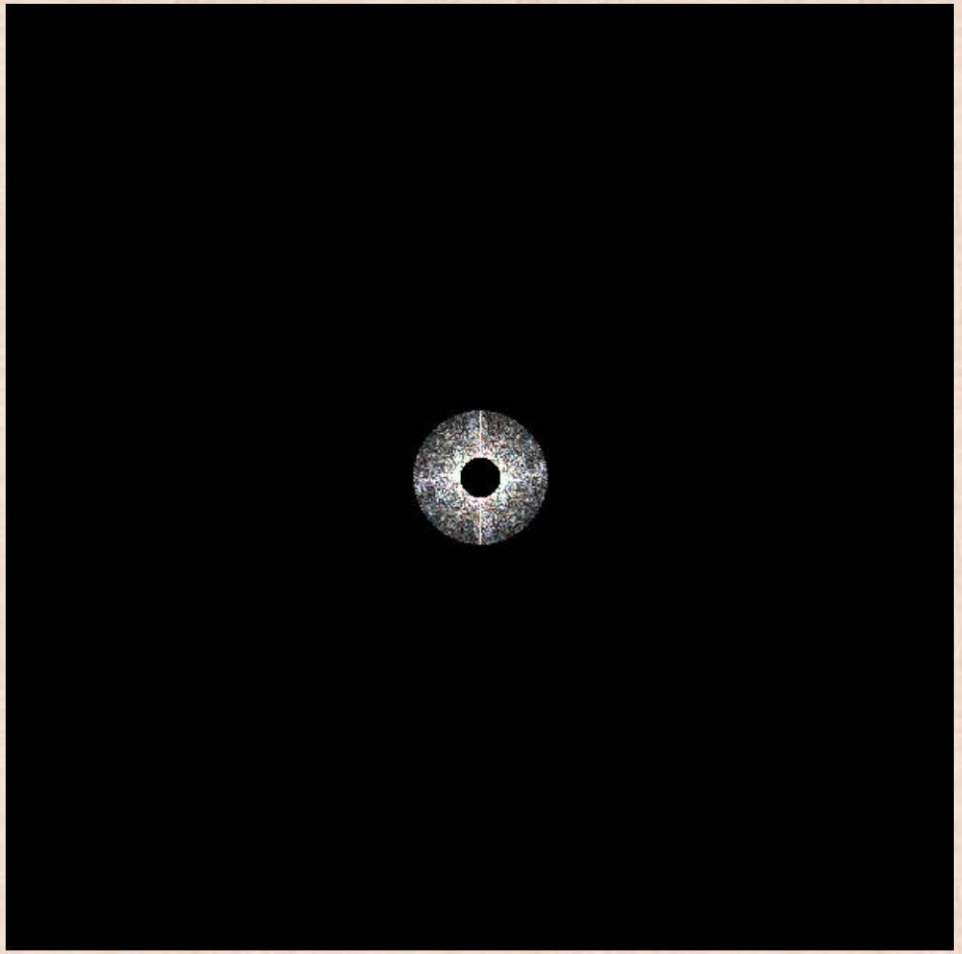




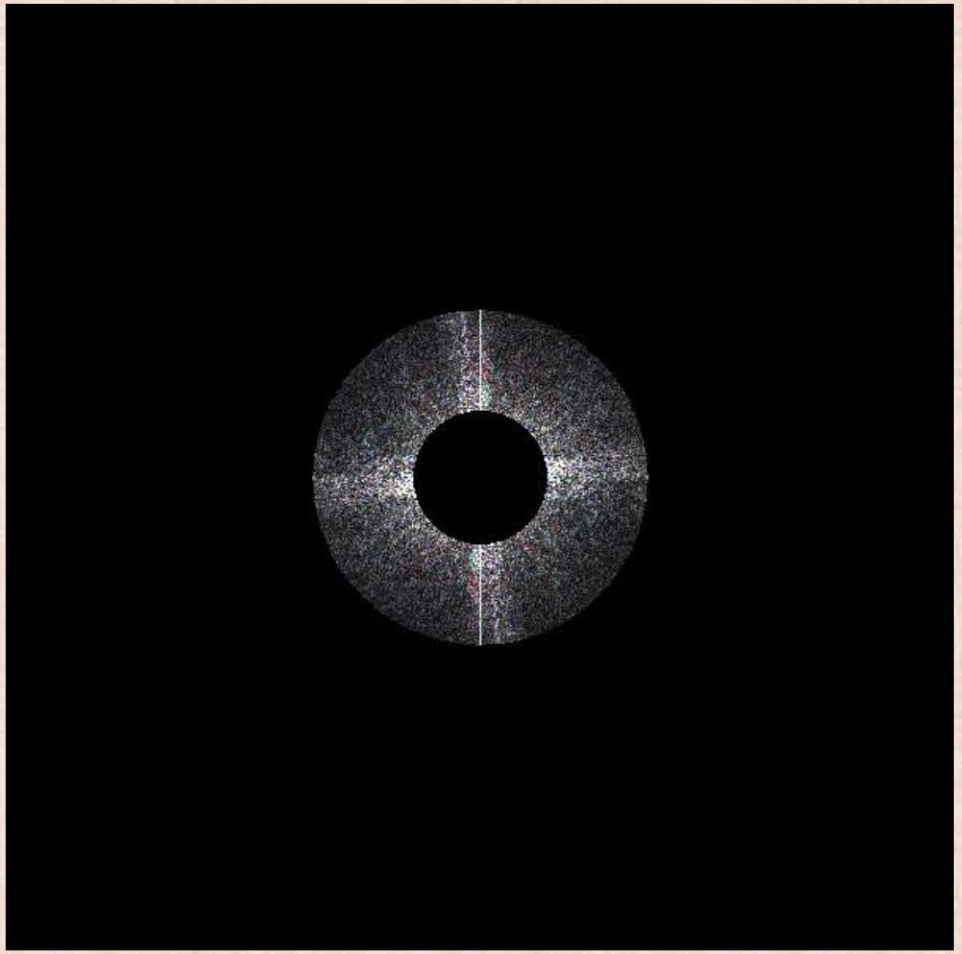


Intermediate Frequencies

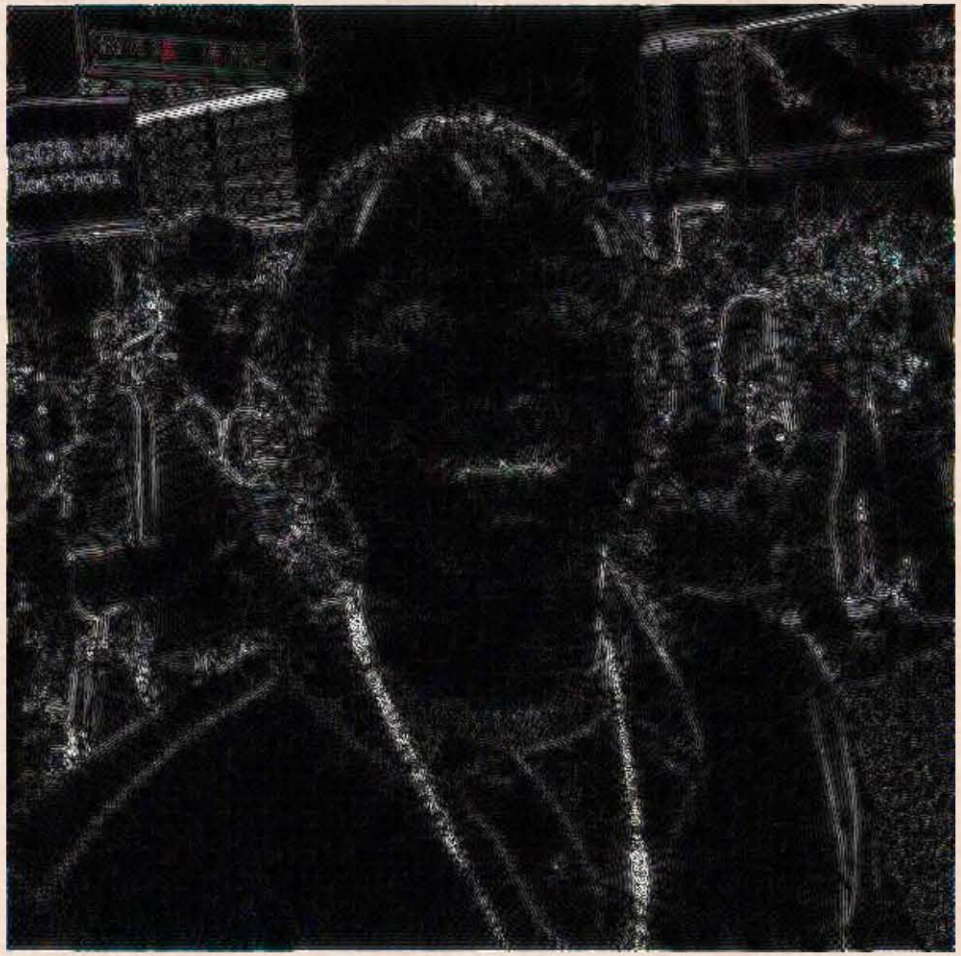


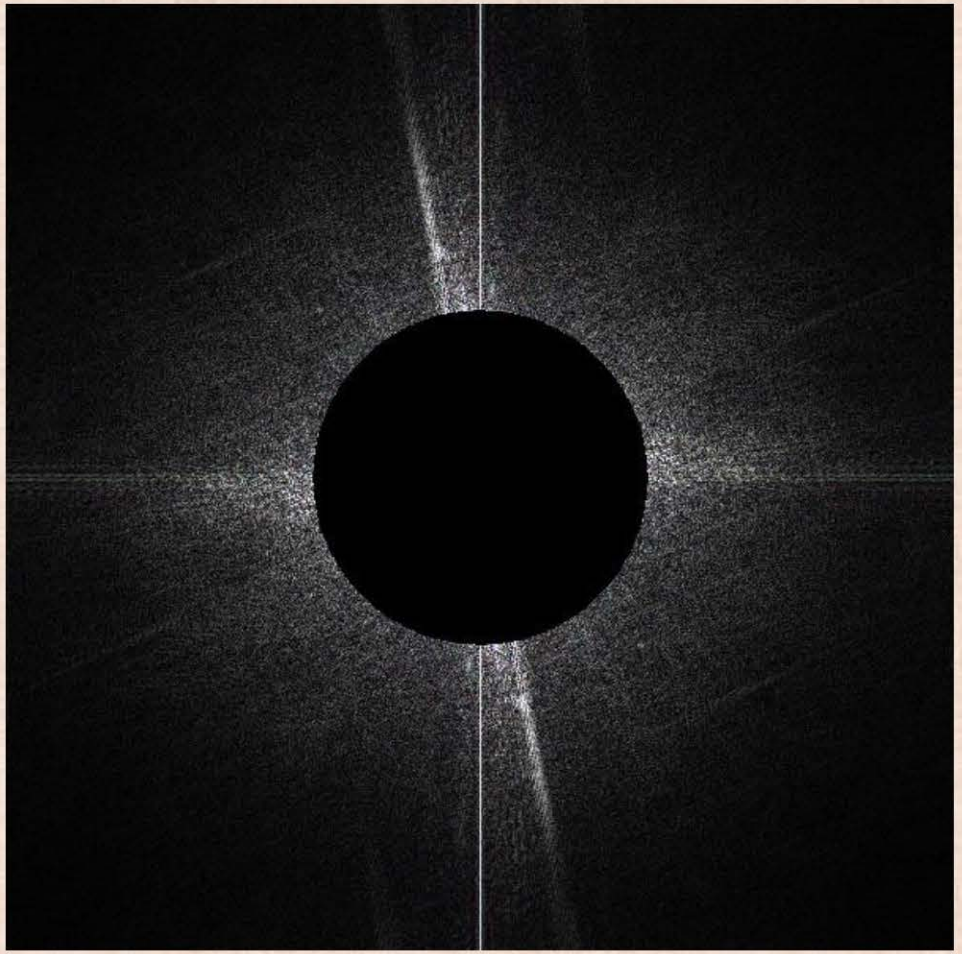


Larger Intermediate Frequencies



Highest Frequencies (edges!)





Convolution

- Let f and g be functions in the spatial domain (e.g. images), and $F(f)$ and $F(g)$ be transformations of these functions into the frequency domain
  - In our prior examples f was the image (on the left), and $F(f)$ was the frequency domain version of the image (on the right)

- Removing higher frequencies of $F(f)$ is equivalent to multiplying $F(f)$ by some Heaviside function $F(g)$ (equal to one for smaller frequencies and equal to zero for larger frequencies)
- Then, the inverse transform $F^{-1}(F(f)F(g))$ gives the final result

- This entire process is called the convolution of f and g:

$$f * g = F^{-1}(F(f)F(g))$$

Convolution

- Convolution can be done without the Fourier Transform as well:

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

- A narrow $F(g)$ removes the higher frequencies, but has a wide g
  - Recall: the narrower Gaussian had wider frequencies, and the wider Gaussian had narrower frequencies
- A wider g makes the integral (above) nonzero in a wider range, and thus expensive to compute
- For efficiency (in computing the integral), we prefer a narrower g, but that makes $F(g)$ too wide
  - A wider $F(g)$ fails to remove out some unwanted higher frequencies (that will alias)

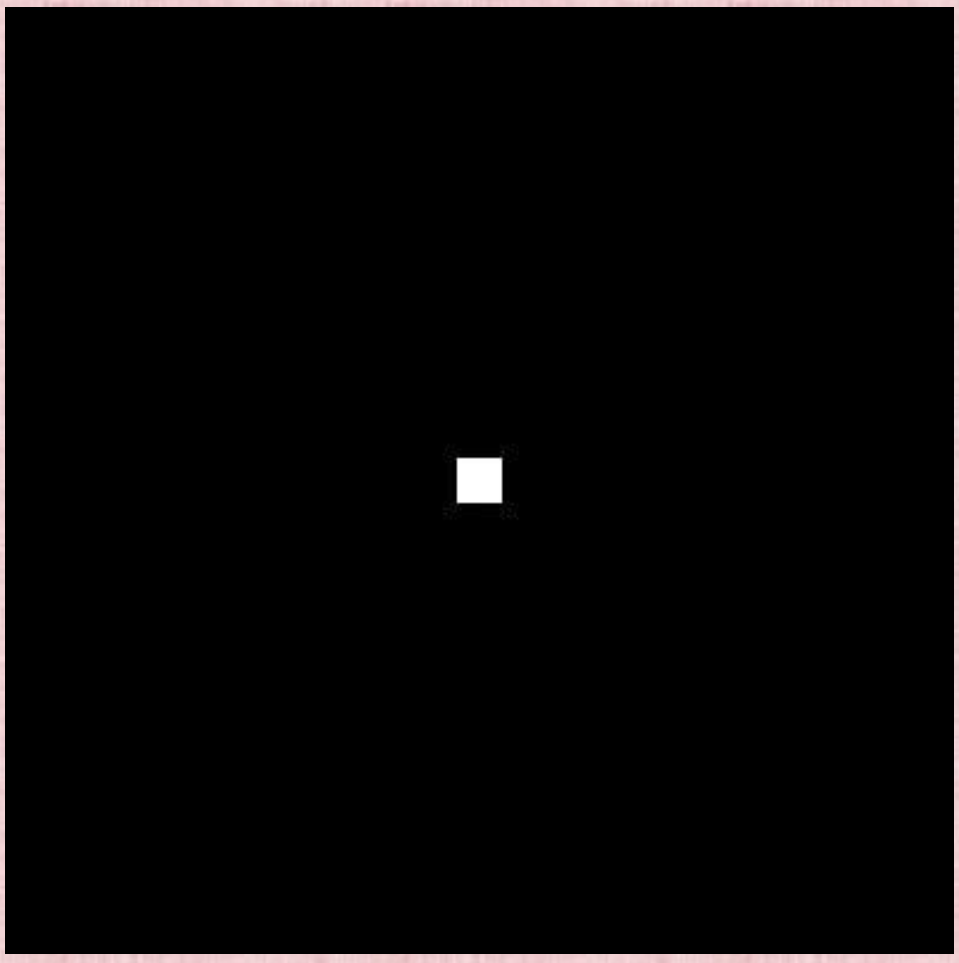
Box Filter

- Let g have nonzero values in an NxN block of pixels surrounding the origin, and be zero elsewhere
- The discrete convolution (integral) is performed by overlaying the filter on the image, multiplying corresponding entries, and summing
- The result is typically defined at the center of the filter

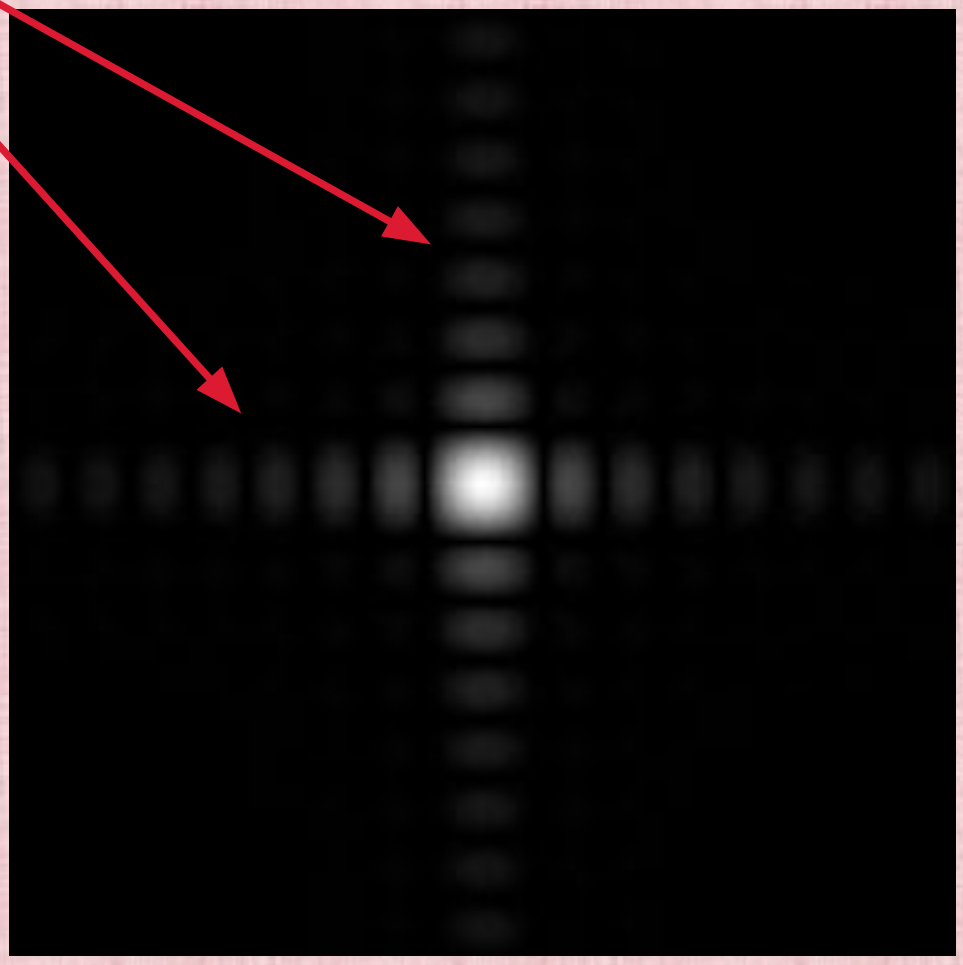
$1/16$	$1/16$	$1/16$	$1/16$
$1/16$	$1/16$	$1/16$	$1/16$
$1/16$	$1/16$	$1/16$	$1/16$
$1/16$	$1/16$	$1/16$	$1/16$

Filters Most (but not all) High Frequencies

g



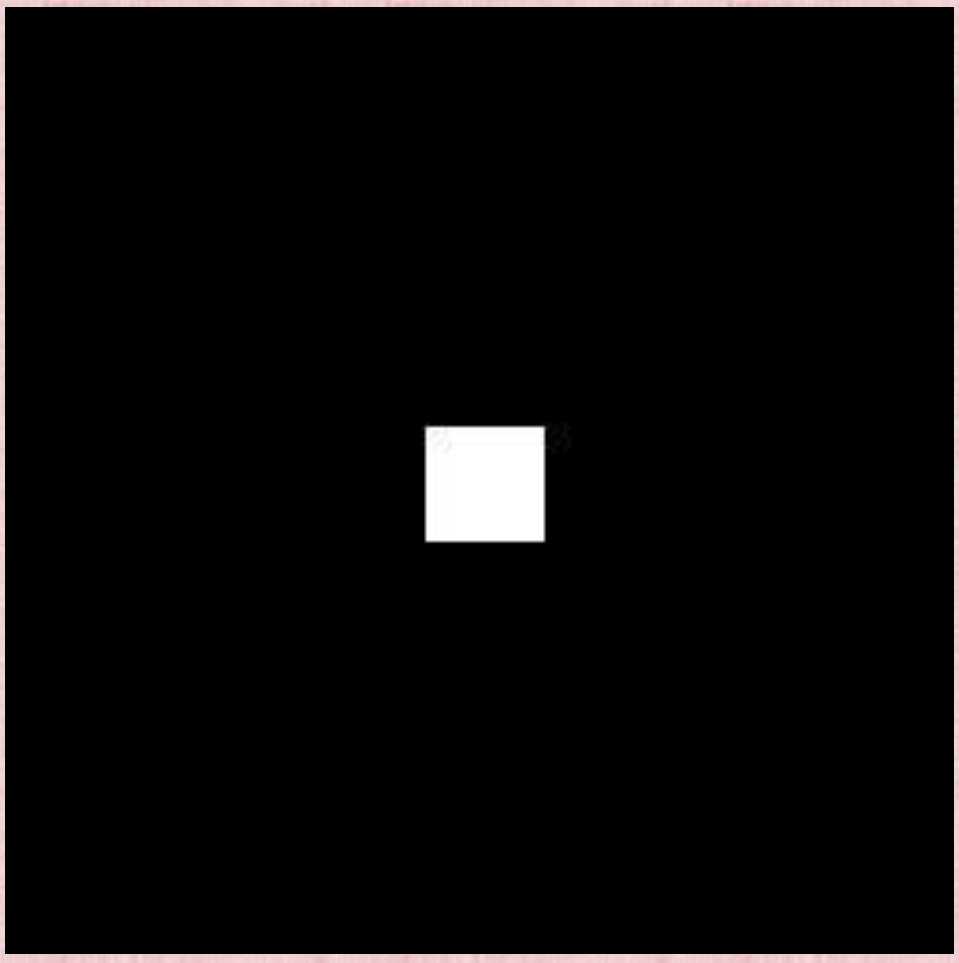
$F(g)$



narrower is computationally cheaper

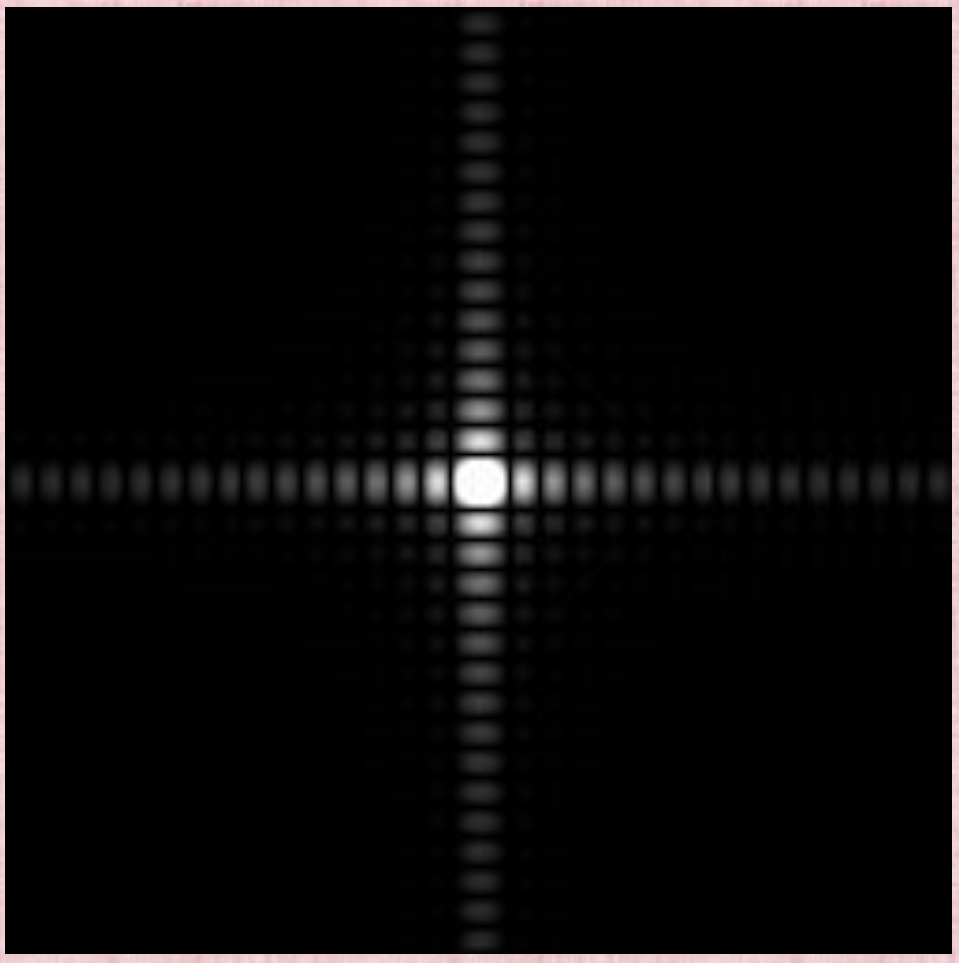
Wider Box Filters More High Frequencies

g



wider is computationally more expensive

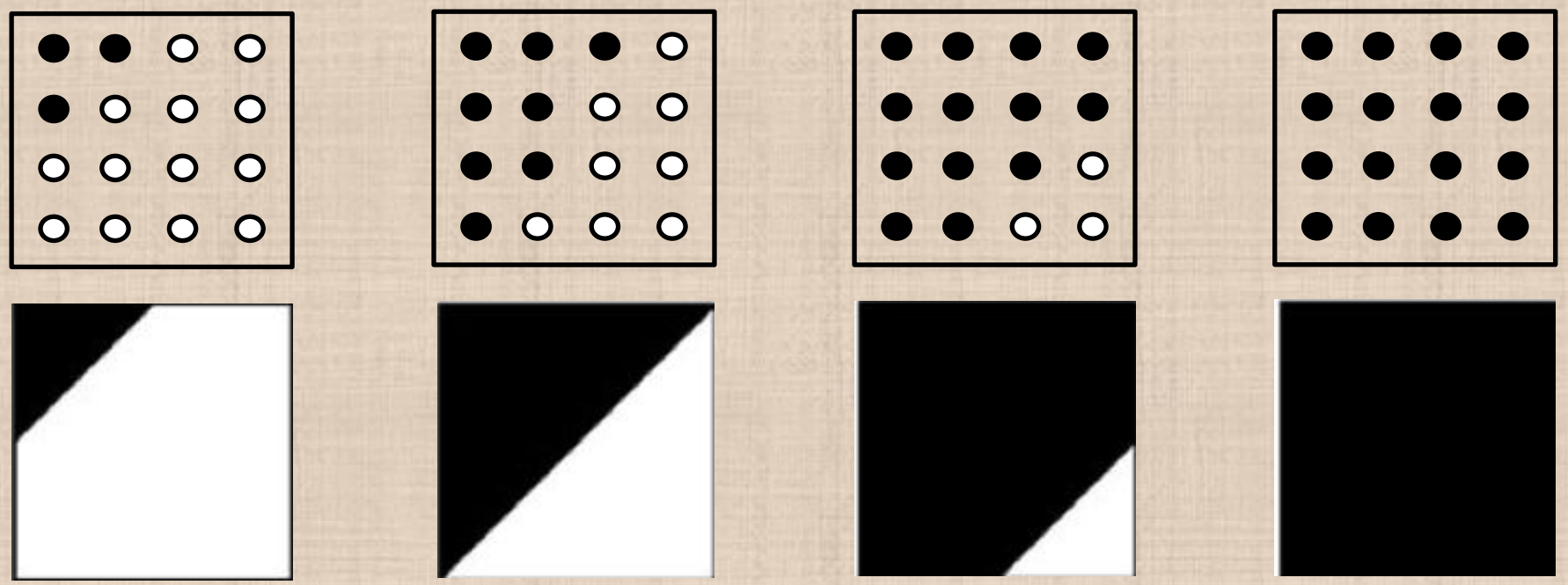
$F(g)$



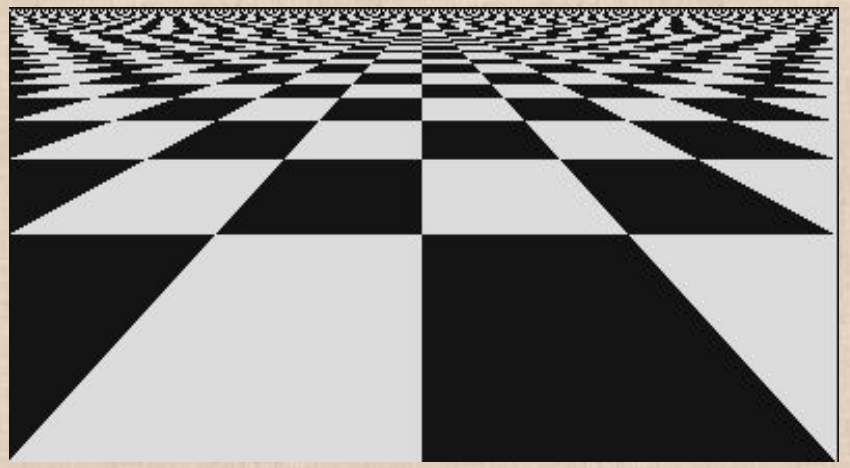
but removes more of the unwanted high frequencies

Super-Sampling

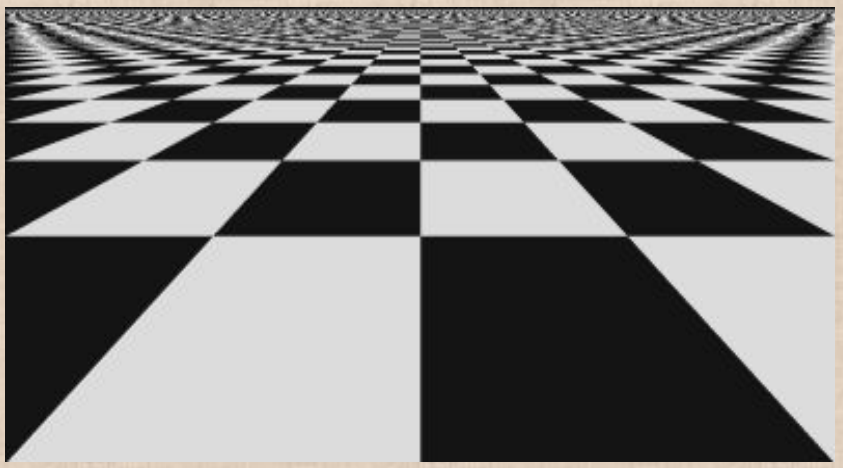
- Average multiple point samples distributed throughout a single pixel to obtain the value for that pixel
  - Rendering a 100 by 100 image using uniform 4 by 4 super-sampling is equivalent to rendering a 400 by 400 image, applying a 4 by 4 box filter 16 points at a time, and keeping only the (convolution) results
  - Storage: sampling at a higher resolution, but still only storing the same (lower) resolution image
  - Computational Cost: only super-sample pixels that have high frequencies (e.g., edges)
- NB: As the number samples per pixel approaches infinity, this approximates the area coverage integral (a “smart” random distribution is often used…)



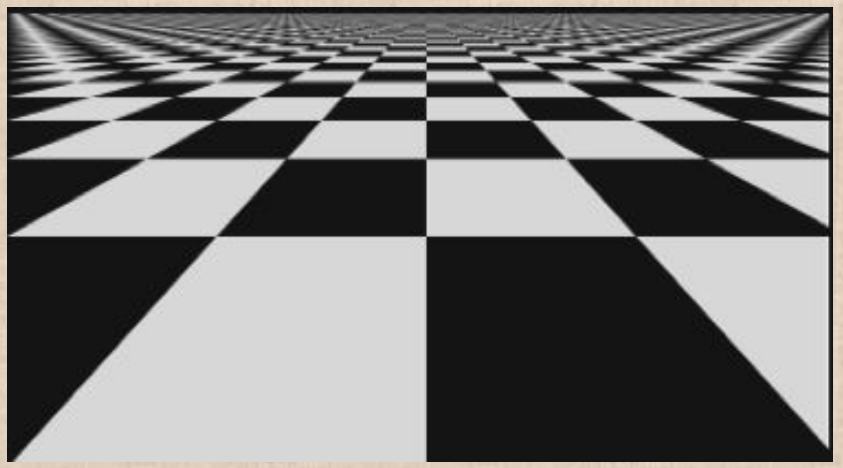
Super-Sampling



Point Sampling

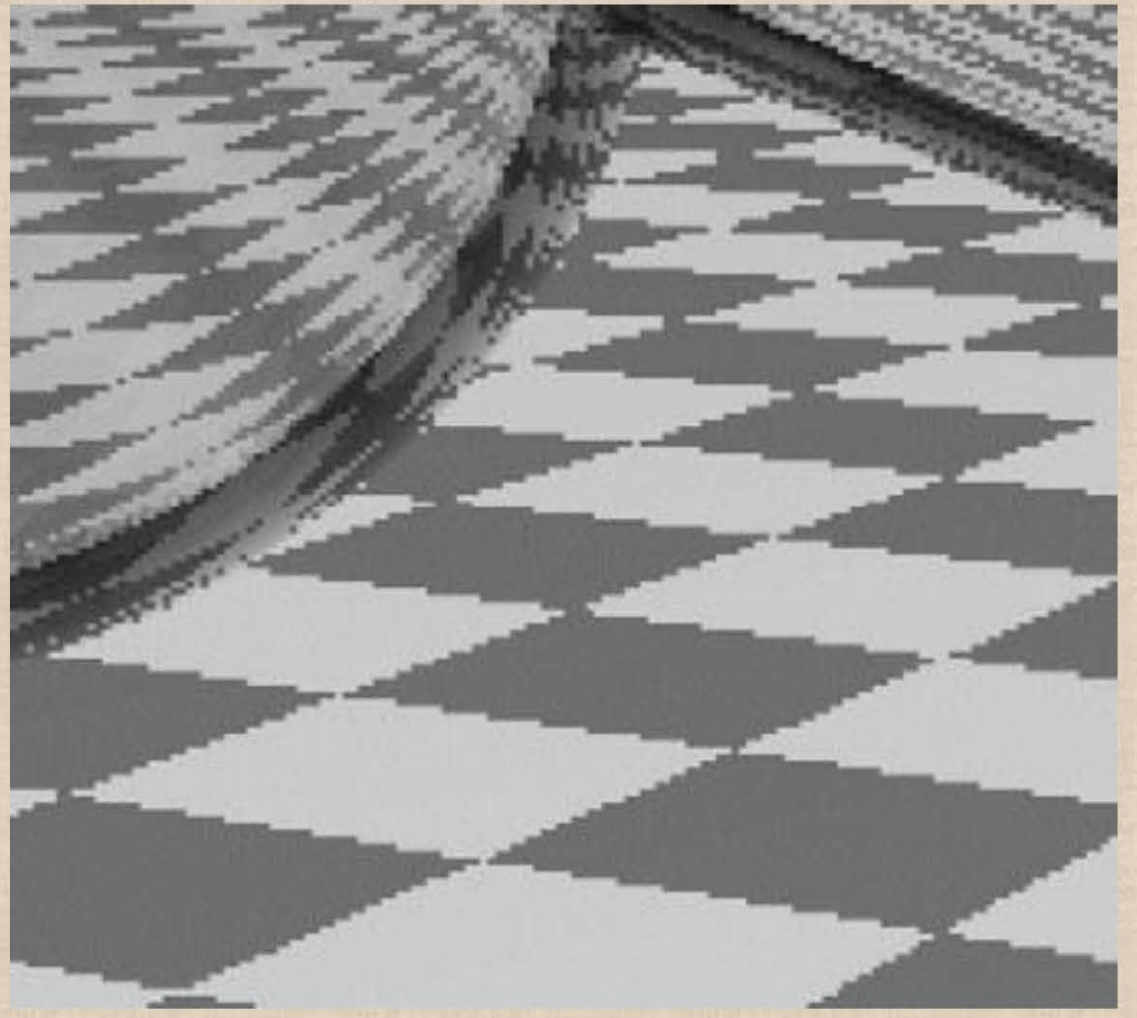


4 by 4 Supersampling

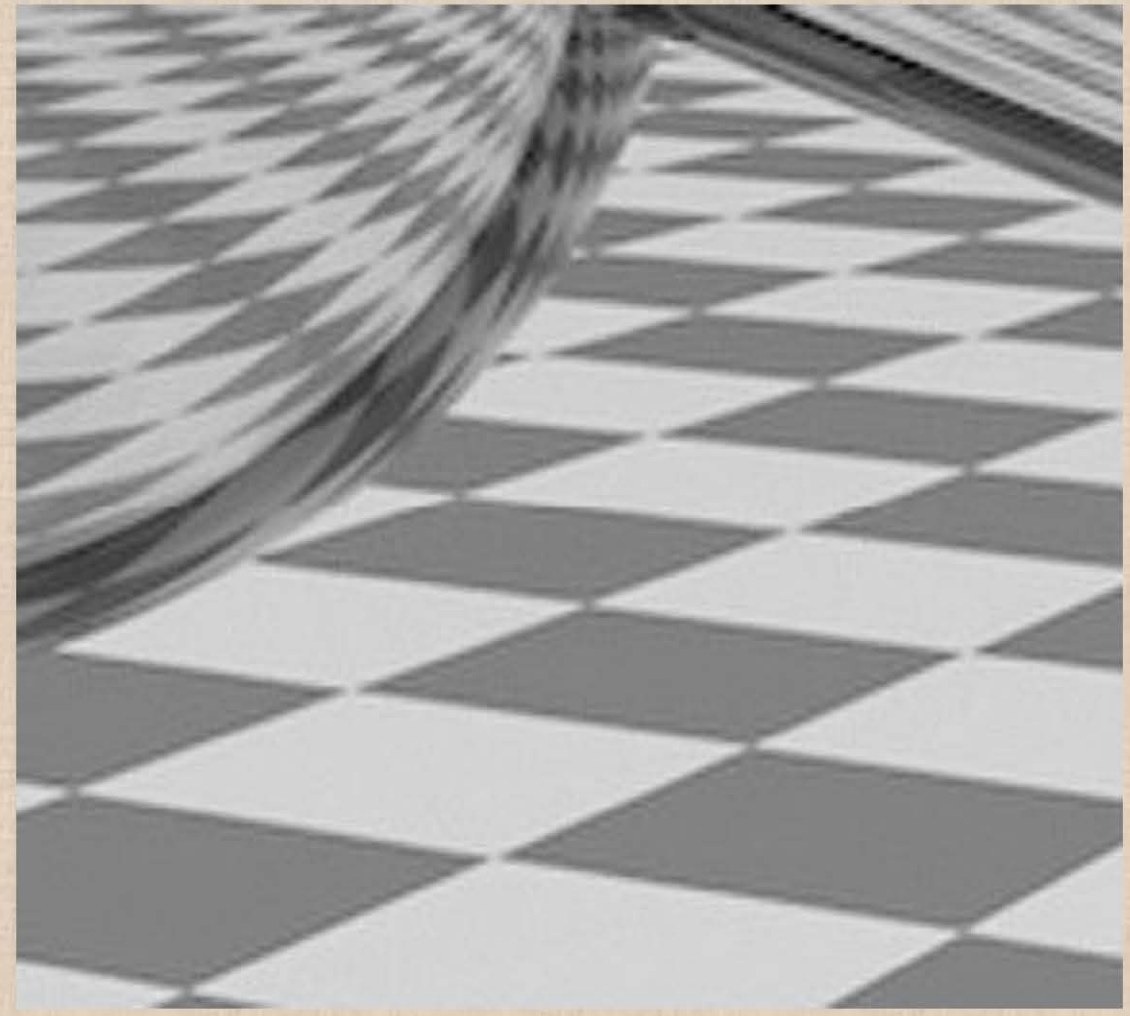


Exact Area Coverage

Super-Sampling



Jaggies



Anti-Aliased