CSE 152 Section 3
Review of Filters and Frequencies

October 19, 2018

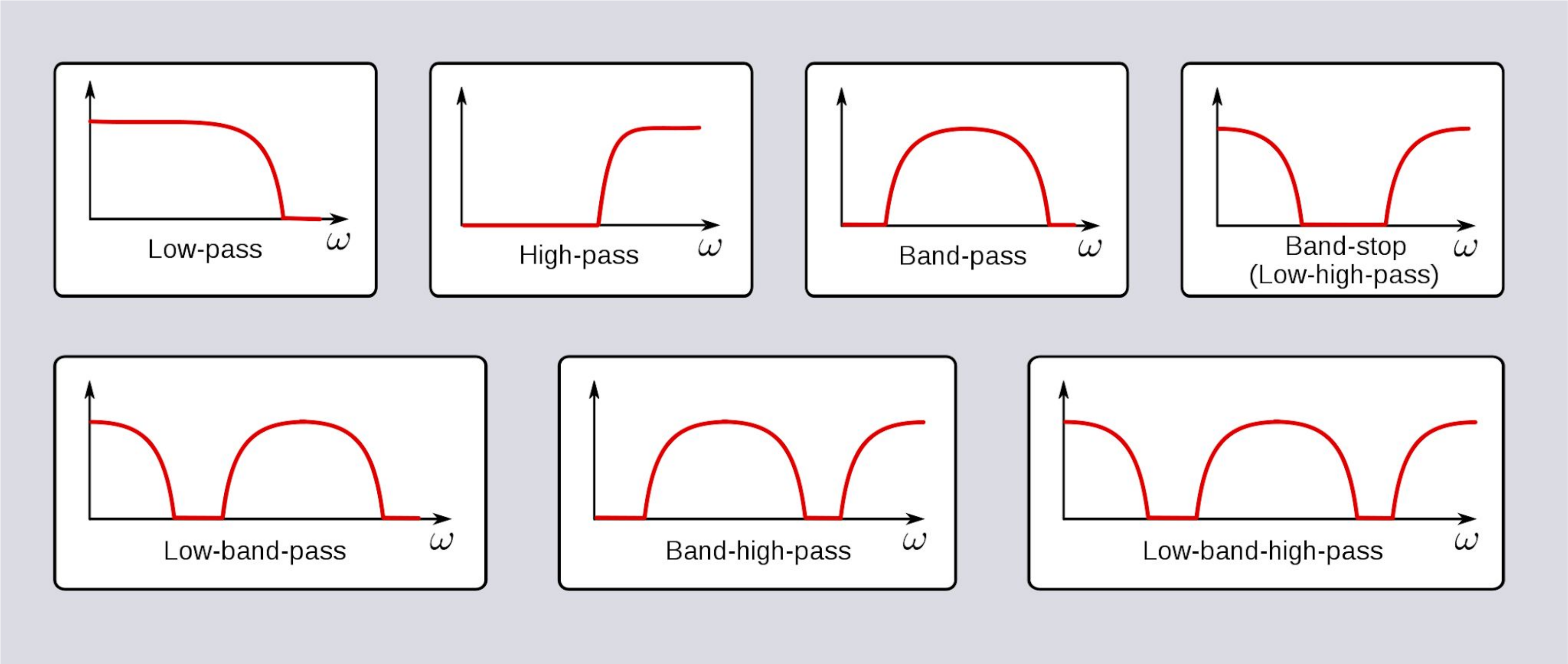
Owen Jow

What is a filter?

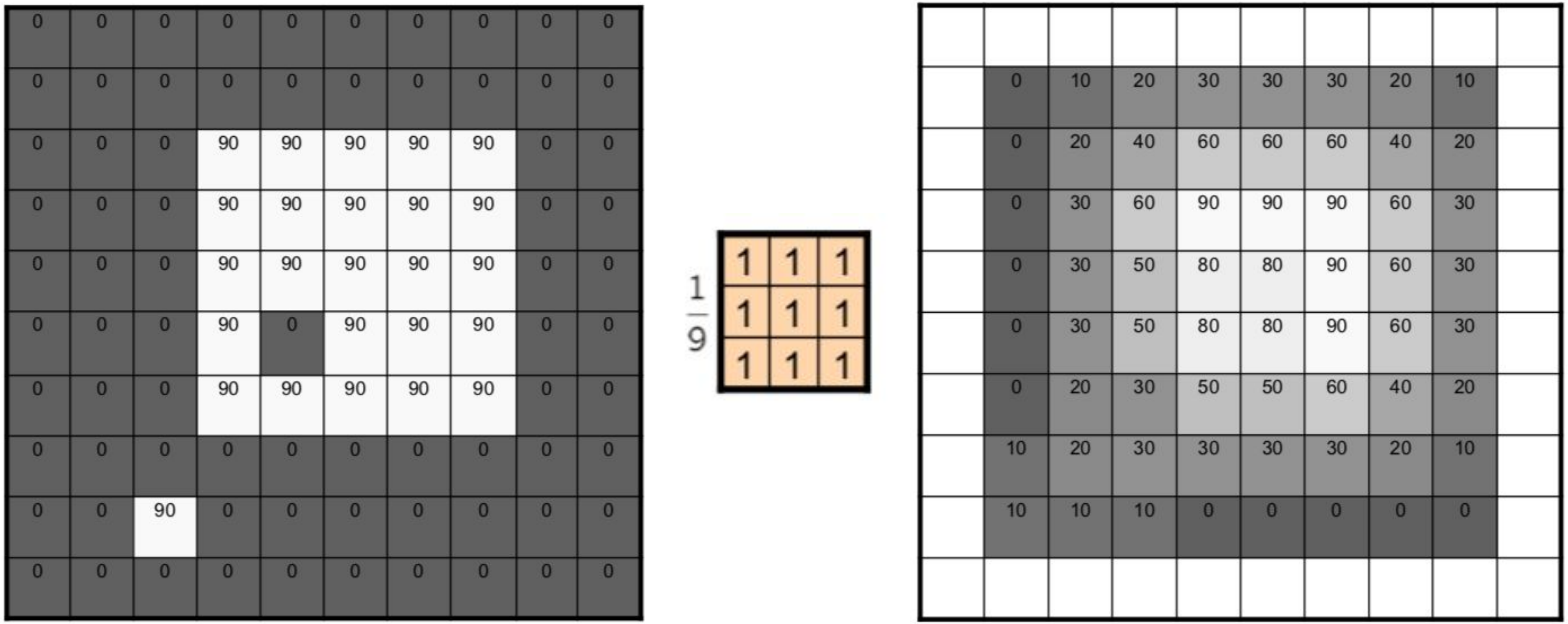
What comes to mind?

A device that lets only some of its inputs through

A process that lets only some of its inputs through



A process that lets only some [scalings] of its inputs through



source: Steven Seitz

and thereby transforms content

source: TapSmart

What does a linear filter do to an image?

Hint: why is it called a *linear* filter?

Linear Filter (An Image Processing View)

Replace each pixel with a **linear combination** of values in its neighborhood.

Linear Filter (An Image Processing View)

Replace each pixel with a **linear combination** of values in its neighborhood.

Correlation

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x+k, y+l)}_{\text{image}}$$

- not commutative or associative
- (preferably) use for measuring similarity

Linear Filter (An Image Processing View)

Replace each pixel with a **linear combination** of values in its neighborhood.

Correlation

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x+k,y+l)}_{\text{image}}$$

- not commutative or associative
- (preferably) use for measuring similarity

Convolution

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x-k,y-l)}_{\text{image}}$$

- commutative and associative
- **(preferably) use for filtering**

Linear Filter (An Image Processing View)

Replace each pixel with a **linear combination** of values in its neighborhood.

Correlation

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x+k,y+l)}_{\text{image}}$$

- not commutative or associative
- (preferably) use for measuring similarity

Convolution

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x-k,y-l)}_{\text{image}}$$

- commutative and associative
- (preferably) use for filtering
- **“flip filter horizontally and vertically”**

Linear Filter (An Image Processing View)

Replace each pixel with a **linear combination** of values in its neighborhood.

Correlation

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x+k, y+l)}_{\text{image}}$$

- not commutative or associative
- (preferably) use for measuring similarity

Convolution

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x-k, y-l)}_{\text{image}}$$

- commutative and associative
- (preferably) use for filtering
- "flip filter horizontally and vertically"
- **denoted h = f * I**

Linear Filter (An Image Processing View)

Correlation and **convolution** are both valid approaches to linear filtering.

Correlation

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x+k,y+l)}_{\text{image}}$$

- not commutative or associative
- (preferably) use for measuring similarity

Convolution

$$\underbrace{h(x,y)}_{\text{output}} = \sum_{k,l} \underbrace{f(k,l)}_{\text{filter}} \underbrace{I(x-k,y-l)}_{\text{image}}$$

- commutative and associative
- (preferably) use for filtering
- “flip filter horizontally and vertically”
- **denoted h = f * I**

Why do we care about associativity?

Why do we care about associativity?

Associativity means that $\mathbf{f * (g * I) = (f * g) * I}$*.*
If we want to apply multiple filters, we can pre-convolve them and use (/reuse) them as a single filter!

Properties of Linear Filters

- **They are linear.**

  $f * (\alpha I + J) = \alpha(f * I) + f * J$

  meaning they obey the superposition principle, shown here

Properties of Linear Filters

- **They are linear.**

  $f * (\alpha I + J) = \alpha(f * I) + f * J$

- **They are shift-invariant.**

  $f * \text{shifted}(I) = \text{shifted}(f * I)$

  “we can shift the image to the left by one pixel, then filter – or we can filter, then shift the result to the left by one pixel”

Properties of Linear Filters

- **They are linear.**

  `f * (αI + J) = α(f * I) + f * J`

- **They are shift-invariant.**

  `f * shifted(I) = shifted(f * I)`

  "we apply the same computation to all of the neighborhoods"

Filter Break

sharpening via
unsharp filtering

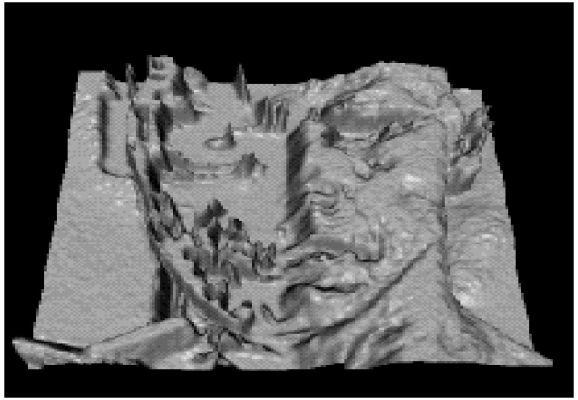
Filter Break

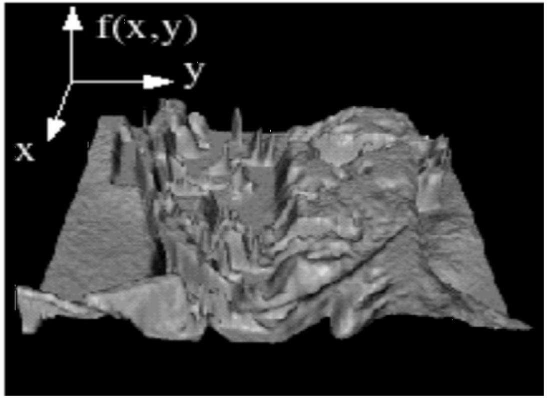
denoising via
median filtering

(nonlinear)

An image is a function f(x, y)

It is a mapping from pixel locations $\in \mathbb{R}^2$ to intensities $\in \mathbb{R}$.

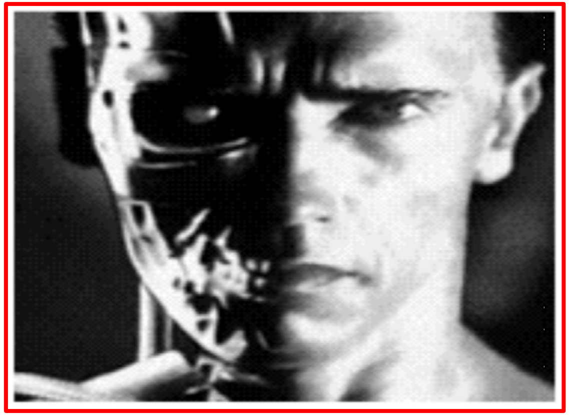


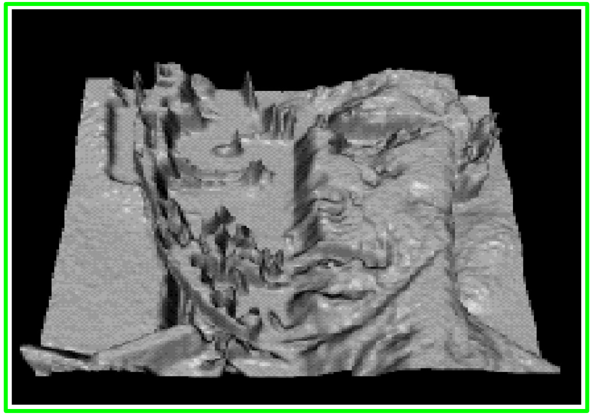


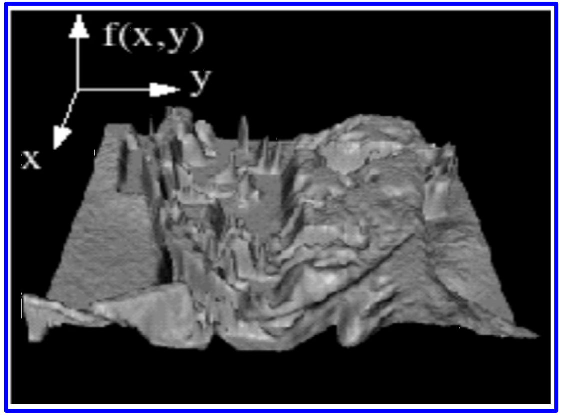
source: Seitz, Szeliski

An image is a function f(x, y)

A color image is a mapping from pixel locations $\in \mathbb{R}^2$ to RGB intensities $\in \mathbb{R}^3$.



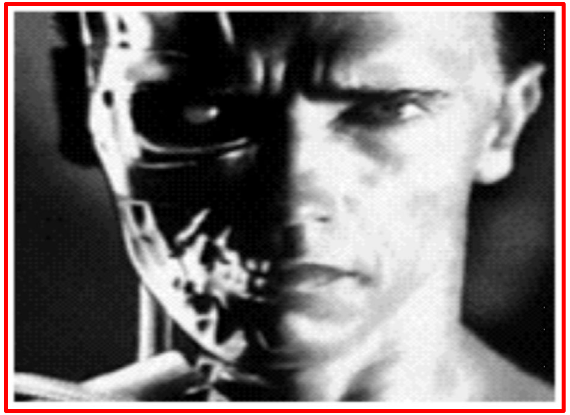


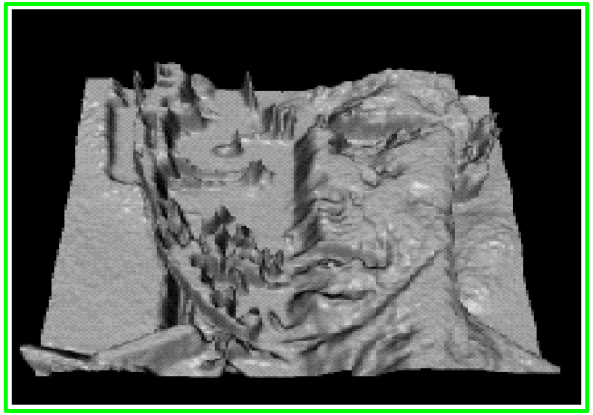


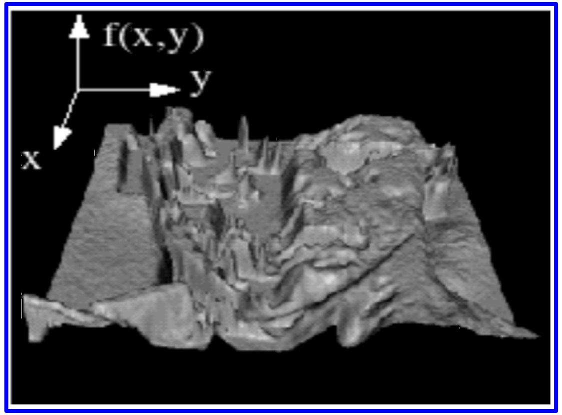
source: Seitz, Szeliski

An image is a function f(x, y)

In the case of digital images, we discretely sample an underlying continuous function.



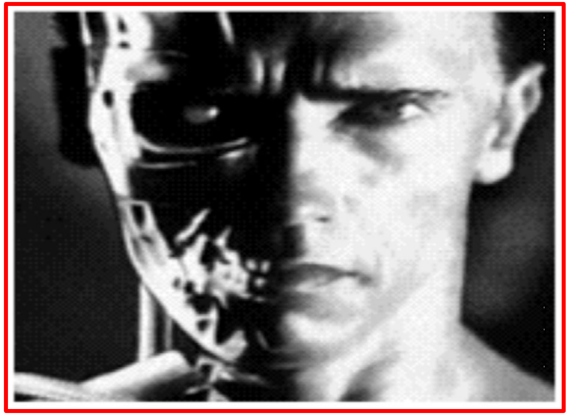


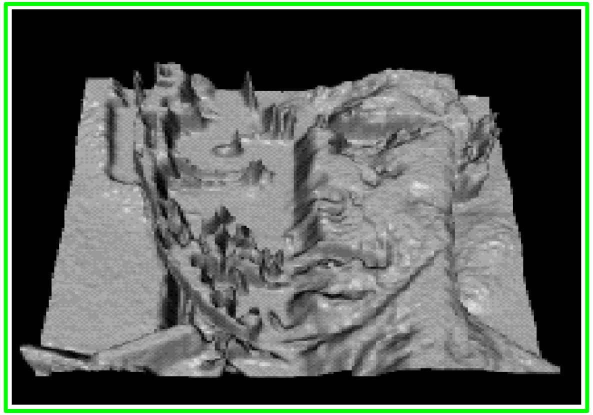


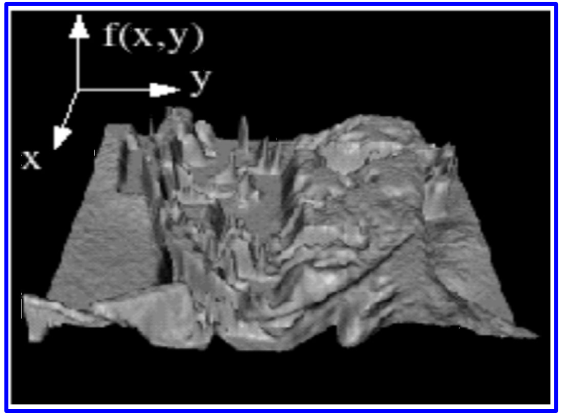
source: Seitz, Szeliski

An image is a signal f(x, y)

In the case of digital images, we discretely sample an underlying continuous function.







source: Seitz, Szeliski

So what we're really doing...

...is **signal processing**

image source: Daniel Sierra

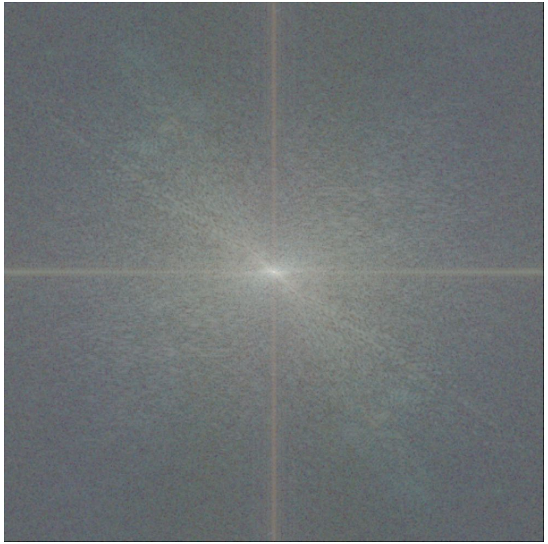
A digital image is a discrete 2D signal (function) (vector).

Traditionally, we think of them as they exist in the spatial domain.

A digital image is a discrete 2D signal (function) (vector).

Traditionally, we think of them as they exist in the spatial domain.

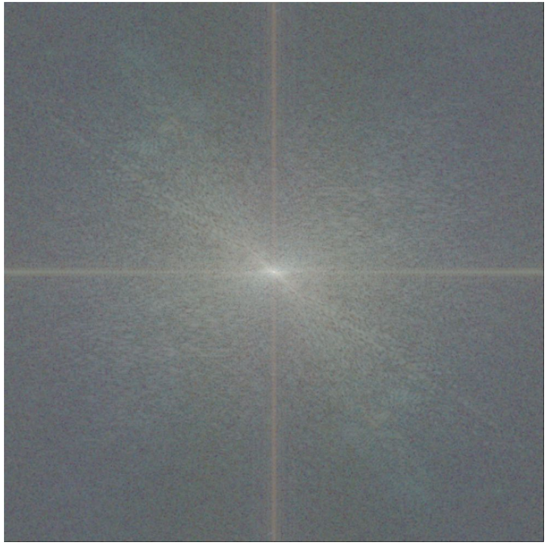
But signal processing gives us a new way to think about things...



A digital image is a discrete 2D signal (function) (vector).

Spatial Domain

Frequency Domain



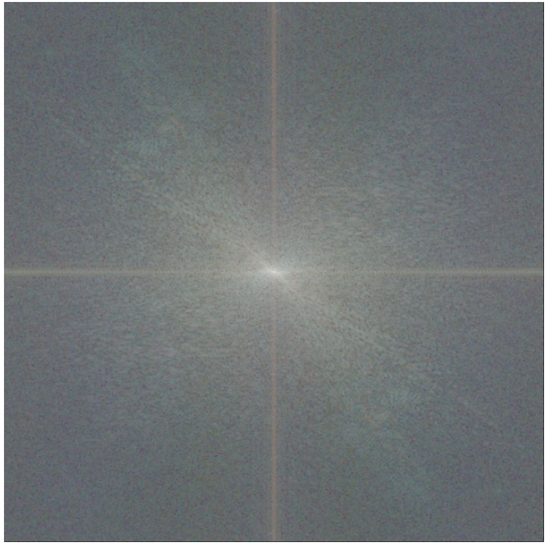
A digital image is a discrete 2D signal (function) (vector).

Spatial Domain

Frequency Domain

Fourier Transform

Inverse Fourier Transform



A digital image is a discrete 2D signal (function) (vector).

Spatial Domain

$$f(x, y)$$

x: distance (px) in horizontal direction
y: distance (px) in vertical direction

f(x, y): intensity at (x, y)

$$f(x, y) \underset{\text{Inverse Fourier Transform}}{\overset{\text{Fourier Transform}}{\rightleftarrows}} F(u, v)$$

Frequency Domain

$$F(u, v)$$

u: frequency (cycles/px) in horizontal direction
v: frequency (cycles/px) in vertical direction

F(u, v): magnitude of frequency (u, v)

1D case (scan line)

Spatial Domain

Frequency Domain

$$f(x) \underset{\text{Inverse Fourier Transform}}{\overset{\text{Fourier Transform}}{\rightleftarrows}} F(u)$$

x: distance (px) in horizontal direction
f(x): intensity at pixel x on scan line

u: frequency (cycles/px)
F(u): magnitude of frequency u

1D case (time-varying signal)

Spatial Domain | **Frequency Domain**

$$f(t) \underset{\text{Inverse Fourier Transform}}{\overset{\text{Fourier Transform}}{\rightleftarrows}} F(\omega)$$

(1D Discrete) Fourier Transform

A discrete Fourier transform (DFT) turns a function into a weighted sum of sines and cosines.

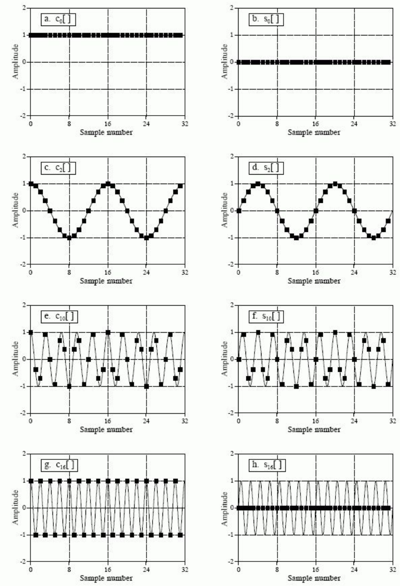


FIGURE 8-5
DFT basis functions. A 32 point DFT has 17 discrete cosine waves and 17 discrete sine waves for its basis functions. Eight of these are shown in this figure. These are discrete signals; the continuous lines are shown in these graphs only to help the reader's eye follow the waveforms.

(1D Discrete) Fourier Transform

A Fourier transform is a **change of basis** into a basis of sine and cosine functions.

$$\cos(2\pi kt/N)$$

$$\sin(2\pi kt/N)$$

If the signal contains **N** samples, the basis will contain **N** sine/cosine functions with different frequencies.

$$\begin{aligned}
F(k) &= \sum_{t=0}^{N-1} f(t) e^{(-2\pi kt/N)i} \\
&= \sum_{t=0}^{N-1} f(t) \left[\cos(2\pi kt/N) - i\sin(2\pi kt/N)\right] \\
&= \sum_{t=0}^{N-1} f(t) \cos(2\pi kt/N) - i \sum_{t=0}^{N-1} f(t) \sin(2\pi kt/N)
\end{aligned}$$

$$\boxed{0 \leq k \leq N-1}$$

(1D Discrete) Fourier Transform

F(k) is a complex number from which we can obtain the magnitude (amplitude) of frequency **k** in the Fourier decomposition.

We can think of the output of our Fourier transform as a **magnitude** for each **frequency**.

$$\cos(2\pi kt/N)$$

$$\sin(2\pi kt/N)$$

$$\begin{aligned} F(k) &= \sum_{t=0}^{N-1} f(t) e^{(-2\pi kt/N)i} \\ &= \sum_{t=0}^{N-1} f(t) \left[\cos(2\pi kt/N) - i \sin(2\pi kt/N)\right] \\ &= \sum_{t=0}^{N-1} f(t) \cos(2\pi kt/N) - i \sum_{t=0}^{N-1} f(t) \sin(2\pi kt/N) \end{aligned}$$

$$\boxed{0 \leq k \leq N-1}$$

Incidentally

$$A\sin(2\pi kt + \varphi)$$

A: amplitude, magnitude, strength, “how much”
k, 2πk: frequency, cycles per pixel or second
φ: phase, shift, “where” the sinusoid is

Incidentally

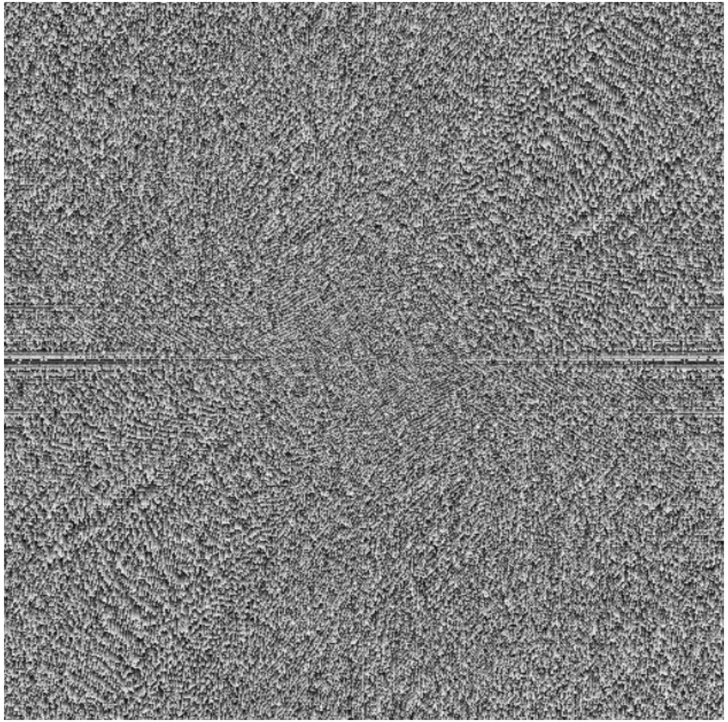
$$A\sin(\omega t) + B\cos(\omega t) = C\sin(\omega t + \varphi)$$

adding a sine and cosine of the same frequency
gives a phase-shifted sine of that frequency

total amplitude is sqrt(A^2 + B^2)
phase shift is arctan(A / B)

(1D Discrete) Fourier Transform

We can also get phase information out of a Fourier transform. But we won't talk about phase much because it isn't very helpful for interpretability.



In Summary: The 1D Fourier Transform

converts a signal `f(t)` into the frequencies that compose it.

$$F(\omega)$$

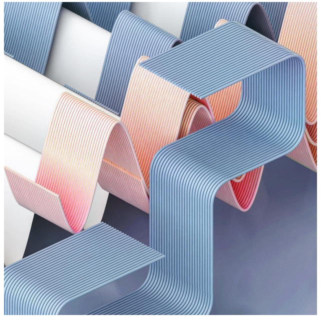
"*what is the strength of the frequency-ω sinusoid in the decomposition of* `f(t)`?"

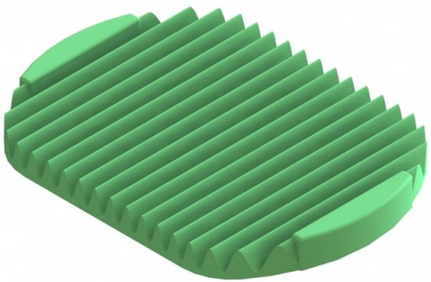
2D DFT

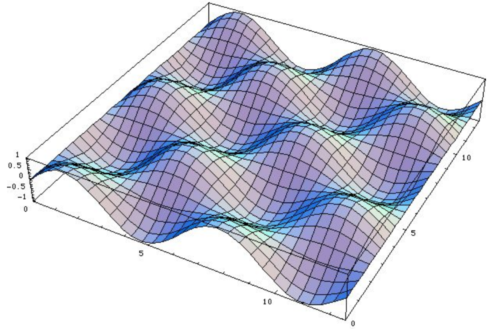
- The 2D DFT is analogous to the 1D DFT; just add another dimension to the input.

$$F(u, v)$$

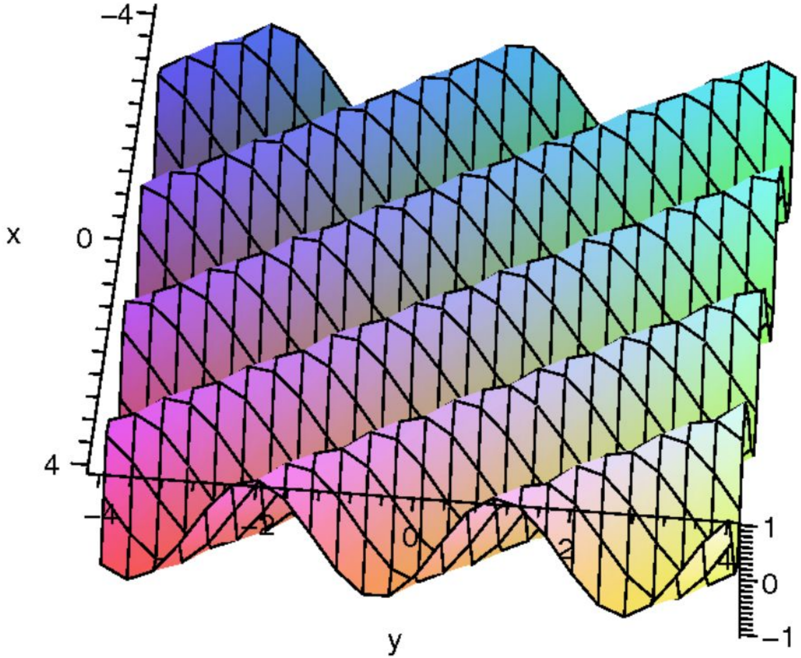
- The main difference is that the sines/cosines can now be *oriented* in 2D.
The orientation depends on the values of u and v passed in. If you draw out the vector (u, v), its direction is the sinusoid's orientation and its length *corresponds* to the sinusoid's frequency.

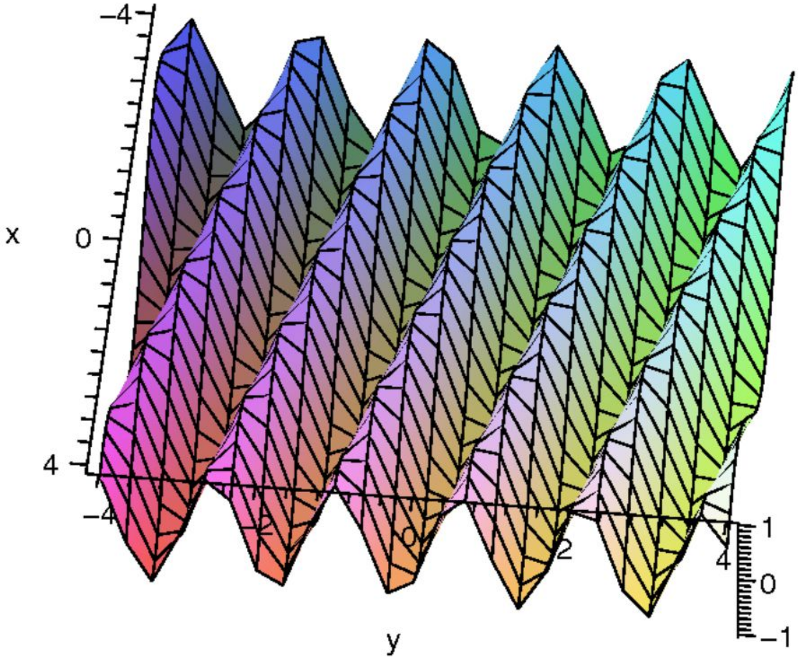






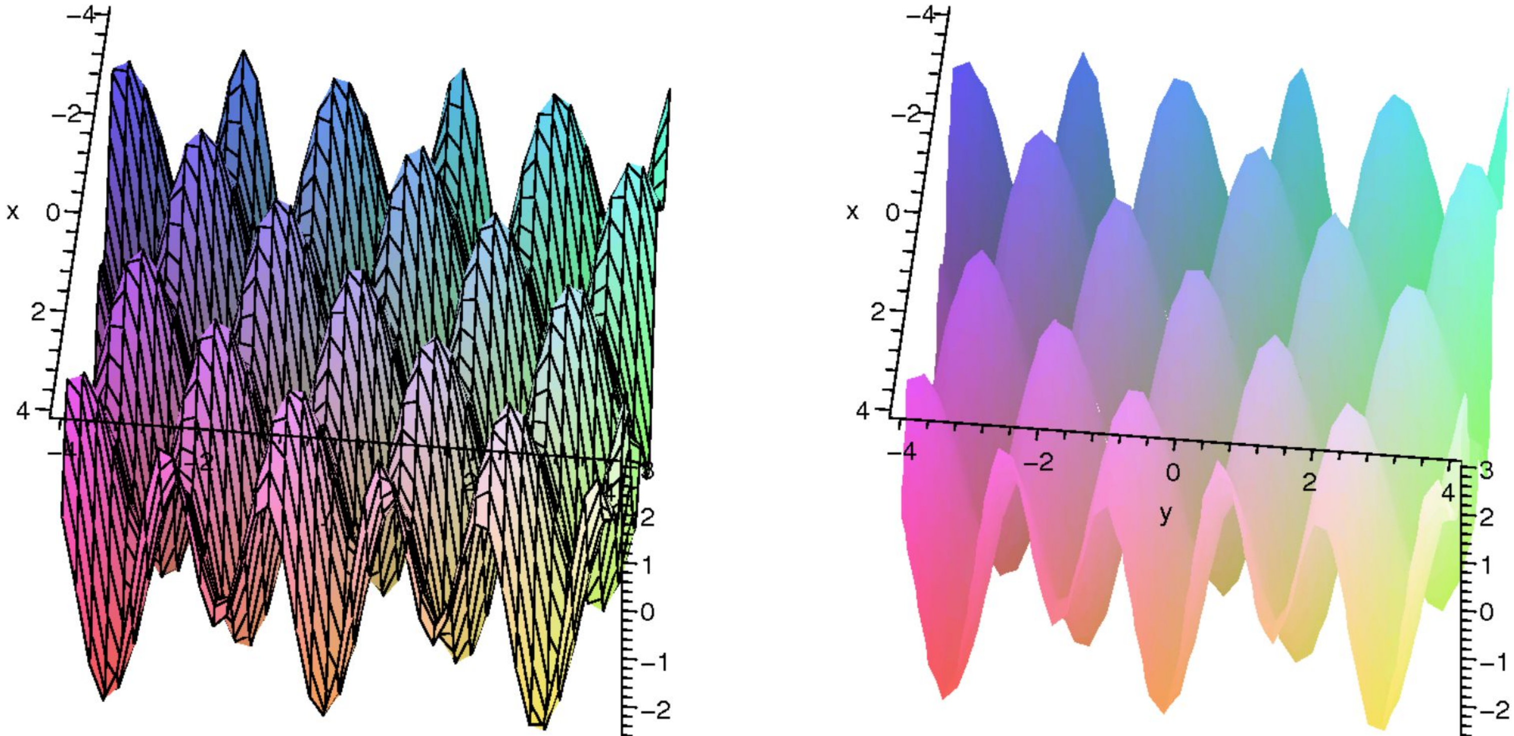
2D Basis Functions





source: Václav Hlaváč

Sum of 2D Basis Functions



source: Václav Hlaváč

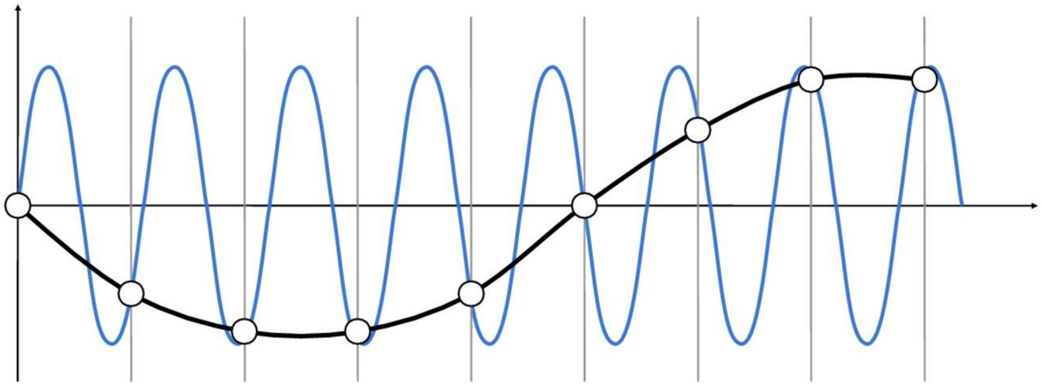
Cycles per pixel?

Cycles per pixel?

We are thinking about **spatial frequency**. The basis sinusoids appear as oriented, repeating stripes. The number of pixels it takes to move along a sinusoid from some intensity back to the same intensity is 1 / (the frequency).

Nyquist frequency

To avoid aliasing, the maximum frequency (*bandwidth*) we can have in a signal is ½ of the number of samples.



Aliasing

source: Ren Ng

Nyquist frequency

To avoid aliasing, the maximum frequency (*bandwidth*) we can have in a signal is ½ of the number of samples.

In other words, the maximum frequency we can have in an image is 0.5 cycles per pixel. What does this mean?

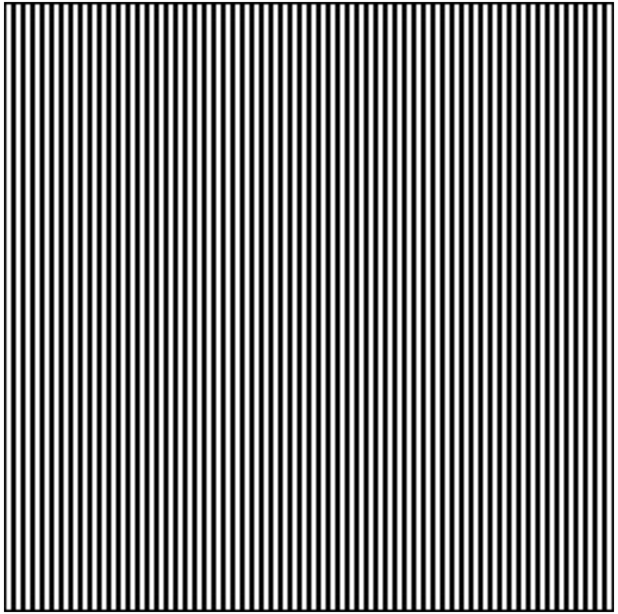
Nyquist frequency

To avoid aliasing, the maximum frequency (*bandwidth*) we can have in a signal is ½ of the number of samples.

In other words, the maximum frequency we can have in an image is 0.5 cycles per pixel. What does this mean?

at max 1 stripe per pixel extent
0.5 cycles per pixel → intensity alternates between low and high every pixel

Half of the Nyquist frequency



- stripe width 2px
- period 4px
- frequency 0.25 cycles/px

High vs. low frequencies

High frequency means a signal is changing quickly over its domain.

- In the previous visualization, the pixel values were changing very quickly from left to right, and the frequency was almost at its maximum (the Nyquist frequency).

In images,

- **high frequencies correspond to...**

?

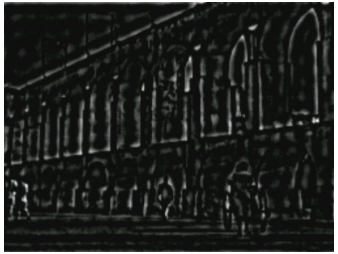
High vs. low frequencies

High frequency means a signal is changing quickly over its domain.

- In the previous visualization, the pixel values were changing very quickly from left to right, and the frequency was almost at its maximum (the Nyquist frequency).

In images,

- high frequencies correspond to **rapid/sharp changes in intensity (edges)**



High vs. low frequencies

High frequency means a signal is changing quickly over its domain.

- In the previous visualization, the pixel values were changing very quickly from left to right, and the frequency was almost at its maximum (the Nyquist frequency).

In images,

- high frequencies correspond to **rapid/sharp changes in intensity (edges)**
- **low frequencies correspond to...**

?

High vs. low frequencies

High frequency means a signal is changing quickly over its domain.

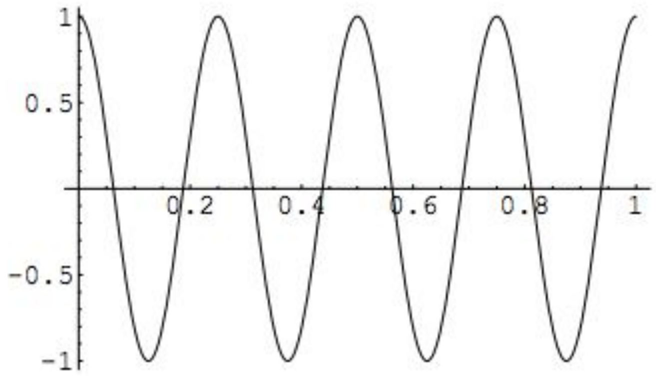
- In the previous visualization, the pixel values were changing very quickly from left to right, and the frequency was almost at its maximum (the Nyquist frequency).

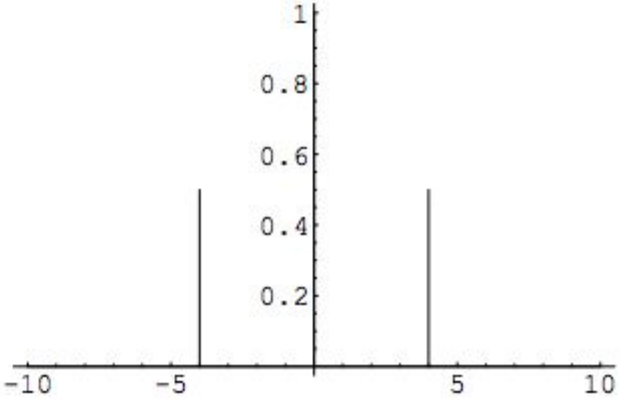
In images,

- high frequencies correspond to rapid/sharp changes in intensity (edges)
- low frequencies correspond to **smooth/slow changes in intensity (blurred/smoothed regions)**

Frequency Domain Images (1D) `understanding the frequency domain`

For a 1D signal, we visualize the frequency domain as a 2D plot
of **frequency** on the horizontal axis and **magnitude** on the vertical axis.

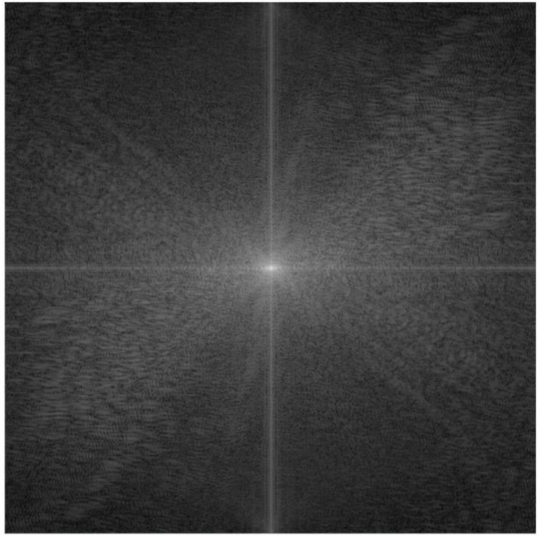




source: Bryan Morse

Frequency Domain Images (2D) `understanding the frequency domain`

For a 2D signal, we can think of the frequency domain as a 3D plot with **oriented frequency** on the xy-plane and **magnitude** on the z-axis.

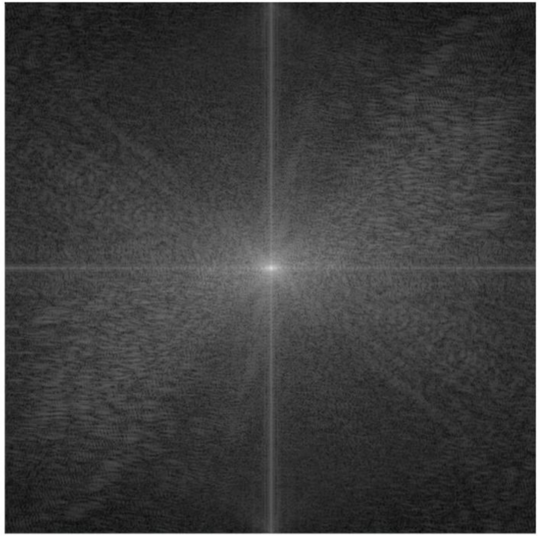


unlike a standard image,
the origin of the plot is in the center

source: Ren Ng

Frequency Domain Images (2D) `understanding the frequency domain`

However, we pretty much always view magnitude as brightness
(i.e. as part of an image), instead of plotting it on a z-axis in 3D.

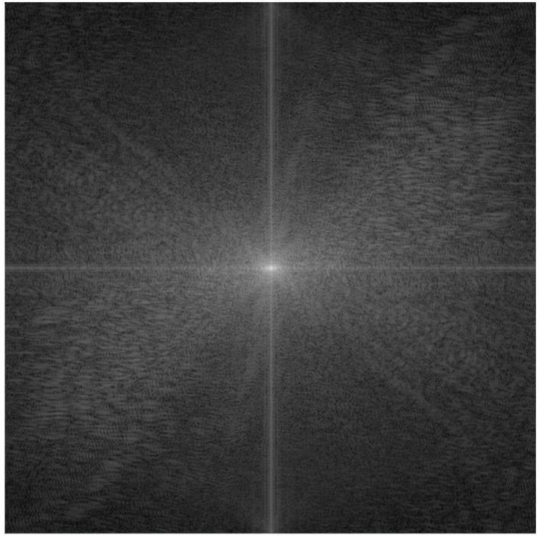


higher magnitude
means **higher brightness**
means **closer to white**

source: Ren Ng

Frequency Domain Images (2D) `understanding the frequency domain`

The spatial domain and frequency domain images are the same size, i.e. the number of frequencies is equivalent to the number of pixels.

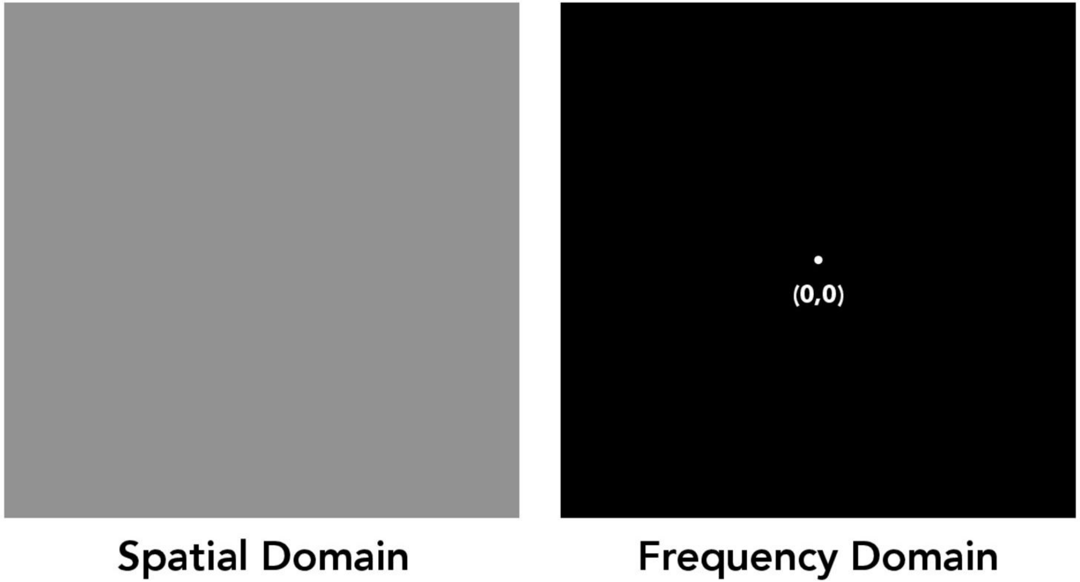


note that frequencies always range from -0.5 to 0.5, so what changes is the step between successive frequencies

source: Ren Ng

Frequency Domain Images (2D) `understanding the frequency domain`

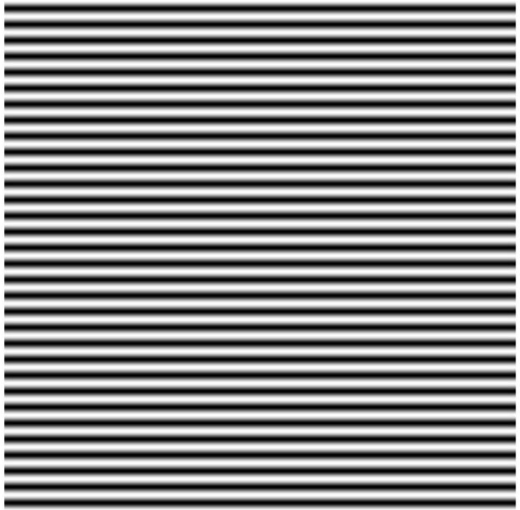
zero frequency (constant; average value in image)



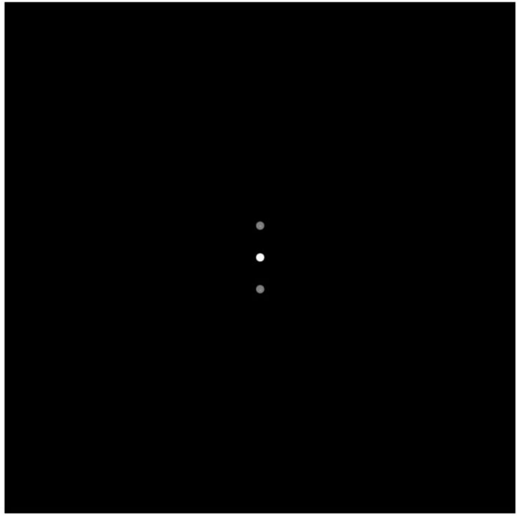
source: Ren Ng

Frequency Domain Images (2D) `understanding the frequency domain`

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



Spatial Domain



Frequency Domain

source: Ren Ng

Interpreting Frequency Visualizations

Let’s say we’re interested in the point **(u, v)**.

- Draw an arrow from the origin to **(u, v)**. This is a vector.

There are three pieces of information we can obtain.

1. The direction of vector **(u, v)** gives the **direction of the sinusoid**.

Interpreting Frequency Visualizations

Let's say we're interested in the point **(u, v)**.

- Draw an arrow from the origin to **(u, v)**. This is a vector.

There are three pieces of information we can obtain.

1. The direction of vector **(u, v)** gives the **direction of the sinusoid**.
2. The length of vector **(u, v)** gives (note: is not directly equal to) the **frequency of the sinusoid**.

Interpreting Frequency Visualizations

Let’s say we’re interested in the point **(u, v)**.

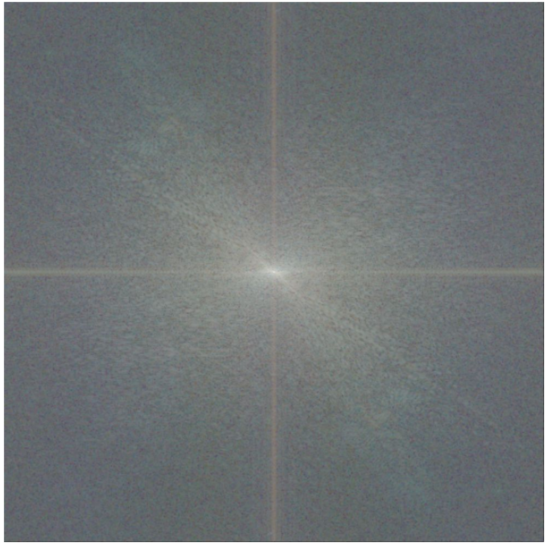
- Draw an arrow from the origin to **(u, v)**. This is a vector.

There are three pieces of information we can obtain.

1. The direction of vector **(u, v)** gives the **direction of the sinusoid**.
2. The length of vector **(u, v)** gives (note: is not directly equal to) the **frequency of the sinusoid**.
3. The brightness at point **(u, v)** gives the **magnitude of the sinusoid** (contrast from low to high).

The + Artifact

The DFT does its computation for an image that is tiled infinitely, meaning we (usually) end up with high-frequency edges where the top/bottom and left/right of the tiled images meet.



Convolution Theorem

Convolution in the spatial domain is equivalent to point-by-point multiplication in the frequency domain.

and what do we use convolution for?

Convolution Theorem

Convolution in the spatial domain is equivalent to point-by-point multiplication in the frequency domain.

and what do we use convolution for?

FILTERING

Convolution Theorem

Convolution in the spatial domain is equivalent to point-by-point multiplication in the frequency domain.

and what do we use convolution for?

FILTERING

So we have a choice.

- we can filter by sliding window in the spatial domain `(convolve)`

  - or -

- we can filter by multiplication in the frequency domain `(fftconvolve)`
  (multiply frequency version of the image by the frequency version of the convolution filter)

Why would we want to filter in the frequency domain?

Efficiency.

Note: we don't get the element-wise multiplication completely for free.
We also have to perform Fourier transforms ***to*** *and* ***from*** *the frequency domain.*

For large arrays, it's faster to go to the frequency domain and filter there.
For smaller arrays, it's faster to stay in the spatial domain and do a gridded convolution.

Why would we want to filter in the frequency domain?

Interpretability.

We can look at convolution as an operation over a grid of numbers,
or as modifying the frequencies of an image. It is often intuitive to think in terms of frequencies.

Why would we want to filter in the frequency domain?

Interpretability.

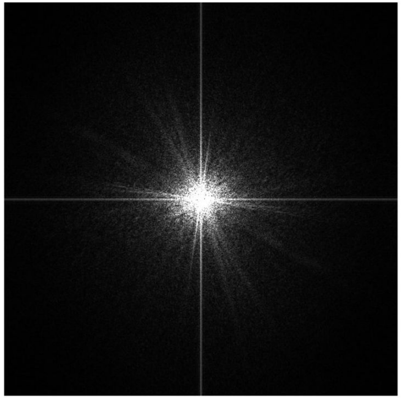
We can look at convolution as an operation over a grid of numbers,
or as modifying the frequencies of an image. It is often intuitive to think in terms of frequencies.

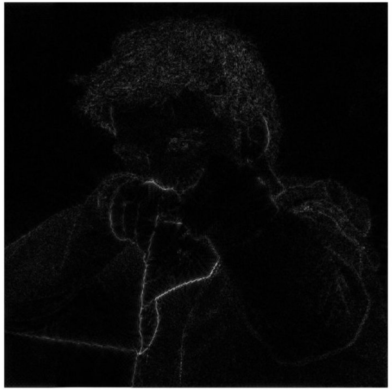
source: Ren Ng

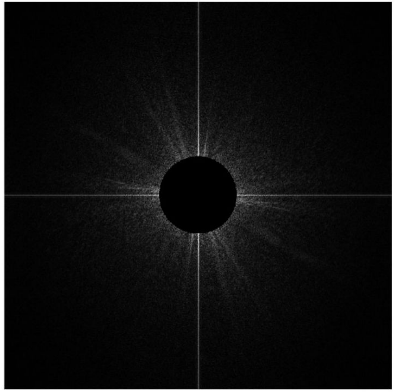
Why would we want to filter in the frequency domain?

Interpretability.

We can look at convolution as an operation over a grid of numbers,
or as modifying the frequencies of an image. It is often intuitive to think in terms of frequencies.







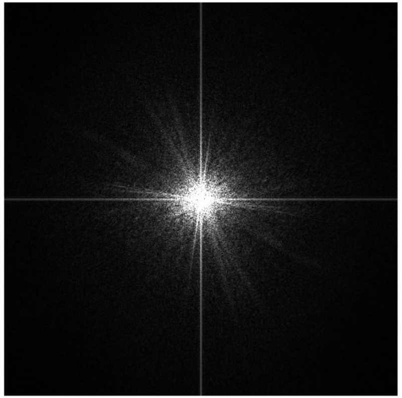
```
high-pass filtering:
pass only the high frequencies
```

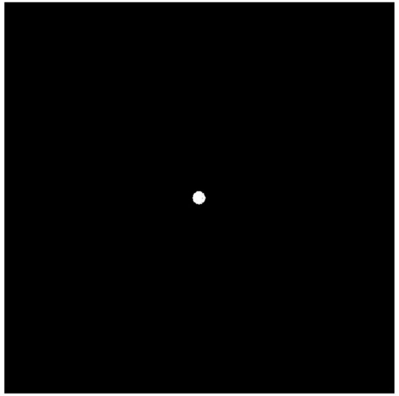
source: Ren Ng

Why would we want to filter in the frequency domain?

Interpretability.

We can look at convolution as an operation over a grid of numbers,
or as modifying the frequencies of an image. It is often intuitive to think in terms of frequencies.





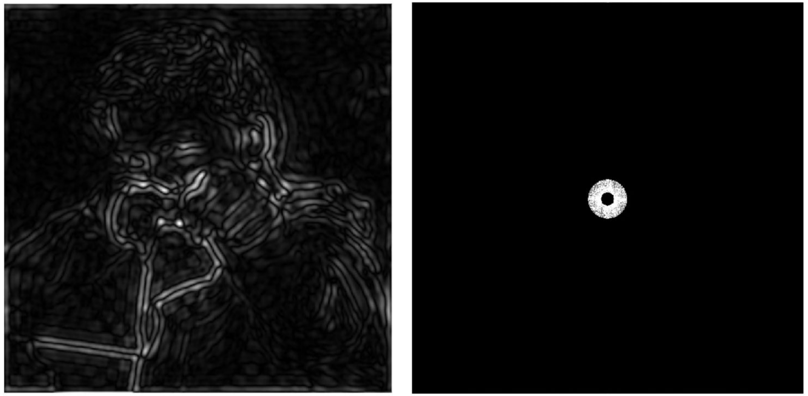
`easy to see smoothness in frequency domain!`

source: Ren Ng

Why would we want to filter in the frequency domain?

Interpretability.

We can look at convolution as an operation over a grid of numbers,
or as modifying the frequencies of an image. It is often intuitive to think in terms of frequencies.



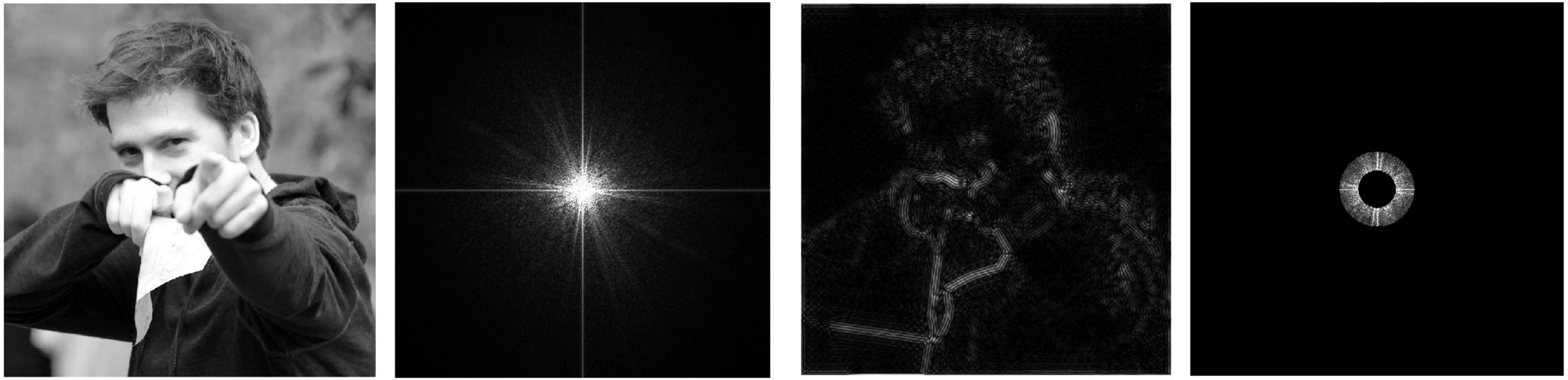
`band-pass filter: filters out both high and low frequencies, looks like a band`

source: Ren Ng

Why would we want to filter in the frequency domain?

Interpretability.

We can look at convolution as an operation over a grid of numbers,
or as modifying the frequencies of an image. It is often intuitive to think in terms of frequencies.



source: Ren Ng

(Extra) Basic Convolution Practice

Convolve the signal **[1, 2, 3, 4, 5]** with the filter **[1, 2, 3]**.

(No need to pad.)

(Extra) Basic Convolution Practice

Convolve the signal **[1, 2, 3, 4, 5]** with the filter **[1, 2, 3]**.

(No need to pad.)

[10

1x3 + 2x2 + 3x1

(Extra) Basic Convolution Practice

Convolve the signal **[1, 2, 3, 4, 5]** with the filter **[1, 2, 3]**.

(No need to pad.)

[10, 16

2x3 + 3x2 + 4x1

(Extra) Basic Convolution Practice

Convolve the signal **[1, 2, 3, 4, 5]** with the filter **[1, 2, 3]**.

(No need to pad.)

[10, 16, 22]

3x3 + 4x2 + 5x1

The Edge Case

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

source: Steven Seitz

Padding Options

cv2.**filter2D**(src, ddepth, kernel[, dst[, anchor[, delta[, **borderType**]]]])

How to extrapolate pixels over the border?

```
/*
 Various border types, image boundaries are denoted with '|'

 * BORDER_REPLICATE:     aaaaaa|abcdefgh|hhhhhhh
 * BORDER_REFLECT:       fedcba|abcdefgh|hgfedcb
 * BORDER_REFLECT_101:   gfedcb|abcdefgh|gfedcba
 * BORDER_WRAP:          cdefgh|abcdefgh|abcdefg
 * BORDER_CONSTANT:      iiiiii|abcdefgh|iiiiiii  with some specified 'i'
 */
```

Padding Options

scipy.signal.**fftconvolve**(in1, in2, **mode='full'**)

Do we even want to extrapolate?

mode : *str {'full', 'valid', 'same'}, optional*

A string indicating the size of the output:

`full`

The output is the full discrete linear convolution of the inputs. (Default)

`valid`

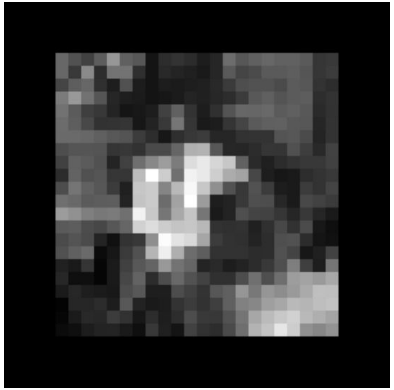
The output consists only of those elements that do not rely on the zero-padding.

`same`

The output is the same size as *in1*, centered with respect to the 'full' output.

Padding Options

- **Constant**
  Pretend that everything outside the image is some specified constant (commonly zero).



source: Richard Szeliski

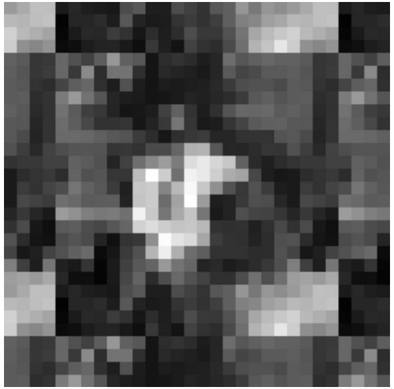
Padding Options

- **Constant**
- **Replicate**
  Pretend that everything outside the image is whatever's already on the edge.

source: Richard Szeliski

Padding Options

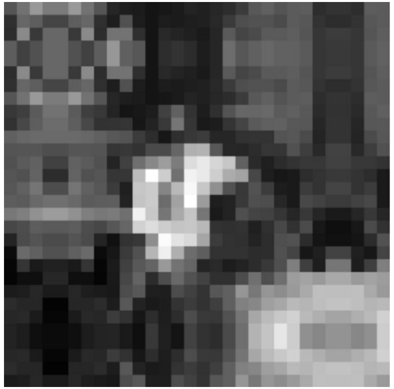
- **Constant**
- **Replicate**
- **Wrap**
  Pretend that the image is tiled indefinitely, i.e. “loop over the image.”



source: Richard Szeliski

Padding Options

- **Constant**
- **Replicate**
- **Wrap**
- **Mirror**
  Reflect the image across its edges.



source: Richard Szeliski

Padding Options

- Constant
- Replicate
- Wrap
- Mirror
- ...

Additional Readings

Sobel filter

- https://stackoverflow.com/questions/17078131/why-sobel-operator-looks-that-way

Fourier domain images

- http://cns-alumni.bu.edu/~slehar/fourier/fourier.html
- https://www.cs.toronto.edu/~guerzhoy/320/lec/FreqDomain.pdf

Circular paths

- https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/