Image Processing II

Shree K. Nayar

Lecture: FPCV-1-5

Module: Imaging

Series: First Principles of Computer Vision

Computer Science, Columbia University

March 30, 2022

FPCV Channel

FPCV Website

Image Processing II

Shree K. Nayar
Columbia University

Topic: Image Processing II, Module: Imaging
First Principles of Computer Vision

1

Recap: Image Processing I

Transform image to new one that is clearer or easier to analyze.

Topics:

(1) Pixel Processing

(2) LSIS and Convolution

(3) Linear Image Filters

(4) Non-Linear Image Filters

(5) Template Matching by Correlation

2

In the previous lecture, we described how image processing can transform an image into one that is clearer or easier to analyze. We first discussed pixel processing, which is the simplest type of image processing. Then, we discussed linear shift invariant systems and their properties. We showed that any linear shift invariant system is performing a convolution. Based on this theory, we described linear image filters that can smooth an image or reduce noise in it. Next, we covered nonlinear image filters which cannot be implemented using convolution but can, for some tasks such as denoising, do better than linear filters. In particular, we looked at the median filter and the bilateral filter. Finally, we presented template matching, which uses correlation — a concept that is closely related to convolution — to find a given pattern in an image.

In this lecture, we will begin with the Fourier transform, which will allow us to switch from the spatial domain to the frequency domain, also known as the Fourier domain. Many image processing methods are easier to develop and analyze in the frequency domain. We will discuss how convolution in the spatial domain is equivalent to multiplication in the frequency domain. This important result can be used to design a variety of linear filters in the frequency domain. Next, we will describe deconvolution, which is the process of undoing the effect of an undesirable convolution. We show that deconvolution is easier to formulate and implement in the frequency domain. We develop a deconvolution algorithm for removing motion blur from images.

Finally, we will study sampling theory. While capturing an image, we are sampling a continuous optical image with a pixel grid. If we sample the image beyond a certain sampling frequency, the continuous image can be recovered without any loss of information. However, under-sampling of the image can result in information loss as well as the introduction of unwanted artifacts in the image. We will study this phenomenon, known as aliasing, and describe how cameras avoid aliasing.

Fourier Transform

Shree K. Nayar
Columbia University

Topic: Image Processing II, Module: Imaging
First Principles of Computer Vision

4

Jean Baptiste Joseph Fourier

(1768-1830)

Any Periodic Function can be rewritten as a Weighted Sum of Infinite Sinusoids of Different Frequencies.

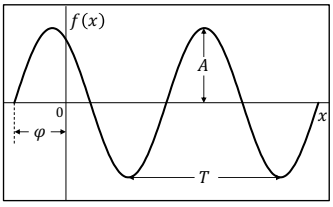
5

The Fourier transform is named for Joseph Fourier, who worked for Napoleon. Fourier was very interested in how heat propagates through materials of different shapes, which led him to develop the Fourier transform. This transform states that any periodic function can be written as the weighted sum of an infinite number of sinusoids of different frequencies.

The basic building block of the Fourier transform is the sinusoid. Shown here is the expression for the sinusoid, when A is its amplitude, u is its frequency, and φ is its phase (or shift). The period T of the sinusoid is the reciprocal of its frequency.

Sinusoid

$$f(x) = A\sin(2\pi ux + \varphi)$$

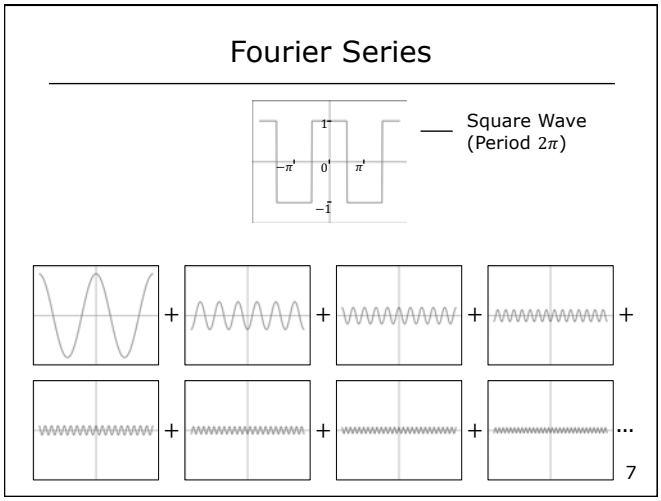


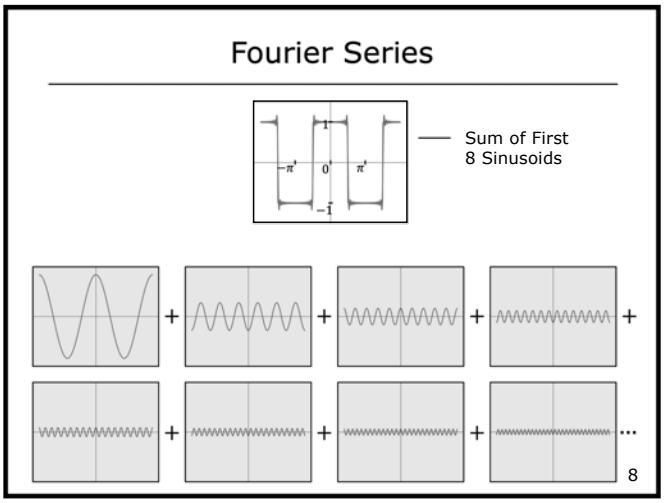
A: Amplitude T: Period

φ: Phase u: Frequency $(1/T)$

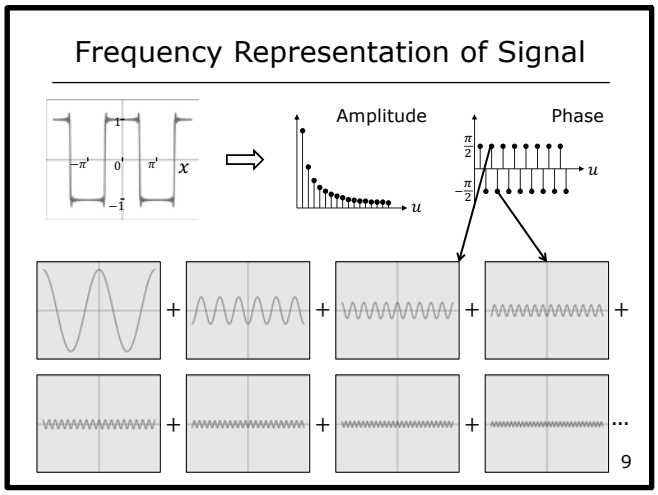
MATH PRIMER 6

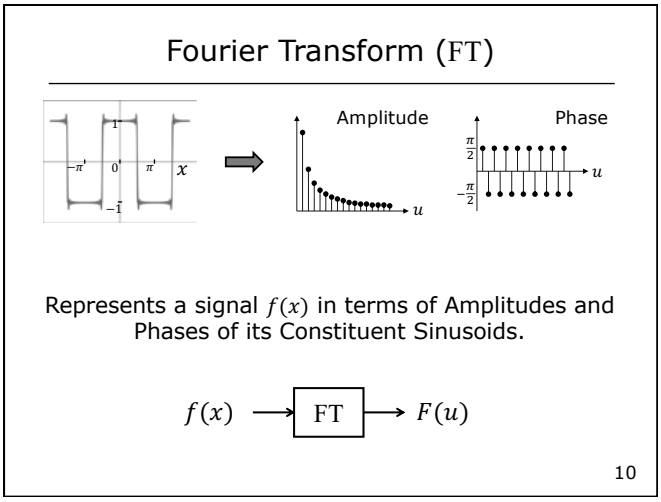
Shown below is a square wave and the first eight of its component sinusoids, as determined by the Fourier transform. These sinusoids are shown in increasing order of frequency. Note that they have different amplitudes and phases. By simply adding these eight sinusoids, we get the fairly good approximation of the square wave shown on the right. It turns out that the Fourier transform of the square wave has an infinite number of sinusoids with non-zero amplitude. If we could add all of these sinusoids, we would get exactly the square wave.

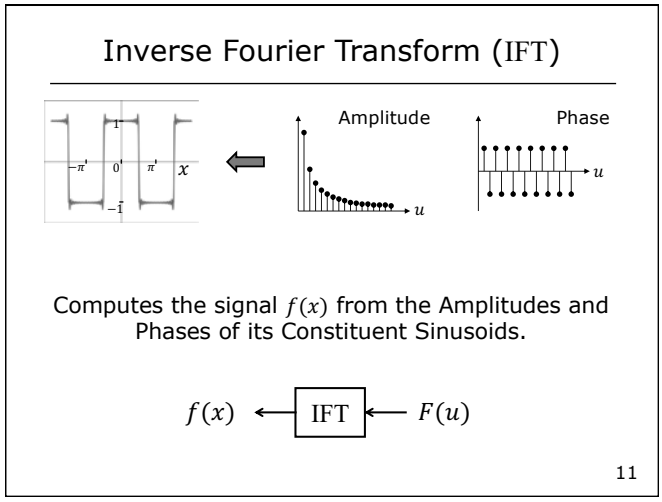




An important thing to note here are the amplitudes and phases of the component sinusoids. Note that the phases flip between $\pi/2$ and $-\pi/2$ in the case of the square wave. While we have used the square wave as an example here, the Fourier transform can be applied to any signal. In each case, it gives an alternative representation of the signal in terms of the amplitudes and phases of its component sinusoids. This representation is referred to as the frequency domain.







As shown here, when the Fourier transform (FT) is applied to a function $f(x)$ we get the function $F(u)$, which represents the amplitude and phase corresponding to the sinusoid for each frequency u. $F(u)$ is the frequency representation of the spatial function $f(x)$. We can also go from $F(u)$ back to the original function $f(x)$ by using the inverse Fourier transform (IFT). It is important to note that we can switch between the spatial and frequency representations, without any loss of information.

Here are the mathematical expressions for the Fourier transform and the inverse Fourier transform. For now, we will stick to signals with a single dimension (x), bearing in mind that all the expressions we present are easily extensible to higher dimensions. Let us first take a close look at the inverse Fourier transform. This tells us that the original signal is an integral (sum) of sinusoids of different frequencies. As we will see shortly, the sinusoids reside within the complex exponential inside the integral. The amplitudes and phases are captured by the Fourier coefficient $F(u)$ associated with each frequency u. The Fourier transform looks similar to its inverse, except you have a negative i in the exponent. Sometimes, the Fourier transform is referred to as the plus i transform, and the inverse is called the minus i transform.

Finding FT and IFT

Fourier Transform:

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi ux} dx$$

x: space
u: frequency
$e^{i\theta} = \cos\theta + i\sin\theta$
$i = \sqrt{-1}$

Inverse Fourier Transform:

$$f(x) = \int_{-\infty}^{\infty} F(u) e^{i2\pi ux} du$$

12

Shown here is the relation between the sinusoid and the complex exponential. Let's take a look at why this relation is valid. We expand the complex exponential using Taylor series and then separate the terms into two groups – the first one has terms with even powers and the second group has terms with odd powers. Each term in the group with odd powers is multiplied by i on the outside. We see that the first group is the Taylor series expansion of cosine θ while the other is of sine θ.

Complex Exponential (Euler Formula)

$$e^{i\theta} = \cos\theta + i\sin\theta \qquad i = \sqrt{-1}$$

Expand $e^{i\theta}$ using Taylor Series:

$$e^{i\theta} = 1 + i\theta + \frac{(i\theta)^2}{2!} + \frac{(i\theta)^3}{3!} + \frac{(i\theta)^4}{4!} + \frac{(i\theta)^5}{5!} + \frac{(i\theta)^6}{6!} + \ldots$$

$$e^{i\theta} = \underbrace{\left(1 - \frac{\theta^2}{2!} + \frac{\theta^4}{4!} - \frac{\theta^6}{6!} + \cdots\right)}_{\cos\theta} + i\underbrace{\left(\theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \cdots\right)}_{\sin\theta}$$

MATH PRIMER 13

Note that the Fourier transform is complex because the coefficient $F(u)$ is complex, as it needs to capture both the amplitude and phase of the sinusoid. Also note that the integral goes from minus infinity to infinity, so frequencies are represented using both positive and negative numbers. The formulas for the amplitude and phase are shown here, where $\mathfrak{Re}$ denotes real and $\mathfrak{Im}$ denotes imaginary.

Fourier Transform is Complex!

$F(u)$ holds the Amplitude and Phase of the sinusoid of frequency u.

$$F(u) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi ux} dx$$

$$F(u) = \mathfrak{Re}\{F(u)\} + i\,\mathfrak{Im}\{F(u)\}$$

Amplitude: $A(u) = \sqrt{\mathfrak{Re}\{F(u)\}^2 + \mathfrak{Im}\{F(u)\}^2}$

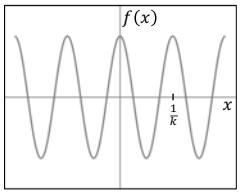
Phase: $\varphi(u) = \text{atan2}(\mathfrak{Im}\{F(u)\}, \mathfrak{Re}\{F(u)\})$

14

Now let's take a look at the Fourier transforms of a few simple functions. Here, we see a cosine function. Since it is basically a sinusoid that is shifted, it is a signal with a single non-zero frequency. Given that k is the frequency of this cosine function, we get a delta function at k and $-k$ in the Fourier transform. The two delta functions happen to reside in the real domain of the Fourier transform.

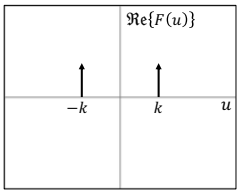
Fourier Transform Examples

Signal $f(x)$



$f(x) = \cos 2\pi kx$

Fourier Transform $F(u)$



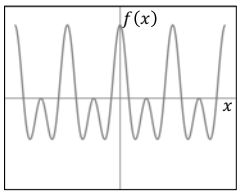
$F(u) = \frac{1}{2}[\delta(u+k) + \delta(u-k)]$

15

If we have a signal that is the sum of two cosine functions with different frequencies, The Fourier transform is simply the sum of the transforms of the two cosine functions. As a result, we get the four delta functions seen here.

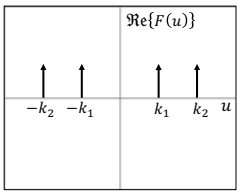
Fourier Transform Examples

Signal $f(x)$



$f(x) = \cos 2\pi k_1 x + \cos 2\pi k_2 x$

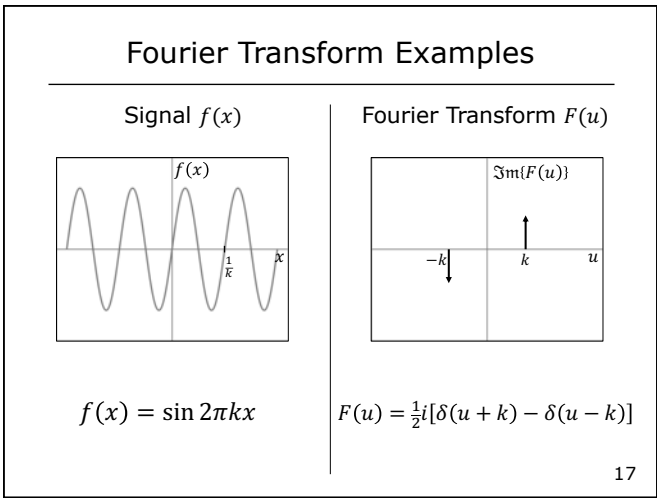
Fourier Transform $F(u)$



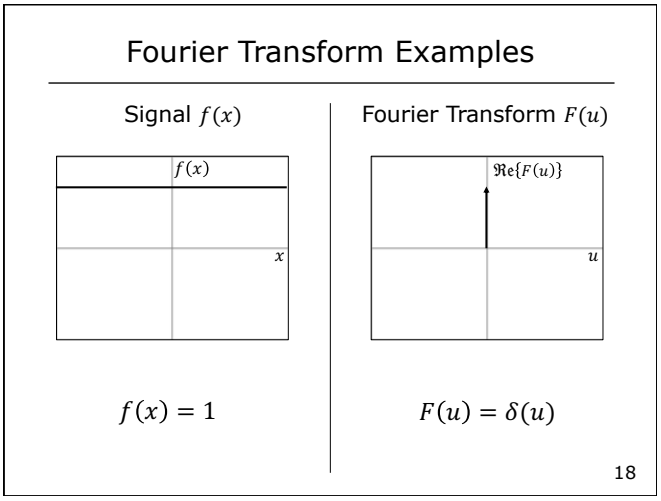
$F(u) = \frac{1}{2}[\delta(u+k_1) + \delta(u-k_1) + \delta(u+k_2) + \delta(u-k_2)]$

16

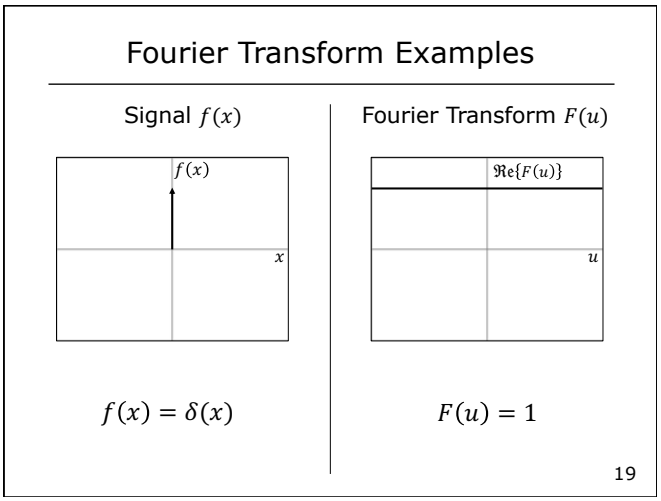
Now let us take a look at the case of a sinusoidal function. As before, we only have one frequency k, which again results in delta functions, at k and $-k$. In this case, however, the two delta functions are flipped. In addition, they reside in the imaginary domain of the Fourier transform.



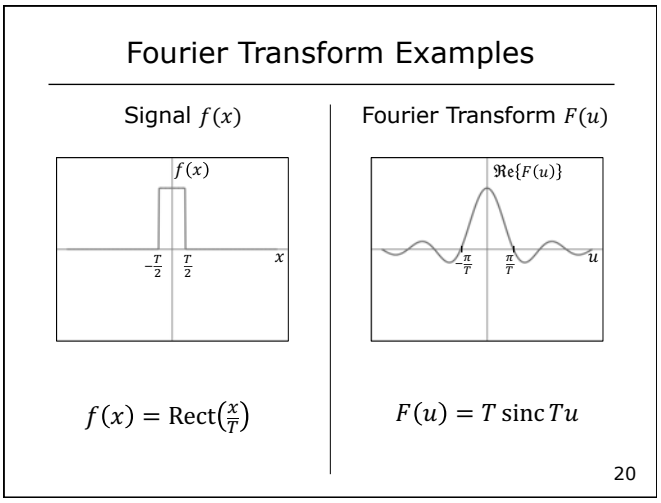
Now, consider a signal that is a constant. In this case, there are no component sinusoids. The constant value of the function can be accounted for by viewing it as the amplitude of a sinusoid with frequency equal to zero. Therefore, we have a single delta function at u equal to zero.



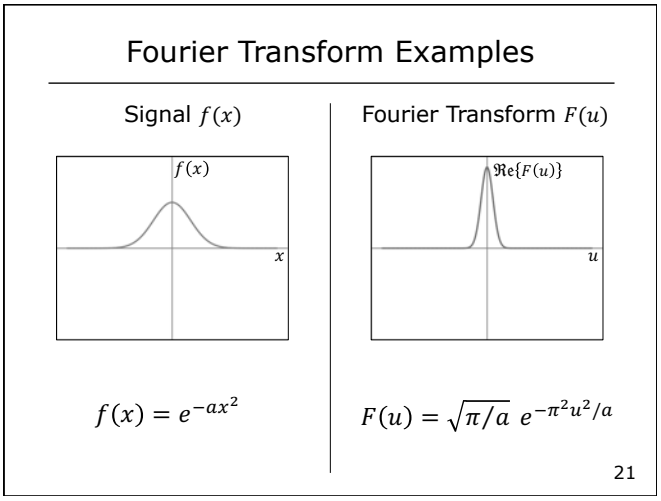
What if we have a signal that is itself a delta function? In this case, we need equal amplitudes of sinusoids of all frequencies to represent it. As a result, the Fourier transform is flat.



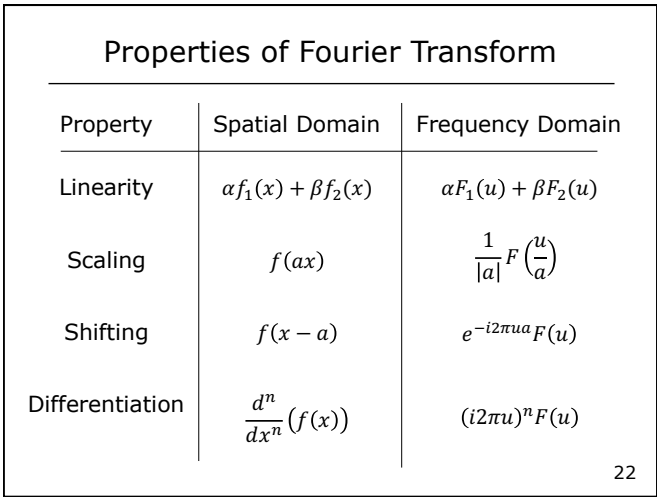
Here is the rectangular function, also known as a pulse. If you remember, in the previous lecture, we talked about the box filter. The pulse is a one-dimensional box filter. The Fourier transform of this rectangular function is the sinc function. It starts off with a maximum value at zero frequency and then tapers down. Note that it oscillates as it tapers.



Finally, we have the Gaussian. Here, a is related to the sigma (width) of the Gaussian. Its Fourier Transform is also a Gaussian, but with a width proportional to 1 over a. For any function, if we widen it in the spatial domain, we actually compress it in the frequency domain and vice versa. This is referred to as inverse scaling.



Let us take a look at some important properties of the Fourier transform. The first is linearity. If we take a linear combination of two functions in spatial domain, then the Fourier transform of that sum is the same linear combination of the Fourier transforms of the two functions. The second property is the inverse scaling we discussed above. If we shift a function, the Fourier transform is simply the Fourier transform of the original function multiplied by a complex exponential. Perhaps, most interesting is the differentiation property. The Fourier transform of the n^{th} derivative of a function is simply the Fourier transform of the original function multiplied by $i2\pi u^n$.



We have already seen that convolution is a very useful concept in image processing. It turns out that there is a very close relationship between convolution and the Fourier Transform. First, let us take a look at convolution once again. Below, we see the convolution of two functions, f and h, and the result, g. Let us recap how convolution works. We take the function h, flip it, move it to a point x, and overlay this function on f. We then find the product of the two functions. The integral of the product of the two functions is a single number, which is the value of g at x. If we want to get the entire function $g(x)$, we take the flipped h, move it to minus infinity, slide it and perform the same operation we just discussed for each value of x.

Convolution Theorem

Shree K. Nayar
Columbia University

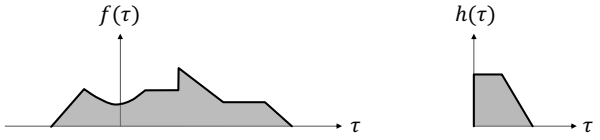
Topic: Image Processing II, Module: Imaging
First Principles of Computer Vision

23

Convolution

Convolution of two functions $f(x)$ and $h(x)$

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

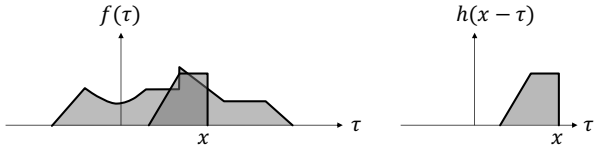


REVIEW 24

Convolution

Convolution of two functions $f(x)$ and $h(x)$

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

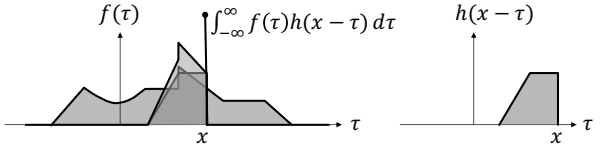


REVIEW 25

Convolution

Convolution of two functions $f(x)$ and $h(x)$

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

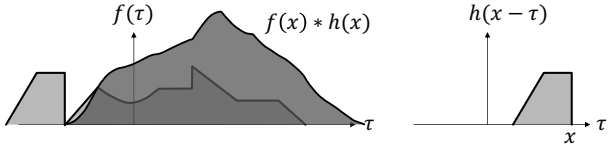


REVIEW 26

Convolution

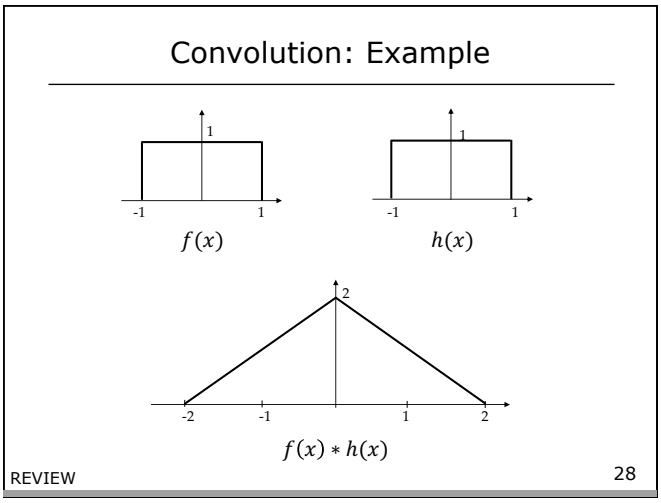
Convolution of two functions $f(x)$ and $h(x)$

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

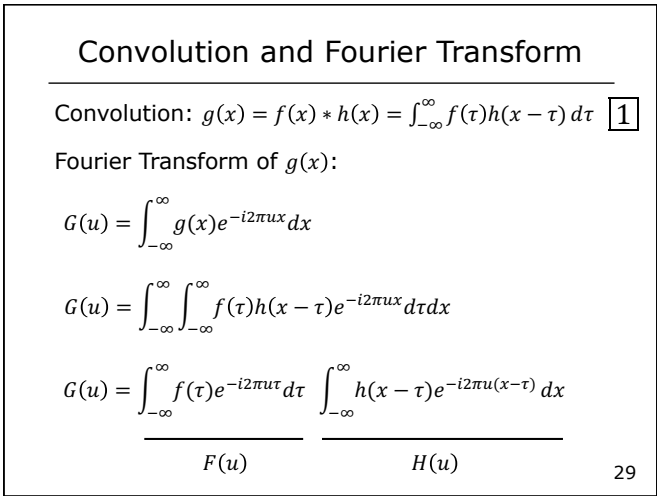


REVIEW 27

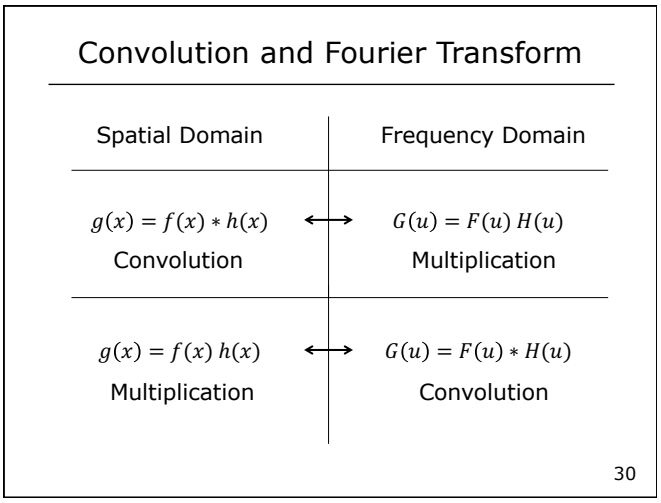
Previously, we looked at this simple example where we have the convolution of a rectangular function with itself. In this case, h is the same as f. As the rectangular function slides over itself, the area of the overlap increases linearly and then decreases linearly. The result of the convolution is therefore a triangle.



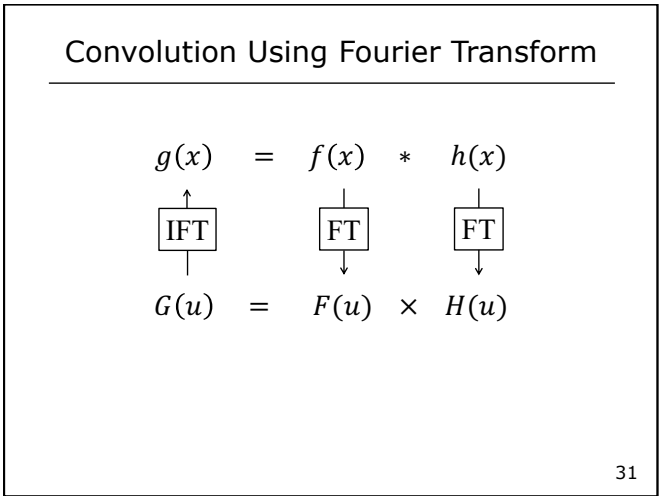
Now let us take a look at the relationship between convolution and the Fourier transform. Let us start with convolution. g is the result of f convolved with h, and here is the expression for convolution we are familiar with [1]. Now, we are going to find the Fourier Transform of g, which is $G(u)$. If we substitute the expression for g from convolution, we end up getting a double integral, which can be written as the product of two single integrals. We see that the first integral is indeed the Fourier transform of $f(x)$, which is $F(u)$. Now let us take a look at the second integral. If we substitute $y = (x - \tau)$, the limits of the integral remain the same since τ is finite. Therefore, what we have here is the Fourier transform of $h(x)$, which is $H(u)$. Essentially, we have that convolution of f with h in spatial domain corresponds to multiplication of the Fourier transforms of f and h in frequency domain.



The above is a very important relationship between convolution and the Fourier Transform, also referred to as the convolution theorem. It tells us that if we are finding the convolution of two functions in spatial domain, that is equivalent to multiplying the Fourier transforms of the two functions in frequency domain. Using the same approach, we can also show that if we are taking the product of two functions in spatial domain, that is equivalent to convolving the Fourier transforms of the two functions in frequency domain.

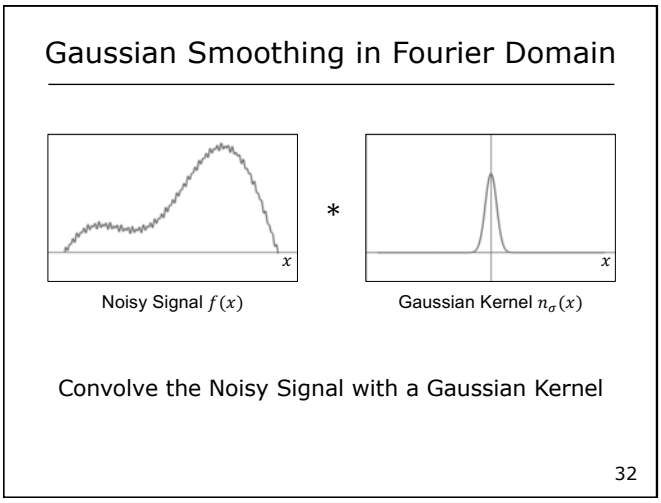


What does the convolution theorem allow us to do? Well, if we want to find the convolution of f with h to get g, we no longer have to do it in spatial domain — we can use the frequency domain instead. We first take the Fourier transform of f to get $F(u)$ and the Fourier transform of h to get $H(u)$. Then, we multiply the two to get the Fourier transform $G(u)$, and then find its inverse Fourier transform to get $g(x)$.

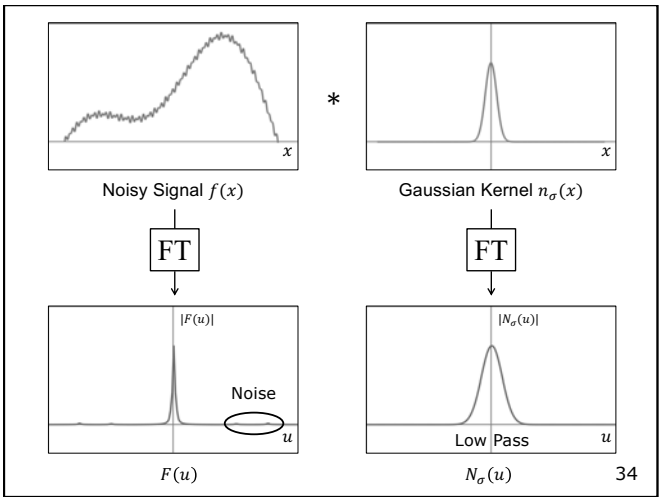


Why is this useful? As it turns out, there are very fast algorithms for finding the Fourier transform and the inverse Fourier transform. So, when we have a very large filter we want to convolve an image with, it is much cheaper and hence much more efficient to do the convolution in the frequency domain. In addition, it allows us to understand the effects of applying a filter. We can take any filter that we design in spatial domain and see what it really does to the frequencies in frequency domain, and vice versa.

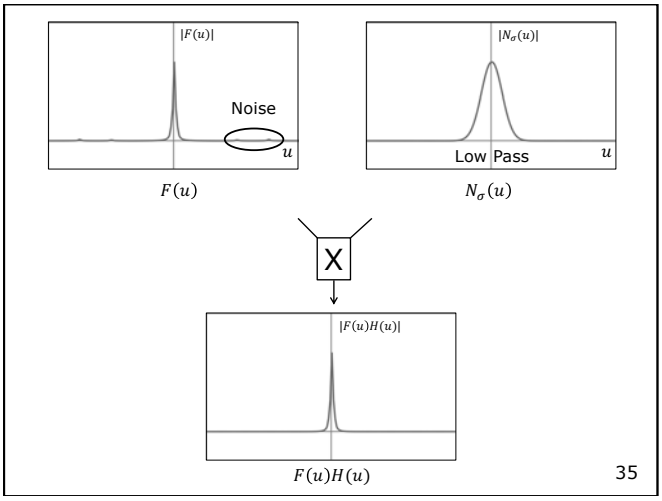
Let us look at a very simple example — Gaussian smoothing in frequency domain. Imagine we have this signal shown on the left. As we can see, there is some noise in the signal which we wish to remove, or at least reduce. What we can do is convolve this signal in spatial domain using the Gaussian kernel shown on the right. Alternatively, we can achieve the same in frequency domain using the convolution theorem.



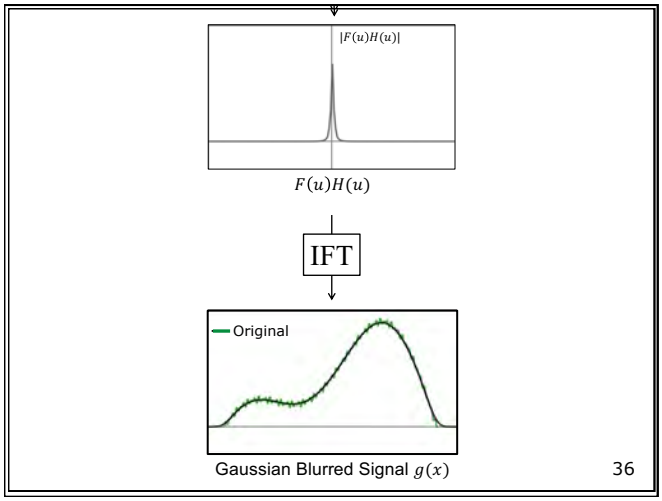
We first take the Fourier transforms of the signal and the Gaussian kernel to get $F(u)$ and $G(u)$. We can see here that the signal that we are interested in is really the central peak in the bottom left plot, while the two little bumps off to the side are high frequency noise that we want to remove. As we previously discussed, the Fourier transform of a Gaussian is also a Gaussian. This Gaussian will act like a low-pass filter that removes the high frequencies.



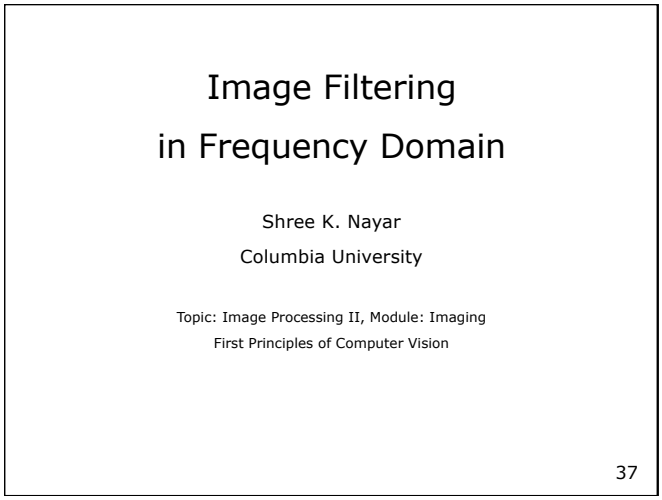
Now, remember that convolution in spatial domain is multiplication in frequency domain. Thus, what we are going to do is find the product of these two Fourier transforms. In the result, we see that the little bumps that correspond to noise are more or less gone.



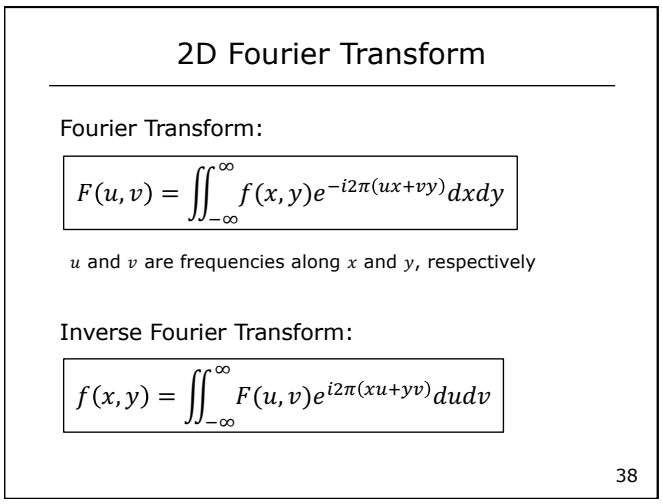
If we find the inverse Fourier transform of the above result, we end up with this clean signal (black). For comparison the original signal (green) has been overlayed on the clean signal.



Now, let us take a look at image filtering in frequency domain. Since images are two-dimensional, we will first need to extend the Fourier transform from one dimension to two dimensions.



Shown here is the expression for the two-dimensional Fourier transform. Our image is $f(x,y)$ where x and y are the spatial coordinates. We now also have two frequencies: u along the x-direction and v along the y-direction. A similar expression for the inverse Fourier transform is also shown at the bottom of the slide. In fact, the expressions for the Fourier transform and its inverse can be extended to any number of dimensions.



Since images are discrete, we need discrete forms of the expressions for the Fourier transform and its inverse, which are shown here. In this case, m and n are used for the spatial coordinates and p and q are used for the frequency coordinates. Using these expressions, we can now find the Fourier transform of any discrete image.

2D Fourier Transform: Discrete Images

Discrete Fourier Transform (DFT):

$$F[p,q] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f[m,n]e^{-i2\pi pm/M}e^{-i2\pi qn/N} \qquad \begin{matrix} p = 0 \dots M-1 \\ q = 0 \dots N-1 \end{matrix}$$

p and q are frequencies along m and n, respectively

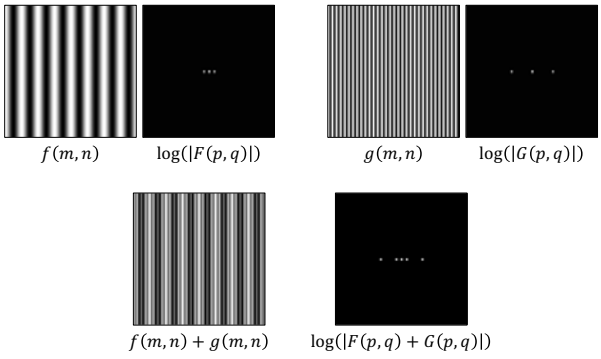
Inverse Discrete Fourier Transform (IDFT):

$$f[m,n] = \frac{1}{MN}\sum_{p=0}^{M-1}\sum_{q=0}^{N-1} F[p,q]e^{i2\pi pm/M}e^{i2\pi qn/N} \qquad \begin{matrix} m = 0 \dots M-1 \\ n = 0 \dots N-1 \end{matrix}$$

39

Let us take a look at the Fourier transforms of a few simple images. While the Fourier transform includes both the magnitude and the phase, for our purposes here, we are going to ignore the phase. We will take the absolute of the magnitude for each frequency and, for visualization purposes, we will also take the log of that value in order to compress the wide range of magnitude values into a smaller range. The center of the magnitude of the transform (shown as an image) is the zero frequency, and the frequency increases with distance (in both direction) from the center.

2D Fourier Transform: Example 1



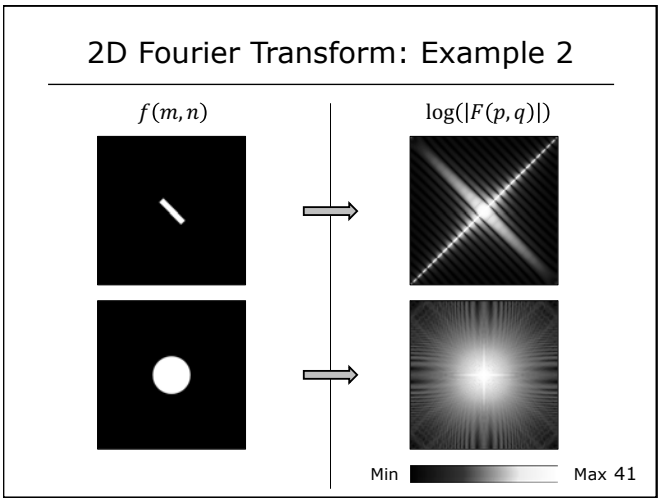
40

Note: $\log(|F|)$ is used just for display

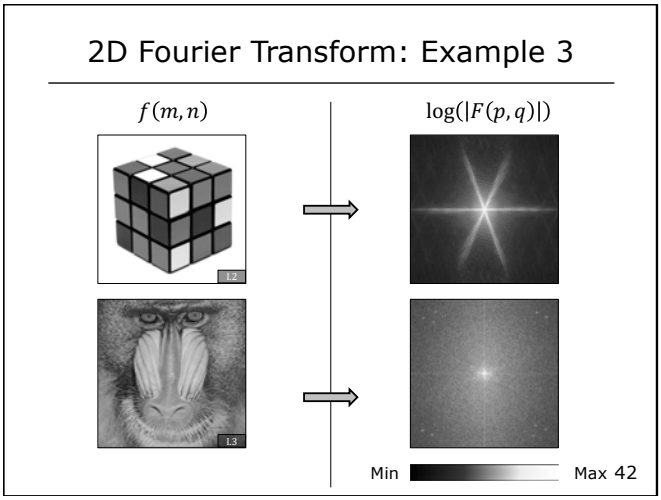
On the left is an image which is a cosine function in the x direction. We know that the Fourier transform of a cosine is just one frequency — the frequency k of the cosine function. So, we end up getting two dots on the horizontal axis in the Fourier transform, one at k and one at -k. Note that there is a third dot in the center at the zero frequency. This results from the fact that images cannot be negative. Let us say that the image intensity values are in the range from 0 to 256 and that the mean value of the cosine function is 128. Hence, the image has an average value of 128. This constant results in a non-zero value at the zero frequency.

The image shown on the right is another cosine function, but one which is of higher frequency. Because it has a higher frequency, the three dots are now more separated from each other. If we take the average of the two cosine images in the top row, we get the image in the bottom row. It's Fourier transform has five dots — we get the center dot again and two dots for each of the two cosines.

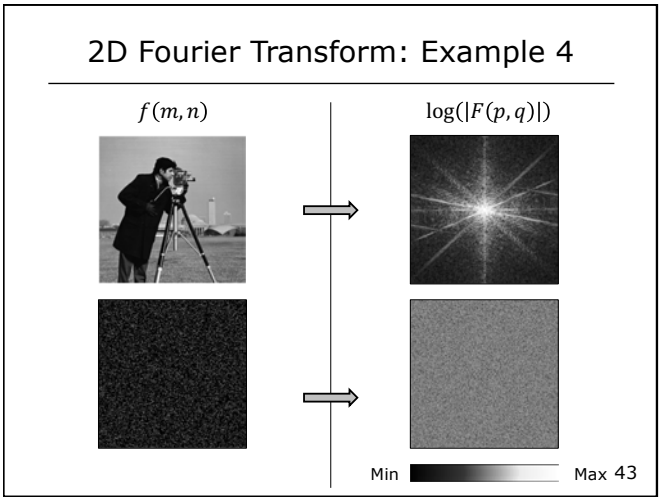
Let us now take a look at two simple binary images. For the top left image, the Fourier transform is shown on the top right. Once again, the intensity at each point in the transform image is proportional to the log of the magnitude of the transform value for the corresponding frequency. As we can see, we get very strong frequencies along two lines. That is because we have two pairs of strong edges and to re-create each edge we need frequencies in the direction perpendicular to the edge. The second image is that of a disk. The flat part of the disk produces some low frequencies. Again, the edges produce high frequencies. Note that since the image is rotationally symmetric, its Fourier transform is rotationally symmetric as well.



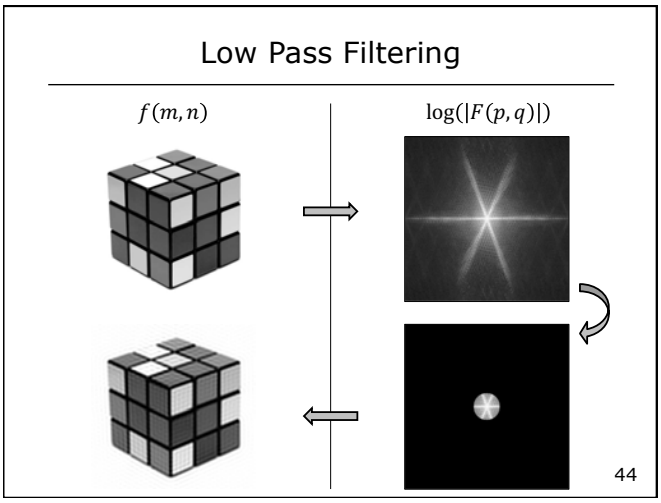
Moving to more natural images, here we have a Rubik's cube on the top. In this case, we see that we get three strong lines in the Fourier transform, indicating high frequencies along these lines. This is because the original image has edges along three dominant directions: vertical edges, horizontal edges, and slanted edges. In the case of the Mandrill, there is not really much that can be said. Since the image is complex it is hard to interpret its transform. As humans, we are not wired to easily interpret signals in frequency domain.



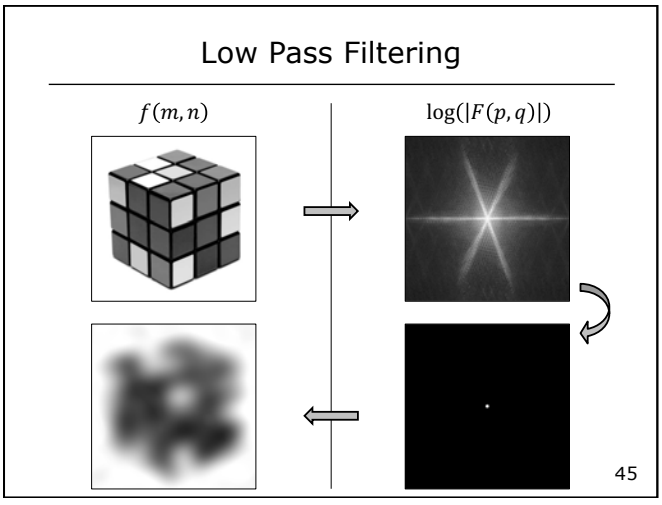
Here is a more complex image with lots of strong edges. The second example is an image which is just noise. In this case we see that we get very strong values in the Fourier transform, even for high frequencies. As we will see soon, this makes the removal of noise from images a challenging problem.

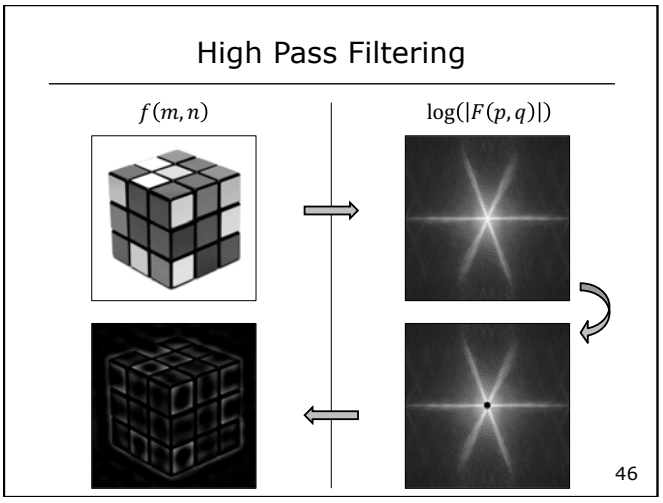


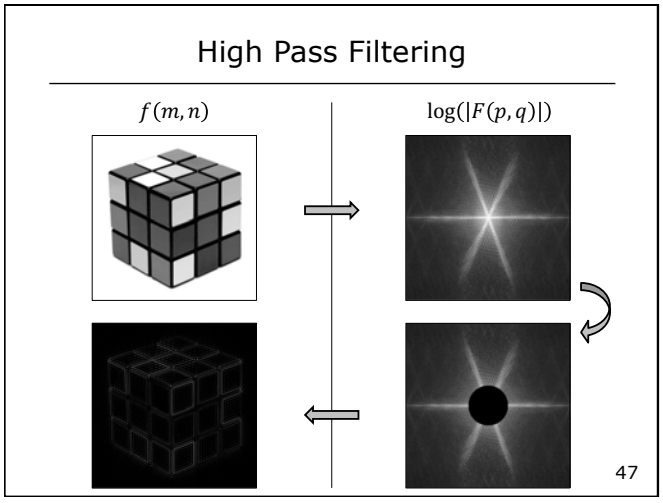
Now let us take a look at some simple types of filtering. Taking the Fourier transform of the image of the Rubik's cube, we get the same output we saw earlier. If we want to low-pass filter this image, we can simply cut out the outer parts of the Fourier transform. Taking the inverse Fourier transform, we see that we get a blurrier (smoother) version of the image. This image has some block artifacts, and that is because we have done a very harsh low-pass filtering — the sharp boundary of the cut-out region causes the artifacts.



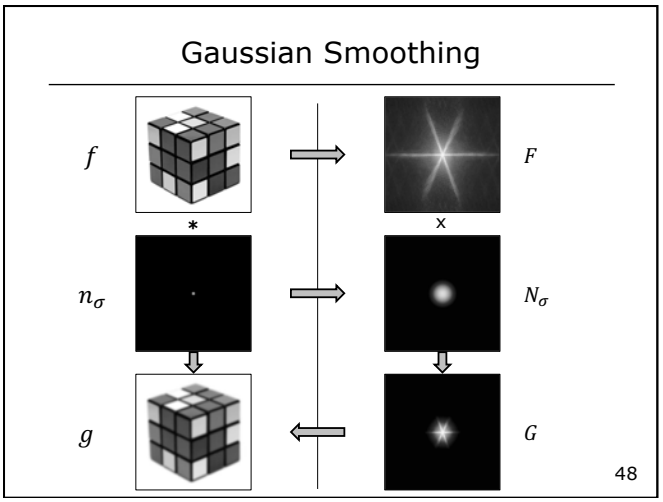
If we more severely low-pass filter this image, meaning we use a smaller cut-out region in frequency domain, as expected, we get an even blurrier image.

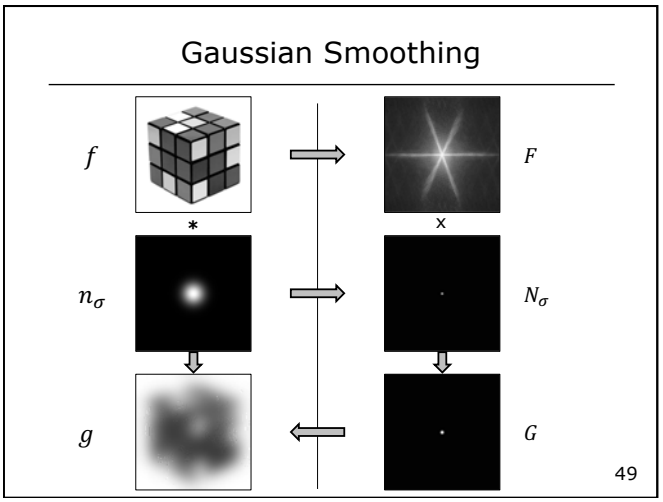






We can also do the opposite, which is high-pass filter the image. Here we have the Fourier transform again, but now we are going to eliminate the low frequencies and keep the high frequencies. When we do that, we see that we essentially remove all of the constant brightness regions in the image, leaving us with the regions where there is a rapid change in brightness. If we increase the severity of our high pass filtering by using the larger cutout on the right, we see that the edges and corners begin to appear more prominent. In the next lecture, we will develop methods for detecting edges and corners. The above examples hint at the fact that edge and corner detectors should be designed to high-pass filter images.

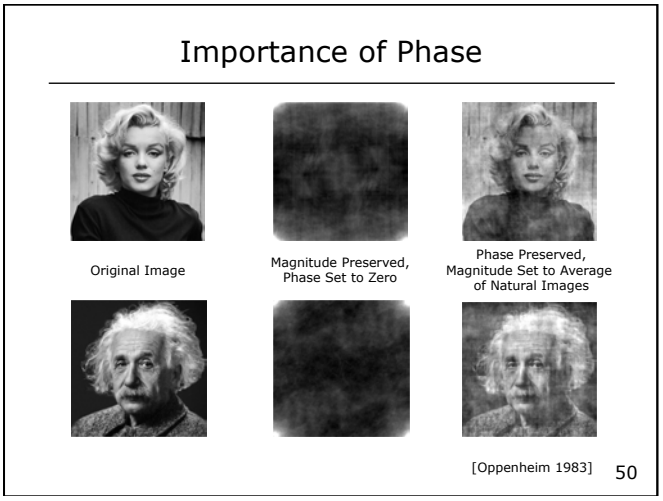




We have talked previously about Gaussian smoothing, so now let us take a look at the same in frequency domain. Here we wish to convolve the Rubik's cube image with a Gaussian kernel. As we discussed earlier, we can perform this convolution by taking the product of the Fourier transforms of the image and the Gaussian kernel. By applying the inverse Fourier transform to the product, we get the result of

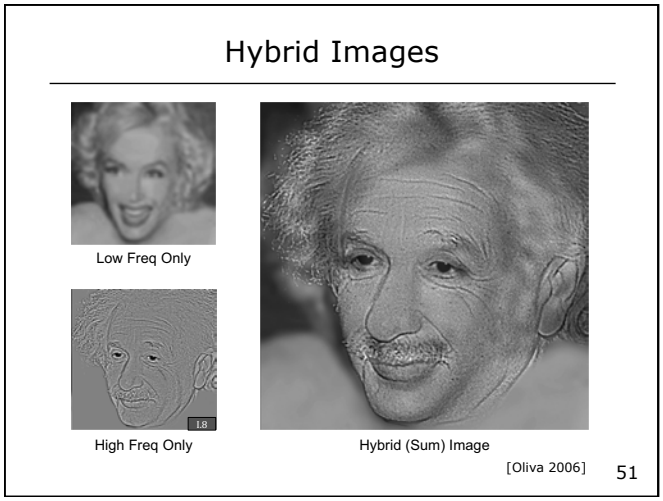
the convolution, which is a blurred version of the original image. It is only slightly blurred because the Gaussian kernel in spatial domain is a very narrow kernel. If we choose a wider kernel (right slide), then we see that we end up getting an even blurrier image.

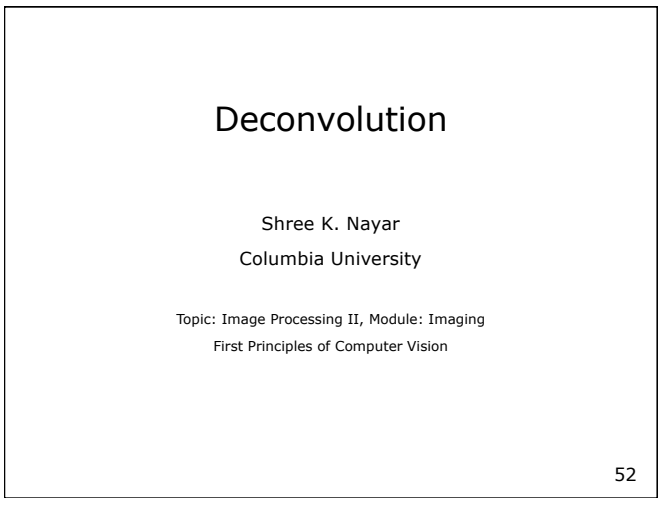
In the above examples we have taken the Fourier transform of an image which gives us both the magnitude and the phase, but we have focused our attention on the magnitude. It turns out, however, that the phase is often more important in terms of preserving information than the magnitude. Let us demonstrate this. Shown here is an image of Marilyn Monroe, to which we have applied the Fourier transform to get the magnitude and the phase. The first thing we are going to do is just set all of the phases to zero while preserving the magnitudes and then apply the inverse Fourier transform. The result, shown in the middle, reveals that if we remove all the phase information, we end up with an image that is unrecognizable.

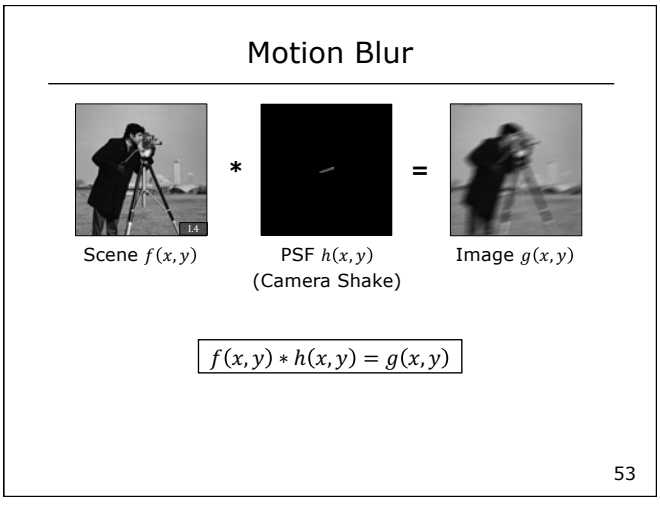


Next, we are going to do the opposite. In this case, we are going to keep the phases, but change the magnitudes. We cannot set the magnitudes to zero, because then we would just get a black image. Instead, we take a bunch of natural images, find the Fourier transforms of those images, and take the average of the magnitudes of those images. We will use that average as the magnitudes and the original phases of the Marilyn Monroe image to construct a Fourier transform, and then apply the inverse Fourier transform to it. As seen, we get an image that is actually recognizable — although the magnitudes do not correspond to those of the original image, Monroe is clearly visible in this image. This simple demonstration tells us that the phase information is extremely important. In the bottom row is the same process applied to an image of Albert Einstein.

Finally, let us talk about hybrid images. On the top left, we see an image of Marilyn Monroe. This image has already been low-pass filtered, meaning the high frequencies were cut out and the low frequencies were preserved, using the techniques that we just discussed. Therefore, we end up with this blurred version of the image. On the bottom left, we see an image of Albert Einstein where the opposite was done. The original image was high-pass filtered, meaning the low frequencies were cut out and the high frequencies were preserved. Hence, we see the fine details exaggerated here. Then, we take an average of these two images and get what is called the "hybrid image." When we look at this hybrid image from up close, we see mostly Albert Einstein. But if we walk away from the image, we see Marilyn Monroe. What is going on here is that when we are close to the display, we are able to see all the fine details (high frequencies) which are of Einstein. But when we are far away, the high frequencies in the image get even higher. At some distance, because of the point-spread function of the lens of the eye, these frequencies are filtered out and the image falling on the retina has mostly the low frequencies that belong to Marilyn Monroe.

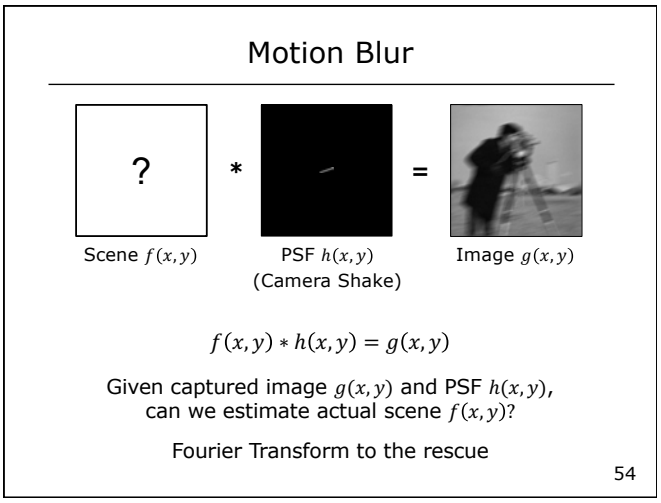




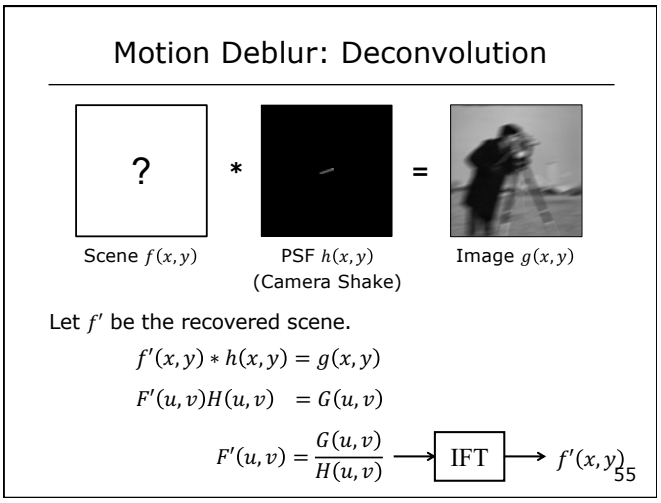


There are instances when, while we are taking an image, the image is inadvertently convolved with an undesirable function. An example of this is motion blur. While taking an image with your smartphone camera, the camera can shake a bit when you press the shoot button. In this case, the captured image is a sharp image of the scene convolved by a motion blur function.

Thus far, we have been talking about how images can be convolved with filters to enhance their quality. In the case of motion blur, though, we have the opposite problem, where we have an image that has been unintentionally convolved with a function and we want to undo the effect of this convolution. That process is called deconvolution. It turns out that it is much easier to do in frequency domain using the Fourier transform.



Shown here is the motion blur problem. Imagine we have an ideal image f of the scene, but because of motion blur, we end up convolving this image with a point spread function h. The result is a smeared image g. To recover the ideal image f we need to know the point spread function h. One way to estimate h is by using an inertial measurement unit, or an IMU, embedded within the capture device (phone, tablet, or camera). The IMU includes accelerometers that can provide an estimate of the 3D motion of the camera during image capture, which can be used to estimate the point spread function h.



Let f' be the image that you want to recover. We call it f' and not f because after the recovery process, f' may be slightly different than the original image. One way to approach the problem is to find the Fourier transforms H and G of h and g, respectively, and then divide G by H and take the inverse Fourier transform to get f'. This is a pretty straightforward solution, so let us see how well it works.

Motion Deblur: Deconvolution

$$F'(u,v) = \frac{G(u,v)}{H(u,v)} \longrightarrow \boxed{\text{IFT}} \longrightarrow f'(x,y)$$

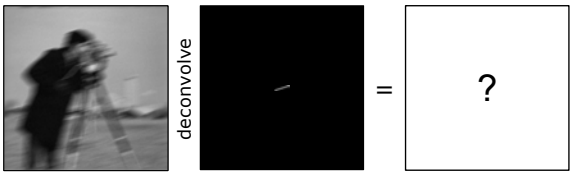


Image $g(x,y)$ PSF $h(x,y)$ Recovered $f'(x,y)$

56

Motion Deblur: Deconvolution

$$F'(u,v) = \frac{G(u,v)}{H(u,v)} \longrightarrow \boxed{\text{IFT}} \longrightarrow f'(x,y)$$

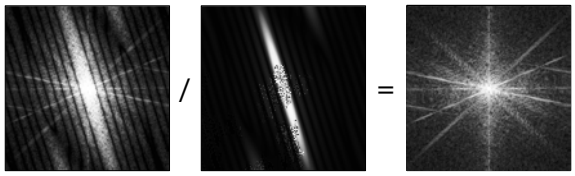


Image $G(u,v)$ PSF $H(u,v)$ Recovered $F'(u,v)$

Step 1: Recover $F'(u,v)$ in Fourier Domain

57

Here we have our original captured image and our point spread function, which has been estimated using the IMU. We take the Fourier transform of the image and divide it by the Fourier transform of the point spread function. This yields the Fourier transform of our recovered image. The three Fourier transforms are shown in the top right slide. Now, if we take the inverse Fourier transform, we end up with our result, which is shown in the slide on the right. We see that this simple solution to deconvolution works quite well — the recovered image is of high quality.

Motion Deblur: Deconvolution

$$F'(u,v) = \frac{G(u,v)}{H(u,v)} \longrightarrow \boxed{\text{IFT}} \longrightarrow f'(x,y)$$

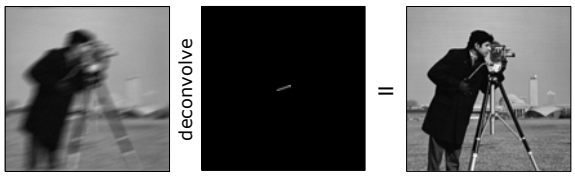


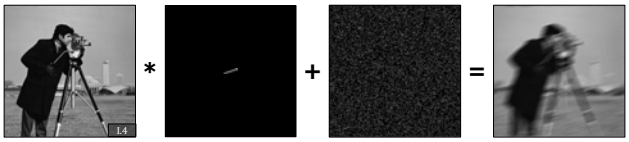
Image $g(x,y)$ PSF $h(x,y)$ Recovered $f'(x,y)$

Step 2: Compute IFT of $F'(u,v)$ to recover scene

58

In our discussion above, we have ignored the effect of image noise. When we shake the camera, the optical image that is falling on the sensor itself is changing while the sensor is integrating the image. This is why the blur happens. Note that this blurred image, when it is read out of the camera, will include various types of noise, including, photon noise, read noise, quantization noise, thermal noise, and so on. The end result is a noisy motion blurred image.

Adding Noise to the Problem



Scene $f(x,y)$ PSF $h(x,y)$ (Camera Shake) Noise $\eta(x,y)$ Image $g(x,y)$

$$f(x,y) * h(x,y) + \eta(x,y) = g(x,y)$$

Can we afford to ignore noise?

59

How well does the above deconvolution process work when the image includes noise? Here is the same captured image, but with noise added to it. Shown next to it is our point spread function. Again, we find the Fourier transforms of both, divide them, and get the Fourier transform shown in the bottom left slide. If we take the inverse Fourier transform, we get an image that is overwhelmed by the noise (bottom right slide). Our simple deconvolution method amplifies the noise and drowns out the signal that we are looking for.

Motion Deblur: Deconvolution

If we ignore the noise ($\eta(x,y)$):

$$\frac{G(u,v)}{H(u,v)} = F'(u,v) \longrightarrow \boxed{\text{IFT}} \longrightarrow f'(x,y)$$

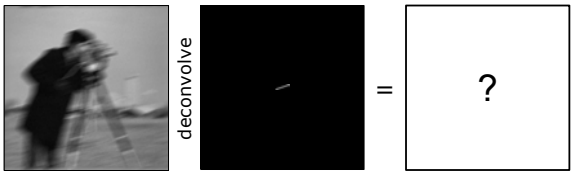


Image $g(x,y)$ (with noise) PSF $h(x,y)$ Recovered $f'(x,y)$

60

Motion Deblur: Deconvolution

If we ignore the noise ($\eta(x,y)$):

$$\frac{G(u,v)}{H(u,v)} = F'(u,v) \longrightarrow \boxed{\text{IFT}} \longrightarrow f'(x,y)$$

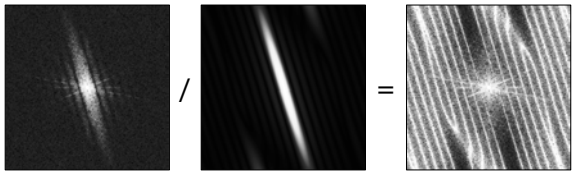


Image $G(u,v)$ PSF $H(u,v)$ Recovered $F'(u,v)$

Higher frequencies in $F'(u,v)$ are amplified

61

Motion Deblur: Deconvolution

If we ignore the noise ($\eta(x,y)$):

$$\frac{G(u,v)}{H(u,v)} = F'(u,v) \longrightarrow \boxed{\text{IFT}} \longrightarrow f'(x,y)$$

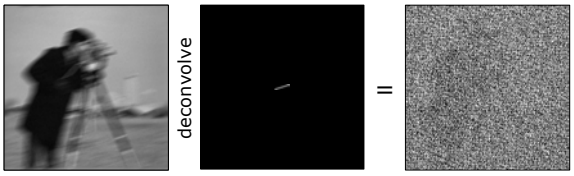


Image $g(x,y)$ (with noise) PSF $h(x,y)$ Recovered $f'(x,y)$

Noise is significantly amplified

62

Let us take a closer look at what is happening here. Remember that the Fourier transform H of the point spread function of the motion blur could be zero for certain frequencies. Additionally, since it is a low-pass filter, it could just be zero for all frequencies above some value. For all frequencies for which the magnitude is zero, F' is simply not recoverable because we would be performing division by zero. Furthermore, while the point spread function is a low-pass filter with low values for high frequencies, we know that noise in the captured image includes high values for high frequencies. As a result, the division stated above ends up amplifying the noise in the recovered image

Deconvolution: Issues

$$\frac{G(u,v)}{H(u,v)} = F'(u,v) \longrightarrow \boxed{\text{IFT}} \longrightarrow f'(x,y)$$

1. Where $H(u,v) = 0$, $F'(u,v) = \infty \rightarrow$ Not recoverable

2. Motion blur filter $H(u,v)$ is a low pass filter.

   For high frequencies (u,v):
   - Noise $N(u,v)$ in $G(u,v)$ is high
   - Filter $H(u,v) \approx 0$

   } Noise in $G(u,v)$ is amplified

We need some kind of Noise Suppression.

63

f'. That is why we end up getting the recovered image we did, which is basically salt and pepper noise over the entire image.

We need some form of noise suppression during the deconvolution process, which brings us to Wiener deconvolution. Once again, we have that F' is equal to G divided by H, except that this time, we are going to multiply it with this term $\boxed{1}$. In this term, NSR is a noise-to-signal ratio. It is, for any given frequency, the power of noise divided by the power of the signal itself. Admittedly, we do not know the power of the noise for any given frequency. We also do not know the signal itself for that frequency because that is what we are trying to recover. However, let us pretend for a moment that we do know these two values. Then, if the noise is high for that frequency, NSR is going to be high, which means that the entire term $\boxed{1}$ is going to be small. In effect, the recovery of the signal for that frequency is attenuated. Also, if H happens to be small because the motion blur kernel itself has low power for that frequency, then, once again, this factor $\boxed{1}$ is going to be small.

Noise Suppression: Weiner Deconvolution

$$F'(u,v) = \frac{G(u,v)}{H(u,v)}\left[\frac{1}{1+\frac{NSR(u,v)}{|H(u,v)|^2}}\right]$$

$\boxed{1}$

Where:

Weiner Filter $\stackrel{\text{def}}{=}$ $$W(u,v) = \frac{1}{H(u,v)}\left[\frac{1}{1+\frac{NSR(u,v)}{|H(u,v)|^2}}\right]$$

Noise-to-Signal Ratio, $NSR(u,v)$:

$$NSR(u,v) = \frac{\text{Power of Noise at } (u,v)}{\text{Power of Signal (Scene) at } (u,v)} = \frac{|N(u,v)|^2}{|F(u,v)|^2}$$

64

We have established here that if we know the Fourier transforms of both the noise and the signal that we are looking for, we can actually develop a recovery process in which the amplification of the noise during deconvolution is suppressed.

Unfortunately, we do not know the Fourier transform of the noise nor the Fourier transform of the signal that we are trying to recover, thus NSR is unknown. It turns out, though, that if we use a carefully chosen constant value (λ) for NSR, we often obtain impressive deconvolution results.

Noise Suppression: Weiner Deconvolution

$$F'(u,v) = \frac{G(u,v)}{H(u,v)}\left[\frac{1}{1+\frac{NSR(u,v)}{|H(u,v)|^2}}\right]$$

- Determining NSR requires us to have prior knowledge of the noise "pattern" and the scene (or of a similar scene).

$$NSR(u,v) = \frac{|N(u,v)|^2}{|F(u,v)|^2}$$

- Often NSR is set to a single suitable constant λ.

$$\boxed{NSR(u,v) = \lambda}$$

65

Let's take a look at how well Weiner deconvolution works. Shown here is our example image with noise added to it, as well as the point spread function due to motion blur. We end up recovering this image with an NSR equal to 0.002. While this is not a perfect image — it has some graininess and some ringing artifacts — but, it is clearly much sharper than the captured image.

Noise Suppression: Weiner Deconvolution

Noisy, Blurred Image $g(x,y)$ | PSF $h(x,y)$ | Recovered $f'(x,y)$

$NSR(u,v) = \lambda = 0.002$ was used to recover image

66

Sampling Theory and Aliasing

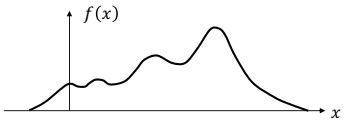
Shree K. Nayar
Columbia University

Topic: Image Processing II, Module: Imaging
First Principles of Computer Vision

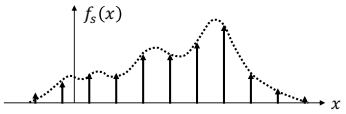
67

From Continuous to Digital Image

Continuous Signal:



Digital Signal:



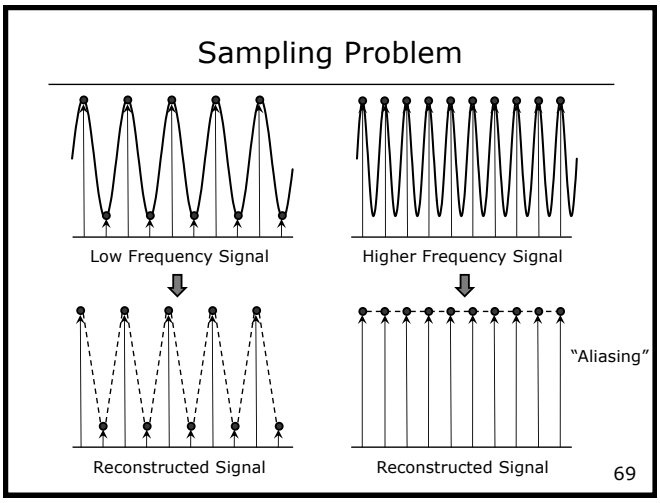
How "dense" should the samples be?

68

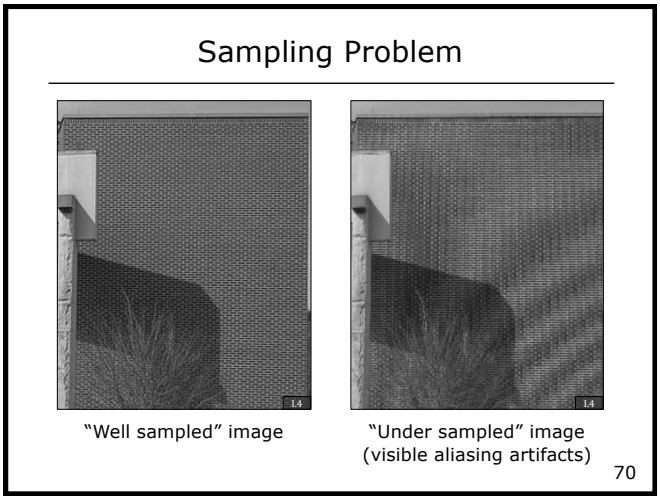
We know that the lens of the camera creates a continuous optical image on the image plane. On this image plane sits the image sensor, which converts the continuous image into a discrete digital image. In other words, the continuous image is sampled and, while this happens, we could lose information in the continuous image as well as introduce undesirable artifacts in the sampled image. So, the question is, how densely should we sample the continuous image to avoid both these phenomena.

First, let us take a look at how we go from a continuous image to a digital or discrete image. In the right slide is shown a continuous image (shown as a one-dimensional signal), formed on the image plane by the lens. We are going to uniformly sample this image so as to get the discrete digital signal shown below.

Let us use a simple example to illustrate the significance of sampling. On the left is a sinusoid, and on the right is a higher frequency version of it. Let us sample both of these functions with the same sampling frequency. The sampled (discrete) signals are shown below. We can try to reconstructed the original signals from the discrete samples using interpolations. For our purposes here we use linear interpolation, where we simply connect consecutive discrete samples using straight lines. We see that, for the low frequency signal, the reconstructed signal looks close to the original signal. Not much damage has been done. However, for the higher frequency signal, we end up getting a flat reconstructed signal. All the information is lost in this case. This phenomenon is called aliasing and results from the fact that we have under-sampled the signal. In fact, aliasing can also introduce new frequencies in the reconstructed signal, which did not even exist in the original signal. It is in our best interest, therefore, to adequately sample the image and avoid aliasing.



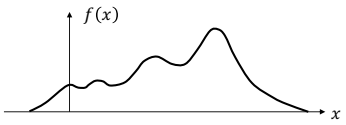
Let us look at how sampling manifests visually in images. On the left is an image of a brick wall that is well-sampled — everything looks fine in the image. When we under-sample the image, however, we end up with the ringing pattern seen in the image on the right. This is often referred to as a Moiré type of pattern. It is a common visual manifestation of aliasing.



Let us now develop a theory of sampling that reveals, for any given continuous signal, the sampling frequency needed to avoid aliasing. We start with a continuous signal and sample it by multiplying it with a train of impulse functions, or delta functions shown below. This function is called the Shah function.

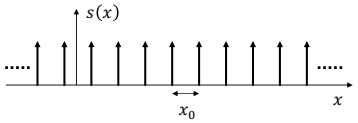
Sampling Theory

Continuous Signal:



Shah Function (Impulse Train):

$$s(x) = \sum_{n=-\infty}^{\infty} \delta(x - nx_0)$$



Sampled Function: $f_s(x) = f(x)s(x)$

71

It turns out that the Fourier transform of the Shah function is also a Shah function, as shown in the left slide below. The period of this function — the distance between consecutive delta functions — is the inverse of the period of the original Shah function.

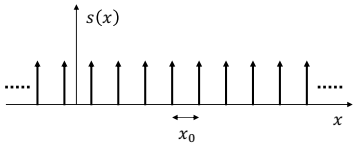
Let us take a look at what happens in Fourier domain when we sample a continuous signal. In the right slide, we have our continuous function f multiplied by the Shah function. We know that multiplication in spatial domain is equivalent to convolution in Fourier domain.

Let us say that we have a continuous image with the trapezoid shaped Fourier transform $F(u)$ shown below. We convolve $F(u)$ with the transform of the Shah function, which is also a Shah function. Given the sifting property of the delta function, we know that if we convolve any function with a single delta function, we will end up reading out that function itself. Thus, if we convolve the function $F(u)$ with the Shah function, we obtain multiple copies of the original function $F(u)$.

Shah Function (Impulse Train)

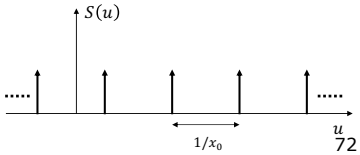
Shah Function (Spatial Domain):

$$s(x) = \sum_{n=-\infty}^{\infty} \delta(x - nx_0)$$



Shah Function (Fourier Domain):

$$S(u) = \frac{1}{x_0} \sum_{n=-\infty}^{\infty} \delta\left(x - \frac{n}{x_0}\right)$$



72

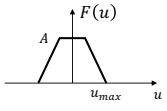
Fourier Analysis of Sampled Signal

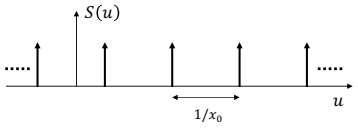
Sampled Signal:

$$f_s(x) = f(x)s(x) = f(x)\sum \delta(x - nx_0)$$

$$F_s(u) = F(u) * S(u) = F(u) * \frac{1}{x_0}\sum \delta(u - n/x_0)$$

For example:





73

Shown here are a few of the copies of $F(u)$. In the example shown here, the copies are non-overlapping. In this case, it is straightforward to exactly recover the original continuous signal $f(x)$. To do so, we can simply cut out one of the copies and find the inverse Fourier transform of it. That is, we can actually recover the original continuous signal from the discrete signal, without any loss of information.

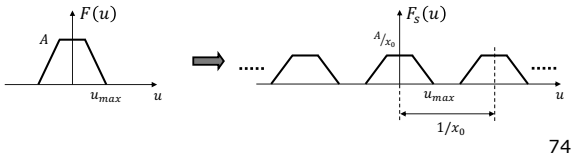
Fourier Analysis of Sampled Signal

Sampled Signal:

$$f_s(x) = f(x)s(x) = f(x)\sum\delta(x - nx_0)$$

$$F_s(u) = F(u) * S(u) = F(u) * \frac{1}{x_0}\sum\delta(u - n/x_0)$$

If $u_{max} \leq \frac{1}{2x_0}$



74

However, if the copies overlap as shown here, we end up introducing new frequencies into the transform of the sampled signal. From the overlap region, there is no way to figure out what the original function was since there are many original signals that could have created the same overlap region. The creation of these new frequencies in the overlap regions is what is referred to as aliasing. It produces undesirable artifacts in sampled images.

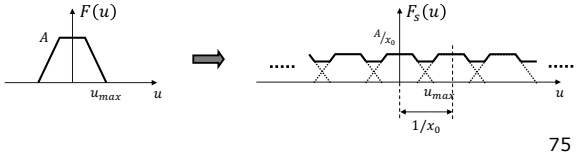
Aliasing

Sampled Signal:

$$f_s(x) = f(x)s(x) = f(x)\sum\delta(x - nx_0)$$

$$F_s(u) = F(u) * S(u) = F(u) * \frac{1}{x_0}\sum\delta(u - n/x_0)$$

If $u_{max} > \frac{1}{2x_0}$ Aliasing



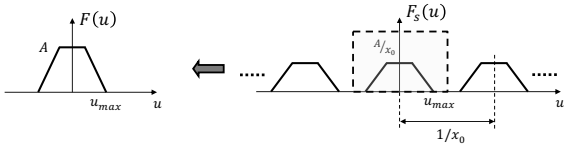
75

That brings us to the Nyquist theorem, which tells us when we have aliasing and when we do not. Note that the distance between the copies of $F(u)$ is one over the period x_0 of the sampling function $s(x)$. Let us assume that the maximum non-zero frequency in the original continuous image is u_{max}. Observe from the figure shown here that the overlap does not occur when u_{max} is lesser or equal to $1/2x_0$, which is called the Nyquist frequency. When this condition is satisfied, the signal is well-sampled and there is no aliasing or loss of information due to sampling. In this case, we can recover the original continuous signal by simply multiplying $F_s(u)$ with a rectangular function

Nyquist Theorem

Can we recover $f(x)$ from $f_s(x)$? In other words, can we recover $F(u)$ from $F_s(u)$?

Only if $u_{max} \leq \frac{1}{2x_0}$ (Nyquist Frequency)



$$F(u) = F_s(u)C(u)$$

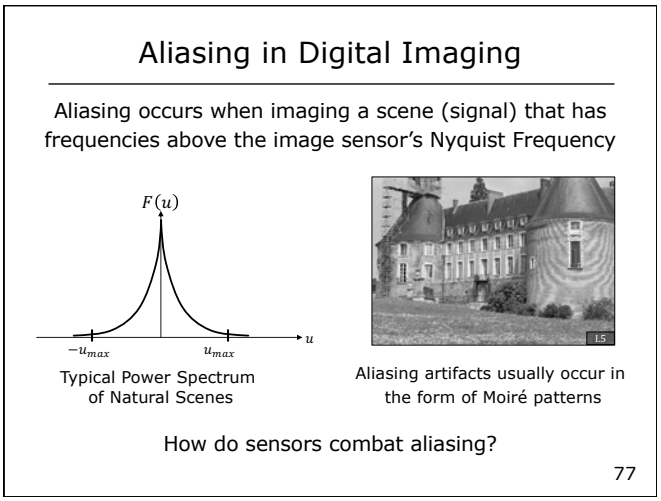
$$\boxed{f(x) = IFT(F(u))}$$

$$C(u) = \begin{cases} x_0, & |u| < 1/2x_0 \\ 0, & Otherwise \end{cases}$$

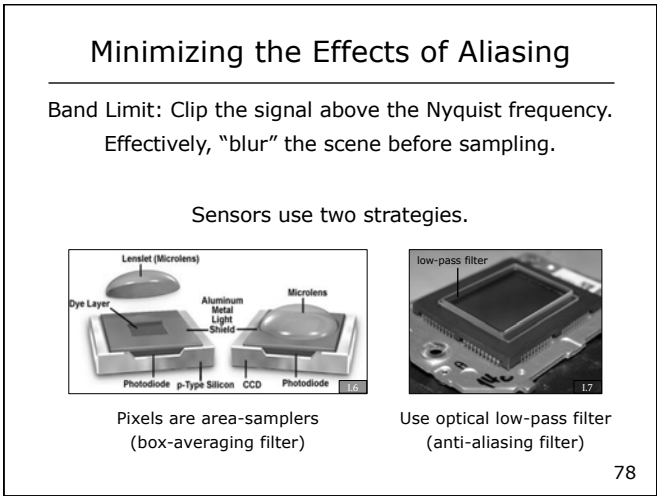
76

$C(u)$ to cut out a single copy of $F(u)$. By applying the inverse Fourier transform to the cut-out function, we get the original continuous signal $f(x)$.

Aliasing, or under-sampling, produces a variety of artifacts in images. That signal arriving at the image sensor depends on the scene that we expose the camera to. For most natural scenes, the Fourier Transform of the scene looks like the plot shown on the left — $F(u)$ falls off as a function of u, the frequency. Let us assume that the maximum non-zero frequency that falls on the image sensor is u_{max}. To avoid aliasing, we need to make sure that the sensor samples the image at a frequency that is at least two times u_{max}. On the right is another example of aliasing, where on the curved wall we can see the ringing (Moiré) effects we saw earlier. Can a camera be designed such that it can always avoid aliasing?



One factor that helps us mitigate aliasing is the inherent structure of the image sensor itself. Remember that each pixel on the image sensor has a sensing area associated with it, as shown here. Thus, we can model the discrete image as the continuous image convolved with a box filter (where the box has the size of a pixel). Then, that image is sampled to produce the final discrete image. The convolution with the box filter results in low-pass filtering of the image. It serves to cut out the highest frequencies in the continuous image. In short, the finite area of each pixel helps reduce aliasing.



One way to entirely avoid aliasing is to use an optical filter that sits on top of the image sensor (see right image). The filter is designed to cut out high frequencies, or specifically those frequencies that are above the Nyquist frequency. In this case, the captured image with be free of aliasing, irrespective of the scene.

References and Credits

Shree K. Nayar

Columbia University

Topic: Image Processing II, Module: Imaging

First Principles of Computer Vision

79

References: Textbooks

The Fourier Transform and Its Applications
Bracewell, R. N., Mc-Graw Hill

Digital Image Processing (Chapter 3)
González, R and Woods, R., Prentice Hall

Computer Vision: Algorithms and Applications (Chapter 3)
Szeliski, R., Springer

Robot Vision (Chapter 6 and 7)
Horn, B. K. P., MIT Press

Computer Vision: A Modern Approach (Chapter 7)
Forsyth, D and Ponce, J., Prentice Hall

80

References: Papers

[Oppenheim 1983] A. V. Oppenheim. J. S. Lim, and S. R. Curtis, "Signal synthesis and reconstruction from partial Fourier-domain information," Journal of the Optical Society of America, Vol. 72, page 1413, November 1983.

[Oliva 2006] A. Oliva, A. Toralba, and P. G. Schyns, "Hybrid Images," ACM Transactions on Graphics, Vol. 25, No. 3, pages 527-532, 2006.

81

Image Credits

I.1 http://en.wikipedia.org/wiki/File:Fourier2.jpg. Public Domain.
I.2 MathWorks.
I.4 Licensed under CC BY SA 3.0.
I.5 http://www.svi.nl/wikiimg/StFargeaux_kasteel_buiten1_aliased.jpg. Huygens software: Scientific Volume Imaging B.V. Used with permission.
I.6 http://hamamatsu.magnet.fsu.edu/articles/microlensarray.html.
I.7 Astrosurf. Used with permission.
I.8 https://en.wikipedia.org/wiki/Hybrid_image.

82

Acknowledgements: Thanks to Nisha Aggarwal and Jenna Everard for their help with transcription, editing and proofreading.

References

[Bracewell 1963] The Fourier Transform and Its Applications, Bracewell, R. N., Mc-Graw Hill, 1963.

[Horn 1986] Robot Vision, Horn, B. K. P., MIT Press, 1986.

[Forsyth and Ponce 2003] Computer Vision: A Modern Approach, Forsyth, D and Ponce, J., Prentice Hall, 2003

[González and Woods 2009] Digital Image Processing, González, R and Woods, R., Prentice Hall, 2009.

[Oppenheim 1983] A. V. Oppenheim. J. S. Lim, and S. R. Curtis, "Signal synthesis and reconstruction from partial Fourier-domain information," Journal of the Optical Society of America, Vol. 72, page 1413, November 1983.

[Oliva 2006] A. Oliva, A. Toralba, and P. G. Schyns, "Hybrid Images," ACM Transactions on Graphics, Vol. 25, No. 3, pages 527-532, 2006.

[Nayar 2022B] Image Formation, Nayar, S. K., Monograph FPCV-1-1, First Principles of Computer Vision, Columbia University, New York, February 2022.

[Nayar 2022C] Image Sensing, Nayar, S. K., Monograph FPCV-1-2, First Principles of Computer Vision, Columbia University, New York, February 2022.

[Nayar 2022D] Binary Images, Nayar, S. K., Monograph FPCV-1-3, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022E] Image Processing I, Nayar, S. K., Monograph FPCV-1-4, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022F] Image Processing II, Nayar, S. K., Monograph FPCV-1-5, First Principles of Computer Vision, Columbia University, New York, March 2022.

[Nayar 2022G] Edge Detection, Nayar, S. K., Monograph FPCV-2-1, First Principles of Computer Vision, Columbia University, New York, May 2022.

[Nayar 2022H] Boundary Detection, Nayar, S. K., Monograph FPCV-2-2, First Principles of Computer Vision, Columbia University, New York, June 2022.

[Nayar 2022I] SIFT Detector, Nayar, S. K., Monograph FPCV-2-3, First Principles of Computer Vision, Columbia University, New York, August 2022.